CSE 534
FUNDAMENTALS OF COMPUTER NETWORKS
Department of Computer Science
Stonybrook University

PROJECT REPORT

Deeper Dive into PiTree: A scalable framework to convert ABR algorithms to Decision Trees

Github Link: https://github.com/zzzeeshannn/pitree

Zeeshan Shaikh (113221938)
Zhilin Zou (113273588)
Yiyun Yang (113031195)

Introduction

PiTree was first introduced in the paper named PiTree: Practical Implementation of ABR Algorithms using Decision Tree.
The need for PiTree came in to solve the issue of deployment overhead and performance tradeoff. PiTree aims to solve that by using Decision Trees to develop ABR policies to predict the optimal bitrates to maximize QoE. With the choice of using Decision Trees, the deployment overhead drops significantly while showing only a loss of 3% in performance in the worst case scenario.

Since PiTree is a scalable framework that supports conversion with any ABR algorithms, the paper uses three ABR algorithms to show their work, namely RobustMPC, Pensieve and HotDash.
We briefly summarize these as this connects to the problem statement that has been formulated.

MPC: Assumes throughput to be the same over a short time period (t - t+k), uses the current buffer, previous bitrate and throughput prediction to find optimal bitrate.
RobustMPC: Assumes throughput to be in a range (T_L- T_U), solves the optimization problem with the same parameters as above.
Problem with RobustMPC: Overhead and has to solve optimization problems online.

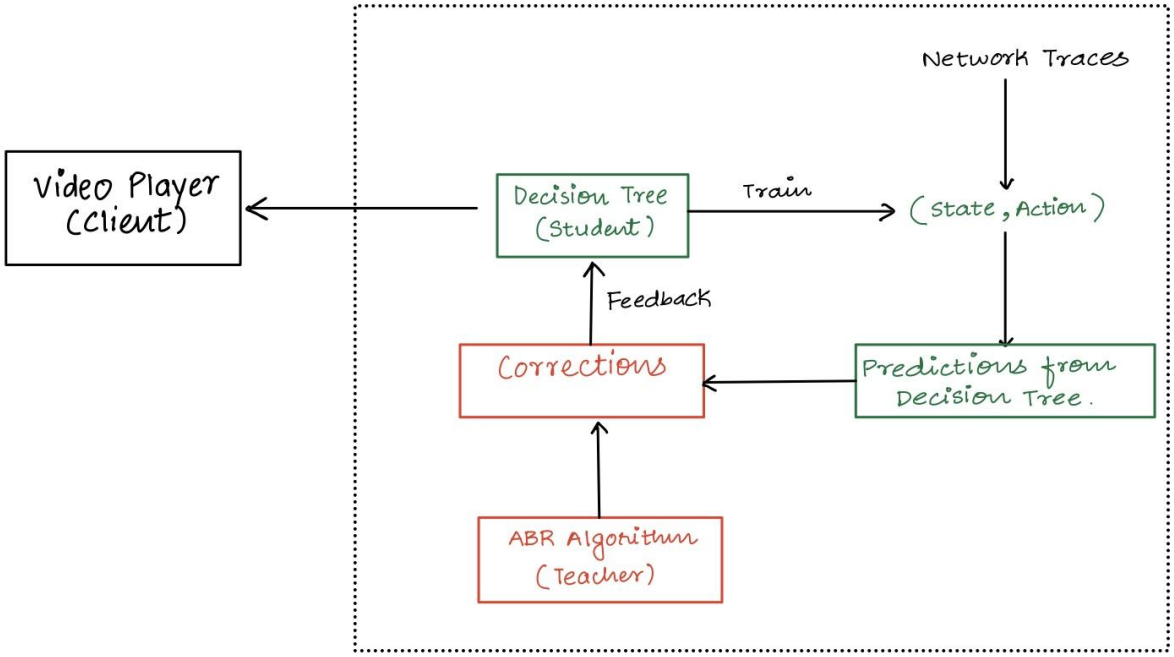
FastMPC: Creates a table offline over the entire state-space, online step includes looking up the store optimal control decisions based on current conditions.
Problem with FastMPC: Performance tradeoff (upto 30%)

Pensieve: Uses Reinforcement learning (Actor-Critic) to learn and predict ABR policies.
HotDash: Prefetches user-preferred temporal segments and uses A3C (Reinforcement Learning) to predict bitrate/ABR policies.
Problem: High deployment overhead for such complex Neural networks.

Here, we can see that Robust/FastMPC is the case where we tradeoff performance for reduced deployment overhead whereas Pensieve and HotDash are complex neural networks that increase the performance but have a very heavy overhead. Thus, the ABR algorithms chosen are done so, to select algorithms from both sides of the tradeoff.

Thus, PiTree provides a common framework to address all of these. It does so by using Imitation learning to train the decision tree. The decision tree is trained on state-action pairs retrieved from the network trace. Here, action is an optimal bitrate and state is the state of the client side media player where the tree is deployed which includes factors such as network throughput.
The decision tree (student) then makes a prediction which is corrected by the original ABR algorithm (teacher), creating a continuous loop of a constantly evolving tree.

This is illustrated by the figure below:



Problem Statement Formulation and Experimental Setup

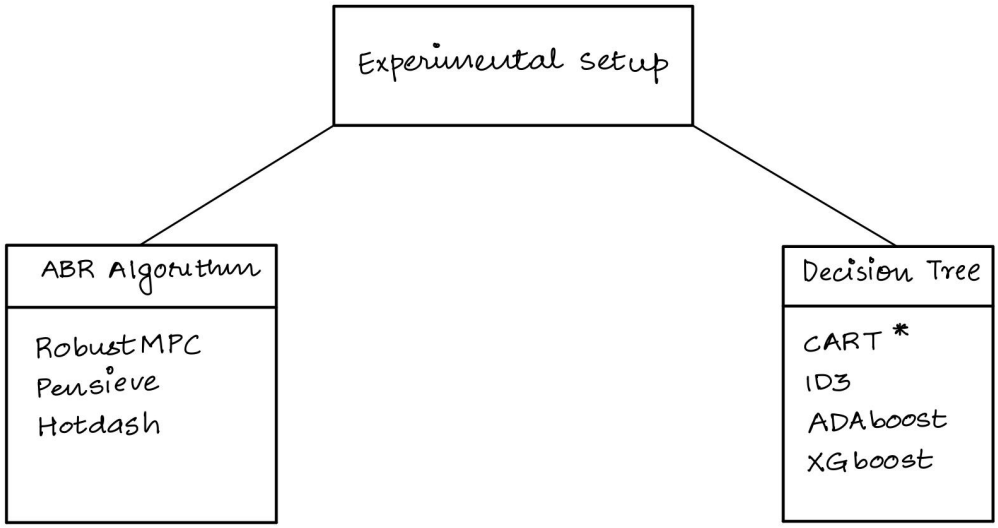
While we were going through the paper, trying to find areas where we could experiment or find exceptions, we finalised these three potential areas:

1. Look into the absence of an explanation for the solution to cascading-effect between actions/performance in high variation throughput networks.
2. Test new-er ABR algorithms that have been released since the paper was released to see if it is truly “scalable”
3. Compare and contrast different Decision Tree models to see how PiTree performs (Possibility of better performance or find a flaw: Does it only work with one type of tree?)

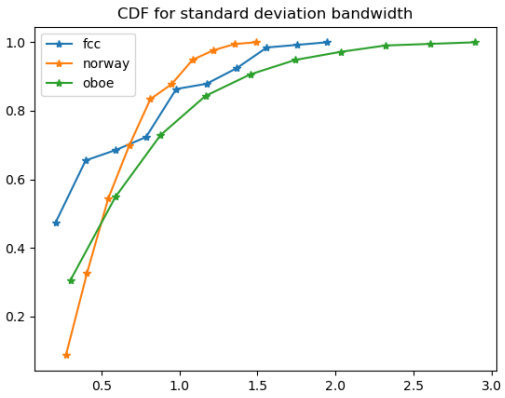
We decided to compare and contrast the performance of various Decision Trees.

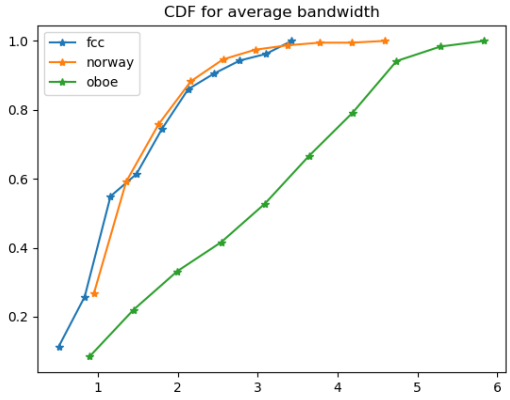
For our experiments, we decided to implement 3 different types of Decision Trees, namely ID3, XGBoost and ADABoost. We use ID3 as a comparison metric to the decision tree they’ve implemented, which is an optimized CART. We then move to Ensemble methods, where we decide to use XGBoost as it operates upon CART trees and thus, would be a good measure to see if an increased complexity or layer over the base CART used, would have any impact. Since ensemble learning methods are a bit more complicated, we introduced ADABoost as a comparison metric to XGBoost within the ensemble learning category.

Thus, our experimental setup follows the image shown below:



Here, we train and test the different decision trees with all the three mentioned ABR algorithms. As for our dataset, we use the network traces provided by the authors of the paper, namely NORWAY - OBOE - FCC. These have different properties which are shown below.

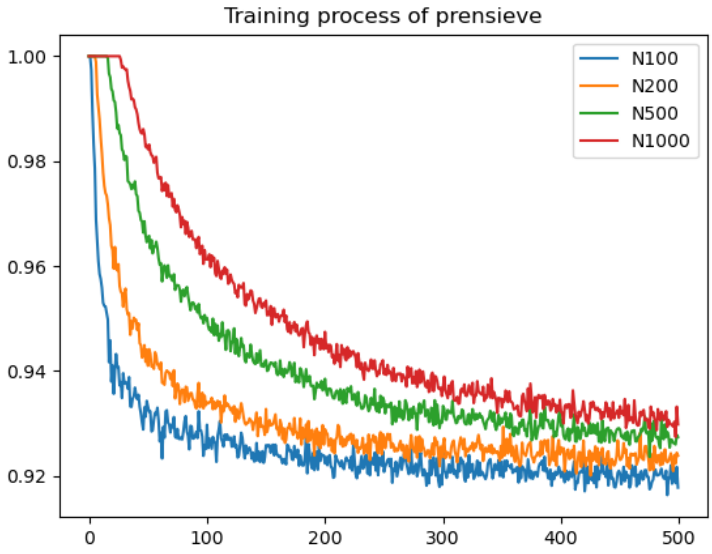




Later, we migrate the decision tree generated by PiTree into dash.js in a light-deployment way. We first convert the tree model generated by scikit-learn, i.e. DecisionTreeClassifier into JavaScript codes, incorporating into the video player. Then we deploy it with Apache and run this Web Application on Google Colab, with the local address exposed publicly through proxy. In this way we can not only evaluate the overhead(page size, decision-making latency), we can also evaluate the impact given different Network latencies.

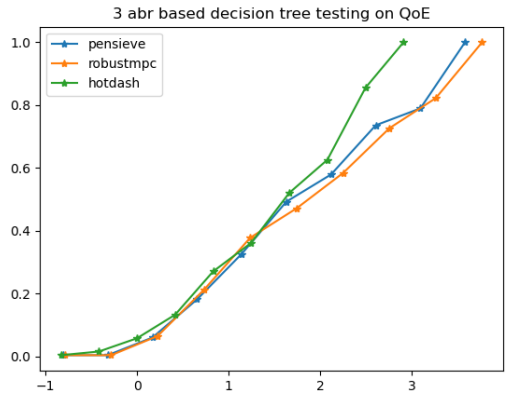
Results

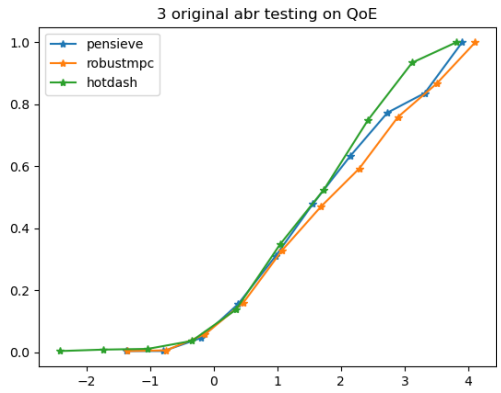
Training



Here, we notice that the decision tree with a larger number of nodes takes a longer time to converge but provides a higher accuracy. That being said, we noticed in our experiments that the increase in performance over the validation set for iterations >200 was not worth the invested time in training given the performance gains. Thus, the number of iterations for at least our testing and analysis work was set at [100, 200].

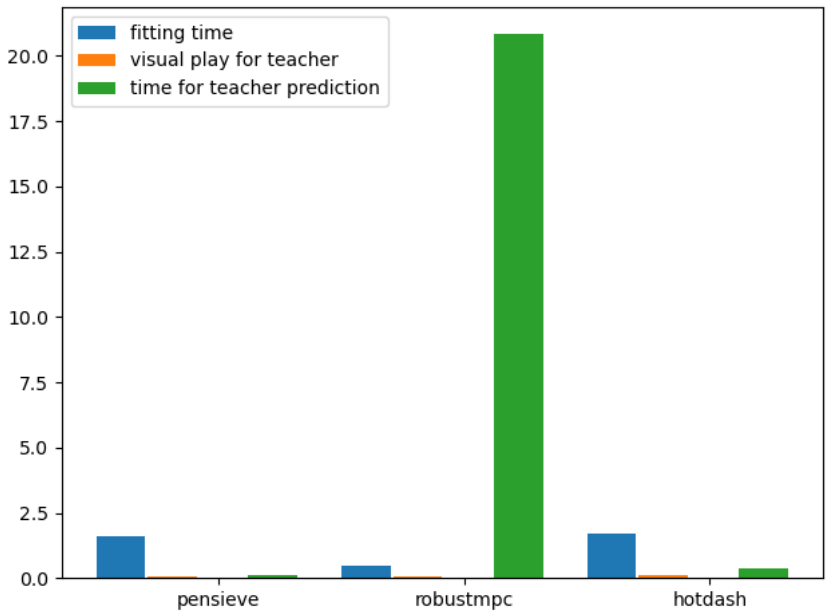
Evaluation Standards





To our surprise RobustMPC had a better performance in both case scenarios. We're not exactly

sure why, but this along with some other observations lead us to believe that overfitting may be a probable cause here.

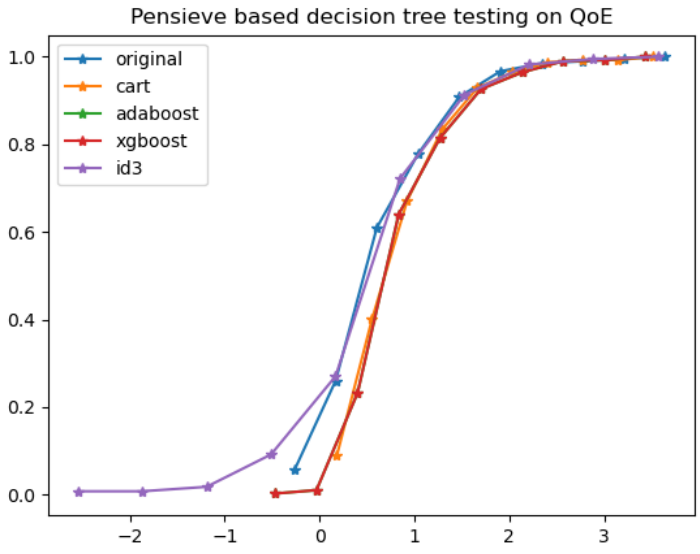


We also noticed that the training time was very high for pensieve and hotdash but RobustMPC was quite fast. Our understanding is that since RobustMPC implements Linear Programming instead of training data, it has an edge over training speed.

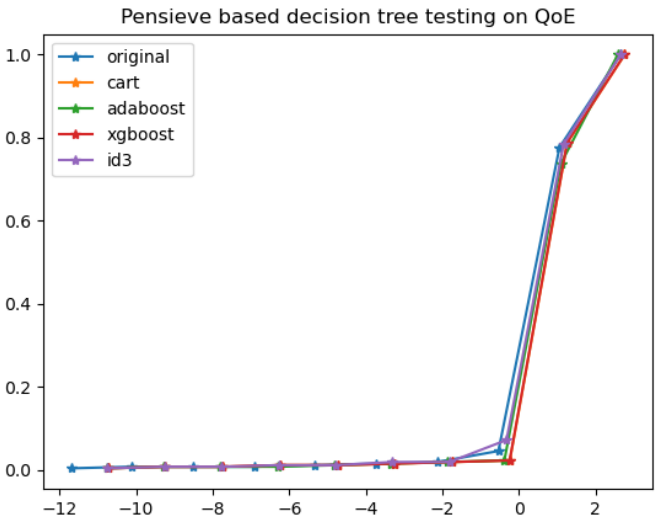
Performance

In the interest of keeping this report to the point, we have shown only specific selections of our results here. We trained 3 models, over 3 algorithms, over 3 datasets. That leads to 9 different results. For each, we have plotted the QoE results at the very least, with individual graphs for each metric to see how each tree performed and where it lacked. This brought our tally upto close to 60 graphs. Thus, we decided to only show specific ones here.

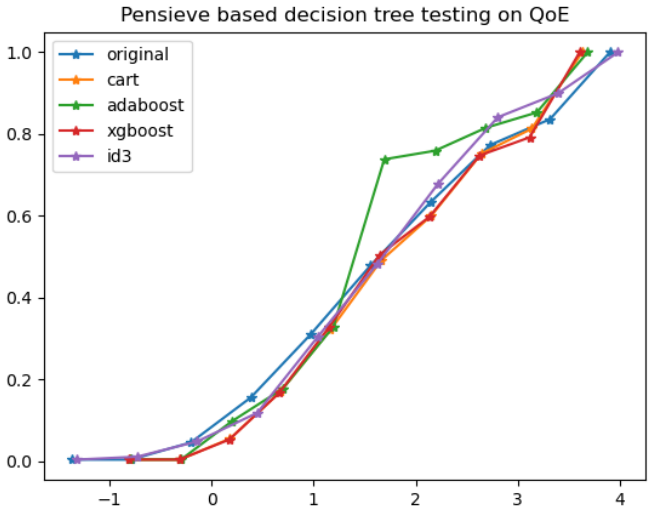
However, we've uploaded the scripts to generate these graphs on our Github page, for further exploration.



Here, Image 1 is Pensieve based Decision Tree over Norway dataset. Image 2 is over the fcc network trace and Image 3 is over OBOE network trace.



Now, here we notice that the QoE performance for all the decision trees is almost the same, with XGBoost having a slight advantage over optimized CART algorithms. This is important because our XGBoost, although optimized, hasn't been optimized to it's peak potential. Thus, a smaller percentage of performance gain remains to be extracted.



This is important because PiTree claims to be much lighter with <3% performance loss, which can be gained back by properly optimizing XGBoost. That being said, the training time and complexity of XGBoost is way more than that of a regular optimized CART tree. But given that initial training is done offline, this remains a tradeoff question of would we rather go over higher training complexity for a minor performance gain or is a 3% gain not worth the effort.

Another interesting thing that we found was that although the QoE was similar for all the trees, on deeper analysis we saw that ADA showed lower average bitrate over chunks of videos. At the same time, XGboost was a bit more towards better average bitrate but also higher rebuffers, which implies that although the QoE metric shows that they are similar in performance, the user may have very different experience on either.

The QoE metric used in the paper is shown below:

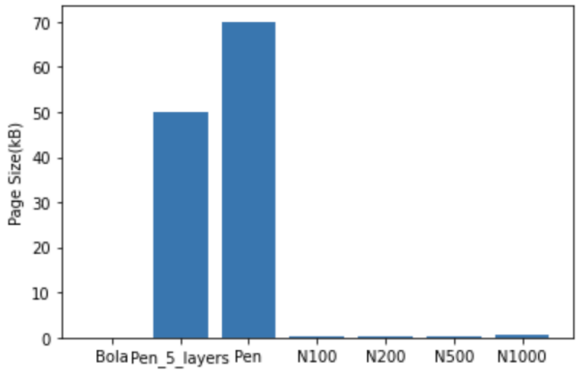
> *4.1.2 QoE Metrics.* The QoE metric can be expressed as:
>
> $$QoE = \sum_n q(R_n) - \mu \sum_n T_n - \sum_n |q(R_{n+1}) - q(R_n)| \qquad (8)$$
>
> where R_n represents the bitrate of chunk n. T_n is the rebuffering time of chunk n. $q(\cdot)$ is the utilization function as defined in Table 2.

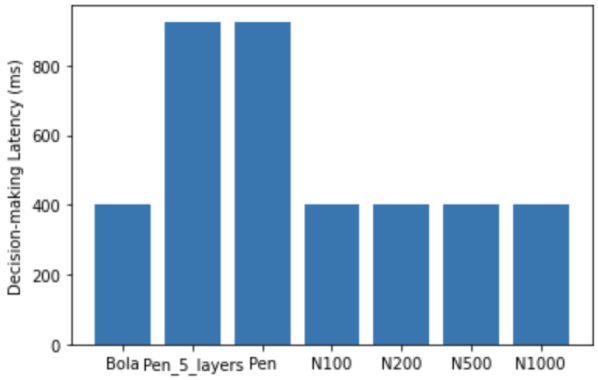
Thus, given our observation, we decided to come up with our own QoE metric. We played around with different equations with terms involving rebuffer time, initial delay, sleep time etc but did not have anything concrete to present by the deadline. One of the biggest problems we faced in this section and also our issue with this paper, was they did not explain how they came up with the value of “mu” for different utilization functions. If we had insights into that, we could’ve better optimized our terms with coefficients to show something meaningful.

Overhead

We evaluate the overhead on 3 metrics: pagesize, decision-making latency, and Network latency. We evaluate them on the following algorithms: Bola(buffer-based), Pensieve(ML-based), DT(decision tree) methods with different nodes, noted as N100, N200, N500, N1000.

1. Page size and Decision-making Latency (tested on local server)





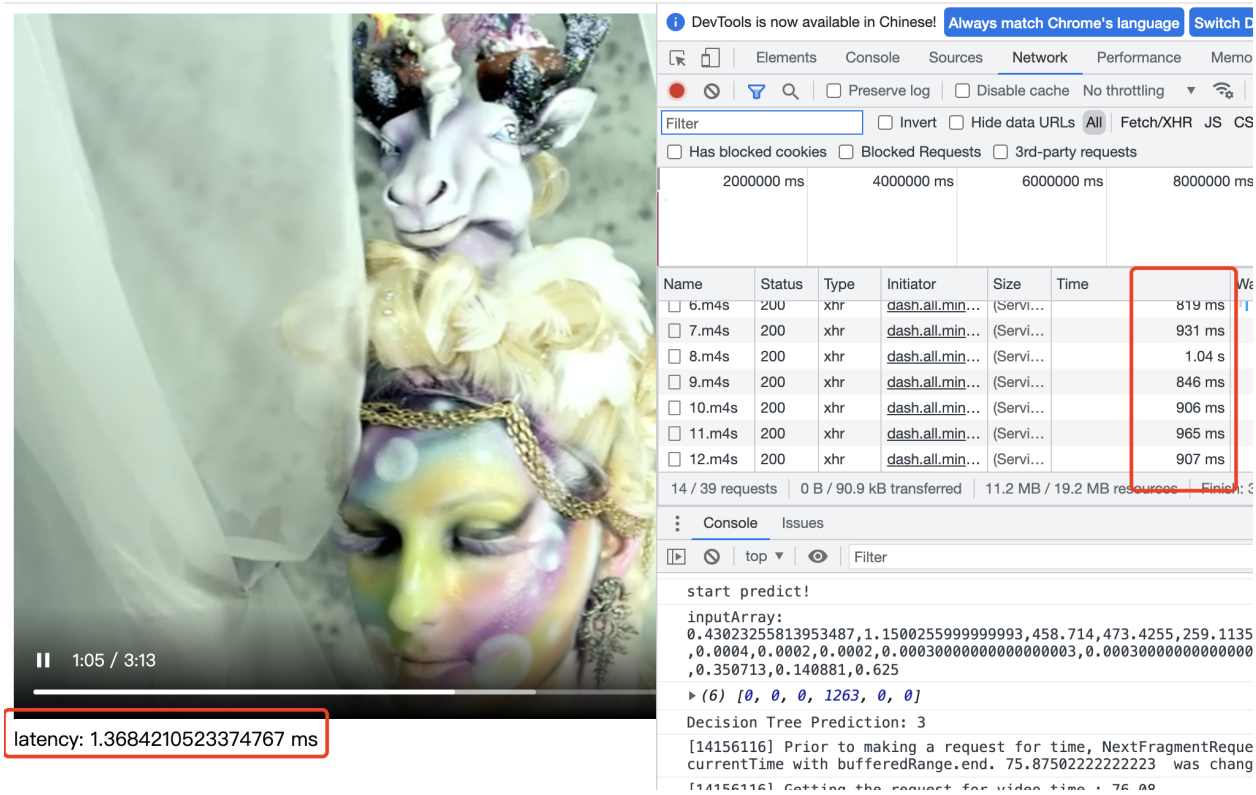
Page Size(kB)

Decision-making Latency(ms)

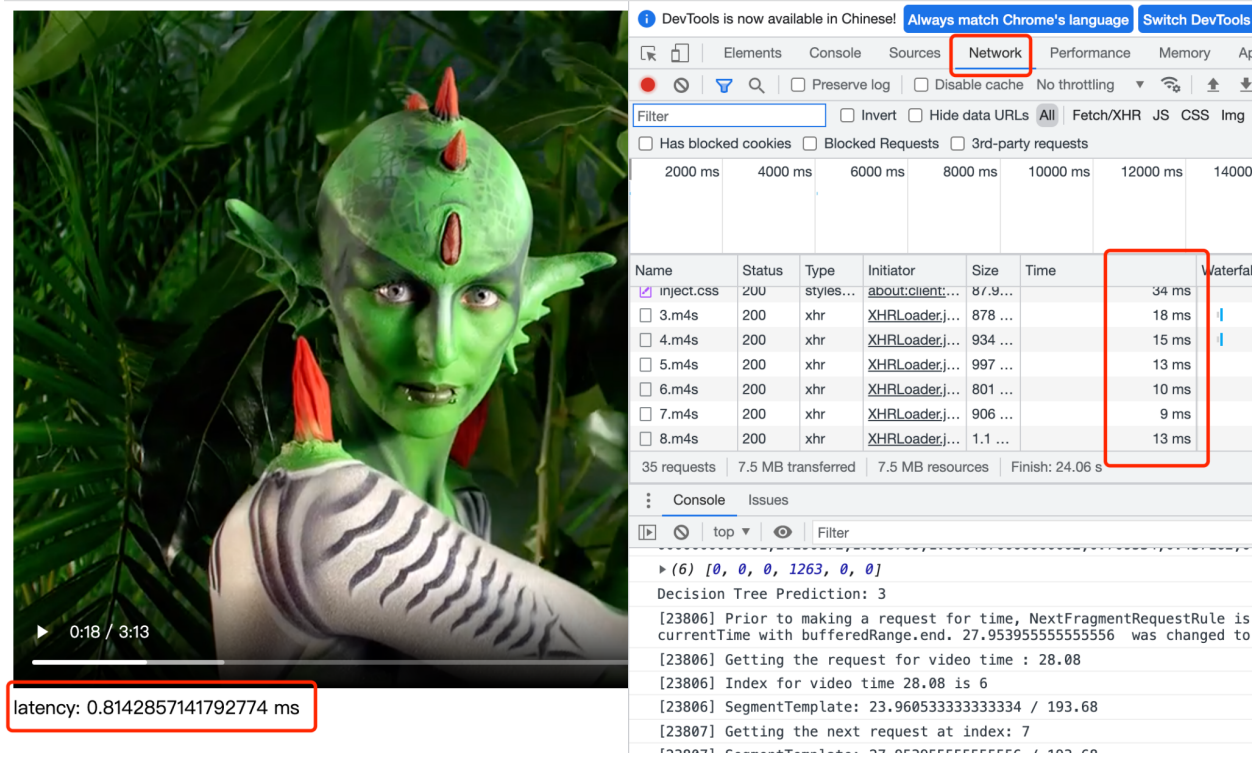
We can see from the results that, Pensieve can have much larger page size because it has included Tensorflow.js, as well as higher latency in decision making. But the Decision Tree only brings up a little these two metrics compared to Bola, even more leaf nodes does not increase the page size much. Moreover, we do not need a very large number of nodes to get faithful results.

2. Network Latency

We use Google Chrome inspect tool to check the Network latency. From the observation, it is obvious that the latency from the server deployed remotely is nearly 100X more than the local server. Their decision-making latencies do not vary. However, the bitrate was changed differently given the Network conditions.



Network Latency on Remote Server



Network Latency on Local Server