ABSTRACT
For our CS 244 final project, we present the recreation of two results from the paper Neural Adaptive Streaming with Pensieve. First, we present a recreation of Figure 11, which compares Pensieve to other ABR algorithms across three different network types. Next, we present a recreation of Figure 13, which compares Pensieve algorithms trained on multiple simulated videos to Pensieve algorithms trained on a single test video. In order to gain additional insight into the robustness and flexibility of Pensieve ABR algorithms, we extended the experiments conducted by the original authors. Specifically, we tested each algorithm in Figure 11 on additional network types, and we trained and evaluated Pensieve algorithms using additional test videos beyond the lone video discussed in the original paper.

An in depth discussion of our recreation and additional experiments follows.

1 INTRODUCTION
The authors of Pensieve present a system that generates adaptive bit rate algorithms (ABR) for Internet video streaming players using Reinforcement Learning (RL). The Pensieve system works by training a neural network using a specific Quality of Experience (QoE) metric evaluated over a large dataset of real network traces. In the paper, the Pensieve authors test the algorithm against several other ABR algorithms, including robustMPC and BOLA, and compare the performance of Pensieve with these algorithms across multiple different network types. Throughout all of these tests, Pensieve outperforms every other algorithm for each examined QoE metric. Further, the authors claim that Pensieve performs just under the theoretical optimal for several of the experiments. Additionally, the paper claims that Pensieve trained on a single video performs very close to Pensieve trained on multiple videos, indicating that the algorithm is highly robust and able to generalize from a small subset of training videos to serve a wide variety of content.

In this report, we examine and attempt to reproduce several of the results given in the paper. In particular, we focus on two concrete claims; first, that Pensieve works across a wide variety of network types, and second, that Pensieve trained on a single video can generalize to many different videos. While these claims are not the primary claims of the paper, they are crucial in demonstrating Pensieve’s generality across varied network conditions and video types. Our primary motivation for examining the first claim is that while the authors mention that they tested Pensieve on some lower-bandwidth connections, they do not provide concrete bandwidth, latency, or loss number for these networks, and we are interested in seeing whether these results hold under further examination. Further, we are simply interested whether the simulation based testing of Pensieve on network traces using MahiMahi actually translates well to strong real-world performance. For the second claim, we are interested in examining why the authors used a single video in particular (the EnvivioDash3 video) across all of their tests, and whether the results generalize across different video selections.

In order to evaluate these two claims, we attempt to reproduce Figures 11 and 13 (originals shown below). Figure 11 compares Pensieve to other ABR algorithms across three different network types (LTE, WiFi, and an international link) using the QoE_{lin} metric. We are particularly interested in examining Pensieve’s behavior on these networks with significant time-based variability and varying amounts of data loss. Figure 13 compares Pensieve trained on multiple videos to Pensieve trained on a single video, and evaluates the results with respect to the QoE_{lin} metric. We are particularly interested in whether the results in this figure still holds if Pensieve is trained on a different single video (instead of the selected EnvivioDash3 video).

![Figure 11: Comparing Pensieve with existing ABR algorithms in the wild. Results are for the QoE_{lin} metric and were collected on the Verizon LTE cellular network, a public WiFi network, and the wide area network between Shanghai and Boston. Bars list averages and error bars span ± one standard deviation from the average.](image-url)
2 OVERVIEW

2.1 Prior Work

Pensieve [1] builds on the work of many prior ABR algorithms designed to improve the performance of video players on low bandwidth networks with unpredictable throughput patterns. Such algorithms include BOLA [3], an algorithm which was originally designed to make bitrate decisions purely on the basis of buffer occupancy, but has since been adapted to take into account throughput predictions in certain situations, particularly when video playback first begins. BOLA is a good benchmark because it has been integrated into the popular dash.js video player, an open source video player which is used widely across the Internet in various video-on-demand and live streaming applications. Pensieve also builds heavily on the work of the "MPC" algorithm. MPC stands for Model Predictive Control, and Yin et. al. [4] published a paper using this control-theoretic approach to generate ABR algorithms. This paper presented a thorough mathematical evaluation of user quality of experience (QOE) which many other video-streaming researchers have used to evaluate various systems. The Pensieve authors base their QOE metrics on those presented in the MPC paper.

2.2 Reproduction Goals

2.2.1 Figure 11. Our reproduction of Figure 11 attempts to determine several things. First, we simply attempt to recreate the results of the paper, in which the authors found that Pensieve outperformed BOLA and RobustMPC when evaluated in real time on real-world networks. Second, we hope to learn more about the extent to which these results are the result of variation in network conditions during the test as opposed to the result of better decision making by the respective algorithms. Third, we add an additional figure in which we use the real-world test setup provided by the authors, but evaluate it using mahimahi simulated networks, which helps us to determine whether the real-world test setup skews the results compared to the simulated results the authors provide elsewhere in the paper. Finally, we provide an analysis of a bug we discovered in the test system in which BOLA does not always produce complete results, and evaluate how this bug may have skewed the results.

2.2.2 Figure 13. For Figure 13, we sought an extension more than a reproduction – we feel that evaluating Pensieve trained on different single "real" videos would provide more information about the extent to which Pensieve results are dependent on the training video used than a comparison to some collection of "simulated" videos. Specifically, we sought to determine whether Pensieve’s performance on the QoeLin metric was as independent of the training video used as the authors suggest in their description of Figure 13.

3 EXPERIMENTAL SETUP

In order to both reproduce the Pensieve results and test Pensieve against additional network conditions, our experimental setup used a variety of real-world and simulated networks. Below is a brief description of how we setup and used both of these simulations.

For each link type, we test a total of three different ABR algorithms: RobustMPC, BOLA, and Pensieve. For all of these tests, we used a round-robin testing style to ensure that the different algorithms were all temporally close to each other, and transient network congestion or failures impacted each test as equally as possible.

3.1 Generic Setup

Our experiments are based off of code provided by the authors of the Pensieve paper, which can be found at https://github.com/hongzimao/pensieve. For all of the Figure 11-based experiments (the simulated and real world experiments using the authors test video), the basic setup involves an ABR server being set up on the computer that is also serving video chunks, and a video client being run on another computer. The client computer requests video chunks from the serving computer, and reports data such as buffer occupancy, throughput estimates, etc. back to the ABR server. The ABR server responds to the client to inform the client what bitrate to request for the next chunk of video. The client then requests the appropriate bitrate chunk from the server. Once that chunk download completes, the client reports a performance summary for that chunk, describing the download time, amount of time spent rebuffering, buffer level etc. The authors performed all of their tests in a Selenium virtual display, and the code in the publicly available repository did not support actually streaming between two different computers. As a result, we had to modify the code base some to support this to allow for us to faithfully reproduce the results in the paper. Our changes to their code base can be found in the Github repository at https://github.com/hudson-ayers/pensieve. Notably, the authors did not provide any of their code to calculate video QOE – we only found scripts that calculated "reward", and we
were not sure how this "reward" metric mapped to QOE. Accordingly, we had to write scripts to calculate QOE ourselves based on the equations provided in the paper. The paper never provided specific units for the \( Qoe_{n,m} \) equation provided in Section 5 of the original paper. Accordingly, we messaged the authors and confirmed that the equation used Mbps and seconds for \( R_n \) and \( T_n \) respectively. Our scripts apply this equation directly to the results reported back by the video client for each chunk of video downloaded.

### 3.2 Simulated Networks

For the simulated tests, we set up the video server and video client on a single Linux machine with 8 GB of RAM and 2 CPU cores running Ubuntu 16.04. We ran the server directly on the machine for each experiment, and started the client browser from within a mahimahi environment with link capacity determined by network traces. We tested each algorithm (Pensieve, RobustMPC, and BOLA) on each network trace. We ran 10 tests for each algorithm, where each test consisted of the test video playing once (3.5 minutes long), making for a total of 30 minutes of test per algorithm on the network trace. Because each video was evaluated on the same network trace, we believe this is represents a fair comparison of the algorithms. We evaluated each algorithm on two network traces - a TMobile UMTS trace collected by driving around Palo Alto, and a Verizon LTE trace collected by a mobile user driving around. Both of these traces were obtained from the MahiMahi github repository [2]. In addition to using the network traces and mm-link to set the capacity of the link over time, we also used mm-delay to add 20 ms of latency to all of the tests. The results for these tests were plotted using the same method as was used for the real-world tests.

### 3.3 Real-World Networks

For the real-world network tests, we ran the video server on a GCE instance that had 2 vCPUs and 8 GB memory, running Ubuntu 16.04. The GCE instance was either located in the us-west1-a region or the asia-east1-a region for the international tests. For the real-world networks, we look at three connection types which match the ones presented in the paper: a Verizon LTE link, an international link, and a public WiFi link\(^1\). For the Verizon LTE link, a laptop was connected via an LTE hotspot to a GCE instance running in us-west1-a, and we moved to different environments to simulate the standard variability that would be seen with standard LTE usage. For the public WiFi link, a laptop was left connected to the WiFi network and talked to a video server running in a GCE instance running in us-west1-a.

\(^1\)Note that for the public WiFi connection, we used the Stanford Visitor WiFi network, as it is both rate-limited and has a large number of users.

Finally, for the international link, a laptop was connected via a wired Ethernet uplink to a video server running in a GCE instance in asia-east1-a.

For both the international link and the public WiFi link tests, a total of 100 tests were run for each of the three algorithms (RobustMPC, BOLA, and Pensieve) while for the LTE link test, a total of 10 tests were run for each.

### 3.4 Changed Video Retraining + Evaluation

In order to evaluate evaluate the dependence of Pensieve-generated ABR algorithms on the video used to train them, we had to change a good amount of code. First, we selected a new video – we chose the EnvivioDash video used by the authors of the MPC paper, as this video is open source and already available encoded at different bitrates. Next, we had to adjust the neural network to account for the fact that the new video was served at different bitrates (350/600/1000/2000/3000 vs. 300/750/1200/1850/2850/4300), was longer (by about 20 chunks), and had one fewer bitrate available (different Action dimension). We also had to modify all instances in the training process which were tied to these values, which occurred in over a dozen places. In order to test the new video, we also had to modify all aspects of the video serving process that was tied to these values. This included modifying the ABR server to have knowledge of the exact size of every chunk in the video, modifying the client’s dash.js code to have this same knowledge, and modifying the video Manifest to reflect the bitrates of the new video. Once these modifications had been made, we evaluated the QOE using MahiMahi emulated networks just as described for our simulated reproduction of Figure 11.

### 4 RESULTS

#### 4.1 Figure 11

In reproducing Figure 11, we used both simulated network traces and real-world network tests across the three algorithms (Pensieve, RobustMPC, and BOLA). The following sections discuss our experimental results for each of these tests, and interpret how our results were influenced by our experimental set-up.

**4.1.1 Real-World Tests.** For the real-world network tests, we measured the performance of all three algorithms across the three types of links depicted in Figure 11: a Verizon LTE link, an international link, and a public WiFi link. However, we found it difficult to exactly reproduce their results. On all of the listed network types, our QoE results were significantly higher than theirs, as all of the networks we tested on were fast enough to serve video chunks at the highest quality which led to a high QoE.
The original paper states that for Figure 11, each algorithm was run 10 times in round-robin order. Figure 1 shows our results from running each algorithm 10 times. This figure differs substantially from Figure 11 in the paper.

Figure 1: Comparison of Pensieve with other ABR algorithms across 10 tests on real world networks

![Figure 1](image1.png)

We were also interested in evaluating the performance of these algorithms as the number of tests increased. Figure 2 shows the QoE measurements from running 100 tests for the international link and the public WiFi network types. We were unable to run 100 tests for the Verizon LTE link due to monetary constraints, so the results for this network is only over 10 tests. The primary difference between these two figures is that the standard deviation (denoted by the error bars) decreases dramatically with the additional tests. The large sample size combined with the small variance for the international link and public WiFi tests increases our confidence in these results. Note that the only test with a large amount of variance is the Verizon LTE network, which makes sense as this sample size is fairly small and we attempted to mirror an LTE user’s experience by moving to different locations while running the tests.

4.1.2 Simulated Tests. In order to better understand how variance in the real-world networks might have affected the performance of each algorithm, despite our round-robin testing style, we decided it would only be fair to also evaluate each algorithm over simulated mahimahi links. The setup for these tests is described in the previous section. Due to each test being evaluated over identical network traces, we did not need to perform 100 tests for each scenario. Instead we played the video 10 times for each. The results are shown in Figure 3 below:

Figure 2: Comparison of Pensieve with other ABR algorithms across 100 tests on real world networks

![Figure 2](image2.png)

Figure 3: Comparison of Pensieve with other ABR algorithms across 10 tests over simulated cellular links

![Figure 3](image3.png)

These results are discussed in detail in the discussion section.

4.2 Figure 13

In reproducing Figure 13, we first trained Pensieve on a new single video. With this new version of Pensieve, we then measured the $QoE_{lin}$ performance of both the old and
new versions of Pensieve when downloading the original video over a mahimahi simulated link. We ran 10 tests over a mahimahi 3G and a mahimahi LTE connection. As in the original Figure 13, we plotted a CDF of the resulting QoE values, shown below in Figure 4.

These results are examined in detail in the following discussion section.

5 EVALUATION

5.1 Implications of Results

5.1.1 Figure 11. For Figure 11, our results differed significantly from the results provided in the paper. First, our Average QoE measurements were much higher than those reported in the paper. Further, our results had a much lower standard deviation for the public WiFi and international link tests for the same number of tests, as a result of our WiFi/International Links being more than capable of delivering high quality video streams. Our results do indicate that while Pensieve is comparable to the other algorithms for high-quality links, it has a slightly larger standard deviation than the other algorithms. Regardless, our results for these networks are not particularly interesting given that each algorithm managed to deliver the highest quality video possible for the entire duration of the video for most runs.

Our LTE results, however, are far to use to compare the algorithms. We were suprised to see that Pensieve exhibited the worst behavior of the three algorithms tested on this network given the results of the original paper. Given our issues with recording BOLA results, it is not necessarily fair to compare Pensieve against the BOLA results, but it is absolutely fair to compare it against RobustMPC. If the comparison is limited to these two algorithms, it can be seen that Pensieve did not outperform RobustMPC on the real world LTE links. Looking at the simulation based results reveals that Pensieve did outperform RobustMPC on the MahiMahi 3G results, though it did not outperform MPC on any of the LTE results (real or simulated). This suggests that Pensieve may perform better on difficult-to-predict 3G networks than it does on higher capacity networks. It also suggests that the influence of the HSPDA traces on training the Pensieve algorithm may have lead to our Pensieve trained algorithm not adapting as well to these higher capacity LTE networks.

While all of these results do not indicate that Pensieve is a bad algorithm, the difficulty in reproducing the results in the paper indicates that Pensieve may not outperform alternative algorithms across all network types. In particular, it suggests that Pensieve generated ABR algorithms may be more dependent on the type of network traces used to train the algorithms than the authors suggest in their own analysis of Figures 11 and 12 in the paper.

5.1.2 Figure 13. Our reproduction for Figure 13 was much closer to the paper than our results for Figure 11. Both the default Pensieve training and our new training are fairly close in the CDF plot, which is similar to the graph of Figure 13 in the original paper. However, it is important to note that we retrained Pensieve on a single video instead of multiple videos as shown in the paper. This distinction is important for two reasons; first, we were interested in seeing whether the single video the authors trained Pensieve on resulted in particularly good experimental results (as opposed to other single videos), and secondly, whether the authors’ claim that Pensieve was generalizable held for different videos.

As the graph demonstrates, our results appear quite similar. Since we were only able to test on two mahimahi link types (3G and LTE) with 10 tests each, we expect that our results have more variance than the results presented in Figure 13 (which had 100,000 tests).

One significant difference between our results and Figure 13 presented in the paper is the spread of QoE values in our results. Specifically, our graph includes QoE values from −9 to 4, while Figure 13 in the paper have QoE values only between −0.5 and 2.5. This is consistent with our results from reproducing Figure 11, as we also differed significantly from the paper in terms of our absolute QoE values. We find it particularly surprising that the CDF in the original paper indicates no runs with a QOE greater than 2.5 – this indicates that 0 of the traces analyzed was sufficiently high throughput to allow for the video to play at high quality the entire time. This seems to indicate that the set of traces used...
may have been too limited a selection of traces that did not capture enough medium-high capacity links.

5.2 Limitations and Weaknesses

5.2.1 Problems with BOLA. One of the primary limitations of our results is the problems with the way BOLA reports its results to the test server. We are not quite sure of the exact cause, but between 20 – 75% of the measurement results for BOLA are not sent to the test server. Unlike the other algorithms, BOLA is included in the dash.js media player, and the code for sending the measurements was put inside the player itself. However, under certain conditions the code for sending these measurements is not called, and BOLA fails to report measurement statistics for that chunk. The results that are reported are already normalized – that is, the throughput for each chunk is reported individually – so missing results would not drastically affect throughput or switching estimates. We suspect that the BOLA code may report results less frequently when it encounters adverse conditions, but we were unable to conclude this with complete confidence. We made several attempts to fix this bug, but were ultimately unsuccessful, as the dash.js state machine is quite complex and we were unable to isolate the race conditions leading to the missing reports.

If the missing results are in fact random selections from the results which would otherwise be reported, we can simulate and compare how such omissions could impact the other algorithms. We ran an experiment that randomly removed some number of results from each test for RobustMPC and Pensieve, then computed the overall resulting QoE value. We then ran this experiment 20 times, then plotted the lowest computed QoE and the highest computed QoE alongside the true QoE value. Figure 5 plots these results, and gives a rough bound on how much the BOLA error could impact the overall QoE. As the figure shows, these omissions can produce a difference of up to 0.8 QoE, which could significantly impact the accuracy of these results.

As we are still not sure if these omissions are random, we cannot place a precise bound on how much this error impacts either our results or the results in the paper. However, this analysis provides approximate, comparative bounds for random losses, and gives a realistic estimate for how much random omissions could impact the result. It is still an open question to figure out exactly why BOLA fails to report some results, and if these omissions are random or correlated with specific network conditions.

5.2.2 Limited Information in Paper. Another limitation of our results is that the paper does not provide a lot of information with regards to specific network conditions. Although they state what traces they use for the MahiMahi tests, their international link, public WiFi, and Verizon LTE tests do not have any information relating to bandwidth or latency. This proved to be a significant challenge, as our QoE results for each of these network link types are much higher than those provided in the paper. This indicates that our tests may have had links with more bandwidth and lower latency than the links they ran tests on, but since they did not provide any specific network information, we are unable to determine what the precise differences were between our experimental set-up and theirs.

It is important to note that for our tests, our computed QoE values and the QoE values their code computes is the same. Thus, the differences in QoE values is due to network conditions, and not a result of an error in our code.

6 FUTURE WORK

While our tests were fairly comprehensive, there are several aspects that could warrant additional research.

First, the issues with BOLA failing to report results should be examined in greater detail. While for our analysis we assumed that BOLA’s omissions were random, it appears that BOLA is more likely to omit results later in the playback. This may skew the BOLA results differently than random omissions, and it worthy of further investigation. Additionally, it would be ideal to solve the issue with BOLA to get a fair comparison with Pensieve, and see if the results still hold. However, as we mentioned previously, the BOLA error
is a fairly complicated race condition which we were unable to pinpoint.

Second, our results for Figure 11 differ substantially from those reported in the paper. Although we contacted the authors for specific bandwidth and latency numbers for the tests conducted in Figure 11, they have not yet responded. As a consequence, it would be worth it to figure out the exact discrepancies between our experimental set-up and the set-up in the paper.

7 CONCLUSION

The results we present above, and our discussion of these results that follows, may seem somewhat dour. However, we would like to emphasize that we do not believe these results are proof that Pensieve does not work, or that the original paper was intentionally dishonest. Indeed, we still believe that Pensieve outperforms other algorithms on the types of networks it is trained on, which is the primary conclusion made by the authors of the original Pensieve paper. Supporting this theory are our results that Pensieve outperformed RobustMPC on the 3G traces that we tested it on, and that BOLA outperforming it on these traces could be attributed to the issue of inconsistent results reporting by BOLA.

However, we do believe that our reproduction/extension shows that Pensieve trained on HSPDA and FCC broadband networks is not a model which can be expected to outperform other state-of-the-art algorithms such as RobustMPC and BOLA in more general deployments. Figure 11 of the Pensieve paper suggested that this was indeed the case, and we believe that our LTE tests in particular reveal this not to be particularly true. Our difficulties recreating the International Link tests further reveal that the authors did not specify enough about the complete testing setup used for the International link, as we believe the performance they observed is highly dependent on the edge networks that connected the client and server to the backbone link.

Our reproduction of Figure 13 generally agrees with the findings of the authors that the RL trained algorithms are not particularly dependent on the video used to train the algorithm. An algorithm trained on a different video still performs similarly to an algorithm being tested using the same video that trained the algorithm, exhibiting only slight worse performance when evaluated over a number of different traces.

Ultimately, the results of our reproduction lead us to believe two things. First, the primary result of the Pensieve paper is correct: Pensieve can effectively train an ABR algorithm to work well on networks similar to the networks it was trained on. Second, Pensieve ABR algorithms trained on certain network traces do not generalize particularly well to real world networks qualitatively different from the network traces used during training. As a result, it is difficult to imagine Pensieve being used in a large scale production deployment, as clients would be spread out across all different types of end networks, and it would be infeasible to train unique neural networks for each different client. However, it would be interesting to see if Pensieve trained across a much broader range of network traces would in fact perform well across a large variety of networks and successfully outperform other general algorithms such as RobustMPC and BOLA.

8 RECREATING OUR EXPERIMENTS

Our experiments cannot all be perfectly replicated by simply running a single script. There are several reasons for this. First, several of our tests require being run on specific real world networks. Second, several of our tests require setting up a server in a third party location in order to enable testing over these real world networks. Third, one of our tests requires retraining a neural network, which takes over 20 hours. Fourth, reproducing our tests requires navigating to specific websites in real browser windows and disabling the Chrome network cache, which is difficult to automate. Nevertheless, we provide tests that enable the reproduction of most of our experiments. Note that to recreate our Figure 13 tests the new neural network that we trained can be found on the new_training branch of the repository, in rl_server/results/new_video_model.ckpt

These experiments can be replicated by taking the following steps on an Ubuntu Linux Machine. We have tested this script on several different Ubuntu 16.04 installations. Notably, the steps generate output graphics, and thus should be run on a physical machine, or X11 forwarding should be enabled if they are run on a remote virtual machine over an SSH connection. The steps are as follows:

(1) Clone our github repository

```bash
$ git clone https://github.com/hudson-ayers/pensieve
```

(2) Run the setup and test scripts. This will generate the plots we made from the data we collected.

```bash
$ cd pensieve
$ python setup.py
$ cd real_exp
$ ./run_all.sh
```

(3) If you wish to collect new data running in your own browser on real world links:

(a) On the server (probably a google compute instance):

Enable incoming TCP traffic to ports 80, 8333,
8334, and 8335
$ ./run_100_round_robin.sh

(b) On the client machine:
Open Chrome. Ctrl+Shift+I to open dev tools.
Under the network tab, check "disable cache"
Open three chrome browser tabs to:
($SERVER_IP/myindex_RL.html)
($SERVER_IP/myindex_BOLA.html)
($SERVER_IP/myindex_.html)

(c) Wait 48 hours. Upon completion, the new QOE results will be in:
$ real_exp/old_results/Stanford_Visitor

(d) Move the results, run the plot results script:
$ cd real_exp/old_results
$ mv Stanford_Visitor/*
   ../results/figure11_100tests/
$ cd ..
$ python plot_qoe.py

Your new qoe results will be plotted using matplotlib.

(4) If you wish to collect new data using a network simulated via mahimahi:
(a) In one terminal:
$ cd pensieve
$ python2 real_world_run_video.py RL 4000 1
(b) In another terminal:
$ cd pensieve
$ mm-delay 20

$ mm-link
mahimahi_traces/TMobile-UMTS-driving.up
mahimahi_traces/TMobile-UMTS-driving.down
--meter-downlink
$ chromium-browser
(Ctrl+Shift+I -> Network -> check disable cache)
(Navigate to 100.64.0.1/myindex_RL.html)

(c) Wait 1 hour and 15 minutes. Upon completion new results will be in
$ real_exp/results

REFERENCES