Reconstructing proprietary video streaming algorithms

Maximilian Grüner, Melissa Licciardello, and Ankit Singla, *ETH Zürich*

https://www.usenix.org/conference/atc20/presentation/gruener

This paper is included in the Proceedings of the
2020 USENIX Annual Technical Conference.

July 15–17, 2020

978-1-939133-14-4

Open access to the Proceedings of the
2020 USENIX Annual Technical Conference
is sponsored by USENIX.

Reconstructing proprietary video streaming algorithms

Maximilian Grüner, Melissa Licciardello, Ankit Singla
Department of Computer Science, ETH Zürich

Abstract

Even though algorithms for adaptively setting video quality in online streaming are a hot topic in networking academia, little is known about how popular online streaming platforms do such adaptation. This creates obvious hurdles for research on streaming algorithms and their interactions with other network traffic and control loops like that of transport and traffic throttling. To address this gap, we pursue an ambitious goal: reconstruction of unknown proprietary video streaming algorithms. Instead of opaque reconstruction through, *e.g.,* neural networks, we seek reconstructions that are easily understandable and open to inspection by domain experts. Such reconstruction, if successful, would also shed light on the risk of competitors copying painstakingly engineered algorithmic work simply by interacting with popular services.

Our reconstruction approach uses logs of player and network state and observed player actions across varied network traces and videos, to learn decision trees across streaming-specific engineered features. We find that of 10 popular streaming platforms, we can produce easy-to-understand, and high-accuracy reconstructions for 7 using concise trees with no more than 20 rules. We also discuss the utility of such interpretable reconstruction through several examples.

1 INTRODUCTION

Video streaming is one of the most popular Internet services today, forming a majority of downstream Internet traffic [35]. Naturally, there is great interest in optimizing video delivery. One particularly well-studied problem is that of designing adaptive bitrate algorithms that adjust video quality in response to changes in network bandwidth. ABR algorithms attempt to maximize some notion of quality-of-experience, which typically combines video playback metrics like average video quality, and playback interruptions and stability.

The problem of maximizing QoE in video streaming is crisply defined, intellectually interesting, and practically valuable. Thus, numerous ABR algorithms have been suggested in recent work to tackle it, *e.g.,* Oboe [3] and MPC [46]. However, little is known about the proprietary algorithms actually deployed in widely used video streaming services such as YouTube, TwitchTV and Netflix.[1] We attempt to address this gap by exploring whether it might be possible to learn such algorithms by controlled observation of video streams.

Our goal is to produce ABR controllers that: (a) mimic the observed behavior of ABR logic deployed in target online video services across a wide range of network conditions and videos; and (b) are open to easy manual inspection and understanding. Note that the latter precludes the direct use of blackbox machine learning techniques like neural networks.

We are motivated by three factors. First, this effort helps understand the risk of competitors copying painstakingly-engineered algorithmic work simply by interacting with popular, public-facing front-ends. Second, being able to reconstruct widely deployed algorithms would allow head-to-head comparisons between newly proposed research ABRs and industrial ABRs, something lacking in the literature thus far. Third, given that video is the majority of Internet traffic, this traffic being controlled by unknown proprietary algorithms implies that we do not understand the behavior of most Internet traffic. This makes it difficult to reason about how different services share the network, and interact with other control loops such as congestion control and traffic shaping.

The above use cases help sharpen the goals for our reconstruction effort. Simplifying our task is the fact that instead of exact algorithm recovery, we need functional equivalence of a reconstruction with its target algorithm over a large, varied set of inputs – note that the same set of outcomes could be arrived at by two substantially different algorithms, making exact recovery of a particular algorithm impossible. However, our use cases also impose a difficult additional requirement: our reconstructions must be human-interpretable, allowing not only the mimicking of observed behavior, but also manual inspection and understanding. A competitor seeking to copy the ABR logic of an online service needs interpretability to be able to modify it as necessary for their use.[2] They would also like to ensure the robustness of the obtained logic, something that is difficult with blackbox learning — prior work has shown corner cases with undesirable behavior in blackbox learning methods applied to networking [17]. Likewise, in terms of comparisons between industrial and academic ABRs, we would not only like to observe the performance

[1] Researchers at Netflix published, in 2014, work on this problem [15], including tests on their commercial deployment. Per our conversations with them, their current deployment incorporates some features of this published work, but they are unwilling to share more details, including the differences from this published approach.

[2] Our work enables an understanding of whether this risk *exists*: "Can a competitor reconstruct an ABR in a meaningfully beneficial, robust way?" We leave the question of how this risk may be tackled to followup work.

differences empirically, but also understand where they stem from. Lastly, reasoning about interactions with other network control loops and competing services also requires having a richer understanding of the control logic under study than blackbox learning can provide.

Algorithmic reconstruction of this type is an ambitious goal, with the current tools available for general-purpose program synthesis still being fairly limited. However, there are two reasons for optimism if we can suitably narrow our scope: (a) the core of ABR algorithms involves a small set of inputs, and has a limited decision space; and (b) it is easy to collect large amounts of curated data for analysis.

Our approach automatically generates concise, human-interpretable rule-sets that implement ABR by learning from an existing target ABR algorithm. These rule-sets map the client and network environment, video features, and state over the connection, to a video quality decision for the next video chunk. To obtain generalizable, succinct, and interpretable pseudocode in a reconstruction, we find that it is insufficient to directly use sophisticated techniques from imitation learning [5, 34]. As we shall show later, such methods can either mimic the behavior of a target accurately with a large set of complex rules, or, when limited to a small set of rules, lose accuracy. Our approach sidesteps this tradeoff by embedding suitable domain knowledge in the learning mechanism: framing intuitive primitives familiar to domain experts, and making them available to the learning mechanism, results in rule-sets that are accurate, concise, and meaningful.

We use our approach to obtain concise reconstructions that can successfully mimic the decision-making of several target academic and industry ABR algorithms, achieving high agreement with their decisions and similar video QoE behavior. Of the 10 online streaming services we evaluate across, our reconstruction achieves behavior similar to its target for 7 services. In each case, we produce a concise decision-tree with 20 or fewer short rules, using primitives that are intuitive and easy to understand. We also explain the reasons for failure for the remaining 3 services.

We make the following contributions:

- We describe an approach for deriving accurate *and* concise rule sets for ABR, using a corpus of decision outcomes over network traces and videos. Our approach handles the complex output space corresponding to diverse video encodings, as well as noise in the data.
- We apply our method to the reconstruction of algorithms deployed in 10 popular streaming services. For 7 services, we successfully achieve high agreement with their decisions and closely similar streaming behavior.
- The rule sets we obtain are concise, with 20 or fewer rules in each case. Our code also generates a loose natural language translation, which we used extensively in understanding problems and improving performance.
- We also expose a likely fundamental compromise necessary for interpretable and effective learning: the time-consuming encoding of domain knowledge.
- Our code and reconstructed ABRs are open-source [12].

Beyond the above results, our ambitious effort raises several exciting questions for future exploration, such as: (1) on the tradeoffs between the effort invested in embedding domain knowledge, and the quality of the inferred pseudocode; (2) to what extent such domain knowledge may itself be learnt from a corpus of hand-designed algorithms broadly from the networking domain; (3) applying our approach to other networking problems, like congestion control, and newer problems where we have more limited experience, such as multipath transport; (4) and how online service providers may obscure their logic against reconstruction, if so desired.

2 RELATED WORK

Numerous high-quality ABR proposals have appeared just within the past few years [3, 11, 31, 37, 45], but relatively little is known about widely deployed industrial ABR algorithms.

There is a large body of work on reconstructing unknown algorithms. One may approach this using code analysis, like Ayad et al.'s analysis of Javascript code for some online video services [16]. However, some targets can be too large and obfuscated for such analysis – YouTube, for instance, comprises 80,000+ lines of obfuscated Javascript. We used JS NICE [36], the state-of-the-art in Javascript deobfuscation, but even coupled with a step-through debugger and with help from the authors of JS NICE, this provided little insight – ultimately, manually examining such a large piece of code with meaningless variable names to reconstruct its functionality seems futile. It also has the downside of potentially requiring substantial rework for even small changes in the target. Even more fundamentally, the code may not be available at the client at all, with decision-making residing on the server side.

Several prior efforts have used manual experimentation and analysis for dissecting the behavior of a variety of online services [2, 9, 16, 19, 21, 27, 44]. For instance, Mondal et al. [27] used network traces to experimentally study the behavior under changing network conditions, and then manually draw coarse inferences, such as that YouTube's requested segment length varies with network conditions. An earlier effort on inferring Skype's adjustment of its sending rate [19], was based on the researchers making experimental observations, then manually hypothesizing a control law, and finally tuning its parameters to fit the data. Our own parallel measurement study [21] experimentally examined the behavior of several deployed ABR algorithms in terms of metrics like stability of playback and convergence time after bandwidth changes. In concurrent work, Xu et al. [43] propose a method for inferring the quality of video chunks downloaded within encrypted streams, and apply it to experimentally study the streaming outcomes in response to different traffic throttling

schemes. In contrast to all the above efforts, our goal here is to *automatically generate logic that mirrors a target ABR algorithm's behavior* by observing the target ABR's actions in response to variations in the environment and inputs.

There are also efforts in networking to inspect the internals of learning-based networked systems. This work is not directly applicable to our goal of reconstructing arbitrary ABRs, which are most likely non-ML, and more importantly, are not available to us. However, one could first train a blackbox-ML algorithm to mimic any reconstruction target, and then use such tools. Recent work on inspecting [10] or verifying [17] systems built using ML has examined Pensieve [23]. The authors frame hypotheses/questions about the system's behavior, and then evaluate them. However, this (a) requires knowing what hypotheses to examine, and (b) does not yield a reconstruction. Among efforts in this vein, the most closely related are the concurrent TranSys [26] and PiTree [25] studies. PiTree focuses on converting ABR algorithms to decision trees, and TranSys broadens this approach to NN-based strategies in networking. Both are networking applications of a broader paradigm in ML, which we discuss next.

Beyond networking efforts, *imitation learning* is a rich discipline in its own right. Most work in this direction uses (uninterpretable) neural networks [6, 14, 42], but recent work has also developed model-free approaches to approximate the learned neural network via, *e.g.,* a decision tree [5, 34]. As we show later in §6.2, directly using this approach (like TranSys and PiTree) does not meet both of our accuracy and interpretability goals simultaneously, instead requiring the sacrifice of one or the other. While complex decision trees, with a large number of rules with many literals, can robustly imitate a target algorithm, they are difficult, if not impossible, for even domain experts to understand and work with. On the other hand, restricting the complexity of the generated trees results in a loss of imitation accuracy and robustness. While the expressiveness and compactness of these approaches can be improved by employing genetic algorithms to craft features for use therein [13], this often leads to both overfitting, and complex, non-intuitive features.

Lastly, program synthesis is a rich and growing field. While we use one particular strategy for ABR reconstruction, there are other tools we plan to examine in future work. The most promising perhaps is recent work combining learning with code templates [41], where the core idea is to modify templates to minimize the distance from a target learning algorithm. An alternative "deductive synthesis" approach, as employed in Refazer [33], could also be fruitful.

To the best of our knowledge, our work is the first to attempt an interpretable reconstruction of unknown deployed ABRs.

3 DATA PREPARATION

We extend a trace collection harness that we built for a measurement study, where we used manual analysis to comment on the behavior of deployed ABR algorithms across 10 streaming platforms [21].

We launch a video stream on a target service, and according to an input network trace, shape the throughput at the client using Linux `tc`. We record the current client buffer occupancy, the video chunk qualities played out, video metadata, etc. The client buffer occupancy is directly measured through the instrumentation of the HTML5 player element. If the HTML5 player element were not available, we could instead use the captured HTTP chunk requests (locally at the client, making encryption irrelevant) to reconstruct the buffer occupancy — this strategy may be of use for future work exploring mobile ABR implementations. This alternative can be less accurate though, as "redownloading" (*e.g.,* to replace already downloaded low-quality chunks in the client player buffer by higher-quality ones) introduces an ambiguity into which chunk is actually played.

For each platform, by appropriate tailoring of HTTP requests, we also fetch all chunks for the test videos at all qualities, such that we can use these videos in an offline simulation, allowing the learned ABR to make choices different from those in our logs, as well as to enable us to study the behavior of academic ABRs. Ultimately, we obtain the following measurements:

- $\mathbb{C}_t$: segment size (Mb) downloaded for request t
- $\mathbb{R}_t$: segment bitrate (Mbps) for request t
- $\mathbb{V}_t$: segment VMAF[3] for request t
- $\mathbb{D}_t$: download time for request t
- $\mathbb{Q}_t$: quality level requested in request t
- $\mathbb{S}_t$: segment length (seconds) downloaded for request t
- $\mathbb{P}_t$: Percent of the video played at request t
- $\mathbb{B}_t$: buffer size (seconds) when requesting t
- $\mathbb{RB}_t$: rebuffer time (seconds) when requesting t
- $\mathbb{C}^i_{t+n}$: segment size of quality i for n^{th} chunk after t
- $\mathbb{R}^i_{t+n}$: segment bitrate of quality i for n^{th} chunk after t
- $\mathbb{V}^i_{t+n}$: segment VMAF of quality i for n^{th} chunk after t

4 RULE-SET BASED INFERENCE

We shall first consider a motivating example for why rule-sets are a simple and potent representation for our type of target algorithms, and then present our recipe for constructing succinct rule-sets that capture the target algorithm's behavior.

4.1 Motivating example

Let us examine a simple throughput-based ABR algorithm, similar to that described in prior work [15]. It uses only the throughput estimate for the last video chunk fetched, $\mathbb{T}_{N-1}$,

[3] VMAF is a video perceptual quality metric [20].

$$\text{quality } 0 \rightarrow \mathbb{T}_{N-1} \leq 4.99$$
$$\text{quality } 1 \rightarrow \mathbb{T}_{N-1} > 4.99 \;\&\; \mathbb{T}_{N-1} \leq 6.97$$
$$\text{quality } 2 \rightarrow \mathbb{T}_{N-1} > 6.97$$

Fig. 1: Minimal rule-set for a reservoir-based algorithm, which uses only the last chunk's throughput estimate to pick a quality level.

and two thresholds: *reservoir* and *cushion*. If $\mathbb{T}_{N-1} <$ *reservoir*, the lowest quality is served. If $\mathbb{T}_{N-1} >$ *reservoir* + *cushion*, the highest quality is served. For other values of $\mathbb{T}_{N-1}$, quality is linearly interpolated between these levels.

This algorithm, regardless of its specific instantiation with particular values of the thresholds, can be easily expressed as a set of rules. For a simple instantiation with only 3 quality levels, and both *reservoir* and *cushion* set to 4 Mbps, this rule-set is shown in Fig. 1.[4] The rule-set is inferred (which is why the rules contain imprecise values like 6.97) by the process we shall describe shortly.

We caution the reader against concluding from this small motivating example that only simple, stateless, "templates with parameters / thresholds" type of algorithms can be expressed in this way. Rule sets are capable of capturing complex stateful behavior, as long as primitives encoding this state are made available for their use.

4.2 Decision trees and rules

We first learn binary decision trees [18] that encode conditions for specific outputs, *e.g.,* the video quality levels requested. Further, in such a decision tree, each path from the root to a leaf can be naturally interpreted as a descriptive rule capturing the conditions for the outcome at the leaf to be reached.

Consider a single-threshold decision: "If throughput < 5 Mbps, pick low quality; otherwise, pick high quality.". This can be captured in a 3-node tree with the conditional at its root, and the two outcomes as leaves. In this case, the rule-lengths, *i.e.,* the path lengths from the root to the leaves, are 1; and so is the number of "splits" in the tree.

Fig. 2 shows a more complex target decision plane with two inputs (along the x and y dimensions), where there are still only two outcomes (labels), but the data samples that map to these labels are separated by more complex logic. If we constrain the decision tree that approximates this decision plane to use rules of only length one, we can use only one line separating the labels, as shown in the top-left smaller figure. Allowing more and more complex (longer) rules, allows a tighter representation of the target decision plane. Of course, using too many rules risks overfitting, especially under noisy data that is typical in networking. Fortunately, our goal to obtain concise rule sets aligns well with that of avoiding overfitting and preserving generalization.

[4] Readers may expect the rule-set in Fig. 1 to mean that *reservoir* = 5 and *cushion* = 2. The discrepancy stems from the discreteness of the interpolation: for some $\mathbb{T}_{N-1} >$ *reservoir i.e.,* $\mathbb{T}_{N-1} \in [4,5]$, quality 0 will be chosen.

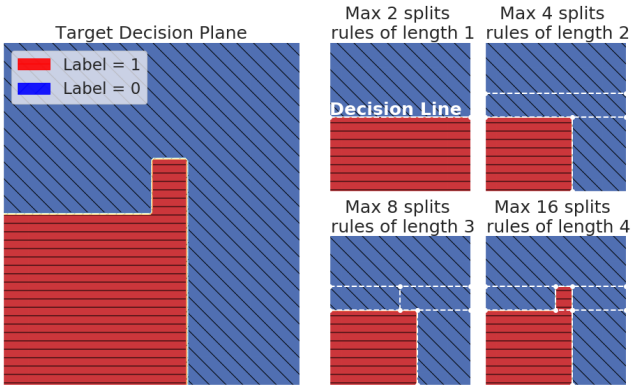

Fig. 2: The big image is the target decision plane, with 2 labels. On the right are its approximations with decision trees of different rule-lengths, going from 1 to 4.

Framing the output space: A key design consideration in using decision trees is the framing of the output decision space. Suppose we frame decision outcomes in terms of the video quality level that the client should fetch the next video chunk at. *If* all the videos being served were encoded with the same quality levels, both in terms of number and their bitrates, *e.g.,* 6 video qualities at bitrates of {200, 450, 750, 1200, 2350, 4300} Kbps, this would be easy: there are 6 *a priori* known outcomes that we can train decision trees for.

However, this is clearly overly restrictive: in practice, ABR algorithms must tackle a variety of videos encoded at different bitrates. The set of different bitrates at which a video is encoded in is referred to as its "bitrate ladder". Providers like Netflix even use per-video customization of bitrate ladders, varying the number and separation of bitrate levels [1]. This diversity in the output space is a challenge for learning approaches: what should we present as the potential output decision space? It is noteworthy that Pensieve [23] does not fully tackle this challenge, instead restricting the video bitrate levels to a small pre-defined set.

To overcome this issue, we formulate the decision process in terms of video quality being upgraded or downgraded relative to the current video quality. With one decision, a maximum of n quality shifts are permitted in either direction, with n being a tunable parameter. Of course, this prevents us from capturing some algorithms, where larger quality changes (than n) may be enforced in a single step. However, this is atypical, as video streaming typically targets smooth output. Even if some algorithm involves such decision making, our approach only differs from such a target's decisions transiently. This small compromise allows us to generalize to arbitrarily diverse video bitrate ladders.

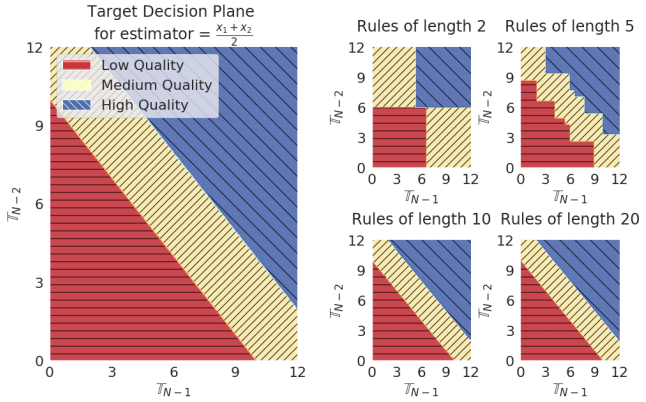

Fig. 3: Here, the target decision plane (big, left) is governed by the mean of $\mathbb{T}_{N-1}$ and $\mathbb{T}_{N-2}$. The smaller figures show that we need long rules to approximate this if we are restricted to using individual literals ($\mathbb{T}_{N-1}$ and $\mathbb{T}_{N-2}$) in our rules.

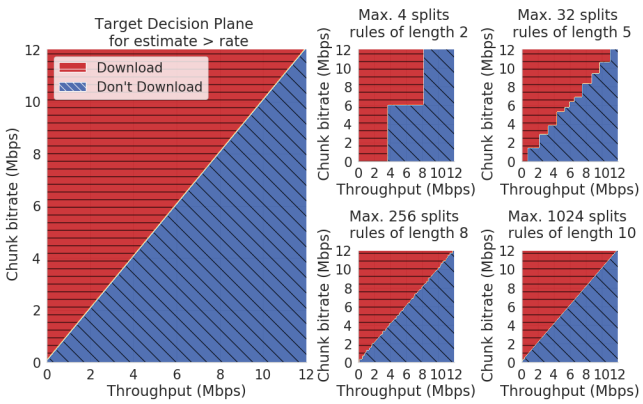

Fig. 4: A decision plane comparing the throughput estimate to the chunk bitrate is difficult to capture without long rules if rules can only be framed in terms of the individual literals.

4.3 Feature engineering

Applying textbook decision-tree inference, with the above framing, one can already infer simple algorithms. However, as we shall see, appropriate customization based on domain expertise is crucial to obtain concise and generalizable rules.

Consider, for instance, a target algorithm that uses the mean throughput across the last 2 video chunks fetched. Naively learnt rules will then contain complex conditionals across both $\mathbb{T}_{N-1}$ and $\mathbb{T}_{N-2}$. Fig. 3 shows this for rules of increasing length, up to 20. The target decision plane uses the mean, $\frac{\mathbb{T}_{N-1}+\mathbb{T}_{N-2}}{2}$ to decide between three video qualities. Rules of length 2 and 5 yield poor accuracy, necessitating much longer (complex) rules.

Of course, if we knew that the building block for throughput estimation is the mean, we could simplify such rules substantially by expressing them in terms of the mean. Thus, we can consider adding common estimators, based on our domain knowledge, to the feature-set available to our learning pipeline. Then, instead of only learning across primitive literals (like each individual chunk's throughput), more compact representation across these non-primitive literals becomes possible. We thus explore three classes of such non-primitive features that are intuitive and likely commonplace in ABR, and even more broadly, other networking algorithms.

Throughput estimators: Clearly, having the most accurate estimate of network throughput is advantageous in deciding on video quality. As such, throughput estimators are potentially useful building blocks. We consider two types of estimators. The first type is only parametrized by the number of past throughput measurements it aggregates using a mean, median, harmonic mean, etc., while the second type involves additional tunable parameters, such as the weight decrease, α, in an exponential weighted moving average (EWMA), which sets the relative weight of a new measurement compared to old measurements.

Encoding these estimators with a range of different parameter choices gives us a large set of features ranging from nearly stateless (*e.g.,* using only the last chunk's throughput) to those with long-term state (*e.g.,* a moving average). In addition to throughput, we also construct features capturing the *variation* across recent throughput measurements as it characterizes the stability of the network.

Comparisons: Decisions often depend on not just thresholding of certain primitive or non-primitive features, but also on comparisons among features. For instance, generalizing even a simple rate-based algorithm to work for arbitrary videos encoded with different bitrate ladders requires a comparison: is the throughput larger than a particular step in the bitrate ladder? Unfortunately, while decision trees can capture such comparisons using only primitive features, they require a large number of rules to do so. This is shown in Fig. 4, where the decision trees with rules of length 2 and 5 do not accurately represent a simple comparison-based target decision plane.

Thus, we must encode potential comparisons of this type as non-primitive features. These can also be parameterized in a similar manner to the throughput estimators discussed above, *e.g.,* by what *factor* should one feature exceed another.

Planning ahead: ABR, like many other control tasks, is not only about making one decision in isolation, but also considering its longer-term influence. For instance, it might be interesting to estimate the max, min, or mean rebuffering one should expect given the decision to download a chunk at a certain quality, assuming the throughput stays the same. We design features along these lines, such as QoE in MPC [46].

More features? Over time more features can be added to enhance our approach without having to reason about their mutual interactions, as would be the case with incorporating new ideas into human-engineered routines. One could also extend this approach by adding automatically engineered features [13]. However, maintaining interpretability would require limiting the complexity of auto-generated features.

5 IMPLEMENTATION

We implement the rule inference pipeline in Python3. For the decision tree, we use the standard implementation provided by the scikit-learn library [30]. If not otherwise mentioned we use a maximum of 20 rules and limit one-step changes in quality to upgrading or downgrading by at most 2 quality levels. The 20-rule limit is somewhat arbitrarily chosen as a quantitative threshold for interpretability, but we also find that for our approach, more rules do not improve performance substantively in *most* cases. This threshold is essentially a hyperparameter that could be tuned by practitioners based on where they seek to operate in the tradeoff space involving interpretability, avoiding overfitting, and performance.

Baselines: To put our results in context, we compare them against three neural network approaches, both as-is (blackbox approaches, always represented by a recursive neural network with GRU cells [7]), and translated to decision trees. The first blackbox approach is the simplest, attempting to directly copy the decisions in a supervised learning setting. The other two use more sophisticated imitation learning methods [14, 42].

For translating the blackbox approaches into decision trees, we test two state-of-the-art methods [5, 34]. One of these [5] needs a reward function. In the vein of other imitation learning approaches, we use a clustering algorithm to assign a similarity reward to every sample. In our implementation we use an isolation forest [22] implemented in scikit-learn [30] with the standard parameters as our clustering approach. At every training step, we sample 3000 samples (as this gave the best results) according to the cluster weighting. We also tried changing the weighting function to a more agnostic divergence measure as the proposed decision by the blackbox approach might not always be what the original algorithm had in mind. This makes the sampling approach more robust.

For each reconstruction, when we compare results to our approach, we use the best blackbox approach and the best tree-translation. Thus, we invested substantial effort in implementing sophisticated baselines from the ML literature.

We also test whether our approach benefits from learning from the blackbox, instead of directly from the data. We find that this yields only minor improvements for 2 of our 10 reconstruction targets. We also explore learning in two passes, where in the first pass, we learn a tree over engineered features, and use a classifier to label its decisions in terms of their similarity to decisions made by the reconstruction target. In the second pass, we re-learn across weighted data samples, such that samples corresponding to more similar decisions are weighted higher. This approach also results in only minor improvements for one of our ten reconstruction targets.

Feature engineering: We instantiate our features (§4.3) with appropriate parameter ranges as below. 'Action' refers to quality decisions, such as maintaining quality, or increasing or decreasing it by up to n quality levels. The 'any' operator instantiates all options for a parameter or primitive.

1. Standard deviation, mean, harmonic mean, EWMA, and q^{th} percentile over the last $\boldsymbol{n}$ chunks, with $n \in \{1 \ldots 10\}$. Additionally, for EWMA, $\boldsymbol{\alpha} \in \{0.15, 0.35, 0.55, 0.75, 0.95\}$.
2. For q^{th} percentile, $\boldsymbol{q} \in \{15, 35, 55, 75, 95\}$.
3. Reward $\mathcal{R}$ achievable by planning ahead 3 steps for **any** action with **any** throughput estimate. The 'any' operators here imply that we have numerous reward features, each of which combines one of the many available throughput estimators (from 1. and 2. above) with one of the possible actions. As the reward function, we use the linear QoE function introduced by Yin et al. [46], which converts bitrate, buffering and bitrate change per chunk downloaded into a score. Note that this is not necessarily what any of our reconstruction targets is optimizing for – each provider may have their own reward goals. We use this feature simply as a compact representation of QoE components.
4. Fetch time for **any** action, **any** throughput estimate.
5. Bitrate gained weighted by the buffer filling ratio for **any** action, **any** throughput estimate. Intuitively, this captures the gain in bitrate relative to its cost, *i.e.,* how much the buffer is drained by an action if throughput stays the same.
6. VMAF gained weighted by the buffer filling ratio for **any** action, **any** throughput estimate. Same as above, but with VMAF.

Ultimately we make $\sim$ 1300 features available to the learner. Note the multiplicative effect of the **any** operator above.

Throughout, we use a standard training, validation, and testing methodology. The test set contains two videos combined at random with 60 traces randomly sampled from the overall set; these 60 traces are *neither* in the training nor in the validation set. We only discuss results over the test set.

Automated Feature Engineering: As a comparison and future outlook on the possibility of automated feature engineering, which has shown promise in other applications [13], we also coarsely implement an automated feature generator. This generator recombines the raw features in an iterative fashion so that the most used features "survive" and get recombined and the others "die" out. We use the library gplearn [39] with basic mathematical operators as usable functions. We limit the iterations to $s \in [50, 100, 150]$ seconds to avoid overfitting.

6 EVALUATION

We summarize below the experiments we conducted as well as their top-line conclusions:

1. How well can we reconstruct target algorithms? We can mimic the decision-making of 7 of 10 targets to a high degree, and obtain high similarity scores.
2. What influence does domain knowledge have? Certain engineered features are crucial to obtain rules that gen-

eralize beyond training data, are concise, and achieve similar QoE as the target algorithms.

3. How interpretable and generalizable are the output rule sets? We find that we can easily spot flaws in the learned algorithm and propose ways to adapt it. Further, trees with only 20 leaves suffice in most cases.
4. How do deployed ABRs compare to academic ones? We find that academic ABRs generally outperform industrial ones, with the caveat that our evaluation uses metrics from academic work. Interestingly, we observe that one provider's algorithm shows behavior closely matching the well known BOLA algorithm, indicating potentially that this provider uses BOLA or a derivative of it.

6.1 Experimental methodology

Target platforms: We use the same 10 popular streaming platforms we used in our measurement study of deployed ABRs [21]. While certainly not exhaustive, this is a diverse set of platforms, including some of the largest, such as Twitch and YouTube; some regionally focused, such as Arte, SRF, and ZDF; and some serving specific content verticals, such as Vimeo (artistic content), TubiTV (movies and TV), and Pornhub and XVideos. We exclude Netflix, Hulu, and Amazon Prime because their terms of service prohibit robotic interaction with their services [21].

Different platforms encode content at varied resolutions and number of resolutions, ranging from 3 quality levels for TubiTV to 6.5 on YouTube (on average across our test videos; YouTube has available resolutions for different videos.) For Twitch, which offers both live streams and video-on-demand of archived live streams, we only study the latter, as live streaming is a substantially different problem, and a poor fit with the rest of our chosen platforms. For several of the providers we study, there are multiple implementations, such as for desktop browsers, mobile browsers, or mobile apps; we only attempt reconstruction for the desktop versions.

We also evaluate our ability to emulate well-known academic approaches for ABR. We use the Robust-MPC (henceforth, just MPC throughout) and Multi-Video Pensieve (henceforth, NN, because it uses a neural network approach) implementation provided by the authors of the Pensieve paper [24]. We train and test these approaches on the Twitch video data set. To speed up our experiments, for MPC, we use a lookahead of 3 chunks instead of 5, finding that this did not make a large difference in performance.

Videos: The type of content can have a substantial bearing on streaming performance, *e.g.,* videos with highly variable encoding can be challenging for ABR. We thus used a set of 10 videos on each platform. Where a popularity measure was available, we used the most popular videos; otherwise, we handpicked a sample of different types of videos. Videos from each platform are encoded in broadly similar bitrate ranges, with most differences lying at higher qualities, *e.g.,* some content being available in 4K.

Network traces: Our experiments use synthetic and real-world traces from 3 datasets in past work [3, 8, 32]. Unfortunately, a full cross-product of platform-video-trace would be prohibitively expensive — the FCC traces [8] alone would require 4 years of streaming time. To sidestep this while still testing a diversity of traces, we rank traces by their throughput variability, and pick traces with the highest and lowest variability, together with some randomly sampled ones.

Our final network trace collection consists of the 5 least stable, 5 most stable, and 20 random traces from the Belgium trace collection [40]; and 10 most/least stable ones plus 25 random traces from each of the Norway [32], the Oboe [3] and the FCC datasets.[5] We also use 15 constant bandwidth traces covering the range from 0.3 to 15 Mbps uniformly. Lastly we add 10 step traces: after 60 seconds of streaming we suddenly increase/drop the bandwidth from/to 1 Mbps to/from 5 values covering the space from 1.5 to 10 Mbps uniformly. If a trace does not cover the whole experiment, we loop over it.

In total, we use 190 traces with throughput (average over time for each trace) ranging from 0.09 to 41.43 Mbps, with an average of 6.13 Mbps across traces. Note that we make no claim of our set of traces being representative; rather our goal is to test a *variety* of traces.

Evaluation metrics: For training our approach and evaluating its accuracy in a manner standard in learning literature, we use two metrics: one measures *agreement*, and another the *similarity* of sets of decisions. We train towards maximizing the harmonic mean of these. Additionally, for our ABR-specific use-case, we evaluate the video quality of experience [28].

Agreement, F_1 score: For each output decision, we compute the precision and recall of the inferred algorithm against its target. The F_1 score is the harmonic mean of these. $F_1 \in [0,1]$, with 1.0 being optimal. We compute an average over the F_1 scores across the labels in an unweighted fashion.

What is high/low agreement? If we were not interested in interpretability, we could obtain a procedure that mimics any target algorithm by using blackbox learning. We can think of the agreement such a blackbox approach achieves with its target as a baseline free of our conciseness constraint. On the other end of the spectrum, if the inferred rules do not achieve substantially higher agreement with the target than a generic 'reasonable' algorithm, then they are useless: this implies any reasonable algorithm would make at least that many decisions similar to the target. We use the simple rate-based approach as the concrete stand-in for this generic reasonable algorithm.

Similarity: As we cannot assume anything about how each provider designs their algorithm, we must use an agnostic approach in evaluating whether the experience under our reconstruction and the actual ABR is the same. Thus, we choose, as is typical in imitation learning, to learn whether the ex-

[5] Specifically, the stable collection from September 2017 [8].

perience of two streaming sessions is "similar". Similarity measures whether a set of samples (our reconstruction's decisions) is likely to be from a given distribution (the actual ABR's decisions). To classify whether a particular decision looks like it has been taken by the actual ABR or by our reconstruction, we choose an isolation forest [22].

Each of these two metrics is insufficient on its own. High agreement is useful, but making a few important decisions differently can substantially change a video stream's behavior. Thus the need for similarity. However, there's a benign solution to achieving high similarity: most commercial providers tend to keep the quality stable, so, by just keeping the same quality one can get high similarity. Conversely, agreement solves this problem: to get high agreement, we must match a large fraction of *each* decision type, matching only the "keep quality" decisions will result in poor agreement because of low matches on quality changes.

QoE: Agreement and similarity can be thought of as "micro-benchmarks" – these are standard measures in imitation learning, and are useful both for training our approach, and evaluating its learning accuracy. But ultimately, we also want to know: "How different is the user experience from an ABR versus from our reconstruction of it?". We thus directly compare how well different components of a visual QoE metric used in earlier work [28] match up between a target and its reconstruction. As we show below, agreement and similarity correlate well with QoE: when a reconstruction achieves high agreement and similarity, it typically also performs like the target algorithm in terms of different QoE components.

Finally, we also comment on the role of domain knowledge in achieving good results with our approach, and the interpretability of our reconstructions.

6.2 Results

Agreement and similarity, Fig. 5: We compare the agreement and similarity achieved by our rule-set approach against the (best) blackbox approach and the simple rate baselines across all 10 online providers. We also include MPC and Pensieve (NN) evaluated on the Twitch videos.

The rule-sets achieve nearly the same or better agreement than the blackbox approach achieves for a reconstruction target in each case – in the worst case (NN), the rule-set's agreement score is 8% lower. Note that in many cases, we achieve higher agreement than even the blackbox approach. This is due to the imitation learning approaches trying to achieve higher similarity in terms of behavior rather than matching each individual quality decision.

The rule-sets also achieve high similarity in most cases, in the worst case (Twitch), achieving a ≈20% lower similarity score than the best blackbox approach, and in the mean, 5% higher. In contrast, the rate-based approach achieves not only very low agreement, but also very poor similarity.

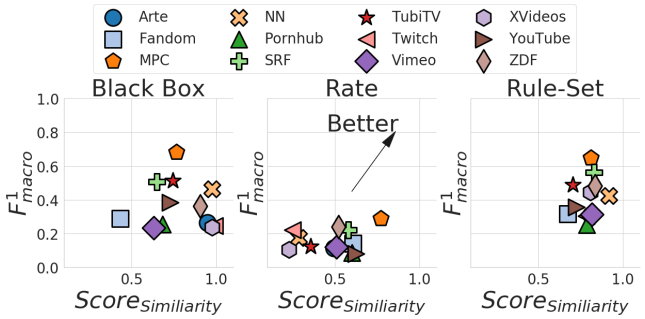

Fig. 5: The generated rule-sets are never worse by more than 8% and 20% than the blackbox approach on agreement and similarity respectively. In contrast, the rate-based approach achieves extremely poor results.

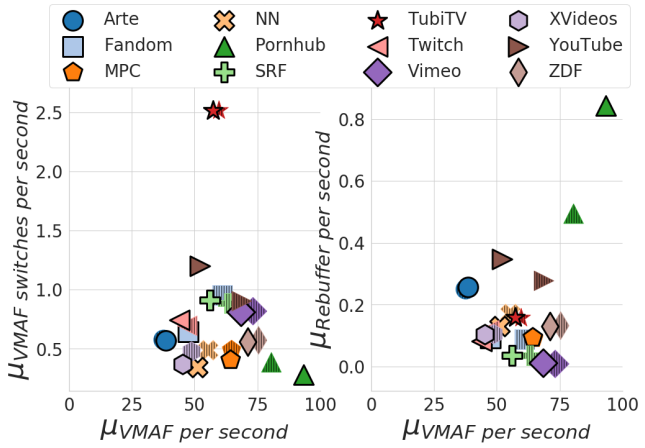

Fig. 6: For all but 3 of the 12 targets, the reconstruction matches the target algorithm very closely. For YouTube, Fandom, and PornHub, there is a substantial difference in performance; these are the same 3 providers in the bottom-left of Fig. 5, for which we achieve the lowest agreement and similarity scores as well.

Some readers may interpret the "low" absolute numbers in Fig. 5, *e.g.,* $F_1 \sim 50\%$, as a negative result. However, note that F_1 differences often don't cause appreciable video session quality differences, *e.g.,* if an ABR switches two quality levels in one step, and its reconstruction switches them in two successive steps, the F_1 score is lowered *twice*, but the video stream behavior changes negligibly. Also, rare labels (*e.g.,* increase quality by three levels) contribute equally to F_1 as common ones (*e.g.,* retain quality), so a few errors on rare labels have out-sized effect.

Video session quality metrics, Fig. 6: We compare metrics used to quantify video playback quality — VMAF [20], VMAF switches, and rebuffers – as seen in the actual algorithm (hatched points in the plot) and its rule-set reconstruction (solid points) across the same set of ABRs as in Fig. 5. For 9 of 12 targets, we achieve a very good match: the mean VMAF (*x*-axis in Fig. 6) for these 9 reconstructions is within 6% of the target ABR's on average; the maximum VMAF difference is 12%. These good matches include Twitch, SRF,

Arte, ZDF, TubiTV, XVideos, Vimeo, MPC, and Pensieve (NN). On the other hand, for the other 3, YouTube, PornHub, and Fandom, there are large discrepancies, with quality metrics being very different for the reconstruction compared to the target. That our reconstruction does not yield good results on these targets is also supported by exactly these ABRs being in the low-agreement-low-similarity part of Fig. 5 (bottom-left in the rightmost plot). We further investigated these 3 negative results:

1. YouTube, in addition to making quality decisions, varies its segment length and can also redownload low-quality chunks to replace them with high-quality ones [27]. Ultimately, learning approaches will not frame new decision spaces, only logic for arriving at the asked-for decisions – in essence, YouTube is solving a different problem than we expected. This is a fundamental gap for efforts like ours: if the *decision space* is not appropriately encoded, outcomes will be sub-optimal. We could add the relevant primitives to achieve better results, but we resist such modifications that use the benefit of hindsight.
2. In a similar vein, we find that PornHub often switches to a progressive download, disabling video quality adaptation altogether. Our approach ends up overfitting to the progressive behaviour as we see few switches. If we exclude data where adaptation is disabled, we're able to match PornHub to within 4%, 0%, and 5% difference in terms of mean VMAF, rebuffering, and switching respectively.
3. For Fandom, we find that the issue is the limited complexity of our tree. A rule-set with a somewhat higher complexity (31 rules) performs substantially better, diverging from the target algorithm by 5%, 11%, and 22% in terms of mean VMAF, rebuffering, and switching respectively. Note that rebuffering and switching, being infrequent events are extremely difficult to *always* match, so a somewhat larger difference there is expected. As noted earlier, the rule-count is a hyperparameter that may need tuning for certain providers.

Role of domain knowledge, Fig. 8, 7: We already discussed in §4 why the use of domain knowledge is critical for interpretation: without simple primitives like moving averages, rules are framed in terms of various basic literals, resulting in complex and incomprehensible rules. Besides interpretation, we find that there is also substantial impact on agreement from adding useful domain knowledge.

We used our modified version of the DASH player to evaluate how the different trees emulating robust MPC generalize to other videos. We selected a mixed subset of 60 traces, that both include challenging and stable throughput measure and generated the distribution across them of linear QoE used in the MPC work [28]. Results are normalized to the mean QoE for the ground-truth MPC implementation across the same video-trace set.

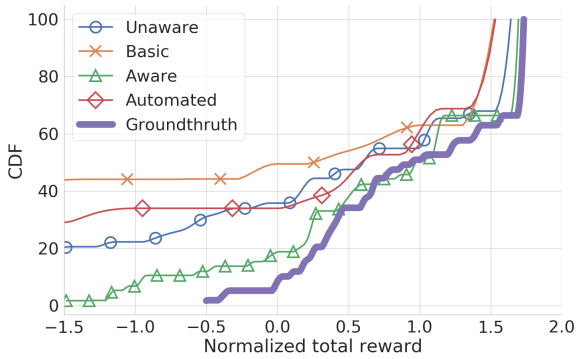

Fig. 7: Domain knowledge helps the rule-set (Bitrate-QoE).

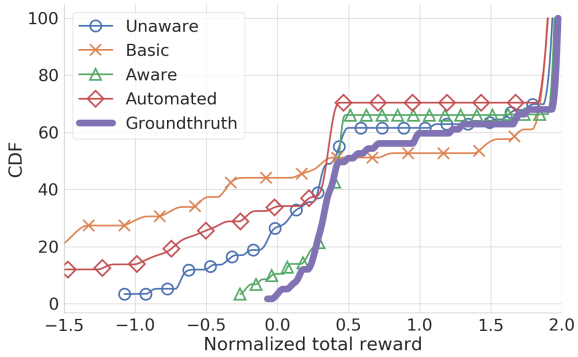

Fig. 8: Domain knowledge helps the rule-set (VMAF-QoE).

Fig. 7 shows how the rule-set reacts to additional building blocks being available for reconstruction in the form of engineered features. The 'Basic' rule-set is framed directly on the data features listed in §3, without any feature engineering. The 'Unaware' and 'Aware' approaches use several engineered features, as described in §4. The difference between them stems from the 'Unaware' approach only using engineered features related to buffer filling and draining in addition to the primitive data features. The 'Aware' approach with the benefit of all engineered features matches MPC the closest. 'Aware' improves average QoE over 'Unaware' by $\sim 5\times$. Thus, encoding domain knowledge helps not only with conciseness, but also performance and generalization. Also of note is the 'Automated' approach, which starts with the 'Basic' features, but can recombine them in the fashion described in §5. While promising for future exploration, it presently performs worse than manually engineering features, and does not produce features that are meaningful to humans.

Fig. 8 repeats the above experiment, but for a VMAF-based QoE function. The results are similar to those for bitrate -QoE. The average QoE of the 'Aware' reconstruction is within 10% of that of the target MPC algorithm, the median being within 2%. This is especially significant because we did not engineer any explicit features similar to this QoE function.

Interpretability: Across our efforts on reconstruction, the

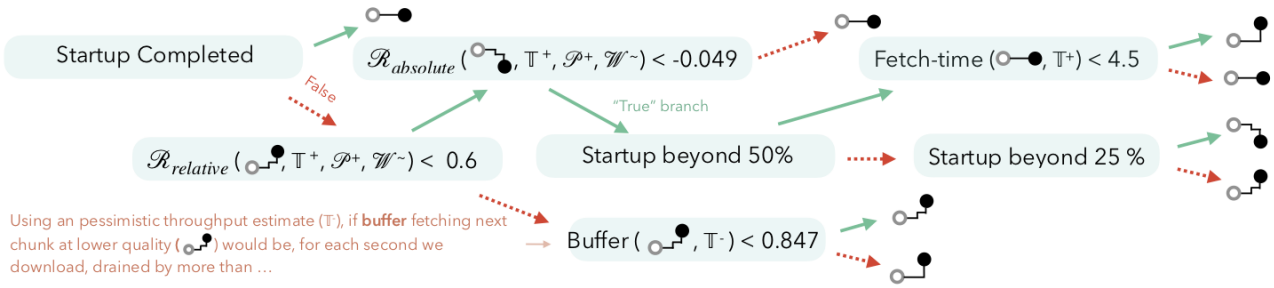

Fig. 9: The core of the decision tree generated by learning from the SRF example data. The green (solid) and red (dashed) arrows are the "True" and "False" evaluations of the predicates at the nodes. The video quality of the last fetched chunk is denoted by ◦ and the next chunk's by • . Potential decisions and decision outcomes are coded in terms of the relationships between these qualities: *e.g.,* denotes that the next chunk is requested at one higher video quality level. The predicates are in terms of expected buffer size after a certain potential decision, based on throughput estimates (*e.g.,* $\mathbb{T}^+$ is an aggressive/optimistic estimator); or on a reward ($\mathcal{R}_{relative}$) calculated relative to the other obtainable rewards involving throughput, rebuffering penalty ($\mathscr{P}$), the lookahead horizon over which these are estimated ($\mathscr{W}$), etc.

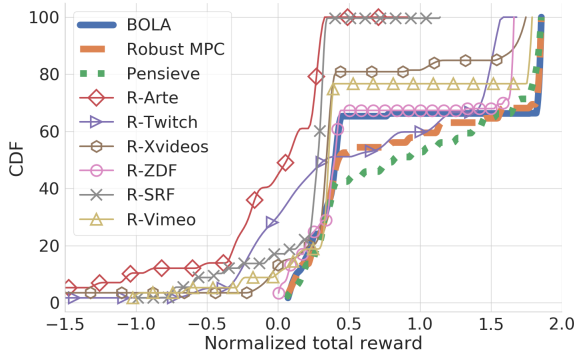

Fig. 10: Reconstructing commercial ABR algorithms allows us to uniformly compare them to both other commercial and academic ones under the same test conditions.

generated rule sets are concise, with no more than 20 rules. We realized early that being able to read and understand the generated trees would make debugging and improvements easier, and thus wrote a small, simple utility to translate the predicates in trees loosely into natural language. Fig. 9 shows an illustration of a tree generated for SRF. This version is hand-drawn for aesthetic reasons, but there is no fundamental reason it could not be auto-generated. Due to space constraints, this version is a compressed tree which was allowed to have at most 10 leaves instead of 20. We extensively examined and used our natural-language trees ourselves throughout this work, as we describe in a few instances in §7.

We also understand, to some extent, *why* small rule-sets suffice: (a) a single rule has implications capturing more than is plainly visible, *e.g.,* if the buffer is running too low, the best recourse is to lower quality, and not much new will be learnt from a long planning horizon; and (b) the domain-specific primitives are a dense encoding of useful knowledge. We caution readers against generalizing these observations to imply that small rule-sets will necessarily suffice for other problems where learning is effective — our exploration and results are limited to ABR. That small rule-sets would suffice for many ABRs, is also supported by prior work [10] showing, for instance, that the neural network ABR approach, Pensieve, largely depends on only 3 features.

Comparing academic and industry ABRs, Fig. 10: For targets we can reconstruct well, having a reconstruction enables us to compare them to each other and academic ABRs in the same test harness. This is non-trivial without a reconstruction, as each video platform has a very different streaming setup in terms of encoding, server architecture, and player design. For instance, if one platform uses more encoding levels than another, then the same ABR algorithm can make more fine-grained decisions on this platform than on the one with coarser encoding. Therefore the same algorithm on the same video would perform differently across video platforms, making it difficult to compare ABRs across providers without removing such confounding factors in a common test harness.

To this end, we extend the DASH architecture [37] with implementations of the (rule-set) reconstructions for the 6 targets we are able to match most closely. The same setup has ground truth implementations for BOLA [38], MPC [46], and Pensieve [23]. We evaluate VMAF-QoE [28] using the Envivio video typically used in academic work, and normalize the results to the mean QoE for Pensieve.

As the results in Fig. 10 show, Pensieve and MPC generally outperform the deployed ABRs' reconstructions, although, for a subset of traces, R-Twitch achieves the same or better performance as MPC. This is perhaps not unsurprising: we are evaluating all providers with QoE functions used in academic literature, while the providers may, in fact, be optimizing for a different goal. Amongst the providers, R-Arte's ABR achieves the worst performance on this QoE metric.

But perhaps most striking is the distribution-wide extremely close match between R-ZDF and BOLA – except for a few outliers at the tails, for most video-trace pairs, their performance is virtually identical. Thus, it is likely that ZDF

is using BOLA, or a derivative of it.

7 THE UTILITY OF INTERPRETABILITY

Human insight can be crucial to robust solutions that account for gaps and unanticipated changes in the data that drives the behavior of learned control procedures. We discuss several ways in which preserving the ability of expert designers to understand the decision procedure helps.

Tracing the input-output mapping: With concise decision trees, human experts can easily trace the decision process used for particular inputs or sets of inputs. For any input conditions, a path can be traced to a leaf (decision output), and for any output, how it was arrived at can be understood as well. Such tracing can allow easy debugging — "Why were bad outcomes seen for these traces?". This also opens the door to more rigorous analyses of the outcomes for sets of inputs, based on methods like symbolic execution [4].

Identifying potential issues: Experts can often identify overfitting and other problems *a priori* if they understand the procedure, as is the case with the concise decision trees we produce. Our experience itself revealed three such instances:

(1) One feature we encoded for use in our decision trees was a prospective reward from fetching the next chunks at different bitrates. This worked for most videos, giving good average performance. However, for some videos with much higher/lower average bitrate than most other videos, results were inferior. This is due to the reward function using absolute bitrates, and thus not being meaningful across videos. Defining reward in relative terms, *i.e.,* normalized to the maximum possible reward, addresses this issue. A blackbox method, by hiding from human experts the logic used in the rules, makes such improvements more challenging.

(2) We noticed that even after training across the sizable network traces used in past work, our rule sets largely depended on optimistic estimators for throughput, unlikely to work well in more challenging environments, *e.g.,* new geographies a video service expands to where connectivity is more limited and variable. To force more conservative behavior, we can either add such traces to the training data, or restrict the learning approach to use only conservative throughput estimators leading to more stable behavior. Another possibility is to add new features to detect situations where conservative or optimistic behavior would be appropriate. Note that while *given enough appropriate data* blackbox solutions would also potentially overcome such problems, this requires noticing the problem in the first place. Also, such data may not always be available: *e.g.,* if the video service performs poorly in a geography, users may self-select themselves out by dropping the service, thus further skewing the data.

(3) Early in our experiments, we observed a peculiar learned rule that translates to "Never fetch the lowest quality after 45 video chunks." This stemmed from overfitting due to training on one video with 49 chunks (on which most other academic ABR work is also evaluated), where even over a sizable set of traces, typically a large enough buffer was built such that the lowest quality was ruled out for the last few chunks. While this particular artifact would be unlikely to arise in a large provider's pipeline given enough diverse training data, similar problems may occur and go undetected in blackbox methods, especially when the underlying data changes, *e.g.,* if a short-form video service introduces longer videos.

Across these examples, blackboxes can hide problems that might otherwise have been obvious to human experts. Prior work [17] has found problems of this type, *e.g.,* Pensieve, even if conditions are highly favourable, does not always download the last chunk of a video at the highest quality.

Finally, when such problems do present themselves, the recourse with blackboxes, depending on the problem's nature, can involve blindly tinkering with the inputs to the blackbox approach until the outcomes improve, or adding extraneous safeguards for each discovered problem.

7.1 Examining two reconstructions

We next give a view of two reconstructions of different complexity: SRF (simplified, same as in Fig. 9) and Twitch.

Simplified SRF: The output tree reveals an intuitive structure and highlights obvious flaws as we discuss below. (These are only present in the simplification, and not SRF's full tree.)

Fig. 9's caption explains how to read the tree. First it checks the point in playback, concluding that it is in a startup phase *if* playtime is below a threshold. In the startup phase, it checks *if* the possible gain in Reward is large enough to warrant the leveling up by two levels. This is only done *if* we deplete the buffer by not too much when doing so; etc. Of course, behind these loose statements are concrete, parametrized features which describe what the particular throughput estimator is, what reward function is used, etc.

An interesting characteristic of the simplified-SRF tree is that there are no quality changes beyond the startup phase. This is clearly unsuitable in practice, and would be an obvious red flag to any domain expert examining this tree. The full tree does, in fact, use adaptation after startup too, although it is infrequent. We have verified this behavior experimentally as well, where SRF makes frequent quality switches during startup, and much fewer later.

Twitch: Having examined a small simplified tree, we discuss the (full) reconstruction for a more complex target, Twitch.

Twitch's tree visually reveals 3 "branches" of activity, which we call panic, cautious, and upbeat. The panic mode is entered when the throughput is very low. Here the tree is most likely to downgrade the quality by two levels to try to build up buffer, regardless of current buffer occupancy. An example trace captured online shows such behavior at roughly 100 s in playback in Fig. 11.

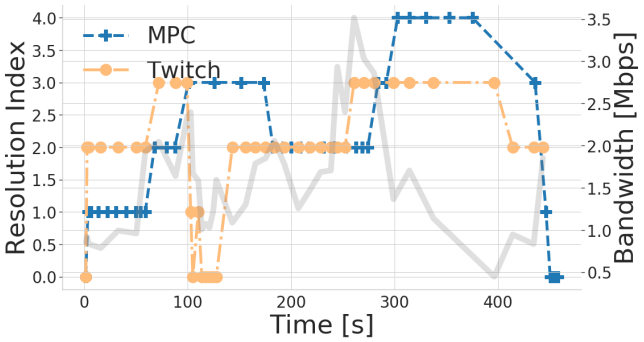

Fig. 11: Twitch shows only marginally more reluctance towards switching when compared to MPC

The cautious mode is entered at mediocre connection quality and, unlike the panic mode, examines the buffer level. In this mode, the most likely action is to either keep the quality or, if buffer-level is low, downgrade it. Downgrading can also be induced by high throughput variance, which indicates uncertain networking conditions.

If throughput is above mediocre, the tree enters the upbeat mode. Here the most common action is upgrading the quality, or if we approach higher quality levels (and therefore, longer download times even with good network conditions), the decision to upgrade is weighted against the buffer drain it would incur, and the current buffer occupancy.

Unlike several other providers, Twitch's reconstruction reveals a willingness to switch qualities often. This is in line with our experimental observation that Twitch and MPC make similar number of switches in the same conditions, while other providers switch much less frequently compared to MPC. Based on this analysis, if a switching-averse provider wanted to adopt Twitch's approach by reconstructing it, they would have to suitably tweak it to reduce switching.

To summarize, with interpretability, we can catch problems before they occur, reason about generalization and behavior across sets of operating conditions instead of just point testing, and methodically discover and fix encountered problems.

8 LIMITATIONS & FUTURE WORK

Over the course of this work, unsurprisingly, we uncovered several shortcomings of our approach, which offer interesting avenues for future exploration:

- Accurate and concise trees require intuitive primitives, *e.g.,* moving averages, which must be manually encoded (§4). Perhaps such primitives can be automatically captured from a corpus of available hand-designed networked algorithms. But this is likely a challenging task.
- We explored a limited set of features, some across only a small part of their possible parameter ranges, *e.g.,* only 5 discrete values for the α parameter in moving averages. A potentially highly effective avenue of improvement lies in tuning the features using a black box optimizer, *e.g.,* a Gaussian Process Optimizer [29], to suggest useful values.
- We can only train for an appropriately specified decision space, as is clear from the failure of our approach for YouTube (§6.2). We can expand the decision space with the benefit of manually-drawn observations from experiments, but automatically discovering it seems difficult.
- We do not expect our approach to always be able to match a target algorithm. However, failures of our approach also help: they often flag "atypical" ABR designs for manual analysis, like for YouTube and Pornhub, and could help uncover unknown (proprietary) insights.
- We used an intuitive but *subjective* definition of "interpretable": trees with under 20 leaves on domain-specific literals. Our own experience with understanding the results was positive, but we hope feedback from other researchers will help sharpen the interpretability goal for future work.
- For providers that customize their ABR for different regions and sets of clients, we can only reconstruct the behavior we observe from our test clients. For future work, this opens an interesting opportunity: observing differently-tuned versions of the same ABR, it may be possible to achieve higher-quality reconstructions, which also identify the parameters whose tuning varies across regions.

9 CONCLUSION

We take the first steps towards an ambitious goal: reconstructing unknown proprietary streaming algorithms in a human-interpretable manner. We customize and evaluate a rule-set approach, achieving good results for reproducing the behavior of algorithms deployed at several popular online services. Our approach produces succinct output open to expert interpretation and modification, and we discuss through several examples, the utility of this interpretability.

While promising, our results also expose a likely fundamental limitation — we need to encode and make available suitable domain knowledge to the learning approach. This can be interpreted as suggesting that we should reconcile learning with our already acquired human expertise, instead of starting afresh. We hope to apply this approach, suitably customized, to congestion control as well, where it is unclear how much diversity there is in actual deployment of different, unknown congestion control algorithms across popular Web services.

Acknowledgments

We are grateful to Martin Vechev, Ce Zhang, our anonymous reviewers, and our shepherd Junchen Jiang, for valuable discussions and feedback. This work is supported by the Swiss National Science Foundation under grant number 182409.

References

[1] Anne Aaron, Zhi Li, Megha Manohara, Jan De Cock, and David Ronca. Per-title encode optimization. https://link.medium.com/jEeb6GV0ZW.

[2] Saamer Akhshabi, Ali Begen, and Constantine Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over HTTP. In *ACM MMSys*, 2011.

[3] Zahaib Akhtar, Yun Seong Nam, Ramesh Govindan, Sanjay Rao, Jessica Chen, Ethan Katz-Bassett, Bruno Ribeiro, Jibin Zhan, and Hui Zhang. Oboe: auto-tuning video ABR algorithms to network conditions. In *ACM SIGCOMM*, 2018.

[4] Roberto Baldoni, Emilio Coppa, Daniele Cono D'Elia, Camil Demetrescu, and Irene Finocchi. A survey of symbolic execution techniques. *ACM Computing Surveys*, 2018.

[5] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *NeurIPS*, 2018.

[6] Lionel Blondé and Alexandros Kalousis. Sample-efficient imitation learning via generative adversarial nets. In *PMLR*, 2019.

[7] Kyunghyun Cho, B van Merrienboer, Caglar Gulcehre, F Bougares, H Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*, 2014.

[8] Federal Communications Commission. Validated data September 2017 - measuring broadband America. https://www.fcc.gov/reports-research/reports/.

[9] Luca De Cicco and Saveri o Mascolo. An experimental investigation of the Akamai adaptive video streaming. In *USAB*, 2010.

[10] Arnaud Dethise, Marco Canini, and Srikanth Kandula. Cracking open the black box: What observations can tell us about reinforcement learning agents. In *ACM NetAI*, 2019.

[11] Anis Elgabli and Vaneet Aggarwal. Fastscan: Robust low-complexity rate adaptation algorithm for video streaming over HTTP. In *IEEE TCSVT*, 2019.

[12] Maximilian Grüner, Melissa Licciardello, and Ankit Singla. Reconstructing proprietary video streaming algorithms. https://github.com/magruener/reconstructing-proprietary-video-streaming-algorithms, 2020.

[13] H. Guo, Q. Zhang, and A. K. Nandi. Feature generation using genetic programming based on fisher criterion. In *IEEE EUSIPCO*, 2007.

[14] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *NIPS*, 2016.

[15] 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 *ACM SIGCOMM*, 2014.

[16] Ayad Ibrahim, Im Youngbin, Keller Eric, and Ha Sangtae. A practical evaluation of rate adaptation algorithms in HTTP-based adaptive streaming. In *Computer Networks*, 2018.

[17] Yafim Kazak, Clark Barrett, Guy Katz, and Michael Schapira. Verifying deep-RL-driven systems. In *ACM NetAI*, 2019.

[18] Breiman L., Friedman J. H., Olshen R. A., and Stone C. J. *Classification and Regression Trees*. Wadsworth and Brooks, 1984.

[19] De Cicco L. and Mascolo S. A mathematical model of the Skype VoIP congestion control algorithm. In *IEEE Transactions on Automatic Control*, 2010.

[20] Zhi Li, Anne Aaron, Ioannis Katsavounidis, Anush Moorthy, and Megha Manohara. Toward a practical perceptual video quality metric. https://medium.com/netflix-techblog/toward-a-practical-perceptual-video-quality-metric-653f208b9652, 2016.

[21] Melissa Licciardello, Maximilian Grüner, and Ankit Singla. Understanding video streaming algorithms in the wild. In *PAM*, 2020.

[22] F. T. Liu, K. M. Ting, and Z. Zhou. Isolation forest. In *IEEE ICDM*, 2008.

[23] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with Pensieve. In *ACM SIGCOMM*, 2017.

[24] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with Pensieve. https://github.com/hongzimao/pensieve, 2017.

[25] Zili Meng, Jing Chen, Yaning Guo, Chen Sun, Hongxin Hu, and Mingwei Xu. PiTree: Practical implementation of ABR algorithms using decision trees. In *ACM Multimedia*, 2019.

[26] Zili Meng, Minhu Wang, Jiasong Bai, Mingwei Xu, Hongzi Mao, and Hongxin Hu. Explaining deep learning-based networked systems. *arXiv:1910.03835*, 2019.

[27] Abhijit Mondal, Satadal Sengupta, Bachu Rikith Reddy, M. J.V. Koundinya, Chander Govindarajan, Pradipta De, Niloy Ganguly, and Sandip Chakraborty. Candid with YouTube: Adaptive streaming behavior and implications on data consumption. In *ACM NOSSDAV*, 2017.

[28] Vikram Nathan, Vibhaalakshmi Sivaraman, Ravichandra Addanki, Mehrdad Khani, Prateesh Goyal, and Mohammad Alizadeh. End-to-end transport for video QoE fairness. In *ACM SIGCOMM*, 2019.

[29] Michael Osborne, Roman Garnett, and Stephen Roberts. Gaussian processes for global optimization. In *International Conference on Learning and Intelligent Optimization*, 2009.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 2011.

[31] Yanyuan Qin, Shuai Hao, Krishna R Pattipati, Feng Qian, Subhabrata Sen, Bing Wang, and Chaoqun Yue. ABR streaming of VBR-encoded videos: characterization, challenges, and solutions. In *ACM CoNEXT*, 2018.

[32] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. Commute path bandwidth traces from 3G networks: analysis and applications. In *MMSys*, 2013.

[33] Reudismam Rolim, Gustavo Soares, Loris D'Antoni, Oleksandr Polozov, Sumit Gulwani, Rohit Gheyi, Ryo Suzuki, and Björn Hartmann. Learning syntactic program transformations from examples. In *ICSE*, 2017.

[34] Stephane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *PMLR*, 2011.

[35] Sandvine. The global Internet phenomena report. https://www.sandvine.com/press-releases/sandvine-releases-2019-global-internet-phenomena-report, 2019.

[36] Reliable Secure and Intelligent Systems Lab. JSNice - statistical renaming, type inference and deobfuscation. http://jsnice.org/, 2018.

[37] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. From theory to practice: Improving bitrate adaptation in the DASH reference player. In *ACM MMSys*, 2018.

[38] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K. Sitaraman. BOLA: Near-optimal bitrate adaptation for online videos. In *IEEE INFOCOM*, 2016.

[39] Trevor Stephens. Genetic programming in Python. https://github.com/trevorstephens/gplearn, 2017.

[40] J. van der Hooft, S. Petrangeli, T. Wauters, R. Huysegems, P. R. Alface, T. Bostoen, and F. De Turck. HTTP/2-based adaptive streaming of HEVC video over 4G/LTE networks. In *IEEE Communications Letters*, 2016.

[41] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *PMLR*, 2018.

[42] Ruohan Wang, Carlo Ciliberto, Pierluigi Vito Amadori, and Yiannis Demiris. Random expert distillation: Imitation learning via expert policy support estimation. In *PMLR*, 2019.

[43] Shichang Xu, Subhabrata Sen, and Z. Morley Mao. CSI: Inferring mobile ABR video adaptation behavior under HTTPS and QUIC. In *ACM EuroSys*, 2020.

[44] Y. Xu, C. Yu, J. Li, and Y. Liu. Video telephony for end-consumers: Measurement study of Google+, iChat, and Skype. In *IEEE/ACM Transactions on Networking*, 2013.

[45] Hyunho Yeo, Youngmok Jung, Jaehong Kim, Jinwoo Shin, and Dongsu Han. Neural adaptive content-aware Internet video delivery. In *USENIX OSDI*, 2018.

[46] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over HTTP. In *ACM SIGCOMM*, 2015.