THE UNIVERSITY OF CHICAGO

USER-CENTRIC OPTIMIZATIONS FOR INTERNET APPLICATIONS

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
XU ZHANG

CHICAGO, ILLINOIS
DECEMBER 2023

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

It is a profound privilege to achieve a Ph.D., an endeavor that has been made possible through the support and guidance of many.

Foremost, I extend my deepest gratitude to my advisor, Prof. Junchen Jiang, whose unwavering patience and invaluable guidance have been pivotal throughout my research journey. His mentorship has been instrumental in teaching me not only the intricacies of conceiving and executing research ideas but also the art of presenting them compellingly. His influence will undoubtedly resonate throughout my professional life.

I am equally indebted to Dr. Siddhartha Sen, whose remarkable ability to navigate research problems has profoundly shaped my approach to formulating research questions and authoring papers. His insightful feedback has bolstered my confidence in my work, and for that, I am immensely grateful.

My sincere thanks go to Prof. Nick Feamster, Prof. Marshini Chetty, and Prof. Paul Schmitt for their substantial contributions to my thesis. Prof. Feamster's insightful comments have broadened my perspective, deepening my understanding of the field. Prof. Chetty's expertise in user studies has been a cornerstone of my thesis, enriching its quality significantly. Prof. Schmitt has been instrumental in refining my research ideas and expanding my grasp of network measurement literature.

I am thankful to Dr. Hui Zhang and Prof. Haryadi Gunawi for their constructive feedback and enriching discussions. Dr. Zhang's contributions as a committee member introduced me to several intriguing future research directions, while Prof. Gunawi's experiences in systems were invaluable in the practical implementation of my ideas.

My colleagues Yihua Cheng, Daniar Kurniawan, Hanchen Li, Tony Ou, Dikaimin Simon, and YiZhuo Zhang deserve special mention for their assistance in building prototypes and conducting experiments. Their commitment and collaborative spirit have been essential to the success of this thesis.

Lastly, my enduring appreciation goes to my family, whose love and unwavering support provided the foundation upon which this achievement was built.

ABSTRACT

Quality of Experience (QoE) plays a key role in determining the revenue of Internet applications, such as web services and video streaming. While researchers work to improve low-level system performance metrics by using system resources (*e.g.,* network bandwidth and compute cycles) more efficiently, better low-level performance does not always correspond to a better QoE. This is because of users' different QoE sensitivities across different contexts (*e.g.,* application content, user preferences, environments, etc.). Not all low-level system performance improvements are perceptible by users.

In this thesis, we introduce a novel *user-centric* approach to QoE optimization. This approach prioritizes resource allocation based on the dynamic user QoE sensitivities towards performance metrics. However, implementing this approach in Internet applications is not straightforward. Firstly, learning QoE sensitivity can be expensive as it varies significantly across numerous contexts, necessitating this learning for each context. Secondly, such fine-grained resource allocation can introduce substantial system overhead.

In this work, we address these challenges and apply our proposed approach to web services and on-demand video streaming. We also introduce a user-study tool to help researchers learn QoE sensitivity more efficiently in terms of monetary cost and latency. Our experiments with real users demonstrate that through this approach, we can either improve user QoE without additional system resource consumption or achieve consistent QoE using fewer resources.

CHAPTER 1

INTRODUCTION

The landscape of networked applications has changed rapidly, with a sharp increase of applications now relying on high-quality video streams and fast web services, a proliferation of higher resolution videos (4K, 8K, and virtual reality), and above all, users becoming less patient with low-quality services. These changes greatly increase resource demands, in both network bandwidth and compute cycles, of today's Internet applications. This has created a significant challenge since scaling applications requires either lowering service quality or upgrading the network and backend system infrastructure (which can be slow and expensive). Although researchers have worked on improving user perception of Internet applications (video and web) for decades, the research community on computer systems has followed chiefly a system-centric approach, intending to maximally utilize system resources to optimize a predetermined set of low-level system metrics (*e.g.,* video bitrate, system processing latency, etc.).

We argue that these system-centric optimizations are not necessarily aligned with improvements in user perception since the users might perceive the same system metric in different ways. For instance, in a sports video, viewers might care more about the video quality of key moments, like a soccer goal, compared to other parts. So, if these important moments are of poor quality, viewers will be significantly more disappointed than if less critical parts were not of good quality. Our key insight is that such differences in user perception are large, and there are many types of user perception differences across different contexts, such as video content, internet connectivity, user preferences, etc. (more analysis for web services and on-demand video streaming are shown in Chapters 2 and 3). Such dynamic differences in user perception provide us with a new design space to optimize Internet application systems.

For instance, in a sports video, viewers might care more about the clarity of key moments, like a soccer goal, compared to other parts. So, if these important moments are of poor quality, viewers will be significantly more disappointed than if less critical parts were unclear. What we've realized is that users perceive quality in varied ways depending on the context, be it the type of video content, their internet speed, or personal preferences. This variation in user perception, which we'll explore further in Chapters 2 and 3, suggests new ways to enhance how Internet applications are designed and optimized.

This thesis delves into a *user-centric* approach for optimizing user perceptions of Internet applications, thereby enhancing the Quality of Experience (QoE). This entails collecting data on user perceptions to understand dynamic QoE perceptions across various contexts, followed by resource allocation based on these insights. Two primary challenges confront this approach: 1) the high cost of understanding perceptions for each context and 2) the potential system overhead from detailed resource allocation. To address the former, we employ crowdsourcing-based user studies for efficient data collection. For the latter, we employ perception dynamics at a coarse granularity to curtail system overhead.

In particular, this thesis focuses on applying our user-centric approaches to two popular applications: web services and on-demand video streaming. We also have designed a user-study tool that enables fast collection of user perception data at a low cost:

E2E is a resource allocator deployed on a web server. The users' sensitivity to web server delay is not the same, and it depends on the external delay experienced on the system components outside the web servers, *e.g.,* data transmission over the Internet. E2E has an efficient algorithm to automatically identify each web request's web server delay sensitivity and prioritizes the sensitive requests by allocating more computational resources with marginal system overhead.

Sensei leverages the significant heterogeneity in user perceptions of video content and user quality preferences to optimize the QoE of on-demand video streaming since users are more sensitive to quality issues when watching interesting content. In Sensei, we first model the user perception sensitivity to different content in a fast and cost-efficient fashion and then utilize the model to optimize video QoE by trading off video-streaming quality during insensitive content and quality during sensitive content under limited network bandwidth.

VidPlat is a crowdsourcing tool that helps researchers measure QoE with low monetary cost and latency. Different from the traditional tools that must determine low-quality issues to measure before the user study begins, VidPlat enables the researchers to dynamically determine those issues during the user study based on the past QoE measurements. It enables dynamic pruning of low-quality issues to measure for saving cost without conducting multiple sequential user-study tasks, significantly reducing user-study latency. Also, researchers can dynamically determine the number of QoE measurements needed for each low-quality issus, further saving the cost. We have open-sourced this tool online: `https://github.com/QoEStudies/Tool`.

The rest of this thesis is structured as follows. We first present E2E in Chapter 2, Sensei in Chapter 3, and VidPlat in Chapter 4, respectively. Chapter 5 concludes this thesis and discusses the future work.

CHAPTER 2

E2E: EMBRACING USER HETEROGENEITY TO IMPROVE QUALITY OF EXPERIENCE ON THE WEB

2.1 Introduction

Improving end-to-end performance is critical for web service providers such as Microsoft, Amazon, and Facebook, whose revenues depend crucially on high quality of experience (QoE). More than ten years have passed since Amazon famously reported every 100ms of latency cost them 1% in sales, and Google found 0.5s of additional load time for search results led to a 20% drop in traffic [15]. Today, latency remains critical but the consequences have gotten steeper: an Akamai study in 2017 showed every 100ms of delay in website load time hurt conversion rates by 7% [16], and Google reported higher mobile webpage load times more than double the probability of a bounce [17]. Naturally, web service providers strive to cut server-side delays—the only delays they can control—to improve the end-to-end performance of each web request. Following this conventional wisdom, a rich literature has developed around reducing web service delays (*e.g.,* [149, 163, 77, 101, 51, 92, 155]).

Our work is driven by a simple observation: although reducing server-side delay generally improves QoE, the amount of QoE improvement varies greatly depending on the *external delay* of each web request, *i.e.,* the total delay experienced prior to arriving at the web service due to ISP routing, last-mile connectivity, and so forth. In other words, if we define *QoE sensitivity* as the amount QoE would improve if the server-side delay were reduced to zero, there is substantial heterogeneity in QoE sensitivity across users. This heterogeneity results from two empirical findings. First, as illustrated in Figure 2.1(a), QoE typically decreases along a sigmoid-like curve as delay increases. When the external delay is very short or very long (*e.g.,* A or C on the curve), QoE tends to be less sensitive to the server-side delay than when the external delay is in the middle (*e.g.,* B on the curve). We verified this trend using traces from Microsoft's cloud-scale production web framework, as well as a user study we

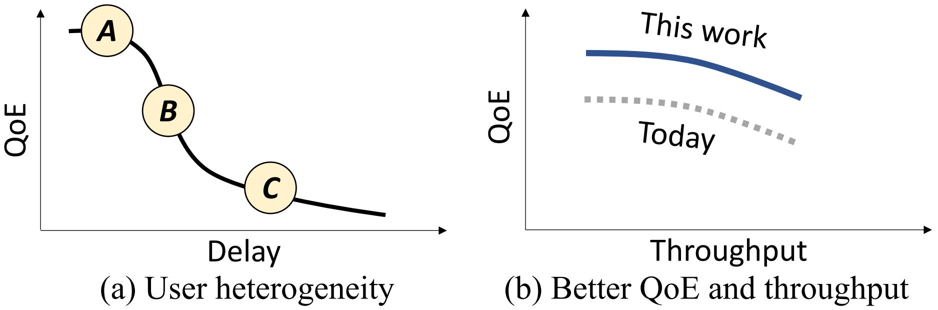


Figure 2.1: (a) An example of three requests with different QoE sensitivities to server-side delays, and (b) the potential QoE/throughput improvement if we leverage user heterogeneity. These figures are illustrative; actual figures from our evaluation and trace analysis appear later (*e.g.,* Figures 2.3, 2.6).

ran on Amazon MTurk to derive QoE curves for several popular websites (§2.2.2).

Second, external delays are inherently diverse across user requests to the same web service, due to factors that are beyond the control of the web service provider: *e.g.,* ISP routing, last-mile connectivity, DNS lookups, and client-side (browser) rendering and processing. Our analysis of our traces reveals substantial variability in external delays even among requests received by the same frontend web server, for the same web content (§2.2.2). The heterogeneity in QoE sensitivity implies that following the conventional wisdom of minimizing server-side delays uniformly across all requests can be inefficient, because resources may be used to optimize requests that are not sensitive to this delay. Instead, we should reallocate these resources to requests whose QoE *is* sensitive to server-side delay.

At a high level, user heterogeneity is inherent to the Internet's loosely federated architecture, where different systems are connected together functionally (client devices, ISPs, cloud providers, etc.), but delay optimization is handled separately by each system. Our work does not advocate against this federated architecture; rather, we argue that web service providers should *embrace the heterogeneity of QoE sensitivity across users* to better allocate server-side resources to optimize QoE. Using our traces, we show that if we could reshuffle server-side delays among concurrent requests so that requests with more sensitive QoE get lower server-side delays, we could increase the average duration of user engagement (a measure of QoE) by 28% (§2.2.3).

To explore the opportunities of leveraging user heterogeneity, we present *E2E*, a resource allocation system for web services that optimizes QoE by allocating resources based on each user's sensitivity to server-side delay.[1] E2E can be used by any shared-resource service; for example it can be used for replica selection in a distributed database to route sensitive requests to lighter-loaded replicas.

The key conceptual challenge behind E2E is that, unlike static properties of a request (*e.g.,* basic vs. premium subscription, wireless vs. wired connectivity), one cannot determine the QoE sensitivity of an arriving request based solely on its external delay. Instead, QoE sensitivity depends on the server-side delay as well. As we show in §2.3.2, if the server-side delay is large enough, it could cause a seemingly less sensitive request (A) to suffer more QoE degradation than a seemingly more sensitive request (B). Thus, one cannot prioritize the allocation of resources without taking into account both the external delay distribution and the server-side delay distribution. The latter distribution, in turn, is affected by the resource allocation itself, which makes the problem circular and computationally expensive to solve at the timescale of a web serving system.

E2E addresses this challenge from both the algorithmic perspective and the systems perspective. From the algorithmic perspective, E2E decouples the resource allocation problem into two subproblems, each of which can be solved efficiently: (1) a workload allocation process, which determines the server-side delay distribution without considering QoE sensitivity; and (2) a delay assignment process, which uses graph matching to "assign" the server-side delays to individual requests in proportion to their QoE sensitivity. E2E solves the two subproblems iteratively until it finds the best workload allocation and delay assignment (§2.4).

From the systems perspective, E2E further reduces the cost of processing each request by coarsening the timescale and the granularity of resource allocation decisions. Observing that the optimal allocation is insensitive to small perturbations in the external delay and server-

[1] E2E takes an "end-to-end" view of web request delays.

side delay distributions, we allow the system to cache allocation decisions in a lookup table and only update them when a significant change is detected in either distribution (§2.5).

We demonstrate the practicality of E2E by integrating it into two open-source systems to make them QoE-aware: replica selection in a distributed database (Cassandra) and message scheduling in a message broker (RabbitMQ) (§2.6). We use a trace-driven evaluation and our testbed deployments to show that (1) E2E can improve QoE (*e.g.,* duration of user engagement) by 28%, or serve 40% more concurrent requests without any drop in QoE; and (2) E2E incurs negligible (4.2%) system overhead and less than 100μs delay (§2.7).

This paper focuses on applying E2E to an individual service, or to multiple services that serve unrelated requests. In a production web framework, it is often the case that multiple backend services work together to complete the same (high-level) web request. Focusing on individual backend services allows us to develop our key idea of prioritizing requests based on how sensitive their QoE is to server-side delays, without the added complexity introduced by dependencies across services. We discuss these issues in §2.9.

2.2 Motivation

We first use our traces to show the prevalence of heterogeneity in how server-side delays impact the QoE of different users (§2.2.2). Then, we analyze the potential QoE improvement that could be attained by exploiting this heterogeneity for server-side resource allocation (§2.2.3).

2.2.1 Dataset

Our dataset consists of the traces of all web requests served by a production web framework cluster during one day in February 2018. The cluster is one of several located in an Eastern US datacenter serving the major websites and online storefront properties of Microsoft.[2]

[2] Examples include: microsoft.com, xbox.com, msn.com, etc..

	Page Type 1	Page Type 2	Page Type 3
Page loads (K)	682.6	314.1	600.2
Web sessions (K)	564.8	265.7	512.2
Unique URLs (K)	3.8	1.5	3.2
Unique users (K)	521.5	264.2	481.8

Table 2.1: Dataset summary (date: 02/20/2018)

Importantly, the traces include both client-side (browser) and server-side event logs: the client-side logs record all page rendering events and issued requests, while the server-side logs record all backend processing operations required to fulfill each request. Overall, the dataset spans 1.17M unique users and 1.6M page load events, as summarized in Table 2.1.

For each web request, we define three delay metrics, shown visually in Figure 2.2:

- The *total delay* (also known as page load time) is the duration between when a user clicks a link that issues the request and when the last object associated with the request is rendered.

- The *server-side delay* is the time to process all server-side operations on the backend, which may involve multiple steps, such as fetching product IDs from a database and then querying a product catalog for HTML description snippets, before aggregating the results and sending them to the user.

- The *external delay* includes all delays beyond the purview of server-side operations, *e.g.,* transferring data over the wide-area network, routing the request to the service, decoding and rendering the response in the client-side browser, etc..

We measure these delay metrics for each web request using the timestamps recorded in our traces. The total delay is measured by the difference between the first and the last timestamps associated with the request.

The server-side delay is measured by the total delay of all backend operations (with overlapping delays excluded). As mentioned above, we assume there is a single backend service; we discuss complex dependencies between backend services in §2.9. Finally, the

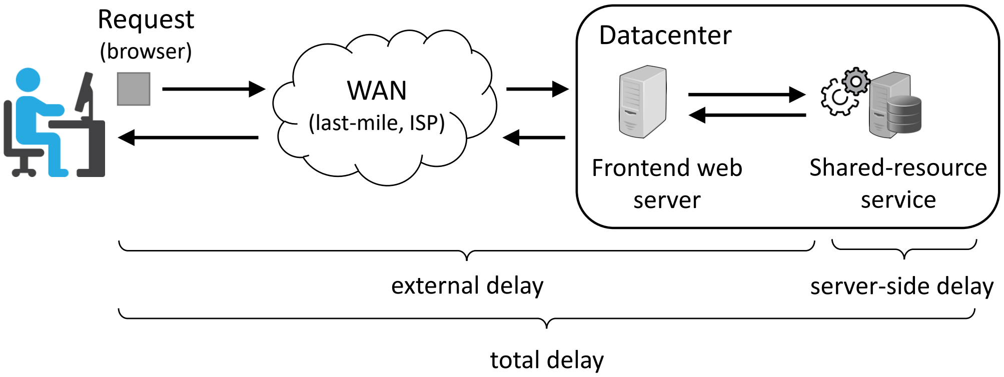


Figure 2.2: The life cycle of a web request, showing the total delay, server-side delay, and external delay.

external delay of a web request is calculated by subtracting the server-side delay from the total delay; it includes both wide-area network and datacenter delays, as shown in Figure 2.2. Note that this estimate of external delay is conservative because the actual delay may be smaller if server-side processing overlaps with wide-area transfers or browser rendering—our results improve as server-side delay becomes larger relative to external delay.

2.2.2 QoE sensitivity and its heterogeneity

Our basic intuition is that the impact of the server-side delay of a request on its QoE, *i.e.,* its QoE sensitivity, varies greatly across users. This follows directly from two observations, which we empirically demonstrate here: the sigmoid-like relationship between QoE and total delay, and the variability in requests' external delays.

Sigmoid-like QoE-delay relationship: The key to understanding the user heterogeneity lies in the non-linear relationship between QoE and delay. Figure 2.3 shows the QoE-delay relationship of requests to one particular page type. Like prior work, we estimate QoE by "time-on-site", measured as the difference between the start and end timestamps of their web session. A *web session* includes all of the user's engagement on the website, such as subsequent clicks and other interactions, with no period of inactivity greater than 30 minutes. Figure 2.3 groups the total delays into equal-sized buckets, each with at least 5,000 users, and plots the average QoE of users in each bucket. The key property of this graph is its *sigmoid-like* shape. Initially the total delay is small and the QoE is almost insensitive to any change in delay (the delay is too short for users to perceive); then the QoE starts to drop sharply with slope peaking at around 2,000 ms (this is the region where reducing total delay makes a difference); finally, when the total delay exceeds about 5,800 ms, the QoE becomes insensitive again (the delay is long enough that a little additional delay, while noticeable, does not substantially affect QoE). Accordingly, we can roughly categorize all user requests into three sensitivity classes:

- *Too-fast-to-matter* (left blue-shaded area): QoE is not sensitive to server-side delay if total delay is below 2000 ms.
- *Sensitive* (middle orange-shaded area): QoE is sensitive to server-side delay when total delay is between 2000 ms and 5,800 ms.
- *Too-slow-to-matter* (right red-shaded area): QoE is not sensitive to server-side delay if total delay exceeds 5,800 ms.

The sigmoid-like curve may look similar to deadline-driven utility curves commonly used in prior work (*e.g.,* [131, 51]), but there is a difference. Traditionally, a service deadline is set where the QoE starts to drop. But our analysis shows that when the total delay exceeds any threshold, the QoE does not drop to zero immediately, and instead decreases gradually as total delay increases. As we will see in §2.7.4, this difference can cause deadline-driven schemes to have suboptimal QoE.

We acknowledge that time-on-site may not always reflect how satisfied users are with the web loading experience. Therefore, we complement the above analysis with an IRB approved user study[3] on Amazon MTurk [1]. We describe the detailed setup in Appendix .2 and only give a summary here. Following similar work in the crowdsourcing literature [156], we asked participants to watch a web page load with different total delays and then to rate their experience on a scale of 1-5. The total delays were randomly permuted per user to avoid

[3] Our study was approved by University of Chicago, IRB18-1096. It does not raise ethical issues.

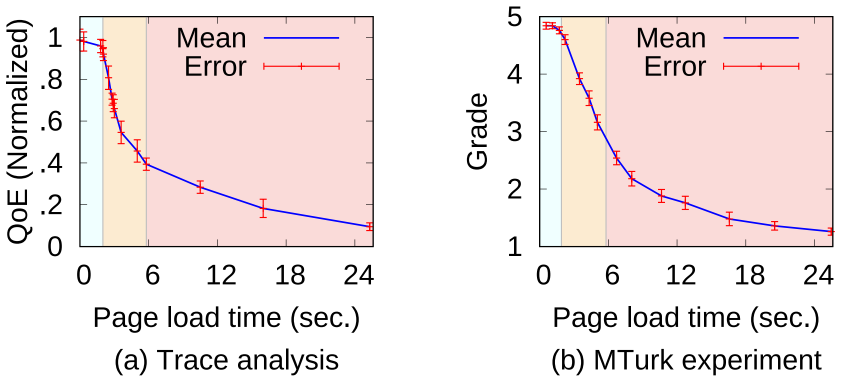


Figure 2.3: We observe a non-linear relationship between QoE and total delay (a), so reducing delay by the same amount can have a dramatically different impact on QoE. We highlight different sensitivity regions with different colors. The same QoE-delay relationship is observed in our MTurk-based user study (b).

any bias due to ordering. We ran this user study on the same web page as in Figure 2.3(a) and plot the resulting QoE curve in Figure 2.3(b). As the figure shows, the curve from the user study shares the same sigmoid-like shape as the curve from our trace analysis. We also repeated the user study on four other popular websites; all websites yielded similar sigmoid-like QoE curves, though the boundaries of the three sensitivity regions vary slightly across the sites.

Although our observations about the QoE-delay relationship do not seem different from prior work (*e.g.,* [35, 55]), they have deeper implications when combined with the next empirical observation on the variability of external delays.

Variability in external delays: The sigmoid-like relationship between QoE and delay means that the sensitivity of QoE to server-side delay depends heavily on the external delay. Figure 2.4 shows the distribution of external delays among requests for the same web page received at the same frontend web cluster. We see a substantial fraction of requests in each of the three sensitivity classes (25% too-fast-to-matter, 50% sensitive, 25% too-slow-to-matter). The same kind of distribution holds across web pages and is stable over time in our traces.[4] Note that the variance in Figure 2.4 is unlikely due to datacenter-level geographical

[4] The total delay distributions in our traces are consistent with those observed in prior work [38], though

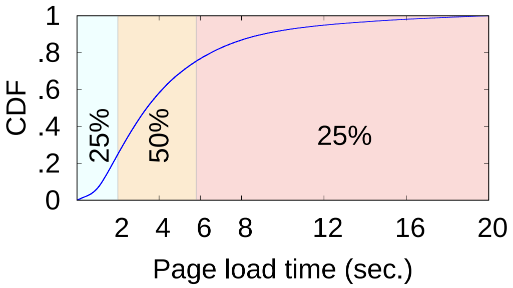


Figure 2.4: External delays exhibit great variance even among requests received by the same web server cluster for the same page content.

differences, since our traces use a global network of edge proxies to route users from the same region to the same datacenter cluster, although this does not exclude geographical differences users in the same region. It is also unlikely due to application-level differences, since the requests are all targeting the same web page. In practice, a web service provider may see even greater variability in external delays if its edge proxies are less widely distributed than our traces (causing each datacenter cluster to serve a larger geographic region), or if requests are processed by a more centralized architecture (*e.g.,* in many video streaming CDNs [170]).

Since external delays are beyond the control of the web service provider, they are an *inherent* property of the request from the perspective of the service provider. This is in contrast to server-side delays, which the service can influence.

2.2.3 Potential for improvement

We now use a trace-driven simulation to demonstrate the opportunity of leveraging the heterogeneity of QoE sensitivity to server-side delays. Suppose the dataset has n requests $R = \{r_1, \dots, r_n\}$, and the server-side delay and external delay of request r_i are s_i and c_i, respectively. Let $Q(\cdot)$ be the QoE function that takes total delay as input and returns the expected QoE. The current QoE of r_i can thus be denoted by $V_i^{old} = Q(s_i + c_i)$. Table 2.2 summarizes our notation.

Reshuffling server-side delays: Now, let us consider a simple analysis to counterfactually

they may still vary with website type (*e.g.,* online shopping vs. search engine).

estimate the benefit of allocating resources based on QoE sensitivity. We preserve both the external delay of each request and the collection of server-side delays, but we *re-assign* the server-side delays to requests as follows. We first rank all requests in order of their derivative on the QoE curve, $-\left.\frac{dQ}{dx}\right|_{x=c_i}$, representing the impact on QoE of a small change in server-side delay. Then, we assign the k^{th}-largest server-side delay to the request with the k^{th}-smallest derivative (*i.e.,* the k^{th}-least sensitive request to server-side delay). Let π denote the resulting permutation of server-side delays, *i.e.,* request r_i now has server-side delay $s_{\pi(i)}$. So the new QoE of request r_i is $V_i^{new} = Q(s_{\pi(i)} + c_i)$.

Intuitively, the above re-assignment gives small server-side delays to requests that are sensitive to them, and larger delays to requests that are less sensitive. If the server-side delays s_i are sufficiently small, this assignment can be shown to be optimal, as follows. The average QoE can be written as $\frac{1}{n}\sum_{i=1}^{n} Q(s_{\pi(i)} + c_i) = \frac{1}{n}\sum_{i=1}^{n} s_{\pi(i)} Q'(c_i) + \frac{1}{n}\sum_{i=1}^{n} Q(c_i)$. Suppose the c_i are given and w.l.o.g. $c_1 \leq \cdots \leq c_n$, then this expression is maximized when $s_{\pi(1)} \leq \cdots \leq s_{\pi(n)}$.

Practicality of simulation: To avoid assigning improbable server-side delays to the requests, we first grouped the requests by page type within one-minute time windows, and only re-assigned server-side delays among requests in the same group and 10-second time window. In other words, we do not assign the server-side delay of an off-peak-hour request to a peak-hour request, or the server-side delay of a simple static page request to a complex page request. We also verified that the server-side delay distributions exhibit only negligible changes within a time window. Nonetheless, there are two important caveats. First, our analysis assumes the server-side delays can be arbitrarily re-assigned among requests, which of course is impractical. Second, the analysis uses a very simple algorithm that assumes the set of server-side delays is fixed. In practice, server-side delays are difficult to predict and depend on how resources are allocated to requests. These issues make it challenging to achieve the QoE gains predicted by our simulation; later sections address the issues to extract as much gain as we can manage.

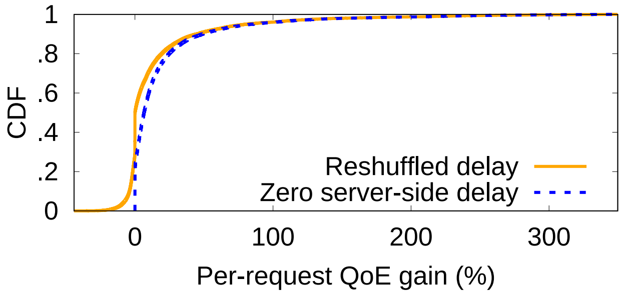


Figure 2.5: Potential QoE gains through better allocation of server-side resources based on QoE sensitivity. By reshuffling server-side delays (solid yellow line), we achieve significant QoE gains that are close to the (unrealizable) ideal of zero server-side delays (dashed blue line).

Potential gains in QoE and throughput: Figure 2.5 shows the distribution of QoE improvements over all requests, *i.e.,* $(Q_i^{new} - Q_i^{old})/Q_i^{old}$, as predicted by our simulation. We see that a small fraction of requests (less than 15.2%) suffer a marginally worse QoE under the new assignment, but a substantial fraction of requests (over 27.8%) see QoE improve by at least 20%. Overall, the new average QoE is 15.4% higher than the old QoE. These improvements are consistent across different page types in the traces. Note that although the new assignment may worsen tail QoE, requests at the tail have such small QoE derivatives that the additional degradation is marginal. We conclude that there is substantial room to improve QoE for a substantial fraction of users, without changing the distribution of server-side delays.

Similarly, we can also support more concurrent requests, *i.e.,* higher throughput, while maintaining a similar level of QoE. To estimate the gain in throughput, we apply our reshuffling of server-side delays to peak hours (higher throughput but worse QoE) and to off-peak hours (lower throughput but better QoE). Figure 2.6 shows the throughput and QoE during these two periods of time. We randomly select web requests from two peak hours (4pm and 9pm) and three off-peak hours (12am, 3am, 10pm), all in the Eastern Time Zone. For every 10 minutes, we pick the last 10-second window, reshuffle the server-side delays within the time window, and measure the new QoE as above. We can see that the new average QoE during peak hours is similar to (even higher than) the old QoE during off-peak hours. In

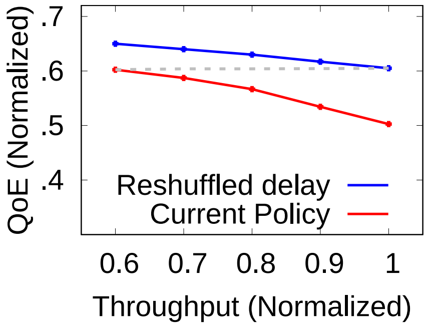


Figure 2.6: Potential throughput improvement with similar QoE, achieved by reshuffling server-side delays during peak hours and off-peak hours.

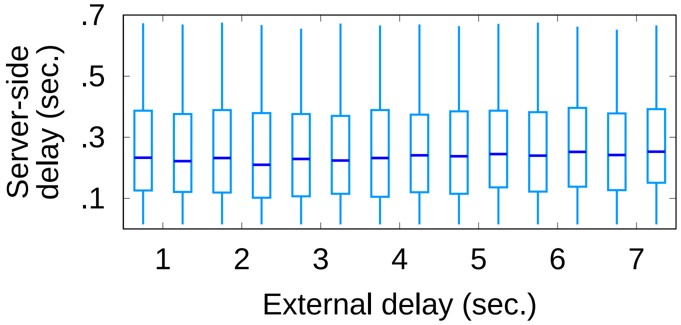


Figure 2.7: Current server-side delays are uncorrelated with external delays, showing that the existing resource allocation policy is agnostic to QoE sensitivity. (Candlesticks show $\{5, 25, 50, 75, 95\}$ percentiles.)

other words, if we only apply our approach during peak hours, we could support 40% more users without any drop in average QoE.

Now, there are two contributing factors that suggest why these potential gains can be realized over existing systems.

1. *Existing systems are agnostic to user heterogeneity.* Figure 2.7 shows the distribution of server-side delays in a 10-second window for requests whose external delays fall into different ranges. We see that there is little correlation between the external delay and the corresponding server-side delay, which suggests that current resource allocation and processing of these requests is agnostic to QoE sensitivity. Our discussions with the Microsoft product teams represented in our traces corroborate this finding.

2. *Server-side delays are highly variable.* Figure 2.8 shows that there is a substantial

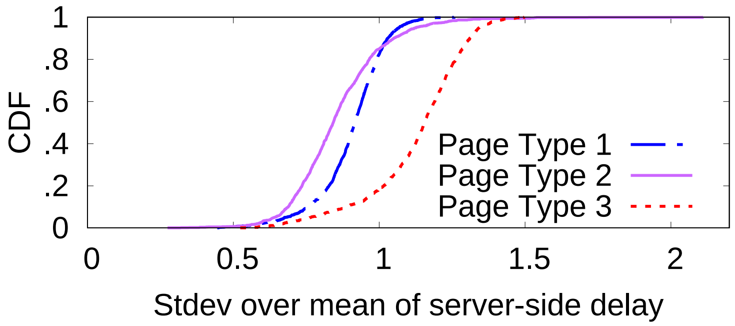


Figure 2.8: Server-side delays are highly variable, and not just at the tail. This holds for different page types.

variability in server-side delays even among requests for the same page type. Part of this variance is due to tail performance (as observed in prior work), but the lower percentiles also show substantial variance. This variance in server-side delays creates the “wiggle room” that makes the improvements in Figure 2.5 possible.

2.2.4 Summary of key observations

The findings in this section can be summarized as follows:

- The variability of external delays across users and the sigmoid-like relationship between QoE and page load time give rise to heterogeneity in the QoE sensitivity of users to server-side delays.

- Our trace-driven simulation shows that by allocating server-side delays based on the QoE sensitivity of each request, one could potentially improve QoE by 20% with the same throughput, or improve throughput by 40% with the same QoE.

- Existing server-side resource allocation is largely agnostic to external delays, while server-side delays exhibit high variance, which together create the opportunity to significantly improve QoE over current schemes.

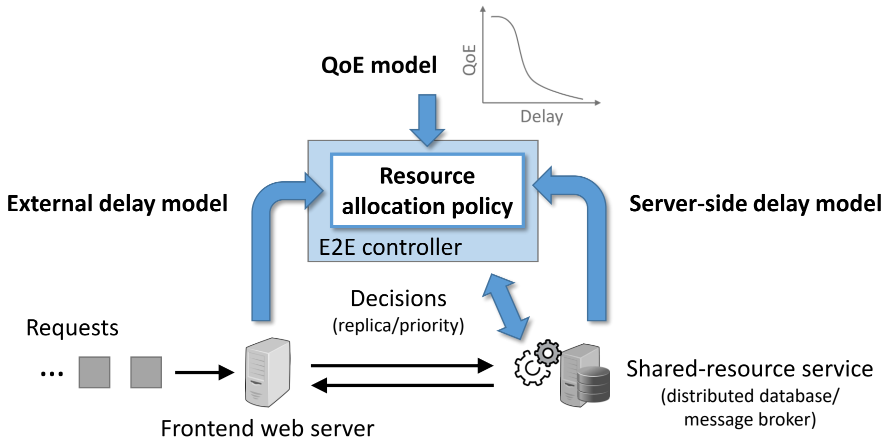


Figure 2.9: Overview of E2E.

2.3 E2E: Overview

The next few sections describe E2E, a general resource allocation system for web services that realizes the potential QoE and throughput gains of leveraging user heterogeneity.

2.3.1 Architecture

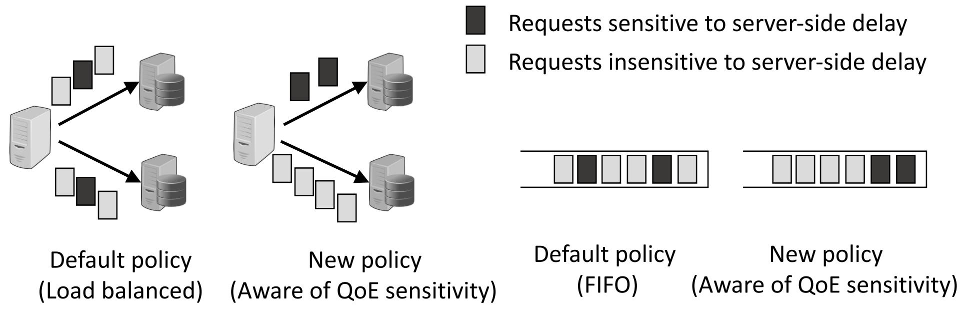
Figure 2.9 illustrates the main components of E2E and how it interacts with a web service system. Typically, a web request is first received by a frontend web server (Figure 2.9 depicts only one web server, but there may be multiple), which then forwards the request to a backend infrastructure service (*e.g.,* a distributed database or a message broker) whose compute/network resources are shared across requests. E2E provides a resource allocation policy for the shared service that makes a *decision* for each request, *e.g.,* telling it which replica to route the request to in a distributed database, or what priority to assign the request in a message broker. Figure 2.9 depicts only one shared-resource service, but in general E2E can serve multiple services (or multiple applications within a service) simultaneously, *provided these services do not interact on the same request.* We discuss interrelated services, such as those used to aggregate results for a high-level web request, in §2.9.

E2E takes as input three variables: an offline-profiled *QoE model* (such as the ones in Figure 2.3), an *external delay model* from the frontend web servers, and a *server-side delay* *model* from the shared-resource service. The external delay model provides the distribution of external delays across requests and an estimate of the current request's external delay. This external delay is then tagged as an additional field on the request and on any associated subrequests (similar to [51]). The server-side delay model provides an estimate of the server-side delay of a request based on the decision and the current workload. Based on these inputs, E2E returns a decision per request for how to allocate resources to it.

We discuss how server-side delays and external delays are estimated in §2.6.

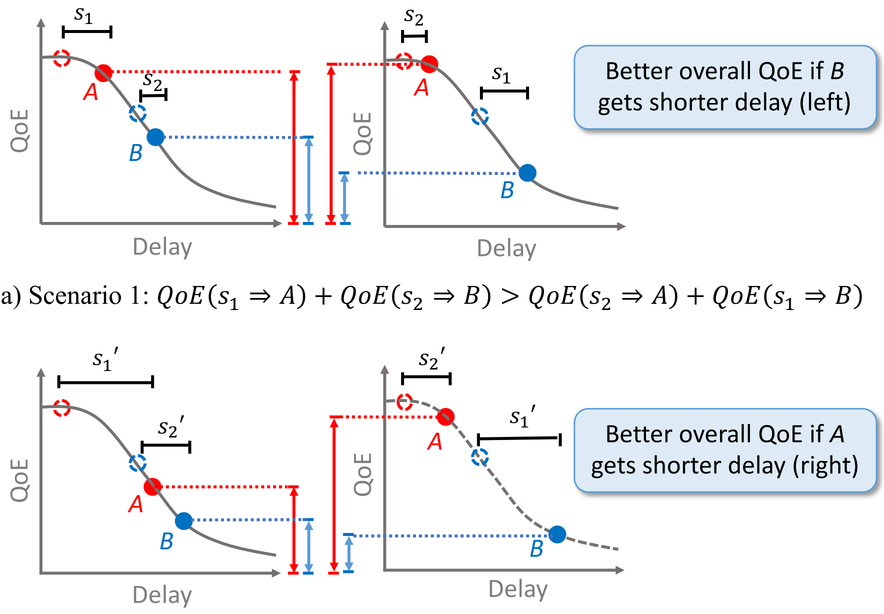
Figure 2.10 gives two illustrative examples of how E2E might affect resource allocation policies, for a distributed database and a message broker. In particular, E2E can improve the requests' QoE in two ways. First, E2E can assign more QoE-sensitive requests to decisions that have lower server-side delays, *e.g.,* a less loaded replica in a distributed database. Second, E2E can allocate resources to affect the server-side delays, in order to reduce the delays for QoE-sensitive requests. Even if E2E cannot predict server-side delays exactly, it can still create a discrepancy between the delays experienced by requests of different QoE sensitivities. For instance, as illustrated in Figure 2.10(a), E2E can assign uneven loads across the replicas of a distributed database, so that less loaded replicas are available to process QoE-sensitive requests with faster response times.

The next two sections present E2E's resource allocation policy and control interface, using the distributed database and message broker as two concrete examples of a shared-resource service. In general, E2E makes very few assumptions about how a shared service processes requests or physically shares its resources; it only requires the service to expose an API for controlling decisions (*e.g.,* the replica to select, the priority of a request, etc.). Also, our work places less emphasis on the prediction of external/server-side delays, or the implementation of a control plane on which E2E's resource allocation policy may run. Existing work already addresses and provides general solutions for these aspects (*e.g.,* [131, 51, 50]).


Requests sensitive to server-side delay
Requests insensitive to server-side delay
Default policy
(Load balanced)
New policy
(Aware of QoE sensitivity)
Default policy
(FIFO)
New policy
(Aware of QoE sensitivity)

(a) Replica selection in distributed database (b) Scheduling in message broker

Figure 2.10: Examples of how E2E may allocate resources differently in (a) a distributed database and (b) a message broker.


s_1
s_2
A
B
QoE
Delay
s_2
s_1
A
B
QoE
Delay
Better overall QoE if B gets shorter delay (left)
s_1'
s_2'
A
B
QoE
Delay
s_2'
s_1'
A
B
QoE
Delay
Better overall QoE if A gets shorter delay (right)

(a) Scenario 1: $QoE(s_1 \Rightarrow A) + QoE(s_2 \Rightarrow B) > QoE(s_2 \Rightarrow A) + QoE(s_1 \Rightarrow B)$

(b) Scenario 2: $QoE(s_1' \Rightarrow A) + QoE(s_2' \Rightarrow B) < QoE(s_2' \Rightarrow A) + QoE(s_1' \Rightarrow B)$

Figure 2.11: Illustration of how allocating resources based solely on requests' external delays can lead to suboptimal QoE. Scenarios 1 and 2 have the same pair of requests but different server-side delays. We use the assignment of server-side delays to represent resource allocation. In scenario 1, assigning the shorter server-side delay (s_2) to B and the longer one (s_1) to A leads to better overall QoE. But in scenario 2, giving the shorter delay (s_2') to A leads to worse overall QoE.

2.3.2 Key challenge

The key challenge behind E2E is that the optimal decision for a request *cannot* be determined from the request alone. Instead, the decision depends on the external delay distribution of other requests as well as the server-side delay distribution, which itself is a function of these decisions. Figure 2.11 illustrates a simple example where prioritizing requests purely based on external delay can lead to a bad decision, and shows how to improve it by taking the server-side delays and other requests' external delays into account. The key observation is that the non-convexity of the QoE-delay curve may cause the sensitivity of a request's QoE to flip depending on the external delay *and* the magnitude of the server-side delay.

This property makes it challenging to design a scalable decision-making policy. In particular, the circular dependence between server-side delays and resource allocation decisions makes the problem algorithmically expensive; and the need to account for other request's external delays adds processing overheads.

The above makes E2E conceptually different from many other request scheduling problems where each request has an innate property that indicates its urgency, such as subscription type (*e.g.,* premium vs regular users) or the application's delay sensitivity (*e.g.,* video streaming vs. web pages). Notably, Timecard [131] and DQBarge [51], two closely related systems to ours, use the external delay to directly determine the processing deadline of each request in isolation, without considering other requests or the global impact on available resources (see §2.8).

2.4 E2E: decision policy

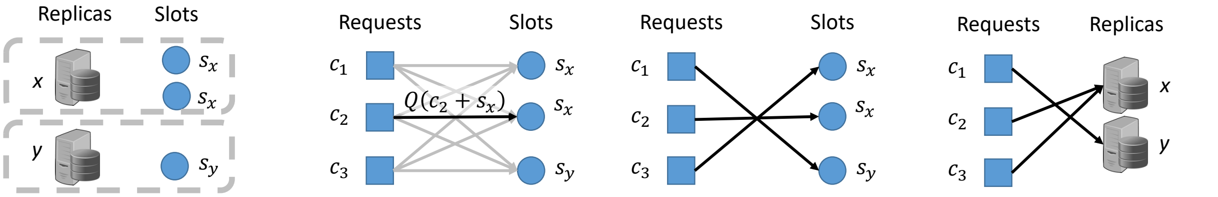
This section describes E2E's decision-making policy for allocating resources to requests.

2.4.1 Problem formulation

We start by formulating the problem of E2E. Table 2.2 summarizes our terminology. We use r_i, c_i, s_i, z_i to denote the i^{th} request, its external delay, server-side delay, and allocation decision, respectively. Given n concurrent requests $r_1,..., r_n$ whose external delays $c_i..., c_n$ are provided by the external delay model, E2E finds the decision vector Z=(z_1,..., z_n) that

Term	Brief description
r_i; R	request; vector of requests
c_i; C	external delay of r_i; vector of external delays
s_i; S	server-side delay of r_i; vector of server-side delays
$Q(\cdot)$	QoE model; $Q(d)$ returns the QoE of total delay d
z_i; Z	allocation decision of r_i; vector of decisions
$G(\cdot)$	server-side delay model; $G(Z)$ returns the server-side delay vector of decision vector Z

Table 2.2: Summary of terminology



(a) Obtain server-side delays of the decision allocation from $G(\cdot)$

(b) Construct a bipartite graph between requests and decisions

(c) Find a maximum bipartite matching

(d) Translate bipartite matching into replica selection decisions

Figure 2.12: Running our request-decision mapping algorithm on an example replica selection scenario with three requests (c_1, c_2, c_3) and two replicas (x, y). The given decision allocation is two requests for replica x and one request for the replica y. The final request-decision assignment is optimal for the decision allocation if and only if the corresponding bipartite matching is maximum.

maximizes

$$\frac{1}{n}\sum_{i=1}^{n} Q(c_i + G(z_i, Z)),$$

where $Q(d)$ is the QoE of a request with total delay d, as estimated by the QoE model; and $G(z, Z)$ is the server-side delay of a request assigned to decision z given that the complete decision vector is Z, as estimated by the server-side delay model. We assume that the QoE, external delay, and server-side delay models are known and provided as input; we discuss their implementation in §2.6. For now we assume the server-side delay model $G(\cdot)$ returns precise (noise-free) estimates; we relax this assumption at the end of §2.4.3.

Unfortunately, solving this problem is computationally hard, because it has to take two dependencies into account:

1. The amount of resource allocated by z_i to a request i depends on how much impact the resource would have on the request's QoE. But this impact is not linear: as more resources are given to the request, the improvement to its QoE may increase or diminish (since Q is non-linear with respect to server-side delay $G(z_i)$).

2. The resource allocation among a set requests depends on the server-side delay distribution, which is itself a function of the resource allocation.

Mathematically, this problem is NP-hard; the proof is beyond the scope of this paper (readers can refer to [154]), but the basic hardness lies in the non-convexity of function Q.

2.4.2 Two-level decision-making policy

Our approach to addressing the above intractability is to *decouple* the problem into two levels, as shown in Algorithm 1). The bottom level finds the best *request-decision mapping* for a given decision allocation, where a *decision allocation* is the number of requests assigned with each possible decision (*e.g.,* in a distributed database the possible decisions are the different replicas). The top level uses a simple hill-climbing search to try different decision allocations, find the best request-decision mapping for each allocation (by invoking the bottom level), and repeating until a decision allocation with the best QoE is found. The rationale behind this search strategy is that requests are functionally identical, so the server-side delay model depends only on the decision allocation—*e.g.,* the number of requests assigned to each replica, not which specific requests are assigned—allowing us to drastically reduce the search space from all possible resource allocations to all possible decision allocations. Since the number of possible decisions is typically small (*e.g.,* the number of replicas or priority levels), this is a large savings.

Algorithm 1: E2E's two-level decision-making policy.

```
Input: 1) A vector of $n$ requests $(r_1, \dots, r_n)$,
2) external delay of $r_i$ is $c_i$,
3) Number of possible decisions $k$
Output: Decision vector $Z = (z_1, \dots, z_n)$, $z_i$ is decision of $r_i$
/* Initialize decision allocation */
$(n, 0, \dots, 0) \rightarrow W$
/* Find the best decisions and QoE under $W$ */
RequestDecisionMappingAlgorithm($W$) $\rightarrow Z$
$\sum_i Q(c_i + G(z_i, Z)) \rightarrow q$
while HillClimbing($W$) $\rightarrow W' \neq \phi$ do
    RequestDecisionMappingAlgorithm($W'$) $\rightarrow Z'$
    $\sum_i Q(c_i + G(z'_i, Z')) \rightarrow q'$
    /* Update $Z$ if hillclimbing step improves QoE */
    if $Q' > Q$ then
        $Z' \rightarrow Z, q' \rightarrow q$
```

On the other hand, finding the best request-decision mapping for a given decision allocation can be done optimally and efficiently, by viewing it as a graph matching problem. We present the details of this algorithm next.

2.4.3 Request-decision mapping algorithm

For a given decision allocation, we compute the optimal assignment of requests to decisions by following a four-step process, illustrated in Figure 2.12 through the example of a replica selection scenario:

1. **Figure 2.12(a)**: Create n "slots" corresponding to the decision allocation and obtain their server-side delays from $G(\cdot)$. In this case there are three slots, two for replica x and one for replica y, with server-side delays s_x, s_x, s_y.

2. **Figure 2.12(b)**: Construct an n-to-n bipartite graph where nodes on the left are requests and nodes on the right are slots, and the weight of the edge from request r_i to slot s is $Q(c_i + s)$, *i.e.,* the expected QoE of the request if assigned with this decision.

3. **Figure 2.12(c)**: Find a maximum bipartite matching, *i.e.,* a subgraph where each node has exactly one edge and the total weight is maximized.

4. **Figure 2.12(d)**: Translate the matching to a request-decision assignment: each request is assigned the decision corresponding to the slot it is linked to. In this example the final decisions are: $c_2 \rightarrow x, c_3 \rightarrow x, c_1 \rightarrow y$.

The key insight is to cast the problem of maximizing the QoE of a request-decision mapping to that of maximizing a matching in a bipartite graph, for which polynomial-time algorithms exist [95, 66]. The polynomial is cubic in the number of requests, so care must be taken to ensure an efficient implementation; this is addressed in §2.5.

In practice, the server-side delay model $G(\cdot)$ estimates a distribution of the server-side delay, not an exact value, so the request-decision mapping algorithm (Figure 2.12) needs to be modified as follows. Instead of labeling each slot with a fixed value in Figure 2.12(a) (*e.g.,* s_x), we label it with a probability distribution $f_x(s)$ (provided by $G(\cdot)$), and label the edge in Figure 2.12(b) between request r_i and the slot with the *expected* QoE over this distribution, *i.e.,* $\int_0^\infty Q(c_i + s) f_x(s) \mathrm{d}s$.

2.5 E2E: Decision overhead

E2E's has to make a resource allocation decision for each request, and this decision might change if one or more of the input variables (QoE model, external delay model, server-side delay model) changes. This overhead can quickly become unscalable if left unchecked.

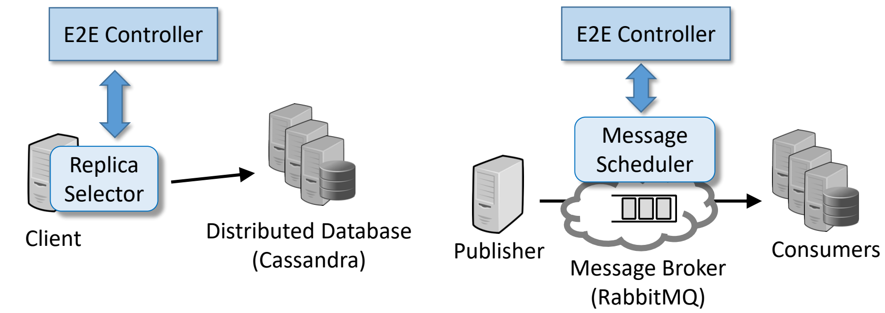
Our idea for reducing the decision-making overhead is to coarsen the granularity of decisions along two dimensions: (1) spatially grouping requests with similar characteristics, and (2) temporally caching decisions that are updated only when a significant change occurs in the input variables. Although these are heuristics with no formal guarantees, we find that they work well in practice (§2.7).

2.5.1 Coarsening spatial granularity

We coarsen decisions spatially by grouping requests into a *constant* number of buckets based on their external delays. Specifically, we split the range of external delays into k intervals, and all requests whose external delays fall in the same interval are grouped in the same bucket. We then run E2E's decision-making policy over the buckets rather than individual requests, and assign the same final decision to all requests in a bucket. This coarsening ensures that the running time of the decision-making process is always constant, rather than growing with the cube of the number of requests (the fastest bipartite matching algorithm [66, 95]). To minimize the amount of QoE degradation caused by making decisions at the bucket level, the external delay intervals satisfy two criteria: (1) they evenly split the request population, and (2) the span of any interval does not exceed a predefined threshold δ. Our evaluation shows these criteria are effective.

2.5.2 Coarsening temporal granularity

We have empirically observed that the same decision assignment can yield close-to-optimal QoE even if some of the inputs to E2E's decision-making policy have changed slightly. Therefore, E2E caches its decision assignment in a *decision lookup table* that the shared-resource service can query for every new request. The keys in this table are the buckets of the external delays, and the corresponding value is the decision assigned to each bucket. The exact definition of decisions varies across use cases. For instance, in a distributed database, the decision of a specific external delay bucket is the probability of sending a request to each of the replicas, if the request's external delay falls in the bucket. The lookup table is only updated when one of the input variables has changed by a "significant amount". The policy for deciding this is orthogonal and not something we prescribe; *e.g.,* it could be if the J-S divergence [116] between the new and old distributions exceeds a certain threshold.



(a) Use case #1: Replica selection in a distributed database

(b) Use case #2: Message scheduling in a message broker

Figure 2.13: Use cases of E2E

2.5.3 Fault tolerance of E2E controller

In E2E, a request needs to wait for its resource allocation decision from the E2E controller, which can therefore become a single point of failure for the whole system. This can be mitigated in three ways. First, if the E2E controller fails, the shared-resource service can still make QoE-aware decisions by looking up the request's external delay in the most recently cached decision lookup table (see above). Second, the E2E controller is replicated with the same input state (QoE model, external delay model, server-side delay model), so when the primary controller fails, a secondary controller can take over using standard leader election [37, 88]. Finally, in the case of total E2E failure, the shared-resource service can simply bypass E2E and use its default resource allocation policy.

2.6 Use Cases

We demonstrate E2E's practical usefulness by integrating it into two popular web infrastructure services, depicted in Figure 2.13: replica selection in a distributed database and message scheduling in a message broker. In both cases, E2E makes minimal changes to the shared-resource service and only relies on the control interface exposed by them. We evaluate E2E's overhead in §2.7.3.

2.6.1 Use case #1: Distributed database.

We choose Cassandra [2] as the distributed database, and use E2E to select the replica for each request (this operation is common to other distributed databases, and not specific to Cassandra). In particular, we made two changes. First, we modified the existing replica selection logic (`getReadExecutor` of `ReadExecutor`) of the Cassandra client. Our new logic stores the decision lookup table (§2.5) received from the E2E controller in a local data structure. When a new request arrives, it looks up the request's external delay in the table to get the selected replica's IP. Second, we modified the client service callback function (in `RequestHandler`) to keep track of the load (number of concurrent requests) and the observed (server-side) delay of each replica. In practice, the replication level, *i.e.,* the number of replicas for each key, is usually much smaller than the total number of servers. A simple replication strategy, adopted by Cassandra and other databases like MongoDB [9], is to divide the servers into replica groups and store a copy of the entire database in each group. This replication strategy is a good fit for E2E, which now simply has to choose a replica group for each incoming request. It also allows E2E to affect server-side delays by ensuring that some replica groups are less loaded and used to process QoE-sensitive requests.

2.6.2 Use case #2: Message broker.

We choose RabbitMQ [10] as the message broker (other message brokers can work with E2E in a similar way). RabbitMQ manages its resource by using priority queues and associating each request with a priority level. Requests with high priority are served before requests with low priority. Similar to the Cassandra implementation, we made two changes to integrate E2E. First, we wrote the E2E controller logic in a python script and pass it to RabbitMQ as the default scheduling policy (through `queue_bind`) when the RabbitMQ service is initialized. Second, we modified the per-request callback function (`confirm_delivery`) to track each request's progress and the queueing delay in the message broker.

2.6.3 Implementation details

E2E requires three models as input in order to run. We describe our realizations of these models below, though other approaches are certainly possible.

- *QoE model:* Our E2E prototype uses the QoE models derived from the Microsoft traces and our MTurk user study, shown in Figure 2.3 and detailed in Appendix .2. The QoE model needs to be updated only when the web service changes its content substantially; we do not update it in our prototype.

- *External delay model:* Our E2E prototype builds the external delay distribution from per-request external delay measurements in recent history. The external delays are currently provided by our traces and are not calculated in real-time for each request, though the latter is necessary in a production deployment (see §2.9). We use batched updates to reduce the overhead of keeping the distribution up-to-date. Specifically, we found in our traces that it is sufficient to update the external delay distribution every 10 seconds, because a 10-second time window usually provides enough requests to reliably estimate the distribution, and the distribution remains stable within this window.

- *Server-side delay model:* Our prototype builds the server-side delay model offline, by measuring the service delay distributions induced by different resource allocations. For instance, to build a server-side delay model for the distributed database, we measure the processing delays of one server under different input loads: {5%, 10%,. . . , 100%} of the maximum number of requests per second. For the message broker the profiling is slightly more complicated: we have to consider both the number of requests at each priority level and the total number of requests at higher priority levels. In practice we need not profile all possible allocations: it is sufficient to sample some of them and extrapolate the others. Also, the requests are homogeneous in both of our uses cases, as is typically the case in web services. For services that serve heterogeneous requests (*e.g.,* both CPU-intensive and memory-intensive jobs), or where the effects of different resource allocations do not easily extrapolate to each other, more advanced techniques may be required to ensure the profiling is efficient.

2.7 Evaluation

We evaluate E2E using a combination of trace-driven simulations and real testbed experiments. Our key findings are:

- *E2E can substantially improve QoE:* Users spend 11.9% more web session time (more engagement) compared to the default resource allocation policy in our traces; this improvement accounts for 77% of the best-possible improvement if server-side delays were zero. (§2.7.2)
- *E2E has low system overhead:* E2E incurs only 0.15% additional server-side delay and requires 4.2% more compute resources per request. (§2.7.3)
- *E2E can tolerate moderate estimation errors* (up to 20%) on the external delays, while still retaining over 90% of the QoE improvement attainable if there are no errors. (§2.7.4)

2.7.1 Methodology

Both our trace-driven simulator and our testbeds use the external delay model derived from our traces (Table 2.1) and the QoE model from Figure 2.3. The simulator is described in more detail in §2.2.3.

Testbed setup: To complement our trace-driven simulations, which unrealistically assume the server-side delay distribution is fixed, we create two real testbeds on Emulab—one for Cassandra and one for RabbitMQ, as described in §2.6. We feed requests from our traces to each testbed in chronological order with their recorded external delays, and use the actual testbed processing time as the server-side delays. To show the impact of system load, we speed up the replay by reducing the interval between two consecutive requests by a *speedup ratio* (*e.g.,* a speed-up ratio of 2 means we halve the interval between every two consecutive requests). In the Cassandra (distributed database) testbed, each request is a range query for 100 rows in a table of 5 million keys, which are replicated to three replicas (three Emulab nodes), so each replica has a copy of each key. The key size is 70B and the value size is 1KB. In the RabbitMQ (messaging broker) testbed, each request is a 1KB message sent to RabbitMQ (one Emulab node), and a consumer pulls a message from RabbitMQ every 5ms. Each Emulab node has one 3.0GHz Intel Xeon processor, 2GB RAM, and 2x146GB HDD storage, and are connected to each other by a 1Gbps Ethernet link.

We do not claim that this testbed is a faithful replication of the production system that generated our traces. Rather, we use the testbeds to allow resource allocation policies to affect the server-side delay distributions, as opposed to being constrained by the fixed server-side delays in our traces. We use the traces only to reflect the real external delays of users issuing requests to a service.

Baselines: We compare E2E against two baseline policies:

- *Default policy* (unaware of the heterogeneity of QoE sensitivity): In the simulator, it simply gives each request its recorded server-side delay. In RabbitMQ, it uses First-In-First-Out (FIFO) queueing. In Cassandra, it balances load perfectly across replicas.

- *Slope-based policy* (aware of the heterogeneity of QoE sensitivity but suffers from the problem described in §2.3.2): In the simulator, it gives the shortest server-side delay to the request whose external delay has the steepest slope in the QoE model, and so forth (see §2.2.3). In RabbitMQ, it gives the highest priority to the request whose external delay has the steepest slope in the QoE model, and so forth. In Cassandra, it is the same as E2E's policy, except it replaces the request-decision mapping algorithm with the slope-based algorithm above.

Metric of QoE gain: We measure the *QoE gain* of E2E (and its variants) by the relative improvement of their average QoE over that of the default policy, *i.e.,* $(Q_{\text{E2E}} - Q_{\text{default}})/(Q_{\text{default}})$.

2.7.2 End-to-end evaluation

Overall QoE gains: Figure 2.14 compares the QoE gains of E2E and the slope-based policy over the existing default policy, in our traces and our testbeds. For page types 1 and 2 we use time-on-site as the QoE metric (with Figure 2.3(a) as the QoE model), and for page type 3 we use user rating as the QoE metric (with Figure 2.3(b) as the QoE model). Using user rating vs. time-on-site has negligible impact on our conclusions, as they lead to very similar QoE models (Figure 2.3).

Figure 2.14(a) shows that in our traces, E2E achieves 12.6–15.4% better average QoE than the default policy, whereas the slope-based policy has only 4–8% improvement. This suggests that E2E addresses the limitation of the slope-based policy discussed in §2.3.2. To put these gains into perspective, we consider an idealized policy (labeled "idealized" in the figure) that cuts all server-side delays to zero (*i.e.,* the best a web service could possibly do by cutting server-side delays). We see that the QoE gain of E2E already accounts for 74.1–83.9% of the QoE gain of this idealized policy.

Figure 2.14(b) also compares the QoE of E2E and the baseline policies when feeding requests of page type 1 to the Cassandra and RabbitMQ testbeds. We used a 20× speedup ratio to sufficiently load the systems (we explore the tradeoff between system load and QoE gain below). The results show similar gains in QoE, with both systems achieving a large fraction of the best possible gains.

Better QoE-throughput tradeoffs: Figure 2.15 compares the QoE of E2E and the default policy under different loads, in our traces and our testbeds. E2E strikes a better QoE-throughput tradeoff than both the default policy and the slope-based policy.

Figure 2.15(a) shows the results for different hours of the day in our traces (12am, 4am,

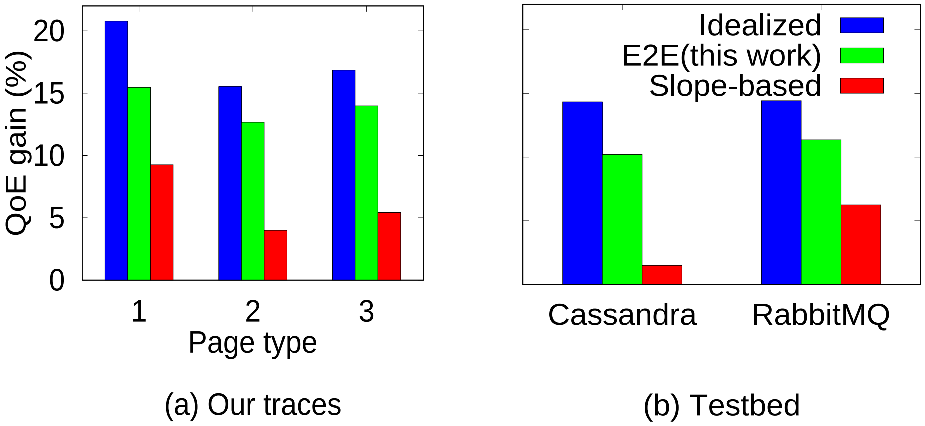


Figure 2.14: Overall QoE improvement of E2E and the slope-based policy over the default policy.

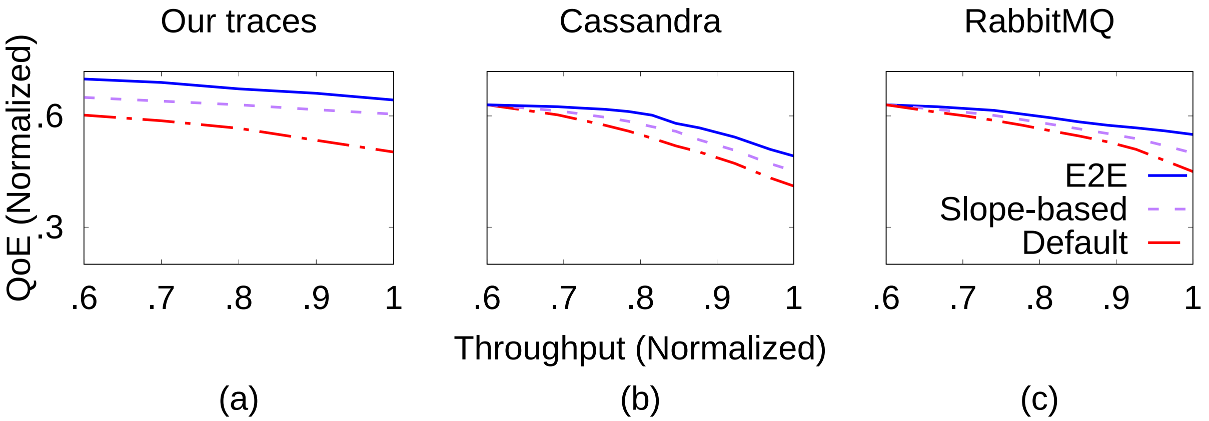


Figure 2.15: QoE improvement of E2E under different levels of loads. Throughput is normalized against the highest per-hour throughput (a) and the total testbed capacity (b, c).

3pm, 8pm, 10pm all in US Eastern Time), which exhibit a natural variation in load. Compared to the off-peak hour (leftmost, at 0.6), the peak hour (rightmost, at 1.0) sees 40% more traffic and, as a result, has 20.1% lower QoE. E2E achieves similar QoE during the peak hour as the default policy does during the off-peak hour. In other words, E2E achieves 40% higher throughput than the default policy without a drop in QoE.

Figures 2.15(b) and (c) compare the QoE of E2E with those of the baseline policies in our testbeds, while varying the load (speedup ratio 15× to 25×, normalized as 0.6 to 1 throughput). E2E always improves QoE, though to varying degrees. E2E's gain is marginal

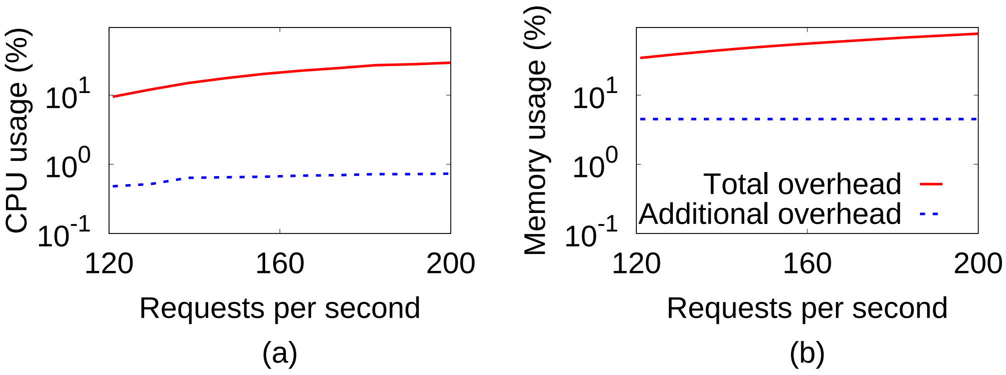


Figure 2.16: The additional overhead of E2E vs. the total overhead of running the testbeds.

under low load, since all decisions have similar, good performance (*e.g.,* all replicas have low read latency when Cassandra is under-loaded). As the load increases, however, E2E's gain grows rapidly: at system capacity, E2E achieves 25% QoE gain over the default policy. This can be explained as follows (using Cassandra as an example). The default policy (perfect load balancing) drives every replica to a moderately high load, so all requests are affected by bad tail latencies. In contrast, E2E allocates load unevenly so that at least one replica is fast enough to serve the QoE-sensitive requests.

2.7.3 Microbenchmarks

We examine the overheads incurred by E2E in computing cost, decision delay, and fault tolerance.

System overhead: We compare the total resource consumption of running each testbed with and without E2E. Figure 2.16 shows the additional overhead of E2E in CPU and RAM usage. We see that the overhead of E2E is several orders of magnitude lower than the total overhead of running the Cassandra or RabbitMQ testbeds themselves. Moreover, the CPU and RAM overheads grow more slowly than those of the testbed service as the load increases.

Decision delay: Figure 2.17 shows the effectiveness of our two decision delay-reduction optimizations (§2.5), using the Cassandra testbed (with speedup ratio 20x). We see that (1) spatial coarsening (bucketization of external delays) reduces the decision delay by four orders

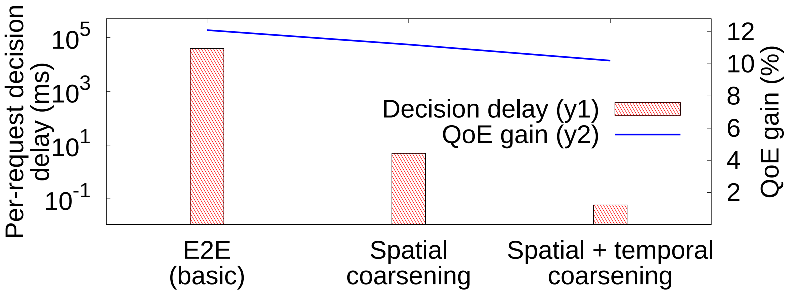


Figure 2.17: Per-request delay reduction due to spatial and temporal coarsening (§2.5).

of magnitude, and (2) temporal coarsening (caching E2E decisions in a lookup table) reduces the decision delay by another two orders of magnitude. The resulting per-request response delay is well below 100μs, less than 0.15% of Cassandra's response delay. At the same time, we see that these reductions in decision-making delay only have a marginal impact on QoE. Note that E2E does not need to make a decision on the arrival of each request, due to these optimizations. Instead, decisions are made periodically and cached in the local memory of each Cassandra client; so when a request arrives, its decision can be read directly from the client's memory.

Fault tolerance: Finally, we stress test our prototype of E2E by disconnecting the E2E controller from the Cassandra testbed. Figure 2.18 shows a time-series of the QoE gain of requests. We disconnect the controller at the 25th second. First, we see that Cassandra's replica selection still uses the latest E2E's decisions cached in the lookup table, so although the QoE gain drops (as the lookup table becomes stale), it is still better than the default policy. At the 50th second, a backup controller is automatically elected, and by the 75th second, the new controller starts to make the same decisions as if the controller was never disconnected.

2.7.4 In-depth analysis

Operational regime: Figure 2.19 tests E2E's performance across a wide range of workloads, along three dimensions that influence E2E's performance. We synthetically generate

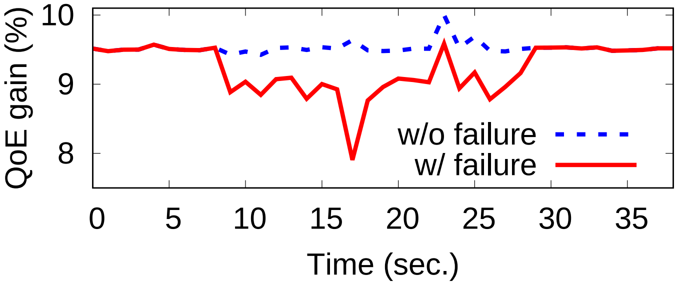


Figure 2.18: E2E can tolerate loss of the controller.

requests by drawing external delays and server-side delays from two normal distributions, respectively, and test them on the trace-driven simulator using the QoE model from Figure 2.3. Although the server-side and external delays in our traces do not exactly follow normal distributions, modeling them in this way allows us to test E2E's performance under different distribution conditions. For instance, we can test the impact of increasing the mean of server-side delay on E2E's performance while keeping the external delay distribution fixed.

We set the default mean and variance of each distribution to match those of the page type 1 requests in our traces, and vary one dimension at a time. We see that at the beginning, E2E does not yield any QoE gain, since there is no variability in the external and server-side delays for it to exploit. Then, the QoE gain of E2E starts to grow almost linearly with the server-side/external delay ratio, external delay variance, and server-side delay variance, which confirms that E2E is able to utilize the variance in external and server-side delays. To put this in the perspective of our traces, the workload in our traces is on the "fast-growing" part of all curves (red spots in Figure 2.19). This means we will see more QoE gain if the workload moves to the right in any of these dimensions.

Robustness to prediction errors: Figure 2.20 shows the impact that prediction errors, in the external delays and the number of requests per second (RPS), have on E2E's performance. We feed page type 1 requests to the Cassandra testbed (speedup ratio 20x), and inject a controllable error on the actual value to obtain the estimated value. Figure 2.20(a) shows that even if the external delay prediction is off by 20% on each request, E2E still retains over

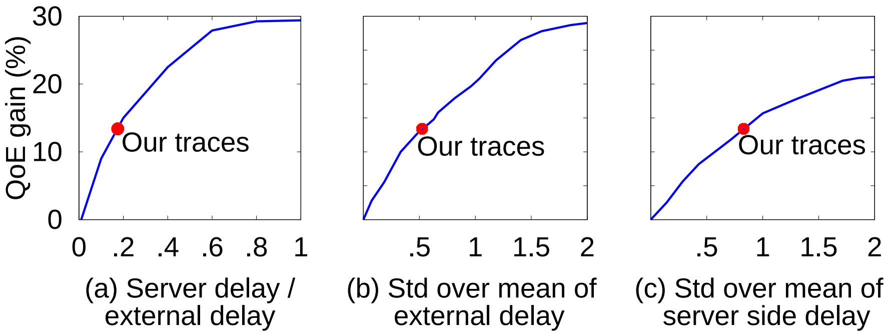


Figure 2.19: The impact of three key workload dimensions on E2E's effectiveness. The red spot shows where the workload in our traces lies.

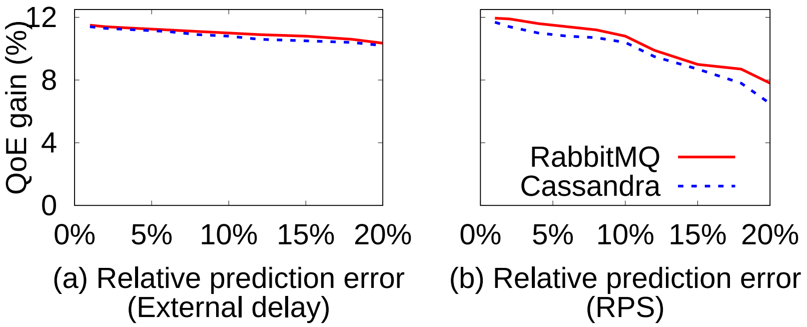


Figure 2.20: Sensitivity of QoE improvement to prediction errors in external delay and requests per second.

90% of its QoE gain. Predicting the external delay with 20% (or 100-200ms) error seems reasonable for most users [131]. Figure 2.20(b) shows that E2E retains 91% of its QoE gain if the RPS is predicted with 10% error. Empirically, we find that 10% prediction error is possible when using the RPS history from the last 10 seconds (not shown).

QoE fairness: A natural concern is that E2E may create a less fair QoE distribution. As an example, we use the QoE distributions of E2E and the default policy from Figure 2.14(a) and page type 1. We calculate Jain's Fairness Index of the requests' QoE values, and find that E2E's Jain index (0.68) is lower but still very close to that of the default policy (0.70). This is because E2E only deprioritizes requests that are insensitive to QoE; these requests experience only a marginal improvement in QoE when using the default policy.

E2E vs. deadline-driven scheduling: Unlike E2E, some prior work (*e.g.,* [131, 51]) models the impact of total delay on QoE as a hard deadline: QoE drops to zero immediately

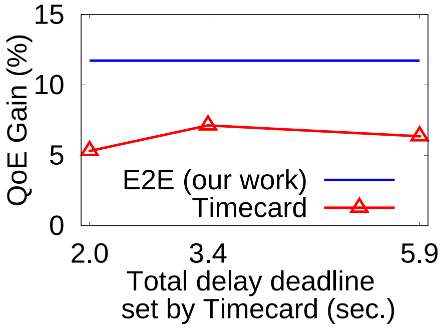


Figure 2.21: E2E vs. Timecard (with different total delay deadlines).

after the total delay exceeds the deadline. We use Timecard [131] as a canonical example of a deadline-driven scheduling policy, and compare E2E to it. Timecard sets a total delay deadline and, given the external delay of each request, tries to maximizes the number of requests served by the deadline. We compare E2E with Timecard under total delay deadlines of 2.0, 3.4, and 5.9 seconds, using RabbitMQ as the testbed. As Figure 2.21 shows, the QoE gain of E2E is consistently better than Timecard under different deadline settings. This is because the deadline-driven scheduler is agnostic to the different QoE sensitivities of requests that have already exceeded the deadline.

2.8 Related work

We briefly survey the most related work on web QoE, cloud resource allocation, and web performance measurements.

2.8.1 Web QoE modeling/optimization

QoE has been intensively studied in the context of web services (*e.g.,* [27, 55]), mobile apps (*e.g.,* [20]), video streaming (*e.g.,* [59, 28]), and measurement tools (*e.g.,* [156]). Prior work (*e.g.,* [35, 69, 123]) has observed a similar non-linear relationship between page loading time and QoE. Although E2E uses a specific QoE model (based on our trace analysis), it can benefit from more precise models of how page loading time affects QoE. Unlike prior QoE optimization techniques that tune client-side knobs [38, 122] or provide server-side resources for individual sessions (*e.g.,* [138, 137]), E2E intelligently allocates server-side resources shared across a large number of heterogeneous users.

2.8.2 Web service resource allocation

There is a large literature on cutting the tail/median server-side delays through better web resource management, including distributed databases (*e.g.,* [149, 163]), partition-aggregation workloads (*e.g.,* [92, 101]), and caching (*e.g.,* [33, 34]). Cloud providers optimize WAN latency through better server selection (*e.g.,* [109, 45]) and WAN path selection [166, 140]. E2E is conceptually compatible with many existing resource sharing techniques (*e.g.,* replica selection and message scheduling). What distinguishes E2E is that it does not seek to minimize the median or tail performance; instead, it takes into account the QoE sensitivity of different users when allocating server-side resources.

2.8.3 End-to-end performance analysis

There have been attempts to measure the contribution of cloud, WAN, and client-side devices to end-to-end delays [130, 50, 48]. Our observations on heterogeneous QoE sensitivity corroborate some prior work (*e.g.,* [50]) that show that cloud-side delays are not a constant fraction of end-to-end delays for all users. These studies offer useful insights for improving web service infrastructure [110, 99, 40] and building real-time resource management systems [51, 131, 19].

The works most closely related to E2E are Timecard [131] and DQBarge [51], which share with us the high-level idea of making server-side decisions based on the QoE of end users [93]. In particular, they estimate the "slack" time between receiving a request and its end-to-end delay deadline, and utilize this slack to maximize the quality of the response. Although they allocate different resources to different requests, they optimize individual requests in isolation, which can cause resource contention when the system is under stress or many requests have low slack time. In contrast, E2E optimizes QoE and resource allocation across requests, by harnessing their inherent heterogeneity. We also empirically show that when the QoE curve is like Figure 2.3, a deadline-based QoE model can be less effective than E2E (§2.7.4).

E2E is similar to work (*e.g.,* [103, 104]) that considers requests with soft deadlines: *i.e.,* QoE decreases gradually to zero after the total delay exceeds a time threshold. These soft-deadline-driven schedulers set the same threshold for all requests and do not take the heterogeneity of web requests into account, whereas the resource allocation in E2E considers different QoE sensitivities.

2.9 Discussion

2.9.1 Incentives of other service providers

One concern about using E2E is that another service provider (*e.g.,* an ISP) may try to manipulate the external delays of its users to get better service from E2E, by making them look more urgent. However, we prove in Appendix .1 that *it is impossible to improve a group of users' QoE without reducing at least some of their external delays.* In other words, E2E creates an incentive for other service providers to reduce their delays, rather than gaming E2E by deliberately adding delays.

2.9.2 Security threat

In theory, E2E may introduce a new attack, in which a large group of users hurt the QoE of other users by making themselves look more urgent, thus starving the other users of resources (similar to a Denial-of-Service attack). We can envision several detection/mitigation techniques for such an attack, such as detecting abnormal changes to the external delay distribution, or adding randomization to the actual server-side delays. We leave investigation of these security issues to future work.

2.9.3 Interaction with existing policies

A web service provider often integrate multiple resource allocation policies. Conceptually, E2E is compatible with other prioritization schemes; they can be included as input into E2E's decision-making policy (*e.g.,* by upweighting the $Q(\cdot)$ values of premium traffic), or E2E can be applied separately to each priority class (*e.g.,* premium users vs. basic users).

2.9.4 Complex request structures

In a real web framework like Microsoft's, a high-level web request usually results in calls to multiple backend services, and the request is not complete until it hears a response from all the backend services [33]. A straightforward way to handle this request structure is to apply E2E to each service in isolation. However, this approach is suboptimal, because it may cause a service to prioritize requests whose server-side delays are determined by other backend services. For example, in Figure 2.11(a), E2E prioritizes request B over A, since prioritizing A would cause B to suffer a significant QoE drop. But if B also depends on another, much slower service, speeding up B will not have a direct impact on the user's QoE. In this case, it would have been better to prioritize A, whose QoE could actually have been improved. We can see that an optimal resource allocation scheme for requests with complex structure needs to take these backend service dependencies into account. We leave this problem to future work.

2.9.5 Deployment at scale

E2E must face the following issues when deployed in a large-scale production system.

- *Multiple agents:* For a web service to scale, it typically uses distributed agents (*e.g.,* Cassandra clients or RabbitMQ message brokers), each making resource-allocation decisions independently. In E2E, although each agent might see a different subset of web requests, its decisions are based on a global decision lookup table built upon the global external delay distribution. In the unlikely event that the requests are load balanced poorly across the agents, it is possible for the resulting decisions to be suboptimal: *e.g.,* in the case of RabbitMQ, if one message broker only sees insensitive requests, those requests will be at the head of its queue (there are no sensitive requests to place ahead of them). We have not investigated such scenarios in our current evaluation.

- *Real-time external delay estimation:* Our current prototype relies on the external delays provided by our traces, but a real deployment would need to compute the external delay in real-time for each request. E2E could accomplish this by borrowing ideas from Timecard [131] and Mystery Machine [50]. Like Timecard, the WAN-induced delay of a request could be derived from the round-trip time of the TCP handshake packets and the TCP sliding window size. To estimate the browser rendering time of a request, E2E could use a model trained on historical traces (Mystery Machine) or on traces and the system configuration (Timecard). Timecard provides more accurate estimates but requires user permission to access the system configuration. Mystery Machine does not need user cooperation but has lower accuracy, especially for first-time users. Since E2E is not very sensitive to the accuracy of the external delay estimates (Figure 2.20(a)), Mystery Machine's method could allow E2E to scale out and support more requests.

CHAPTER 3

SENSEI: ALIGNING VIDEO STREAMING QUALITY WITH DYNAMIC USER SENSITIVITY

3.1 Introduction

An inflection point in Internet video traffic is afoot, driven by more ultra-high resolution videos, more large-screen devices, and ever-lower user patience for low quality [3, 12]. At the same time, the video streaming industry, after several decades of evolution, recent adaptive bitrate (ABR) algorithms (*e.g.,* [117, 87, 165]) achieve near-optimal balance between bitrate and rebuffering events, and recent video codecs (*e.g.,* [147, 114]) improve encoding efficiency but require an order of magnitude more computing power than their predecessors. The confluence of these trends means that the Internet may soon be overwhelmed by online video traffic,[1] and new ways are needed to attain fundamentally better tradeoffs between *bandwidth usage* and user-perceived *QoE* (quality of experience).

We argue that a key limiting factor is the conventional wisdom that users care about quality in the same way throughout a video, so video quality should be optimized using the same standard *everywhere* in a video. This means that lower quality—due to rebuffering, low visual quality, or quality switches—should be avoided identically from the beginning to the end. We argue that this assumption is not accurate. In sports videos (*e.g.,* the one in Figure 3.1), a rebuffering event that coincides with scoring tends to inflict a more negative impact on user experience than rebuffering during normal gameplay. But there are also sports videos where scoring is not the most quality-sensitive part. Thus, user quality sensitivity *varies with the video content dynamically over time.*

Unfortunately, both the literature on ABR algorithms and the literature on QoE modeling adopt the conventional wisdom. Most ABR algorithms completely ignore the content of

[1] This is vividly illustrated by the recent actions taken by YouTube and Netflix (and many others) to lower video quality in order to save ISPs from collapsing as more people stay at home and binge watch online videos [14].

each video chunk: they focus on balancing high bitrates and low rebuffering times, and thus consider only the size and download speed of the chunks. Traditional ways of modeling QoE are also agnostic to the substance of videos, although recent QoE models—*e.g.,* PSNR [74], SSIM [160], VMAF [13], and deep-learning models [65, 98]—try to find frames that users are more sensitive to by studying the structure of pixels and motions to gauge their saliency. These heuristics seek to generalize across *all videos* and thus resort to generic measures (like pixel-level differences), but it is unclear if any heuristic can capture the diverse and dynamic influence a video's content can have on users' sensitivity to quality.

For example, models like LSTM-QoE [65] assume that users are more sensitive to rebuffering events in more "dynamic" scenes. In sports videos, however, non-essential content like ads and quick scans of the players can be highly dynamic, but users may care less about quality during those moments. In the video in Figure 3.1, LSTM-QoE considers normal gameplay to be the most dynamic part, but the most quality-sensitive part of the video according to our user study is the goal. A key insight is that the impact of the substance of a video on users' sensitivity to quality cannot be fully explained by pixel-level patterns or cross-frame motions. Some recent work tries to predict user's dynamic sensitivity, but they either need access to users' viewing history [68] or use off-the-shelf computer-vision saliency models [67] whose predictions have little correlation with quality sensitivity on videos they have never seen before (§3.2.3 elaborates on this).

The dynamic nature of quality sensitivity suggests a new avenue for improvement. One can achieve *higher QoE with the same bandwidth* by carefully lowering the current quality in order to save bandwidth and allow higher quality when users become more sensitive. Similarly, one can attain *similar QoE with less bandwidth* by judiciously lowering the quality when quality sensitivity is indeed low. In short, we seek to *align higher (lower) quality of video chunks with higher (lower) quality sensitivity of users.*

We present Sensei, a video streaming system that incorporates dynamic quality sensitivity into its QoE model and video quality adaptation. Sensei addresses two key challenges.

Challenge 1: How do we profile the unique dynamic quality sensitivity of each video in an accurate and scalable manner?

Crowdsourcing the true quality sensitivity per video: Instead of proposing another heuristic, Sensei takes a different approach. We run a *separate crowdsourcing experiment for each video* to derive the quality sensitivity of users at different parts of the video. Specifically, we elicit quality ratings directly from real users (obtaining a "ground truth" of their QoE) for multiple renderings of the same video, where each rendering includes a quality degradation in some part of the video. Sensei automates and scales this process out using a public crowdsourcing platform (Amazon MTurk), which provides a large pool of raters, while using pruning techniques to reduce the number of rendered videos that need to be rated. We then use these ratings to estimate a *weight* for each video chunk that encodes its quality sensitivity, independent of the quality of other chunks. While crowdsourcing has previously been used to model QoE, Sensei is to our knowledge the first to scale it to *per-video* QoE modeling.

Challenge 2: How do we incorporate dynamic quality sensitivity into a video streaming system to enable new decisions? Today's video players are designed to be "greedy": they pick a bitrate that maximizes the quality of the next chunk while avoiding rebuffering events. But in order to utilize dynamic quality sensitivity, a player must "schedule" bitrate choices over *multiple* future chunks, each having a potentially different quality sensitivity. This means that some well-established behaviors of video players, *e.g.,* only rebuffer when the buffer is empty, may need to be revisited.

Refactoring ABR logic to align with dynamic quality sensitivity: Sensei works within the popular DASH framework. It integrates the aforementioned per-chunk weights into existing ABR algorithms to leverage the dynamic quality sensitivity of upcoming video chunks when making quality adaptation decisions. The per-chunk weights enable new adaptation actions that "borrow bandwidth" from low-sensitivity chunks and give them to high-sensitivity chunks. For example, Sensei may lower the bitrate even when bandwidth is sufficient, or initiate a rebuffering event with a non-empty buffer, to afford higher bitrates when quality sensitivity becomes higher. We apply Sensei to two state-of-the-art ABR algorithms: Fugu [165], a more traditional rule-based algorithm, and Pensieve [117], a deep reinforcement learning-based algorithm.

Using its scalable crowdsourcing approach, Sensei can predict QoE more accurately than state-of-the-art QoE models. For example, with a budget of just \$31.4/minute video, Sensei achieves 55% less QoE prediction error than existing models. Compared to state-of-the-art ABR algorithms, Sensei improves QoE on average by 15.1% or achieves the same QoE with 26.8% less bandwidth across various video genres.

Contributions and roadmap: Our key contributions are:

- A measurement study revealing substantial temporal variability in users' quality sensitivity and its potential for improving video streaming QoE and bandwidth usage (§3.2).
- The design and implementation of Sensei, including: 1) a scalable crowdsourcing solution to profiling the true dynamic quality sensitivity of each video (§3.4,§3.5),[2] and 2) a new ABR algorithm that incorporates dynamic user sensitivity into existing algorithms and frameworks (§3.6).

3.2 Motivation

We begin by showing that existing approaches to modeling video streaming QoE fail to accurately capture the true user-perceived QoE (3.2.1). We then present user studies that reveal a missing piece in today's QoE modeling: users' quality sensitivity varies dynamically throughout a video (§3.2.2), and this dynamic quality sensitivity is hard to capture using prior heuristics or vision models (§3.2.3). However, by incorporating dynamic quality sensitivity into existing ABR algorithms, we can significantly improve QoE and save bandwidth (§3.2.4).

[2] Our study was IRB-approved (IRB18-1851).

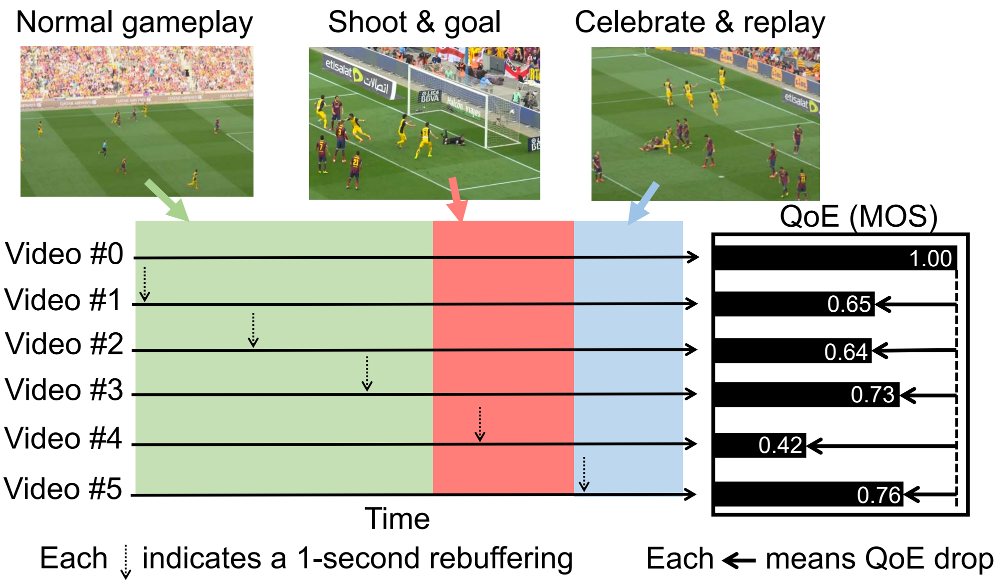


Figure 3.1: Example of dynamic quality sensitivity. Users are asked to rate the quality (on a scale of 1 to 5) of different renderings of a source video (`Soccer1`), where a 1-second rebuffering event occurs at a different place in each rendering. We observe substantial differences in the QoE impact (measured by mean opinion score, or MOS) across the renderings.

3.2.1 Prior QoE modeling and their limitations

QoE models are crucial to modern video streaming systems. A QoE model takes a streamed video as input and returns a predicted QoE as output. When streaming a video, the video player optimizes QoE by adapting the bitrate of each video chunk to the available bandwidth. QoE is often measured by the mean opinion score (MOS) assigned by a group of users to the quality of a video.[3]

Quality metrics: Today's QoE models consider two aspects.

- *Pixel-based visual quality* tries to capture the impact of visual distortion on QoE. These metrics, such as PSNR and VMAF, are based on pixel/motion-based patterns [74, 160, 159, 143, 13, 98, 132] and recently on visual attention [58, 169, 85].
- *Streaming quality incidents* during the streaming process can negatively impact user experience, such as rebuffering, low bitrate, and bitrate switches. Their impact is modeled by metrics, such as rebuffering ratio, average bitrate, and frequency of bitrate switches during

[3] Our methodology extends to other QoE metrics as well.

a video (*e.g.,* [59, 28]).

Some work also considers contextual factors (*e.g.,* viewer's emotion, acoustic conditions, etc.), but these are orthogonal to our focus on the video's content.

QoE models: Recent QoE models combine both pixel-based visual quality metrics and quality-incident metrics for more accurate QoE prediction. We consider three such QoE models: KSQI [60], P.1203 [136], and LSTM-QoE [65], which were proposed within the past two years and have open-source implementations. KSQI combines VMAF, rebuffering ratio, and quality switches in a linear regression model. P.1203 combines QP (quantization parameter) and quality incident metrics in a random-forest model. Most recently, LSTM-QoE takes STRRED [143] and individual quality incidents as input to a long short-term memory (LSTM) neural network designed to capture the "memory effect" of human perception of past quality incidents. (We discuss related work in §3.8.)

User study methodology: We evaluate these QoE models (KSQI, P.1203, LSTM-QoE) on 16 source videos randomly selected from four public datasets [72, 31, 61, 158], covering a wide range of content genres (sports, scenic, movies, etc.). These videos are streamed using one of three ABR algorithms: Fugu [165], Pensieve [117], and BBA [87], over 7 throughput traces randomly selected from real-world cellular networks [135, 135], with bandwidths ranging from 200Kbps to 6Mbps. §2.7.1 and Appendix .3 provide more details on the videos and network traces. This creates 336 (16×7×3) rendered videos. To obtain the ground truth QoE of each rendered video, we elicit QoE ratings from crowdsourced workers on Amazon MTurk [1]. We obtain at least 30 ratings from different MTurkers and use the MOS over these ratings as the true QoE of the rendered video §3.4 and §3.5 describe our crowdsourcing methodology in detail.

QoE prediction accuracy: Given the ground-truth QoE, we evaluate the three QoE models both with their pre-trained parameters and after customizing (retraining) them on 315 of the rendered videos selected at random. All models are tested on the remaining 21 videos; we scale their output range and the true QoE to $[0, 1]$. The x-axis of Figure 3.2 shows

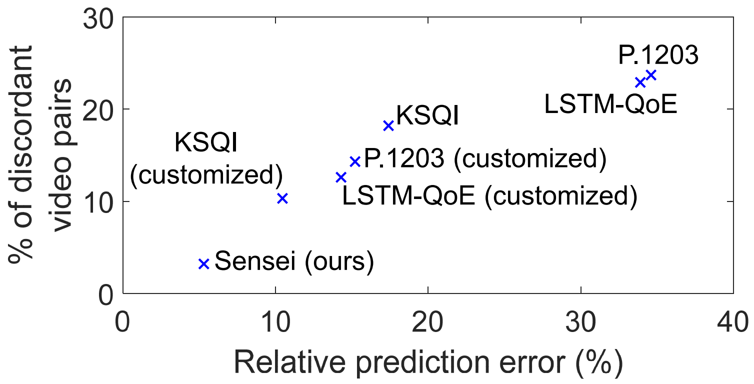


Figure 3.2: Existing QoE models exhibit substantial QoE prediction errors (x-axis), which cause them to frequently mis-predict the relative QoE ranking between two ABR algorithms on the same video, *i.e.,* a discordant pair (y-axis).

the mean relative prediction error of each QoE model on the test set; relative prediction error is defined as $|Q_{predict} - Q_{true}|/Q_{true}$, where $Q_{predict}$ and Q_{true} are the predicted and true QoE of the video. We see that these errors are nontrivial; even the most accurate QoE model has over 10.4% error on average.

We also examine whether these models can correctly *rank* the QoE achieved by two different ABR algorithms. For each pair of source video and throughput trace, we first rank every two of the three ABR algorithms using their true QoE and then again using the predicted QoE. If the rank is different, this pair is called a discordant pair. The y-axis of Figure 3.2 shows the fraction of discordant pairs among all possible pairs (a common measure used in rank correlation): over 10.2% of pairs are discordant even for the most accurate QoE model. This suggests that using QoE predictions to compare different algorithms (*e.g.,* [117, 165, 87]) may not be reliable.

3.2.2 Temporal variability of quality sensitivity

Figure 3.2 shows that, unlike prior methods, our QoE model (§3.4) can predict QoE and rank ABRs significantly more accurately when applied on the same train/test set. We argue that this gap stems from a common assumption shared by all previous QoE models, which is that all factors affecting QoE can be captured by a handful of objective metrics. This premise ignores the impact of high-level video content (rather than low-level pixels and frames) on users' sensitivity to quality at different parts of the video. We now demonstrate how this quality sensitivity varies as video content changes.

Quantifying dynamic quality sensitivity: Users' sensitivity to quality at a certain part of a video is reflected by the *QoE drop* when a low-quality incident occurs at that part of the video, *i.e.,* $\Delta = Q_{before} - Q_{after}$, where Q_{before} is the MOS of the video without the low-quality incident and Q_{after} is the MOS of the video with the low-quality incident. To measure the true quality sensitivity at different parts of a source video, we create a *rendered video series* as follows. Rendered videos in a video series have the same source content and highest quality (highest bitrate without rebuffering), except that a low-quality incident (a rebuffering event or a bitrate drop) is deliberately added at different positions, *e.g.,* at the 4^{th} second, 8^{th} second, and so forth. Then, as before, we use Amazon MTurk to crowdsource the true QoE of each rendered video, following our crowdsourcing methodology (§3.4,§3.5).

Figure 3.1 shows an example video series created using a 25-second soccer video as the source video and a one-second rebuffering event as the low-quality incident. We observe significant differences between the QoE drops caused by the rebuffering event at different parts of the video. The highest QoE drop (caused by rebuffering at the 15^{th} second) is 2.1× higher than the lowest QoE drop (rebuffering at the 10^{th} second). This shows that a low-quality incident can have a significantly higher/lower impact on user experience if it occurs a few seconds earlier or later.

Quality sensitivity is inherent to video content: Our user study also suggests that the type of low-quality incident does not affect the ranking of QoE drops within a video series, even though it affects the absolute QoE drops. In other words, quality sensitivity seems to be *inherent* to different parts (contents) of the video. Figure 3.6 shows the dynamic user sensitivity of three low-quality incidents on the same source video: 1-second rebuffering, 4-second rebuffering, and a bitrate drop from 3Mbps to 0.3Mbps for 4 seconds. Although the absolute values of the QoE drops depend on the particular quality incident, the relative

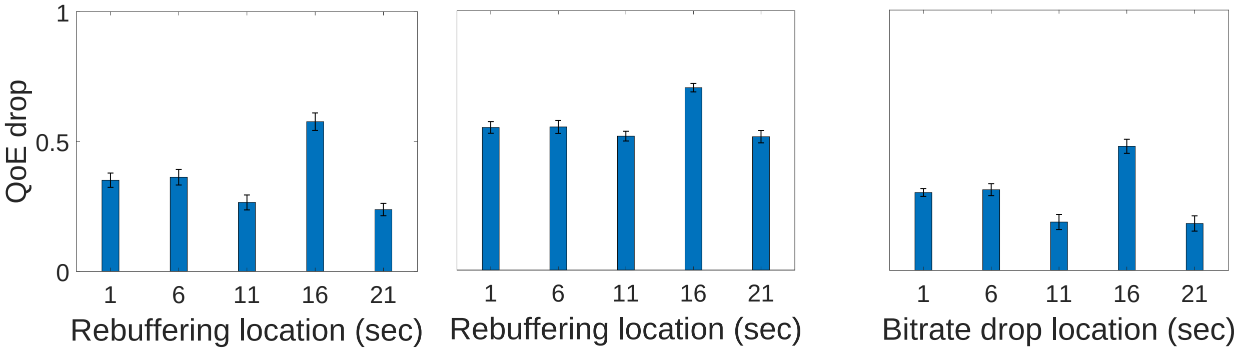


Figure 3.3: 1-sec rebuffering

Figure 3.4: 4-sec rebuffering

Figure 3.5: Bitrate drop

Figure 3.6: Impact of different quality incidents at different points in the video in Figure 3.1. The pattern of variability remains the same across the different quality incidents. Error bars show standard deviation of the means.

rankings are identical. The strong rank correlation (measured by Spearman's rank coefficient) is persistent across all videos in our dataset: 0.95 rank coefficient between the 1-second and 4-second rebuffering events and 0.94 between the 1-second rebuffering and bitrate drop.

Sources of dynamic quality sensitivity: We speculate that the dynamic quality sensitivity stems from users paying different degrees of attention to different parts of a video. In our dataset, we identify at least three types of moments when users tend to be more (or less) attentive to video quality than usual. The first are key moments in the storyline of a video when tensions have built up; *e.g.,* in `BigBuckBunny` (animation) when the bullies fall into a trap set by the bunny, or in `Soccer1` when a goal is scored. The second are moments when users must pay attention to get important information; *e.g.,* showing the scoreboard in sports videos (`Soccer2`), or acquiring supplies after killing an enemy (`FPS2`). The third are transitional moments with scenic backgrounds, when users tend to be less attentive to quality; *e.g.,* the universe background in `Space`.

3.2.3 Modeling quality sensitivity

Can it be captured by QoE models? Traditional QoE models predict the same QoE for all rendered videos in a video series. Even models that do predict different QoE assume

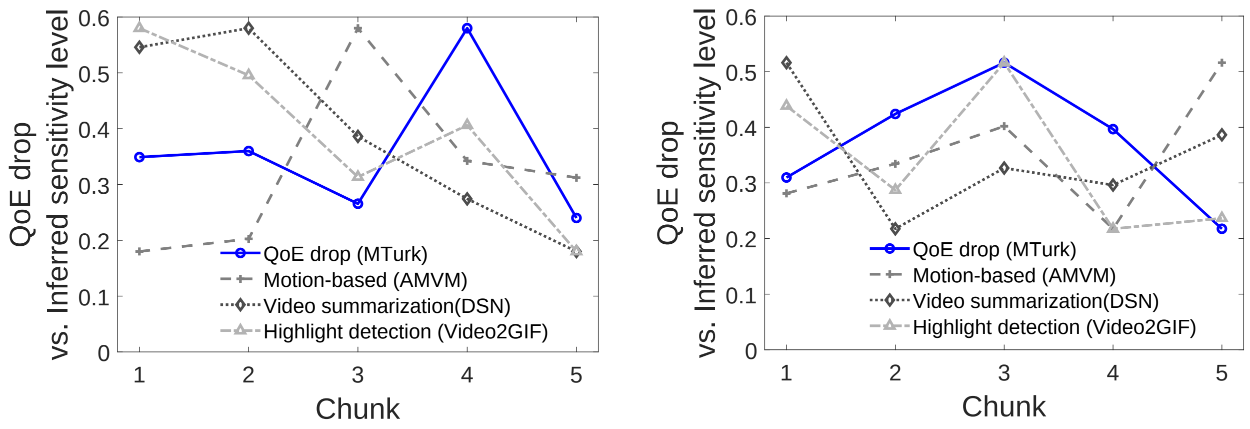


Figure 3.7: `Soccer1`

Figure 3.8: `FPS`

Figure 3.9: QoE rating drop when adding a 1-sec rebuffering at different points in the video, compared to chunk sensitivity levels inferred by saliency models.

that the impact of video content can be captured by pixels and motions; *e.g.,* VMAF [13] (the visual quality metric used by KSQI) gives lower QoE estimates if a bitrate drop occurs when the frame pixels are more “complex”. Unfortunately, the impact of content on user sensitivity discussed above cannot be fully captured by pixel-level patterns. In Figure 3.1, the true highest QoE drop occurs when the low-quality incident occurs during the goal, but both VMAF and LSTM-QoE predict that it occurs during normal gameplay.

Can it be captured by vision saliency? User sensitivity is conceptually similar to temporal saliency in computer vision. Can saliency/highlight detection models capture user sensitivity to quality? We examine three representative approaches.

- *Traditional motion-based models*, such as AMVM (average motion-vector magnitude) [112, 18], use the motion vector magnitudes of pixels in a chunk to indicate user sensitivity—*i.e.,* users are more sensitive to more dynamic scenes.
- *Interestingness score per frame (highlight detection)*, such as Video2GIF [75] and [67], train a regression model (using C3D [152] neural network as the spatio-temporal feature extractor) on videos with human-annotated per-frame interestingness scores.[4] The model

[4] We notice that some content providers passively monitor the number of viewers at different parts of a video (*e.g.,* [6]), which is an alternative way of identifying highlights or high-interestingness chunks.

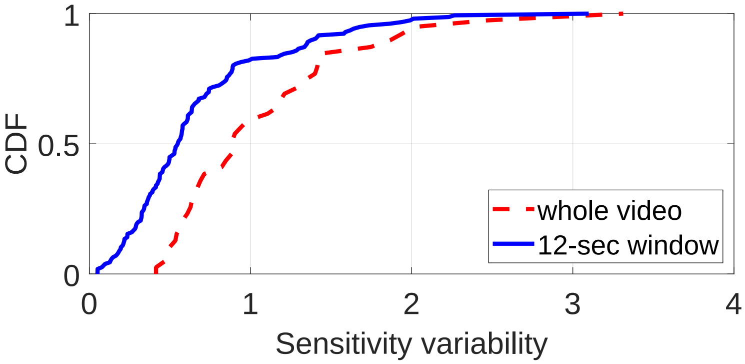


Figure 3.10: Distribution of sensitivity variability when a low-quality incident (1-second rebuffering, 4-second rebuffering, or a bitrate drop for 4 seconds) is added at different points in the same video. The trend is similar even if the low-quality incident and QoE gap are localized to a 12-second window.

then produces a per-frame interestingness score which might indicate user sensitivity.

- *Video summarization models*, such as dppLSTM [171] and DSN [178], infer how important each frame is to the whole story of a video, by extracting vision features [150] and using an LSTM to model temporal dependencies. The more important a frame is, the higher user sensitivity might be.

Figure 3.9 shows the average saliency scores (normalized to $[0, 1]$) returned by these models at each chunk of two example videos. We see a weak correlation between the QoE drops caused by a 1-sec rebuffering event at different chunks and the true user sensitivity. Overall, such correlation is low for all videos in our dataset: less than 0.23 (Pearson's correlation) and 0.18 (Spearman's rank correlation). To see an example, in the soccer video (Figure 3.1), the part right before the goal is the most quality sensitive. However, the highlight detection and motion-based models highlight the highly dynamic scenes that pan across the audience, and the video summarization model picks diverse moments of a video, such as shot/rewind clips, whereas users pay more attention to when a goal might be scored. Appendix .6 gives more discussions. As a result, ABR logic based on saliency scores performs poorly (§2.7).

3.2.4 Potential gains

Dynamic quality sensitivity is prevalent: We repeat the same experiment from Figure 3.1 on all 16 source videos in our dataset and three low-quality incidents: 1-second rebuffering, 4-second rebuffering, and a bitrate drop from 3Mbps to 0.3Mbps for 4 seconds. This creates 48 video series in total. Figure 3.10 plots the *sensitivity variability* defined by $(\Delta_{max}-\Delta_{min})/\Delta_{min}$ for each video series, where Δ_{max} and Δ_{min} are the maximum and minimum QoE drop of the videos in a series. We see that 21 of the 48 video series have a sensitivity variability of over 0.99, while some have less than 0.20 variability. A similar trend holds even if we localize the low-quality incident and sensitivity gap measurement to 12-second windows. The fact that quality sensitivity varies substantially even among very nearby chunks suggests a new opportunity: we can lower the quality when sensitivity is low in order to save bandwidth for nearby chunks whose sensitivity is high.

Potential sensitivity-aware improvement: The above suggests that we can improve ABR algorithms to optimize QoE and save bandwidth by *aligning quality adaptation with dynamic user sensitivity*. We demonstrate the potential gains using an idealistic but clean experiment. We create two simple ABR algorithms whose only difference is the QoE model they optimize: one algorithm optimizes KSQI, the most accurate QoE model from Figure 3.2 that is *unaware* of dynamic quality sensitivity, and the other optimizes our eventual QoE model from §3.4, which *is* aware of dynamic quality sensitivity. Both algorithms take as input an entire throughput trace and the same 4-second video chunks encoded using the same bitrate levels. They then determine a bitrate-to-chunk assignment that maximizes their respective QoE model. Note that these ABR algorithms are idealistic because they have access to the entire throughput trace in advance, and hence know the future throughput variability. However, this allows us to eliminate the confounding factor of throughput prediction. We pick one of the throughput traces (results are similar with the other traces) and rescale it to $\{20, 40, \ldots, 100\}\%$ to emulate different average network throughputs.

For each source video, we create the rendered video as if it were streamed by each ABR

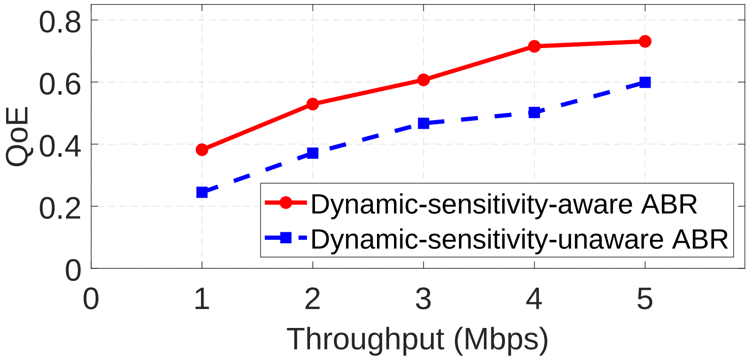


Figure 3.11: Being aware of dynamic quality sensitivity can significantly improve QoE and save bandwidth.

algorithm (with bitrate switches, rebufferings, etc.). We use Amazon MTurk as before to assess the true QoE of the rendered video. Figure 3.11 shows the average QoE of the two ABR algorithms across 16 source videos and different average bandwidths. We see that being aware of dynamic quality sensitivity could improve QoE by 22-52% while using the same bandwidth, or save 39-49% bandwidth while achieving the same QoE.

3.3 Overview of Sensei

To unleash this potential, we present Sensei, a video streaming system that unearths and leverages dynamic quality sensitivity. Here, we overview Sensei (§3.3.1) and then introduce our crowdsourcing-based approach to per-video QoE modeling and its limitations (§3.3.2).

3.3.1 Sensei's approach

As shown in Figure 3.12, Sensei has two main components.

Per-video QoE modeling: Before streaming a video, Sensei profiles the quality sensitivity of its chunks. As we saw in §3.2.2, prior QoE models fail to capture content-induced user sensitivity to quality. Instead, we advocate for directly asking human viewers to rate the quality of rendered videos with quality incidents inserted at various chunks. This reveals the true user sensitivity to quality incidents. Since quality sensitivity is unique to each video,

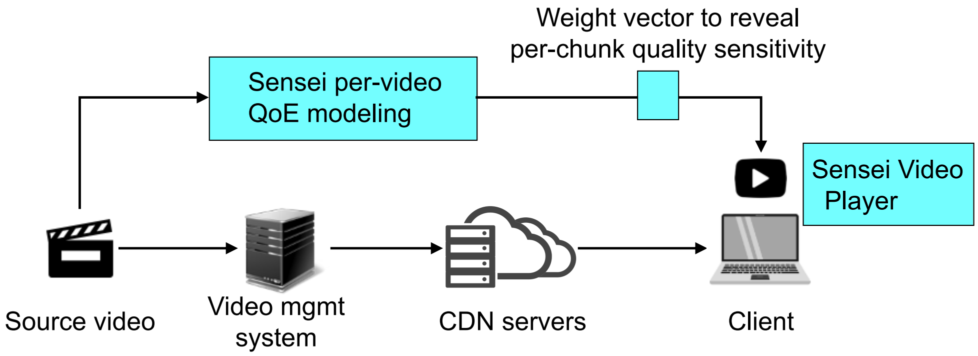

Weight vector to reveal per-chunk quality sensitivity
Sensei per-video QoE modeling
Sensei Video Player
Source video
Video mgmt system
CDN servers
Client

Figure 3.12: Overview of Sensei.

this user study must be scaled to many videos. Sensei uses crowdsourcing to automate and scale the per-video QoE modeling, by addressing two challenges: (1) how many (and which) rendered videos must be rated to build a sensitivity-aware QoE model (§3.4); and (2) how to get reliable ratings from crowdsourced workers (§3.5).

Sensitivity-aware ABR: Video players today are designed to maximize bitrate without rebuffering on *every* chunk. This is ill-suited to our goal of aligning quality adaptation with dynamic quality sensitivity: quality should be optimized in proportion to the quality sensitivity of the content. To achieve this, Sensei refactors the control logic of video players to enable new adaptation actions that "borrow" resources from low-sensitivity chunks and give them to high-sensitivity chunks. We discuss the details in §3.6.

Instead of building a separate QoE model for each video, Sensei reuses existing QoE models but reweights each chunk by its quality sensitivity. This is inspired by our observation that relative quality sensitivity is inherent to the content, rather than the specific quality incident (§3.2.2). Thus we assign a weight to each chunk to encode its inherent quality sensitivity. The abstraction of *per-chunk weights* has two benefits. First, it allows us to reuse existing QoE models by simply reweighting the quality of different chunks. Second, by using the sensitivity weights as input, the same Sensei ABR algorithm can be used to optimize QoE for any new video.

3.3.2 Crowdsourcing quality sensitivity per video

Sensei directly elicits quality ratings from human viewers to reveal their quality sensitivity to various quality incidents. However, these user ratings must be elicited *per video* and the sheer scale of this feedback can be prohibitive! To put it into perspective, QoE models are usually built from user ratings on just a handful of source videos [31, 62], but getting enough user ratings requires a lab environment (or survey platform) to recruit participants and have them watch over two orders of magnitude more video content than the source videos. [5] This does not scale if we repeat the process per video.

To address this, we use crowdsourcing platforms like Amazon MTurk [1] to automate the user studies and scale them out to more videos. Crowdsourcing reduces the overhead of participant recruitment, survey dissemination, and result collection (down to about 78 minutes), and provides a large pool of participants. This allows for repeated experiments to help control for human-related statistical noise. Although the crowdsourcing cost grows with video length, Sensei offers several techniques to reduce the cost (see §3.4). Thus, the content providers can decide whether and how to initiate profiling given their budgets. Note that our reliance on crowdsourcing makes some scenarios, *e.g.,* live video streaming, currently inapplicable (see §3.9).

3.4 Profiling Quality Sensitivity at Scale

In this section, we show how to build an accurate and cost-efficient QoE model using crowdsourcing. We overview our workflow (§3.4.1) and then discuss low-cost methods for chunk-level reweighting (§3.4.2) and crowdsourcing scheduling (§3.4.3).

[5] For instance, in the WaterlooSQOE-III dataset [62], each video is streamed over 13 throughput traces with 6 ABR algorithms, and each rendered video is then rated by 30 users.

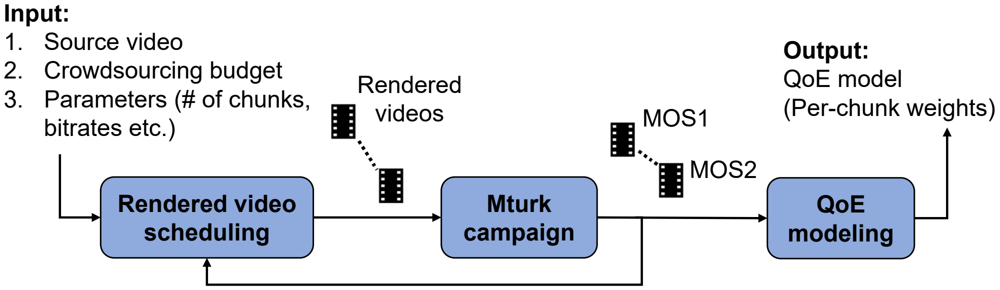


Figure 3.13: Workflow of profiling dynamic quality sensitivity using a crowdsourcing platform. The arrow back to the scheduler means that crowdsourced ratings may be used to suggest more rendered videos to iteratively refine the QoE modeling.

3.4.1 QoE modeling workflow

Figure 3.13 shows Sensei's workflow for QoE modeling. Sensei takes a source video and a monetary budget as input and returns a QoE model that incorporates dynamic quality sensitivity (customized for this video) as output.

- *Rendered video scheduling (§3.4.3):* We first generate a set of *rendered videos* from the source video. Each rendered video is created by injecting a carefully selected low-quality incident at a certain point in the video.
- *MTurk campaign (§3.5):* The rendered videos are published on the MTurk platform and we specify how many *participants* (MTurkers) to recruit for this *campaign*. When an MTurker signs up, they start a *survey* that asks them to watch K rendered videos and, after each video, rate its QoE.
- *QoE modeling (§3.4.2):* Finally, we use the MOS of each rendered video as its QoE and use regression to derive the per-chunk weights, which are then incorporated into an existing QoE model to derive the QoE model for this video.

3.4.2 Cutting cost via chunk-level reweighting

While crowdsourcing scales QoE profiling elastically, profiling each video can still be prohibitively expensive. Since a QoE model must capture the impact of both quality incidents and the quality sensitivity of each chunk, a strawman solution would build a QoE model with

O($N \cdot P$) parameters, where N is the number of chunks and P is the number of parameters in a traditional QoE model. This could require a prohibitive number of ratings to build (*e.g.,* KSQI has tens of parameters).

Encoding quality sensitivity with per-chunk weights: We leverage the insight that quality sensitivity at a chunk is inherent to its video content (§3.2.2). Thus, Sensei assigns a single weight to each chunk irrespective of the quality incident, reducing the number of model parameters to $O(N)$. Then, *Sensei reuses an existing QoE model but reweights the chunks by their quality sensitivity.* If the QoE model is *additive*, *e.g.,* the overall QoE is the sum of the QoE estimates of individual chunks q_i, or $Q = \sum_{i=1}^{N} q_i$, then Sensei can directly reweight the chunks by their quality sensitivity. Though some QoE models are non-additive (*e.g.,* LSTM-QoE), many mainstream QoE models including KSQI and others [168, 117] are. For KSQI, the q_i take into account the impact of visual quality, rebuffering, and quality switches. Sensei reweights the QoE model as follows:

$$Q = \sum_{i=1}^{N} w_i q_i, \tag{3.1}$$

where w_i is the weight of the i^{th} chunk, reflecting how much more sensitive users are to quality incidents in this chunk than in other chunks.

Weight inference: Given any V rendered videos, if Q_j is the QoE (MOS) of the j^{th} rendered video and $q_{i,j}$ is the estimated QoE of the i^{th} chunk of the j^{th} rendered video, then we can write V equations, $Q_j = \sum_{i=1}^{N} w_i q_{i,j}$ for $j = 1, \ldots, V$. We can then infer the w_i using a linear regression.

In the remainder of the paper, we assume that KSQI reweighted by Equation 3.1 is the QoE model of Sensei.

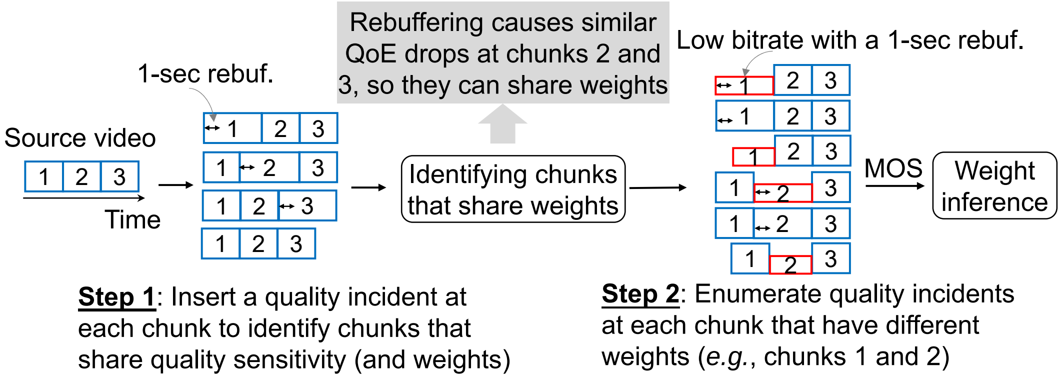

Rebuffering causes similar QoE drops at chunks 2 and 3, so they can share weights
Low bitrate with a 1-sec rebuf.
1-sec rebuf.
Source video
1
2
3
Time
Identifying chunks that share weights
MOS
Weight inference
Step 1: Insert a quality incident at each chunk to identify chunks that share quality sensitivity (and weights)
Step 2: Enumerate quality incidents at each chunk that have different weights (e.g., chunks 1 and 2)

Figure 3.14: A running example of the crowdsoucring scheduler for a source video with 3 chunks, 2 bitrate levels (high and low), 2 rebuffering event levels (0 and 1 second).

3.4.3 Crowdsourcing scheduler

We now turn our attention to compiling a small set of rendered videos that, after being rated, will produce enough data to reliably estimate the per-chunk weights.

Two-step scheduling: Given a source video, Sensei's scheduler uses a two-step process to decide which rendered videos to publish and how many participants to elicit ratings from.

- First, Sensei creates a set of N rendered videos, each with a single 1-second rebuffering event at a different chunk (recall N is the number of chunks). It then publishes these videos and asks M_1 participants to rate each video. The total rendered video duration is $O(N \cdot M_1)$. Once the videos are rated, we infer the per-chunk weights as described above.
- Second, we pick $N' \ll N$ chunks whose inferred weights are α-high or low (*e.g.,* 6 % higher or lower than the average weight). We then repeat the first step with two differences: (1) low-quality incidents are added only to these chunks, and (2) the quality incidents include B bitrates (below the highest bitrate) and F rebuffering events (1,2,...seconds). We publish the rendered videos and ask M_2 participants to rate them, for a total video duration of $O(N' \cdot B \cdot F \cdot M_2)$.

The purpose of the first step is to use a small number of participants (M_1) to get a noisy but indicative estimate of which chunks have quality sensitivity that is very high or low, so we can *focus* the second iteration on these chunks using a larger number of participants

(M_2). In general, for an ABR algorithm to improve QoE-bandwidth tradeoffs, it is more important to identify which chunks have very high/low quality sensitivity than to precisely estimate the quality sensitivity of every chunk. §3.5 discusses the number of participants; we evaluate the effect of α, B and F in §3.7.4. These parameters are empirically selected and held constant throughout our tests.

Figure 3.14 shows an example two-step schedule for a source video. In the first step, we generate a series of rendered video with the same rebuffering event injected at different chunks. By examining the ratings of these videos from the MTurkers, we determine that chunks 2 and 3 have similar sensitivity to the rebuffering event, allowing them to share the same chunk-level weight. Thus, in the second step, we only need to enumerate the quality incidents for chunks 1 and 2. In practice, for a 20-second video, we generate 5 rendered videos in the first step for the $N = 4$ chunks, of which $N' = 2$ chunks may have high/low sensitivity, and generate 15 rendered videos in the second step for these chunks.

Quality incidents used in profiling: For the set of B bitrates, we use the bitrate levels of YouTube videos and pick three of them to cover high, medium and low visual quality; we found this to be a practical compromise. The set of rebuffering events F are chosen to match those we plan to proactively add to the video (see §3.6). Testing on a larger set of quality incidents would yield more data points, but our microbenchmarking results in §3.7.4 show that this only marginally improves model accuracy, while significantly increasing the cost.

3.5 Reliable QoE Crowdsourcing

Sensei's QoE model crucially depends on the *reliability* of MTurkers' quality ratings. This section describes our user survey procedure and techniques for increasing reliability. While Sensei mostly follows known practices [84, 80, 119], we provide some key details that arose from our experience (described below and in Appendix .4).

Single-survey procedure: As shown in Figure 4, each survey starts with the instructions and rejection criteria under which the ratings will be rejected. The MTurker then watches an example video that includes a quality incident, so they know what their ratings should be based on. Then the MTurker is asked to watch a sequence of rendered videos (determined by the scheduler) and, after each video, rate its quality on a scale of 1-5. Finally, the MTurker does an exit survey.

Quality control per survey: Several measures are taken to prevent and filter out spurious user ratings. First, we show the test videos in a *randomized order* to each MTurker. This eliminates biases due to viewing order and which videos were previously watched. Second, we add *reliability checks*: we show a video without any quality incident at a random position among the test videos, and if an MTurker does not give the highest score to this video, we discard all of their ratings. We also ask the MTurker what quality incident(s) they just saw in the last video, and if they report more quality incidents than were included, that rating is discarded. This may occur if the MTurker's network connection is poor and new quality incidents are introduced. Third, we implement an engagement test to verify if the MTurker watched the video in its entirety, by monitoring the time spent on the video playback page and discarding the rating if the time is shorter than the video length. We also implement other filters, such as limiting the number or length of videos per MTurker to prevent fatigue.

Use of Master MTurkers: We follow a common practice (*e.g.,* [113]) and restrict our tests to *Master MTurkers*, a class of reliable MTurkers who have participated in over 1000 surveys and whose feedback was accepted for over 99% of their prior surveys. We find that our rejection rate from Master MTurkers is over 4× lower than normal MTurkers. One lesson we learned is that Master MTurkers are more willing to participate if the publisher (us) historically has a low rejection rate because they wish to maintain their rejection rate below 1%.

Sanity check of our dataset: To check if MTurker ratings are similar to prior lab studies [157, 80], we select three 12-second videos from a public dataset [62] whose quality ratings are collected in a lab environment, and obtain MTurker ratings for these videos. We find that the MTurker responses are similar to the in-lab study: after normalizing the ratings to the same range, the MTurkers' MOS differs by less than 3% from the in-lab study's MOS on the same video.

How many MTurkers are needed? We did a head-to-head comparison with WaterlooSQOE-III [62] and found that we need 17% more MTurkers to reduce the variance of QoE ratings down to the levels of the in-lab study. §3.7.4 shows how the number of MTurkers affects Sensei's performance.

Despite the above, we acknowledge that our MTurk survey methodology could be susceptible to human factors.

3.6 Sensei's ABR Logic

The key difference between Sensei's ABR logic and traditional ABR logic is that Sensei aligns quality adaptation with the temporal variability of quality sensitivity. We first show how Sensei modifies a traditional ABR framework (§3.6.1), and then show how existing ABR algorithms can be minimally modified to benefit from Sensei (§3.6.2).

3.6.1 Enabling new adaptation actions

Sensei takes a pragmatic approach by working within the framework of existing players. It proposes specific changes to their input and output, as highlighted in Figure 3.15.

Input: Besides the current buffer length, next chunk sizes, and history of throughput measurements, Sensei's ABR algorithm takes as input the sensitivity weights of the next h chunks, where h is the *lookahead horizon.* A larger h allows us to look farther into the future for opportunities to trade current quality for future quality, or vice versa. In practice, we are also constrained by the reliability of our bandwidth prediction for future chunks. We microbenchmark the selection of h in §3.7.5.

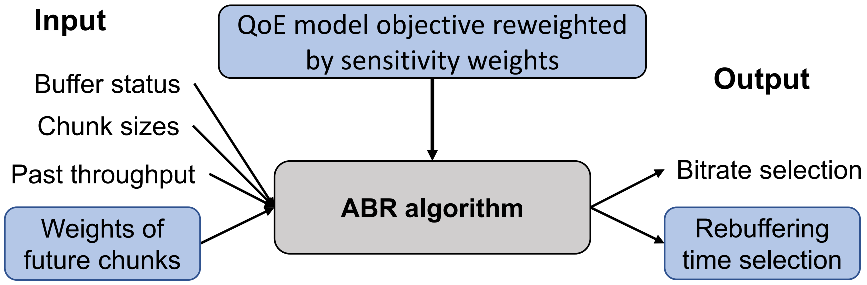


Figure 3.15: ABR framework of Sensei. The differences with traditional ABR framework are highlighted.

Output: Sensei's ABR algorithm selects the bitrate for future chunks as well as when the next rebuffering event should occur.[6] In contrast, traditional players only initiate rebuffering events when the buffer is empty.

QoE model objective: If the ABR algorithm explicitly optimizes an additive QoE model, Sensei can modify its objective as described in §3.4.2. While Sensei can be applied to most ABR algorithms (*e.g.,* [117, 165, 168]), some (*e.g.,* BBA) do not have an explicit objective that Sensei can build on.

In theory, these changes are sufficient to enable at least the following optimizations, which traditional ABR algorithms are unlikely to explicitly do. (1) Lowering the current bitrate so that it can raise the bitrate for the next few chunks, if they have higher quality sensitivity (Figures 3.16(a) and (b)). (2) Raising the current bitrate slightly over the sustainable level if quality sensitivity is expected to decrease in the next few chunks. (3) Initiating a short rebuffering event now in order to ensure smoother playback for the next few chunks, if they have higher quality sensitivity (Figures 3.16(c) and (d)).

3.6.2 Refactoring current ABR algorithms

We apply Sensei to two ABR algorithms: Pensieve [117], based on deep reinforcement learning, and Fugu [165], a more traditional algorithm based on bandwidth prediction.

[6] Sensei currently makes adaptation decisions only for the next chunk, but in principle it could plan adaptations for multiple chunks in the future.

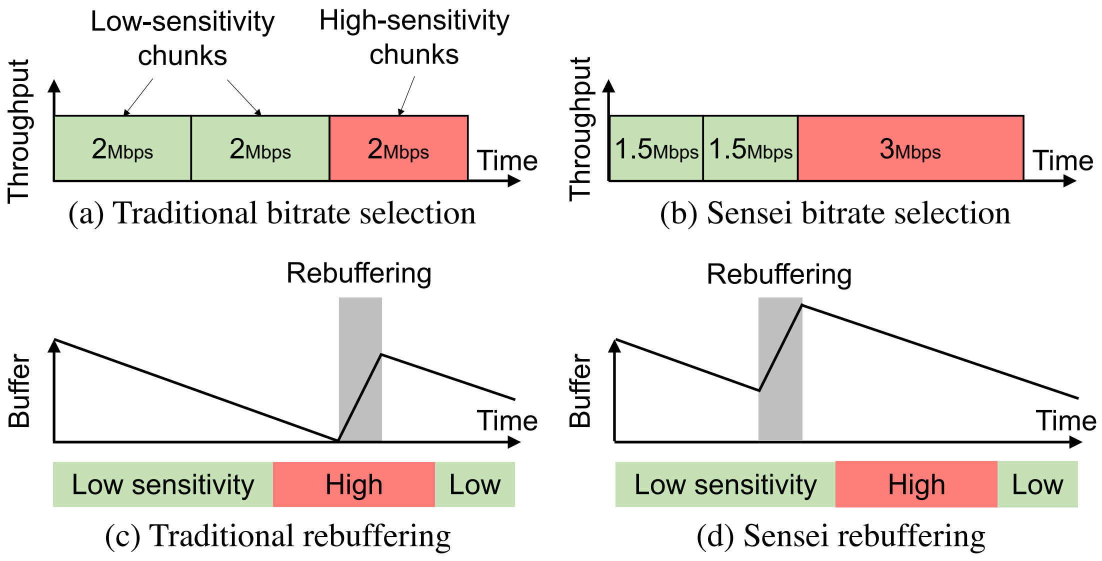


Figure 3.16: Illustrative examples of Sensei vs traditional ABR logic: how Sensei improves quality (a vs. b) or avoids bad quality (c vs. d) for high-sensitivity chunks.

Applying Sensei to Pensieve: Sensei leverages the flexibility of deep neural networks (DNNs) and augments Pensieve's input, output and QoE objective—its states, actions, and reward, in the terminology of reinforcement learning—as described in §3.6.1. It then retrains the DNN model in the same way as Pensieve; we call this variation Sensei-Pensieve. Sensei-Pensive makes two minor changes to reduce the action space (which now includes rebuffering). First, we restrict possible rebuffering times to three levels ({0,1,2} seconds) that can only happen at chunk boundaries. Second, instead of choosing among combinations of bitrates and rebuffering, Sensei-Pensieve either selects a bitrate or initiates a rebuffering event at the next chunk. If it chooses the latter, Sensei-Pensieve will increment the buffer state by the chosen rebuffering time and rerun the ABR algorithm immediately.

Applying Sensei to Fugu: Let us first explain how Fugu works. At a high level, before downloading the i^{th} chunk, Fugu considers the throughput prediction for the next h chunks. For any throughput variation γ (with predicted probability $p(\gamma)$) and bitrate selection $B = (b_i, \ldots, b_{i+h-1})$, where b_j is the bitrate of the j^{th} chunk, it simulates when each of the next h chunks will be downloaded and estimates the rebuffering time $t_j(B, \gamma)$ of the

j^{th} chunk (which could be zero). It then picks the bitrate vector $(b_i, \ldots, b_{i+h-1})$ that maximizes the expected total quality over the next h chunks and possible throughput variations: $\sum_\gamma p(\gamma) \sum_{j=i}^{i+h-1} q(b_j, t_j(B, \gamma))$. Here, $q(b,t)$ estimates the quality of a chunk with bitrate b and rebuffering time t using a simplified model of KSQI.

The Sensei variation of Fugu, which we call Sensei-Fugu, uses Fugu's throughput prediction and the sensitivity weights w_j of the next h chunks. Sensei-Fugu picks the bitrate vector $B = (b_i, \ldots, b_{i+h-1})$ and the rebuffering time vector $T = (t_i, \ldots, t_{i+h-1})$, where t_j is the rebuffering time of the j^{th} chunk, that maximizes the expected total quality over the next h chunks and possible throughput variations:

$$\sum_\gamma p(\gamma) \sum_{j=i}^{i+h-1} w_j q(b_j, t_j) \tag{3.2}$$

Here, the chosen rebuffering times must be feasible, *i.e.,* the buffer length can never be negative.

In short, Sensei-Pensieve and Sensei-Fugu add an extra action (rebuffering time per chunk), and their objective function reweights the contribution of each chunk's quality using the sensitivity weights provided by our QoE model.

3.6.3 Player implementation and integration

We implement Sensei on DASH.js [4], an open-source player that several commercial players are based on. We add a new field in the DASH manifest file (under `Representation`) to represent per-chunk sensitivity weights and change the parser `ManifestLoader` to parse these weights. Unlike other ABR players, Sensei may initiate rebuffering when the buffer is not empty. We use Media Source Extensions [7] (an API that allows browsers to change player states) to delay a downloaded chunk in the browser buffer from being loaded into the player buffer. We also describe the implementation of our crowdsourcing pipeline for MTurk surveys in Appendix .5.

3.7 Evaluation

Our evaluation of Sensei shows several key findings:

- Compared to recent proposals, Sensei can improve QoE by 7.7-52.5% without using more bandwidth or can save 12.1-50.3% bandwidth while achieving the same QoE.
- The performance gains of Sensei come at a cost of $31.4/minute video, which is marginal compared to the investments made by content providers.
- Sensei can improve QoE prediction accuracy by 11.8-37.1% over state-of-the-art QoE models.
- Sensei's ABR algorithm consistently outperforms baseline ABR algorithms even when bandwidth fluctuates.

3.7.1 Experimental setup

Test videos and throughput traces: Our test videos are selected from four datasets: LIVE-MOBILE [72], LIVE-NFLX-II [31], and WaterlooSQOE-III [62] are professional-grade datasets often used to train/compare QoE models in the literature. We complement these sources with videos from a user-generated dataset, YouTube-UGC [158]. The videos are randomly selected from four video genres: sports, gaming, nature, and animation. Appendix §.3 provides more details about the videos. To create an adaptive video streaming setup, we chop videos into 4-second chunks and encode each chunk at 5 bitrate levels: $\{300, 750, 1200, 1850, 2850\}$Kbps. We randomly select 10 throughput traces from two public datasets, FCC [53] and 3G/HSDPA [135], restricting our selection to those whose average throughput is between 0.2Mbps and 6Mbps, forcing the ABR algorithms to adapt their bitrates.

Baselines: We compare Sensei's ABR algorithm with three baselines: Buffer-based adap-

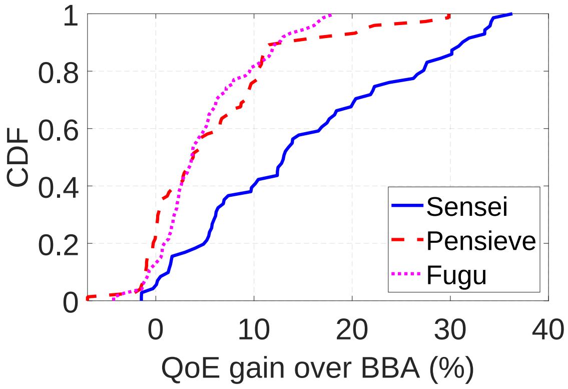


Figure 3.17: QoE gains over BBA

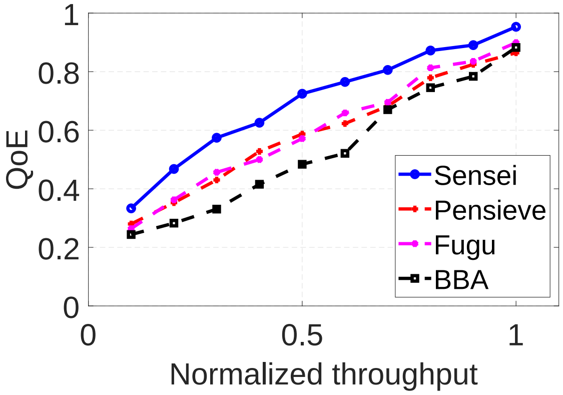


Figure 3.18: QoE vs. bandwidth usage

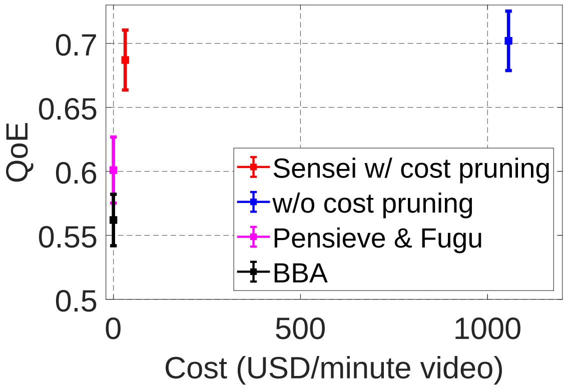


Figure 3.19: MTurk cost vs. QoE

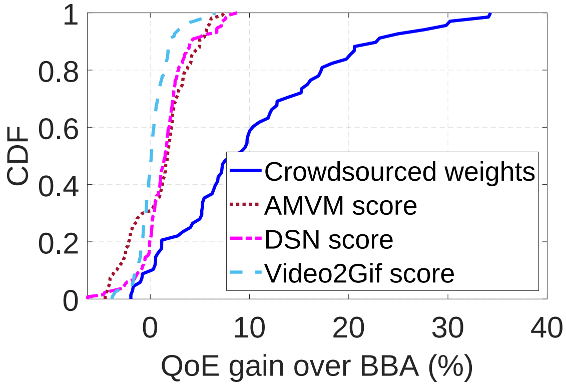


Figure 3.20: Sensei with crowdsourced weight vs. saliency-based weights

Figure 3.21: End-to-end performance of Sensei over traditional and saliency-based ABR baselines across all videos.

tation (BBA) [87], Fugu [165], and Pensieve [117]. We keep their default settings (*e.g.,* same DNN architecture and training network traces for Pensieve, etc.). For fairness, we use KSQI as the QoE model for Pensieve, Fugu, and the Sensei variants. This modification should improve the quality of Pensieve and Fugu, because the QoE models used in their original implementations are special cases of KSQI. We use Sensei-Pensieve (*i.e.,* the application of Sensei to Pensieve) as Sensei, but confirm that the improvements of Sensei-Fugu are similar (Figure 3.29).

Performance metrics: We use three performance metrics. For a given source and video and throughput trace, we report the **QoE gain** of one ABR algorithm (Q_1) over another

(Q_2), *i.e.,* $(Q_1 - Q_2)/Q_2$, where Q_1 and Q_2 are *rated by MTurkers*. We calculate Sensei's **bandwidth saving** by scaling down the throughput traces and determining how much bandwidth each ABR algorithm needs to achieve the same QoE. We normalize all QoE values to the range $[0, 1]$. We measure the **crowdsourcing cost** paid to MTurk to get enough ratings to profile a 1-minute video. Only Sensei incurs this cost.

3.7.2 End-to-end improvement

QoE gains: Figure 3.17 shows the distributions of QoE gains of Sensei, Pensieve, and Fugu over BBA, across all combinations of the 16 source videos and 10 network traces. Compared to BBA, Sensei has at least 14.4% QoE gain for half of the trace-video combinations, whereas Pensieve's and Fugu's median QoE gains are about 5.7%. The tail improvement of Sensei is greater: Sensei's QoE gain at the 80th percentile is 4.8%, whereas Pensieve's and Fugu's are 0.2% and 0.7% respectively. The fact that Sensei's gains over Pensieve (its base ABR logic) are similar to Pensieve's gains over BBA suggests the significant potential in making an existing ABR algorithm aware of dynamic quality sensitivity.

Bandwidth savings: Figure 3.18 shows the average QoE of different ABR algorithms across the source videos, under one throughput trace scaled down by different ratios (x-axis). We confirm the results are consistent across different throughput traces. We see that when setting a target QoE of 0.8, the bandwidth savings of Sensei is about 27% higher compared to Pensieve and Fugu, and 32% higher compared to BBA.

QoE vs. crowdsourcing cost: Figure 3.19 shows the crowdsourcing cost and resulting QoE of Sensei relative to Pensieve, both with and without the cost-pruning optimization (which is evaluated separately in Figure 3.27). Compared to enumerating all combinations of the quality incidents, we see that costs can be reduced by more than 32× with only a 3.1% degradation in QoE, and Sensei is still 14.7% better on average than its base ABR logic (Pensieve with KSQI). This cost is equivalent to ~\$31.4 per 1-minute video, which is a negligible cost for large content providers that may spend on the order of \$10 billion

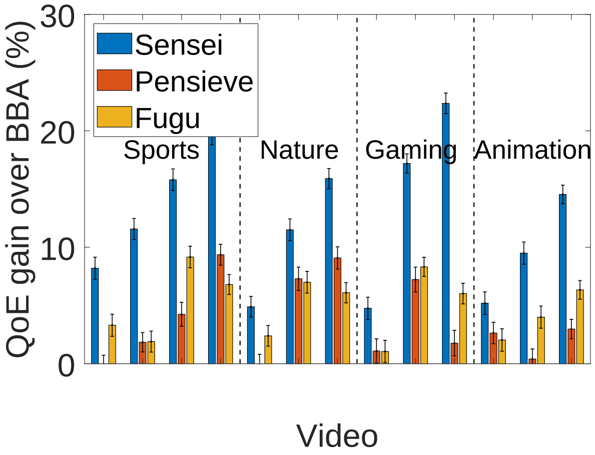


Figure 3.22: QoE gain grouped by genre

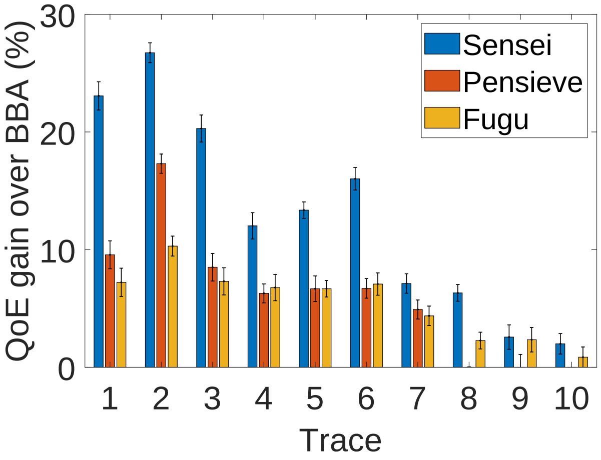


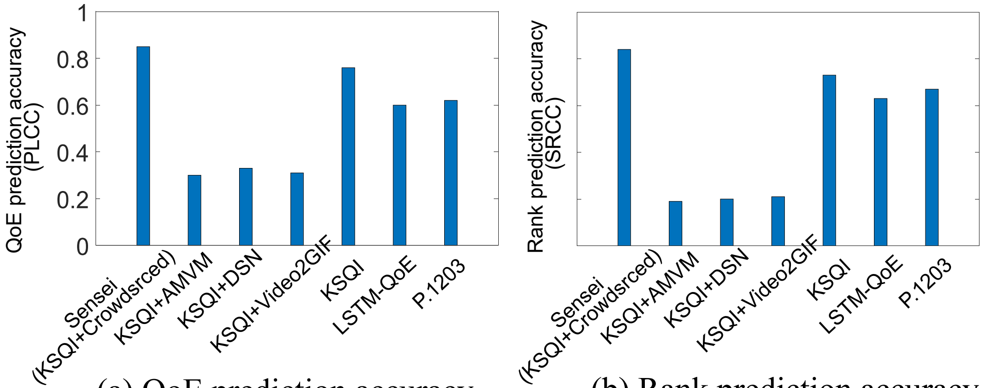
Figure 3.23: QoE gain grouped by trace

Figure 3.24: QoE gains over BBA for genre and for each throughput trace (ordered by increasing average throughput)

annually [5] for licensing popular TV shows (or making such shows).

Improvements by video and trace: Figure 3.22 shows the QoE gains for each video across the network traces. We see significant variability in the QoE gains across videos and even within the same genre. Figure 3.23 shows the QoE gains for each network trace across all videos. Overall, Sensei yields more improvement when the average throughput is lower (towards the left). This shows that Sensei can better maintain high QoE even when the network is under stress.

Sensei vs. saliency-reweighted ABR: Finally, Figure 3.20 shows the QoE gains of Sensei when the per-chunk weights are based on crowdsourcing results (our approach) and when the weights are based on the saliency scores produced by various saliency models (see §3.2.3). We normalize each model's saliency scores to sum to the sum of chunk weights of Sensei. We see that Sensei's gain significantly reduces if the weights are based on these saliency models, because as explained in §3.2.3, they fail to capture users' quality sensitivity.



(a) QoE prediction accuracy

(b) Rank prediction accuracy

Figure 3.25: QoE prediction accuracy of Sensei, Sensei's variants, and baseline QoE models.

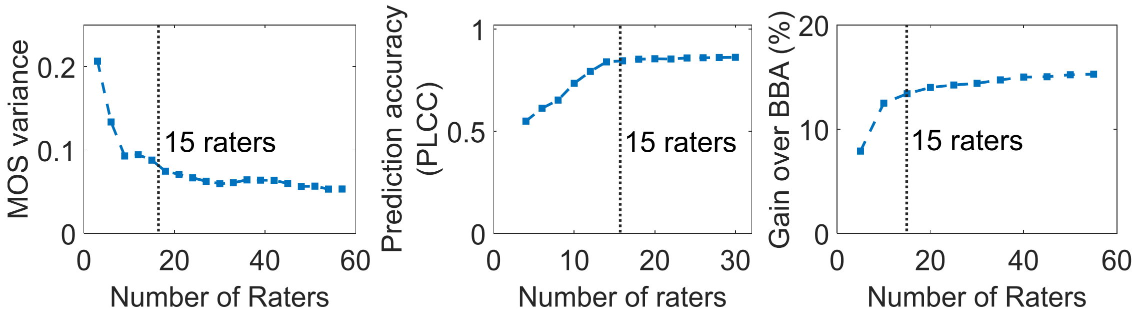
3.7.3 QoE prediction accuracy

We now microbenchmark Sensei's QoE model introduced in §3.4 using all 640 rendered videos generated by running Sensei and the baseline ABR algorithms on all combinations of source videos and network traces. We obtain the "ground truth" QoE of each rendered video using our MTurk survey procedure (§3.4,§3.5). We calculate Pearson's linear correlation coefficient (PLCC) and Spearman's rank correlation coefficient (PRCC) between the predicted QoE and actual user-rated QoE. Figure 3.25 compares Sensei with three baselines QoE models (KSQI, LSTM-QoE, P.1203). The PLCC (and PRCC) of Sensei's QoE prediction is over 0.85 (and 0.84), whereas the baselines are below 0.76 (and 0.73). We evaluated several variants of KSQI (the best baseline QoE model) re-weighted by per-chunk saliency scores from the saliency models in §3.2.3, but their accuracies are even lower.

3.7.4 Cost savings on crowdsourcing

We microbenchmark the effects of different crowdsourcing parameters on Sensei's QoE model.

Impact of number of raters per video: Figure 3.26(a) shows that while the quality ratings have substantial variance with less than 5 raters, their mean value (MOS) stabilizes with more than 15 raters. As a result, having 15 raters per video (as used in Sensei) produces a similar QoE prediction accuracy (b) and QoE gains (c) as having 30 raters.

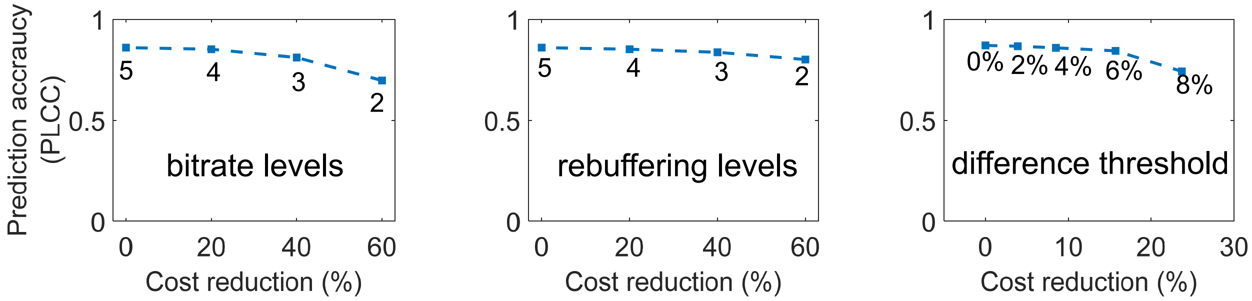


(a) Rating variance vs. # of raters

(b) Model accuracy vs. # of raters

(c) QoE gain vs. # of raters

Figure 3.26: The effect of the number of raters



(a) # of bitrate level

(b) # of rebuffering level

(c) Filtering threshold

Figure 3.27: QoE model accuracy changes with cost.

Impact of crowdsourcing schedule granularity: Figure 3.27 shows the effect of reducing MTurk cost by considering (a) fewer bitrate levels (B), (b) fewer rebuffering events (F), or (c) higher threshold α used to pick which chunks to investigate in the second step. These terms are defined in §3.4.3. By reducing B to 3, F to 2, or raising α to 6%, we greatly reduce the cost while incurring less than 3% drop in accuracy.

3.7.5 Sensei's ABR logic

Finally, we microbenchmark Sensei's ABR logic (§3.6). To scale this experiment out, we use real videos and throughput traces but use the QoE predicted by Sensei (instead of real user ratings) to evaluate QoE. We have confirmed that this yields the same QoE estimates on average as real user ratings under the same setting.

Impact of bandwidth variance: Figure 3.28 shows the performance of Sensei under

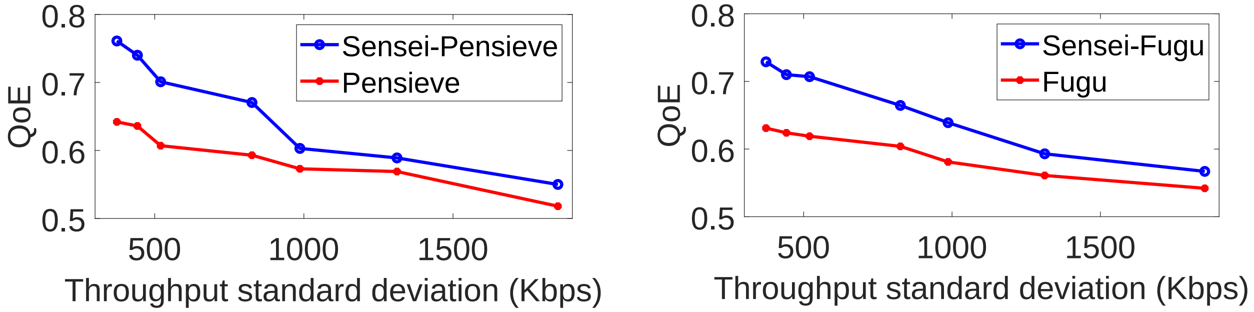


Figure 3.28: QoE under increasing bandwidth variance.

increasing throughput variance. We pick one throughput trace and increase its throughput variance by adding unbiased Gaussian noise. The graph begins at the variance of the original throughput trace; as variance increases, Sensei's QoE degrades gracefully, but it still maintains a significant gain over its base ABR logic (Pensieve or Fugu). This is because Sensei only needs to predict how likely low throughput will occur on high quality-sensitivity chunks, not all future chunks, so if the average throughput until the next such chunk is predictable, it will work well. We confirm the results are similar on other throughput traces.

Performance breakdown: Figure 3.29 shows that Sensei achieves comparable improvement when either Pensieve or Fugu is the base ABR logic. This suggests that Sensei's gains do not depend on the choice of the base ABR logic. Figure 3.30 shows that both aspects of Sensei's control logic contributes to its improvements: (1) making ABR logic aware of dynamic quality sensitivity (1st vs. 2nd bar), and (2) injecting rebuffering judiciously (2nd vs. 3rd bar). Thus, even if a content provider cannot control rebuffering, it can still benefit significantly from Sensei's dynamic quality sensitivity.

Impact of video contents: While the videos in our dataset have varying fractions (from 20% to 60%) of high-sensitivity chunks, Figure 3.31 tests Sensei's performance under an even wider range of high-sensitivity chunk fractions (from 0% to 100%). We create source videos with the specific fractions of high and low quality-sensitivity chunks and randomize the positions of the chunks. Sensei has marginal improvement when the video is dominated by either high or low quality-sensitivity chunks. However, Sensei significantly improves QoE

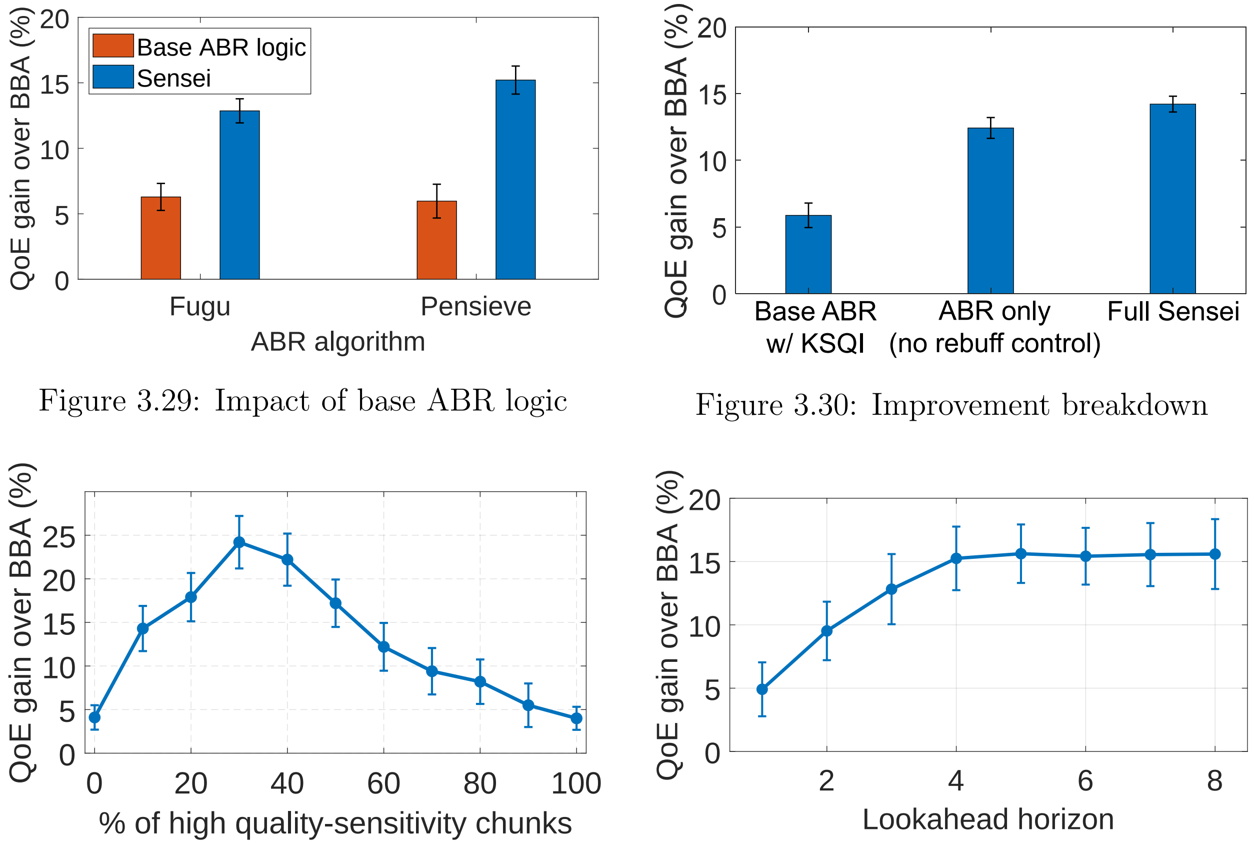


Figure 3.29: Impact of base ABR logic

Figure 3.30: Improvement breakdown

Figure 3.31: Chunk sensitivity

Figure 3.32: Lookahead horizon

Figure 3.33: Understanding Sensei's improvements.

when high quality-sensitivity chunks are 20-40% of a video (most of our videos fall in this range).

Lookahead horizon: Figure 3.32 tests the impact of lookahead horizon—the number of future chunks h whose quality-sensitivity weights are revealed to the ABR algorithm. A longer horizon increases Sensei's ability to schedule quality events between low and high quality-sensitivity chunks. Empirically in our dataset, the QoE gains diminish after the lookahead horizon is greater than 4 chunks.

Systems overhead: We confirm empirically that compared to a video player without Sensei, the runtime overhead of Sensei is less than 1% in both CPU cycles and RAM usage.

3.8 Related Work

ABR algorithms: Mainstream ABR algorithms maximize bitrate under dynamic available bandwidth. Traditional ones are buffer-based (*e.g.,* [94, 107]) or rate-based algorithms (*e.g.,* [87, 145, 144]). Recent ABR algorithms explicitly optimize a given QoE objective via control theory [168], ML-based throughput prediction [148, 165], or deep reinforcement learning [117, 68, 67]. Some ABR algorithms also rely on server-side processing [26, 167, 97]. Key parameters of the ABR logic can be customized to the network conditions or devices [23]. Though E2E reuses existing ABR algorithms, its contribution lies in identifying minimum changes (*e.g.,* adaptation actions they never would have taken) needed for these algorithms to fully leverage users' dynamic quality sensitivity.

Modeling and optimizing user-perceived quality: Visual quality assessment (VQA) traditionally models user's perception of encoded video using pixel-level patterns (*e.g.,* [74, 159, 134, 143]) as well as advanced data-driven models, such as SVM [13] and deep learning models (*e.g.,* [98]). Adaptive quality assessment (AQA), on the other hand, models streaming-related incidents, including join time, bitrate switches, rebuffering (*e.g.,* [59, 28, 100]). Recent QoE models combine VQA and AQA (*e.g.,* [30, 29, 62, 60, 56, 44, 65]) and sometimes uses spatial/temporal visual attention (*e.g.,* [126, 124, 164, 67, 73, 61, 67]).

These perception-centric QoE models have inspired a large body of work that maximizes user-perceived quality with bitrate adaptation [128, 121], adaptive video encoding [177, 142, 26], adaptive bitrate levels [21, 22], dynamic chunk lengths [108], and super resolution [175, 97, 167]. Since the user-perceived quality metrics can vary across chunks, they may also treat video chunks differently, like E2E does. However, as elaborated in §3.2.3, E2E is complementary to these efforts: while they propose heuristics to how pixel-/motion-based visual features affect QoE, E2E customizes itself for each video (in a cost-efficient way) to capture the impact of the *substance of video content* on true user sensitivity to video quality. That said, actions like dynamic bitrate levels, chunk lengths, and super resolution could be used in E2E too, though E2E only considers actions directly supported by current DASH players.

QoE research using crowdsourcing: Prior work (*e.g.,* [84, 46, 129, 119, 81, 162]) provides methodologies for using commercial crowdsourcing platforms, *e.g.,* Amazon MTurk [1] and Microworkers [8], to systematically model user perception using objective quality metrics [129, 84, 162, 81, 46], investigate QoE impact of different types of low-quality events (*e.g.,* [63]), and build crowdsourcing platforms themselves for similar purposes (*e.g.,* [157, 165]). While E2E follows conventional crowdsourcing methodology (§3.5), E2E faces a unique challenge of scaling crowdsourcing to *per-video* QoE modeling. The cost of modeling QoE of each video separately is prohibitive, and E2E drastically prunes the cost by reusing an existing QoE model while profiling only a single weight per representative chunk to encode the content-induced quality sensitivity of each chunk.

3.9 Discussion

Participant selection bias: A concern of any crowdsourced user study is that the results could be biased because the workers who are willing to participate in the user study might have different characteristics than the real video viewers. A common approach to address this bias is to reweight the participant responses based on the demographics of real users (*e.g.,* [146, 115, 153, 57]). Sensei could apply reweighting to the user study if we have knowledge of the target viewers' demographics, or it could directly recruit the user study participants from the target viewers themselves (*e.g.,* subscribers of the content provider).

Inapplicable scenarios: Sensei does not apply to live video streaming and copyrighted videos. Live videos have strict delay requirements which our crowdsourcing-based video profiling cannot meet. Showing copyrighted videos to crowdsource workers poses the risk of copyright violation, though Sensei could be used on already-released videos. Moreover, the profiling cost of $\sim$ \$31.4/minute video may still be impractical for videos with only a few views. Instead, we envision that Sensei will be used for popular on-demand video by content providers who seek to improve their QoE-bandwidth tradeoffs. For example, providers such as Amazon, Netflix and YouTube recently lowered the default bitrate in Europe due to increased network traffic during the COVID lockdown [11].

CHAPTER 4

VIDPLAT: A TOOL FOR FAST CROWDSOURCING OF QUALITY-OF-EXPERIENCE MEASUREMENTS

4.1 Introduction

Measuring Quality of Experience (QoE) *at scale* is increasingly important, for both researchers and video and web service providers. At least two reasons contribute to this trend. First, new optimization methods are proposed every year with the aim of striking various tradeoffs among *objective* quality metrics that are potentially conflicting, such as video bitrate, buffering stalls, and bitrate switches [168, 181, 59] in video, or page load time and time to interactivity [55, 42] in web. Given the complex relationships between quality metrics and QoE [43, 60, 79], it is important to know how these optimizations actually affect QoE. Moreover, recent work increasingly utilizes the heterogeneous sensitivity of QoE to quality metrics across videos, web pages, and even across different segments in the same video (*e.g.,* [139, 172, 89, 39]). Thus, QoE measurements can inform service providers to strategically allocate more compute/network resources or enhance quality at points of higher QoE sensitivity (*e.g.,* 15.4% higher average QoE in video-on-demand [172] and $> 40\%$ better minimum QoE for various web pages [139, 54]).

Measuring QoE is inherently slow since researchers need to ask human users to provide subjective QoE ratings about their experience[1] after they watch a video rendered at certain quality or a web page loaded with a certain page load time. Even though some QoE models exist to predict QoE from objective quality metrics (*e.g.,* VMAF [105] or CNN-based models [75, 178]), user studies are still preferred, because it is hard to analytically capture

[1] As a subjective experience of users, QoE can be influenced by many factors that are not controlled by (or even visible to) service providers, such as device type, customer background, user intention, etc. Therefore, QoE researchers will diversify the recruited human raters to obtain the distribution of QoE over a range of users. There are many existing methods for managing the human raters, *e.g.,* rater recruitment eligibility and validating raters' responses (*e.g.,* [43, 157, 81, 80]), and they complement our work.

all intricate relationships interactions between quality metrics and QoE, and creating these QoE models also requires user studies in the first place.

Scaling QoE measurements involves two challenges, each of which has been tackled *separately.*

- First, how to *automate* QoE measurements while ensuring the collected measurements do reflect users' true QoE? *Crowdsourcing* is promising—researchers can publish a task on MTurk [25] or Prolific [127] to invite crowdsourcing raters to watch a series of *demos* (a video rendered at a certain video quality or a web being loaded with a certain page load time) and provide their QoE ratings of the demos. Previous work has focused on collecting reliable QoE ratings at a fair cost from raters (*e.g.,* [76, 81, 80, 36, 151]), with open-sourced implementations to democratize these efforts (*e.g.,* [157, 46, 162]).
- Second, how to gather QoE ratings for a *potentially huge space* of possible quality (*e.g.,* buffering ratio, average bitrate, bitrate variation, web page load time), whose impact on QoE may compound each other and vary across video or web content? Most works tackle this challenge by *pruning* demos whose QoE can already be inferred by past QoE measurements [172, 111, 43]. For instance, in a user study investigating how video bitrate affects the QoE of a particular video, if human raters are unable to perceive the QoE difference between bitrates of 1 Mbps and 10 Mbps, then no ratings will be needed for the bitrates between 1 Mbps and 10 Mbps [60] on this video. Inspired by such observations, pruning techniques, *e.g.,* based on active learning, can dynamically prune out demos whose QoE could be reliably inferred based on QoE ratings already collected in the past (*e.g.,* [43, 173, 102, 161]).

While these approaches tackle the challenges separately, unfortunately they are not easy to be used together. This is because most demos can be pruned only *after* the QoE ratings of some other demos have been measured, but today's automated crowdsourcing tools require that the researchers must *pre*-determine the demos in a task and the number of QoE ratings needed per demo *before* the task begins. As a result, QoE measurements can be automated by crowdsourcing but still very slow (see §4.2 for concrete examples). First, researchers have to launch a series of tasks, with the QoE measurements collected in one task, deciding which demos will be pruned from the next task. However, each task is treated independently by the crowdsourcing platform and can take significant time to recruit and calibrate each rater. Second, some demos may have less variance in QoE and thus need fewer QoE ratings, but without knowing this in advance, researchers are required to collect more ratings than necessary; otherwise, extra tasks need to be conducted to collect the rest ratings.

To fully realize the speed benefit of crowdsourced QoE measurements, we present *VidPlat, the first re-usable open-source tool that enables dynamic demo pruning to speed up crowdsourced QoE measurements.* VidPlat serves as a shim layer between researchers and crowdsourcing platforms, exposing a more flexible interface. Unlike the traditional crowdsourcing interface that requires researchers to pre-determine the demos upfront, each rater is "redirected" to a website which shows them demos one by one[2], and VidPlat allows researchers to define the pruning logic and a few initial demos, and upon receiving a new QoE rating, VidPlat invokes this logic to *iteratively* determine the subsequent demos based on the past QoE measurements. At the same time, VidPlat must make sure that the demos are shown to raters in a randomized fashion (to minimize biases) and avoid asking a rater to rate too many (similar) demos. To this end, VidPlat puts each demo returned by the pruning logic to a queue, from which VidPlat decides which demo should be the next for each rater.

In short, with VidPlat, researchers do not need to determine all the demos or the required number of QoE ratings before the user study task begins; instead, VidPlat lowers the development burden while still collecting crowdsourced QoE measurements with minimum redundancy. As a result, it greatly reduces the number of demos and QoE ratings collected, thereby saving both time and cost.

[2] This may seem similar to prior solutions like [157], where raters are redirected to a website written by the researchers. But these solutions impose a substantial *development burden* on researchers and also introduce new challenges, such as how to properly randomize the order of demos seen by each rater (which is critical as shown in [43, 157]) and how to prevent habituation (fatigue) effect of each rater. As a result, the potential to speed up QoE measurements by crowdsourcing remains largely unrealized.

Term	**Definition**
Crowdsourcing platform or platform	Online human workforce marketplace, *e.g.,* Amazon Mechanical Turk [25] and Prolific [127].
Source content	The original file of an application with specific content, and this file is with perfect quality, *e.g.,* a RAW video in video streaming.
Low-quality event	A low-quality event happens within a specific time frame, and it can lower the application quality, *e.g.,* bitrate drop, buffering stalls in video streaming, and loading delays of web objects in web services.
Application demo or demo	A source content is rendered with certain low-quality events.
Assignment	In an assignment, a rater needs to rate a demo and answer demo-related questions.
Crowdsourcing task or task	A task includes a set of assignments and the payment structure (*e.g.,* how much will a rater get paid). One or multiple assignments will be allocated to a rater in a task.
Crowdsourcing tool or tool	A shim layer that helps researchers automatically set up tasks on platforms

Table 4.1: A summary of the terms used in the paper.

VidPlat has already been used in three IRB-approved QoE-related projects: *(i)* investigating the relationship between webpage load time and QoE [174]; *(ii)* exploring the correlation between video quality and QoE in on-demand video streaming [172]; and *(iii)* comparing the QoE impact of video bitrate and motion-to-photon (MTP) latency in online video gaming [49]. VidPlat's dynamic demo determination significantly improved the efficiency of our user studies. For instance, compared to Sensei [172], a prior tool employing a traditional interface, VidPlat reduced monetary costs by 31.8% – 46.0% and latency by 50.9% – 68.8% in these use cases, while obtaining QoE models that realize the same QoE improvement as Sensei. These empirical results demonstrate the tangible benefits of our novel approach.

4.2 Motivation

We begin with the background on why QoE measurements need to be made more efficiently. Then we explain why the two existing approaches—crowdsourcing and demo pruning—are not sufficient and use a dataset of crowdsourcing logs to highlight opportunities for a new tool.

4.2.1 Background

QoE measurements: Content providers strive to maintain high QoE (quality of experience), which represents the subjective user satisfaction with the service quality. However, it is hard to directly ask users to rate their subjective experience in real time. As a consequence, researchers and content providers run offline user studies to assess QoE under various objective quality metrics, such as video bitrate and buffering stall or web page load time in web services [111, 59].

In such QoE user studies, the participants are asked to watch an application *demo*. A demo is a video clip that shows the application at a certain quality level. In video QoE, a demo can be a video streamed with a one-second buffering stall deliberately added at a certain point [60]. In web QoE, a demo can be a video that records a web page loaded with a certain page load time (*e.g.,* a certain above-the-fold time) [157]. Then, the participants rate the subjective QoE score in the range from 1 to 5 [91]. With these QoE measurements, we can calculate the mean QoE scores (mean opinion score or MOS) of different demo videos and use them to model the relationship between QoE and quality metrics. Using these QoE models, content providers can online optimize the quality metrics in a way that also optimizes QoE.

More QoE measurements are needed: Traditionally, researchers and content providers expect QoE models to capture *general* relationship between QoE and a few quality metrics. As a result, QoE measurements are not in very high demand because if enough QoE measurements are collected to model QoE on several representative videos or web pages, the QoE models will be re-used on other videos or pages. However, many recent efforts have advocated for more granular, *context-specific* QoE models that quantify the QoE-quality relationship of individual video (or even video segments) [172] or each website (web page) [55]. They show that the context-specific QoE models are more accurate because users may not pay equal attention to different content when watching videos [172] or perceive web objects in the same order when browsing different websites [39, 55][3]. As the research and the industry move from one-size-fits-all QoE models to context-specific QoE models, the frequency and amount of QoE measurement also increase quickly. For instance, Netflix produces on average more than 580 minutes worth of new video content every day [90]. If it builds a separate QoE model for each minute of video, it will ask raters to watch and rate roughly $9,600$ hours of videos in total (or 1,200 raters for 8 hours) [172]. Similarly, web QoE research also shows a similar increase in the demand for QoE measurements [39, 55].

More importantly, they show that these context-specific QoE models can substantially improve QoE without using more bandwidth or compute resources. For example, in video streaming, applying per-video QoE models to adaptive bitrate (ABR) algorithms in video players can improve 15.1% QoE without using more network bandwidth [172]; in web services, we can have 28% QoE improvement by allocating computing resources across different web requests by their QoE models [55, 42, 174].

An additional reason for more QoE measurements is that when a new system optimization is proposed, its impact on QoE may not be captured by existing QoE models (*e.g.,* [176, 118]). For instance, customizing video bitrate encoding ladders for each video or even each video segment has been recently studied, but if the existing QoE measurements only cover bitrate levels from a fixed ladder, QoE might change significantly within a bitrate ladder, and the QoE within this ladder cannot be modeled by the existing QoE measurements.

4.2.2 Two existing approaches

As the need for QoE measurements rises, so does the need to reduce the *latency* and the *cost* of QoE measurements. Here, the cost means the total compensation paid to the raters who provide the QoE ratings, and the latency means the timespan between the recruitment

[3] The fact that the QoE-quality relationship differs across web or video content is hardly surprising and is long known, but it was not until recently that researchers have started to explore the potential improvement in practice, thanks to the use of crowdsourcing platforms to automate QoE studies.

of raters and the end of the collection of QoE ratings. Currently, there have been two approaches to reducing the latency and cost of QoE measurements.

Prior approach #1: Crowdsourcing QoE measurement. While QoE modeling is still sometimes performed in a lab setting with participants recruited by researchers themselves, several efforts (*e.g.,* [46, 80, 157, 47]) have shown the potential of automating QoE measurements using platforms like Amazon Mechanical Turk (MTurk) [25] and Prolific [127]. Efforts along this line of work have been focused on retaining reliable crowd raters [76], validating/-calibrating QoE ratings [81, 80], mitigating hidden confounders (*e.g.,* order of assignment completion or different user devices) [36], reducing per-task cost via dynamic pricing [151]. Several open-source crowdsourcing-assisting tools also bundle the aforementioned optimizations [157, 46, 162, 47]. A more comprehensive survey can be found in [80, 81].

Prior approach #2: Dynamically pruning demos. Crowdsourcing can automate QoE measurements, but it does not reduce the number of QoE ratings to collect. Fortunately, it has been observed that, depending on the QoE ratings already collected, many demos would be *redundant* and can be pruned to let participants rate fewer demos. For instance, in a user study investigating how video bitrate affects the QoE of a particular video, if human raters are unable to perceive the QoE difference between bitrates of 1 Mbps and 10 Mbps, then no ratings will be needed for the bitrates between 1 Mbps and 10 Mbps [60] on this video. Inspired by such observations, pruning techniques based on active learning have been proposed to dynamically prune out demo videos whose QoE can already be reliably inferred based on collected QoE ratings (*e.g.,* [43, 172]). For instance, in [125], active learning helps reduce the number of demos to rate for ranking the QoE of the demos in a given dataset.

Before diving into more discussions of the two approaches, we summarize the terms that will be used in the following sections in Table 4.1.

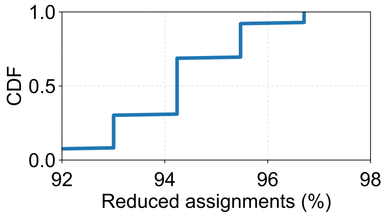


Figure 4.1: Plan-B's assignment saving over Plan-A

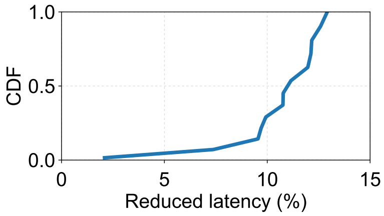


Figure 4.2: Plan-B's reduced latency over Plan-A

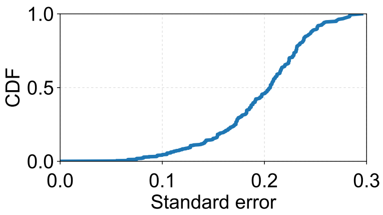


Figure 4.3: MOS stand error across videos if each video is rated by 20 raters

4.2.3 What's missing?

With the use of crowdsourcing tools which automate QoE measurements and the pruning methods which reduce the total number of demos to rate, a natural question is *will QoE measurement be automated and made much faster as promised by these approaches?* Unfortunately, the answer is no, for two reasons.

- First, to dynamically prune demos, researchers have to *sequentially launch a series of crowdsourcing tasks*, which can cause significant latency, as we will show shortly, and use the QoE measurements from one task to decide which demo can be pruned in the next task.
- Second, each demo might have a different variance in its QoE ratings [141], causing the minimum number of ratings needed for different demos can *vary significantly*.

In the following, we will expand on each point using empirical evidence. We use a real example where we want to collect QoE measurements to quantify how QoE changes with the length of a buffering stall at each chunk of a source video[4]. Here, we assume the source video has n chunks and each chunk can have d buffering stall lengths. To measure the latency of crowdsourcing, we use a dataset [172] that consists of user study logs gathered from raters enlisted from MTurk. There are 16 source videos with distinct content, each having 4-second chunks.

The key reason that the promise is not realized is the **standard crowdsourcing inter-**

[4] This is part of the QoE model proposed in [172].

face, which requires that the researchers must *pre*-determine which demos will be rated by each rater and how many QoE ratings are needed per demo *before* the task begins. Given this interface, a typical process to crowdsourcing QoE measurements is as follows.

- *Rater recruitment:* The researcher publishes a task on the crowdsourcing platform (MTurk [25] or Prolific [127]), which matches the task with raters.
- *Training phase:* Once a rater agrees to do the task, the rater will be invited to go through a training phase, during which the rater is explained what to do and the definition of QoE, and in the meantime, the rater also needs to provide some basic information (*e.g.,* age group, gender, devices, etc.). This training phase is necessary to calibrate the QoE ratings of raters, as confirmed in [80, 70].
- *QoE rating:* After the training phase, a rater will be invited to watch a set of demos and provide their QoE rating after each demo.

Limitation #1: Pruning demos causes high crowdsourcing overhead. A naive plan (dubbed *Plan-A*) is to enumerate each of the d buffering stalls on each of the n chunks. This will lead to a sheer d^n demos in total, all of which require QoE ratings from raters. Fortunately, the number of demos can be greatly reduced by pruning redundant demos using two rounds of QoE measurements. In the first round, we ask the raters to rate the demos where only a buffering stall of a fixed length is inserted at different chunks. Based on these QoE ratings, we can identify the chunks where the buffering stall does not cause a QoE drop. In the second round, we only need to enumerate buffering stalls in other chunks because buffering stalls have little impact on QoE in the pruned chunks. This two-round plan (dubbed *Plan-B*) will need two crowdsourcing tasks, but the total number of demos to rate is significantly reduced.

Figure 4.1 shows the fraction of demos pruned by Plan-B compared to Plan-A on different source videos. For more than 64% of source videos, at least 94.2% of demos can be pruned. However, this demos reduction fails to translate to a reduction of latency of the user studies. Figure 4.2 shows Plan-B only has less than 2.1% – 12.5% latency reduction.

To understand why, we break down the latency of each QoE crowdsourcing task into rater recruitment, training phase, and QoE rating, as explained earlier. Among the three latencies, rater recruitment and training will increase dramatically with more crowdsourcing tasks and cannot be reduced by pruning demos. In the aforementioned example, the rating time of each rater *was* reduced from 105.2 minutes in Plan-A to 8.5 minutes in Plan-B. However, because Plan-B launches two crowdsourcing tasks, it has twice the latency of rater recruitment and training than in Plan-A. This significantly increases the latency of Plan-B, despite the reduced number of demos to rate.

Unfortunately, with the current crowdsourcing interface, researchers will have to launch *multiple sequential tasks* if they apply dynamic pruning, inducing high latency. Although in the dataset of the above example, we only have two sequential tasks, and it is not the common case since it is a simple QoE model that only considers the impact of buffering stalls with various lengths. When considering more QoE parameters (*i.e.,* a larger QoE parameter space), using traditional interfaces usually leads to a few tens of sequential tasks [111, 141, 43]. For example, in active learning, different numbers of demos need to be rated in different parameter ranges (*e.g.,* low-bitrate range that users cannot distinguish video quality vs. mid-bitrate range in which QoE changes dramatically) to save cost. However, the numbers of demos in different ranges are unknown before QoE study begins but are dynamically determined as QoE ratings are collected. Thus, to minimize the cost, an extreme solution using traditional interfaces is putting each demo in separate tasks, dramatically increasing latency.

Limitation #2: Different demos need different numbers of ratings. Unlike label annotation in machine learning, rating the QoE of a demo is a subjective task without a definitive ground-truth answer. For a demo, not all raters will provide the same ratings, although all raters are required to use the same rating scale as instructed in the training phase (*e.g.,* Absolute Category Rating [91]), because other than video quality, QoE ratings are also affected by other factors like the video viewing environment (*e.g.,* brightness, device, etc.).

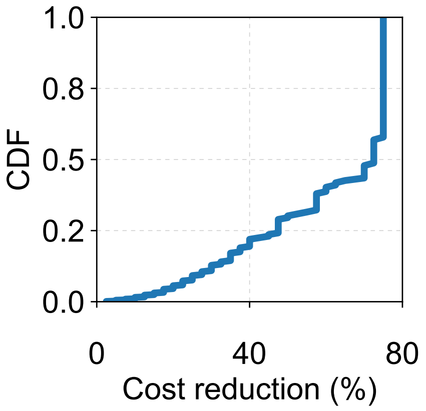


Figure 4.4: Percentage of cost reduction across demos

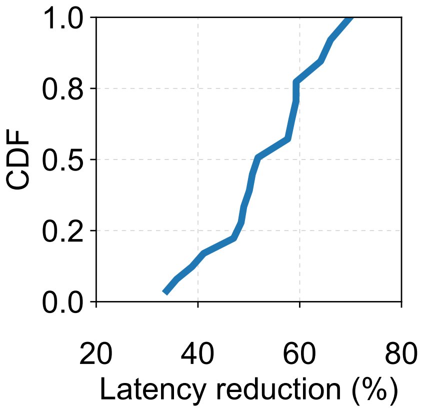


Figure 4.5: Percentage of latency reduction across source videos

The raters might have different environments, and some quality issues are hard to perceive in some environments, so different ratings are provided for the same demo. Figure 4.3 shows that if we fixed the number of ratings (*i.e.*, 20 ratings) collected for each video, the stand error of mean opinion score (MOS) is distributed from 0.05 to 0.3, indicating a large variance across the confidence levels of video MOS.

However, it is hard to determine the number of ratings for each demo without past QoE measurements. Through the traditional interface, for a demo, if researchers set a small number of ratings to collect, there is a risk that the confidence level of the MOS cannot meet our target, leading to significant extra latency for collecting more ratings in a new task. On the contrary, setting a large number of ratings would cause redundant rating collections, leading to extra cost.

Potential improvement by consolidating all QoE measurements into one task: We now show how much cost and latency we can save by overcoming the two limitations by a trace-driven simulation. We consolidate the necessary events, including recruiting raters, watching and rating videos in Plan-B by raters, and training raters, into a single task instead of multiple tasks in the original Plan-B. We align the simulation's settings with those of the data log. The recruitment time of each rater is randomly selected from the log, and for each video, their ratings should make the standard error of MOS smaller than 0.15. Figures 4.4 and 4.5 show the potential cost and latency improvement when compared to Plan-B (using multiple crowdsourcing tasks). We can see that the latency can be improved by 52.1% by consolidating multiple tasks to one task, and the cost can be saved by 57.7% in average cases.

4.2.4 Summary of Key Observations

The key findings in this section can be summarized as follows:

- Through traditional tools, we must conduct *multiple* tasks to minimize QoE measurements for saving cost, but it introduces significant latency.
- If we can consolidate all necessary QoE measurements into one task, we can save 57.7% cost as well as 52.1% latency compared with using traditional tools in average cases.

4.3 VidPlat

To realize the potential improvement from consolidating all necessary QoE measurements as shown in §4.2, we designed a crowdsourcing tool that allows for dynamic assignment allocation during a task. In this section, we first introduce how researchers can utilize our proposed tool, VidPlat, as presented in §4.3.1. We then discuss the design options of VidPlat in §4.3.2 and illustrate how to conduct QoE measurements using VidPlat in §4.3.3.

4.3.1 Programming model

VidPlat enables researchers to set up assignments based on past QoE measurements during a crowdsourcing task to address the two limitations highlighted in §4.2. To achieve this, instead of providing all the demos at the beginning of the task, VidPlat requires researchers to supply 1) a small set of initial demos and 2) a *logic* that takes the collected QoE measurements as input and produces the subsequent demos for the raters as output.

```python
def initialize_assignments(demo_to_emit):
  # Set up the initial videos that are with 1-sec buffering stall at chunks 0 to N-1
  for chunk_id in range(0, N):
    demo_to_emit[generate_demo_with_rebuf(chunk=chunk_id,rebuf_time=1.0)] = 1

def update_assignments(demo_map_ratings, demo_to_emit):
    for (demo, ratings) in demo_map_ratings:
      if stand_error(ratings)>ϵ):
        # we still need more ratings for this demo
          demo_to_emit[demo] = 1
      elif stand_error(ratings)<ϵ) and mean(ratings)<α and demo.rebuf_time==1.0:
        # Need to refine the QoE model by collecting ratings for more buffering stalls stored in rebuf_time_need_to_explore.
        for t in rebuf_time_need_to_explore:
          demo_to_emit[generate_demo_with_rebuf(chunk=demo.chunk_id,rebuf_time=t)]=1
```

Figure 4.6: An code example of setting up QoE measurements for modeling the relationship between video QoE and buffering stall.

Code example: We present a code example showcasing the logic provided by a researcher for the use case mentioned in §4.2, as illustrated in Figure 4.6. The initial demos are defined within `initialize_assignments()`, and are stored as keys in the key-value store named `demo_to_emit`, where the values represent the number of ratings required for the associated demos. As the QoE ratings accumulate, VidPlat invokes the assignment-generation logic, `update_assignments()`, to generate subsequent demos. In this example logic, the researcher stipulates that the standard error of each demo's QoE ratings should be below a threshold ϵ. If a demo's collected QoE ratings have a standard error exceeding ϵ, more ratings for that demo are needed; in this example, the researcher requests an additional QoE rating for the demo each time. Unlike traditional tools which require researchers to execute `initialize_assignments()` after the task concludes, we prompt them to run it during the user task itself. As a result, VidPlat does not notably increase the development burden on researchers, as discussed further in §.8.

4.3.2 Design choices

VidPlat's new interface does not need researchers to specify demos to the raters and is versatile enough to support a wide range of video and web applications. This new interface introduces two primary challenges:

- **Challenge 1: How can we ensure unbiased QoE measurement?** Within VidPlat, different demos may receive ratings from varying numbers of raters who may operate under distinct working environments, potentially introducing bias into the QoE ratings. For example, a worker using a low-resolution device might struggle to discern between high and low-quality videos [180].
- **Challenge 2: How does VidPlat support QoE measurement across various applications?** Unlike conventional tools where demos are directly provided by researchers, in VidPlat, demos are produced based on the assignment-generation logic. Consequently, VidPlat must offer a flexible interface that helps researchers in describing the rendered quality of the demos for rating. A direct approach is to use quality metrics for describing the demo, but enumerating all such metrics is very challenging.

Unbiased QoE measurement: We employ *assignment-level randomization* to overcome bias in QoE measurements: demos are randomly allocated to raters. It requires VidPlat to handle demo distribution as the logic provided by researchers who lack rater information. When deciding on a demo's allocation, VidPlat randomly selects a demo from those generated by the logic, specifically from `demo_to_emit()`.

Since VidPlat does not require every demo to be rated by the same set of raters, unlike the traditional tools, VidPlat does not need to specify the number of raters to recruit. Instead, VidPlat dynamically recruits raters based on how many ratings we need to collect. For example, if we have 100 ratings to collect and 50 raters now, VidPlat will keep recruiting raters for fully parallelizing the QoE rating collection. Such a feature can further reduce the user study latency.

Moreover, VidPlat must validate the QoE ratings for the demos, which is non-trivial due to the absence of ground-truth answers in QoE measurements. In VidPlat, we incorporate a blend of standard practices [157, 80, 81]. Initially, we implement *golden standards* - control questions tied to low-quality events in the assignment (*e.g.,* "did you notice a resolution drop in the video?"). These questions are straightforward for raters who do diligently complete

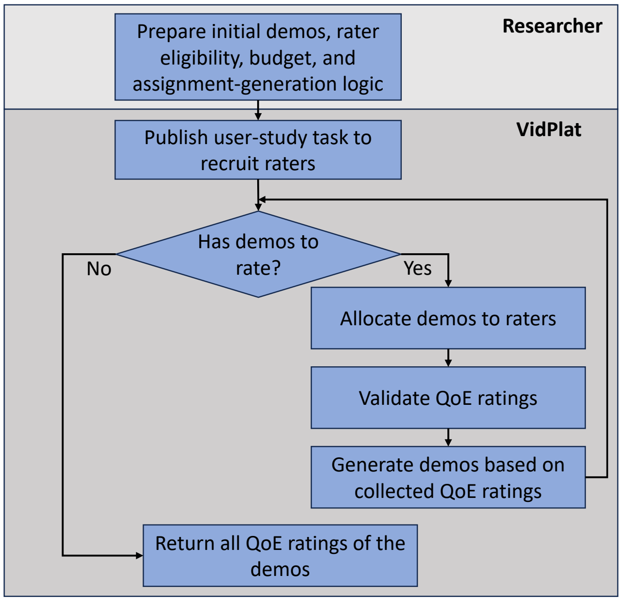
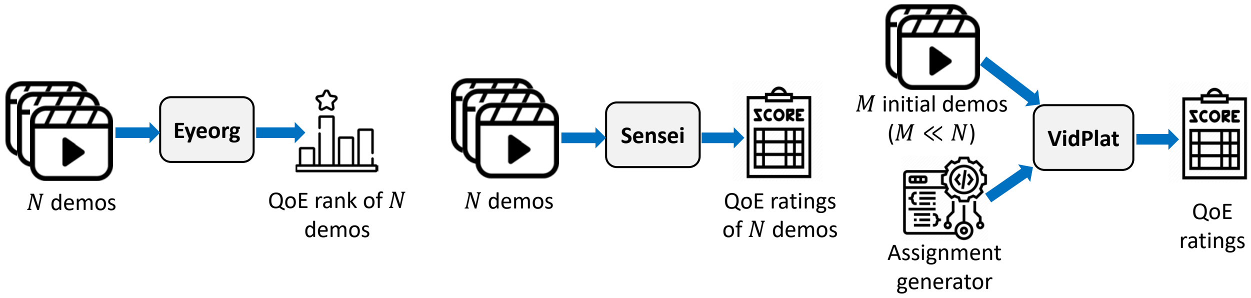

Researcher
Prepare initial demos, rater eligibility, budget, and assignment-generation logic
VidPlat
Publish user-study task to recruit raters
Has demos to rate?
No
Yes
Allocate demos to raters
Validate QoE ratings
Generate demos based on collected QoE ratings
Return all QoE ratings of the demos

Figure 4.7: Workflow of using VidPlat for QoE rating collection

their tasks. Furthermore, VidPlat ensures that raters watch the entire application demo before offering QoE feedback by tracking the active time on the web tab that displays the video players. We disregard the QoE ratings from raters who either skip parts of the video or incorrectly respond to the control questions.

Broad application support: Taking inspiration from Eyeorg [157], VidPlat presents the demos using *videos* that demonstrate the perceived quality. Researchers are required to supply raw videos, meaning videos that present the best quality of the application. Within the logic, researchers must list a sequence of video quality operations to create the desired demo. For instance, to add an extra 0.1-second MTP latency to a one-second clip of an online game, playback rate for that clip could be reduced by 10%. Given these operations, VidPlat can then generate the demos. Using videos offers two advantages. Firstly, the perceived quality remains consistent among raters. Secondly, VidPlat is not burdened with accommodating the myriad quality metrics of various applications, simplifying the development process of VidPlat.



(a) Eyeorg (b) Sensei (c) VidPlat

Figure 4.8: Comparison of researcher interfaces between prior tools and VidPlat.

4.3.3 Set up and conduct QoE study

VidPlat's design facilitates automated QoE measurements. As depicted in Figure 4.7, VidPlat initiates a user study task on crowdsourcing platforms to engage raters for evaluating the demos produced by the assignment-generation logic. VidPlat can persistently allocate demos to raters as long as there are demos available for rating. After the task ends, VidPlat returns all the QoE ratings provided by the raters.

Researcher interface: The distinction between the researcher interface in VidPlat and that in previous tools is illustrated in Figure 4.8. Other than the capacity to produce demos based on accumulated QoE measurements — as is customary in older tools—researchers are tasked with informing VidPlat about their rater eligibility criteria, budget, and raw videos that represent the optimal application quality. Regarding the raw videos, the demos for rating in the QoE study should be derivable from these raw videos by customizing the raw videos' quality. Once the user study is completed, VidPlat returns the QoE ratings of the demos.

Rater interface: For VidPlat, we have adopted the rater interface from Sensei [172]. As raters enroll in our task, they undergo training to ensure they provide valid QoE ratings for the demos. Initially, raters encounter an instruction detailing the task's objectives, outlining their responsibilities and expectations. This instruction also tells them the criteria that might lead to the rejection of their QoE ratings. Subsequently, raters are prompted to watch sample demos showcasing various low-quality events, familiarizing them with the nature of the demos they will rate and instructing them on the rating procedure. Next, raters proceed to evaluate demos designated by VidPlat, assigning scores ranging from 1 to 5. Following each QoE rating, raters are expected to answer control questions to validate their ratings.

4.4 Use cases

VidPlat is particularly advantageous for scenarios that require dynamic demo pruning. Such a dynamic approach enables researchers to efficiently allocate their limited budget across different quality ranges (*i.e.,* determining how many demos within a particular quality range need to be rated) and individual demos (*i.e.,* deciding how many ratings a specific demo requires). In this section, we demonstrate the potential of VidPlat by exploring its application in three distinct use cases: building QoE models for on-demand video streaming, online video gaming, and web services.

4.4.1 On-demand video streaming

With on-demand video streaming, the same video quality may produce varying QoE based on which chunk of the video is viewed, largely due to the content of individual chunks [172]. For instance, a buffering stall right before a crucial moment in a sports video (like just before a goal) can be more frustrating for viewers than during a regular play segment. Thus, to allocate network resources more effectively (*e.g.,* dedicating more resources to pivotal moments rather than a uniform distribution), we should understand and model the QoE sensitivity for every video chunk.

A widely recognized QoE model that accounts for this per-chunk sensitivity is:

$$QoE = \sum_{i=1}^{N} w_i q_i. \tag{4.1}$$

In the equation above, q_i represents the visual quality of the i-th chunk, as explored in prior research [60, 105]. Meanwhile, w_i denotes the weight or QoE sensitivity of chunk i, which we aim to model.

Training for per-chunk QoE sensitivity involves understanding the QoE degradation when certain low-quality events occur in varying chunks. Directly assessing QoE for every possible combination of such incidents across chunks is not practical due to prohibitive costs. Therefore, a pruning method suggested in [172] was introduced to simplify this. This method first pinpoints chunks with similar weightings and then groups them. From each group, a single chunk is selected. By examining the QoE degradation of various low-quality events on this representative chunk, a weight is assigned that applies to all chunks within that group.

Nonetheless, the traditional approach divides the QoE study into two distinct, sequential crowdsourcing tasks: the first task focuses on chunk grouping, while the second task refines the weight for each chunk group. Specifically, for a video comprising N chunks, the first task creates N videos, each experiencing a 1-second buffering stall at varying chunks. The QoE scores for these videos provide a rough estimate of each chunk's QoE sensitivity, enabling the grouping of chunks with comparable one-second buffering stall QoE ratings. During the second task, a random chunk from each group undergoes a combination of B bitrate levels and F buffering stall levels. QoE ratings for these BF videos then determine the weight. Though this approach requires fewer videos for assessment, it necessitates these sequential crowdsourcing tasks to prevent redundant weight refinement.

VidPlat, however, offers the opportunity to combine both tasks. With VidPlat, the refinement of a chunk's weight can start as soon as it's identified as requiring refinement, without waiting for the subsequent stage. The first step remains consistent for any given chunk: gather QoE ratings for a video with a 1-second buffering stall at the chunk (akin to the original first task). Once sufficient ratings are collected (*i.e.,* the standard error falls below a specified threshold), the average QoE rating is computed. If there are not any other chunks with comparable scores, refinement can begin immediately for that chunk, using the BF low-quality events.

4.4.2 Online video gaming

Online gaming's dynamic content arises from the interactions between players and game servers. This distinct characteristic places equal importance on video quality and interaction latency, *i.e.,* motion-to-photon (MTP) latency [24, 64]. Unlike on-demand videos, which can be downloaded over a longer timeframe, online gaming needs immediate decisions on changing video bitrate to download, especially during network congestion. Here, the challenge lies in striking the right balance between video quality and MTP latency to ensure optimal QoE [24].

Different game scenarios prioritize different aspects. For instance, in MOBA (Multiplayer Online Battle Arena) games, where players interact in real-time, the QoE emphasis may shift towards maintaining low latency. On the other hand, for SLGs (Simulated Life Games) which simulate real-life, high video quality might be more important. This highlights the need to craft QoE models specific to each game's scenario. Should network congestions occur, these models would guide decisions on the amount of redundancy (such as the forward error protection level [24, 49]) to add to video packets. This is to avoid packet retransmissions which increase MTP latency.

To establish a QoE model for any game scenario, the QoE ratings of videos spanning different bitrates (*e.g.,* from (D_{min}) to D_{max}) and MTP latencies (*e.g.,* ranging from L_{min} to L_{max}) needs to be collected. The most straightforward strategy is to sample combinations of bitrate and MTP latency by fixed intervals ΔD and ΔL, *i.e.,* $(D_{min}, L_{min}), (D_{min} + \Delta D, L_{min}), \cdots, (D_{min} + \Delta D, L_{min} + \Delta L), \cdots$. However, this approach comes with its challenges, especially when deciding on the ideal intervals, ΔD for bitrate and ΔL for MTP latency. If these intervals are too small, it would require rating an excessive number of videos. Conversely, larger intervals might not yield enough data for precise QoE models, necessitating additional crowdsourcing tasks for smaller intervals and thus increasing latency.

A proposed solution is a dynamic method of interval selection based on the ratings gathered so far. Initiating with larger intervals, the QoE ratings are first collected. If two neighboring samples (for instance, two video demos (D, L) and $(D, L + \Delta L)$) exhibit a QoE rating difference surpassing a pre-defined threshold α, the subsequent sampling task refines this range using smaller steps, $\Delta D/2$ and $\Delta L/2$. If we use traditional tools to realize this solution, we must conduct multiple tasks to collect QoE ratings, which increases latency. Because to minimize the sample number, in each task, we only sample the video demos with the same interval.

In contrast, VidPlat's capability allows for the simultaneous collection of QoE ratings of video demos across diverse intervals. When a significant QoE variance is found between two neighboring samples, VidPlat can immediately initiate the gathering of ratings for intermediary video quality levels. By continuously iterating this process, VidPlat can execute the user study in a single task, thereby reducing latencies introduced by extra tasks.

4.4.3 Web service

Webpage Load Time (PLT) is a crucial determinant of user QoE. It denotes the time span between initiating a web request and the rendering of user-noticeable web objects on the browser. Prior research indicates that the user's perception of PLT is non-linear. For example, while there's negligible perceptual difference if the PLTs are two long, the distinction between PLTs in certain ranges is quite noticeable [174]. Moreover, the correlation between PLT and QoE varies, contingent upon the content of the webpage and the sequence in which web objects load [55, 39]. These nuances necessitate frequent updates to webpage QoE models.

Constructing such models involves collecting QoE ratings across varying PLTs for specific web content and object-loading order. Once the ratings are acquired, QoE models are built correlating PLT values with corresponding QoE ratings. Since QoE typically decreases with increased PLT, if we have data points for PLTs a and b with QoE ratings of Q_a and Q_b, the QoE for an intermediate PLT value can be extrapolated using linear interpolation.

To ensure the accuracy of such extrapolations, it is vital that the QoE difference between two neighboring PLTs remains below a predetermined threshold, α. One simplistic approach to model QoE across a range of PLTs between c and d seconds involves sampling PLTs at fixed intervals δ (*i.e.,* collecting QoE ratings of PLTs $c, c+\delta, c+2\delta, \cdots, d$) and then soliciting user QoE ratings. Traditional tools necessitate deciding on this interval, δ, prior to initiating the task. However, the challenge arises in selecting an optimal δ. While a large δ might necessitate additional tasks due to insufficient PLTs, a smaller δ can result in surplus QoE collection and escalating costs.

VidPlat offers a more flexible approach, allowing developers to adjust δ dynamically during the task. Initially, PLTs are sampled using a large interval,η. If the collected QoE ratings reveal a QoE difference surpassing α between neighboring PLTs, the interval is refined, *i.e.,* collecting QoE ratings with a smaller interval in the range between the two neighboring PLTs. This iterative process continues until the QoE difference for all consecutive PLTs drops below α. Such a dynamic method, as facilitated by VidPlat, not only ensures accuracy but also optimizes task durations, as it accommodates varying intervals within a single task.

4.5 Discussion

Validating QoE measurements: Within VidPlat, the demos presented next are dictated by the assignment-generation logic, and this decision is based on the QoE measurements collected during the task. Although we demonstrate in §4.3.2 that VidPlat can utilize online techniques, such as posing control questions, to validate the QoE ratings of raters, a potential concern arises: the inability to feed the logic with QoE ratings that have undergone offline processing. Offline techniques—like adjusting each individual rater's average score to a predetermined value as cited in [43, 80, 106] — aim to neutralize biases in QoE ratings, ensuring consistent rating scales across raters. In VidPlat, we preemptively train raters to calibrate their rating scales, following the earlier practices in [70]. This training provides raters with guidelines on the appropriate rating scale. Our experimental results, as presented

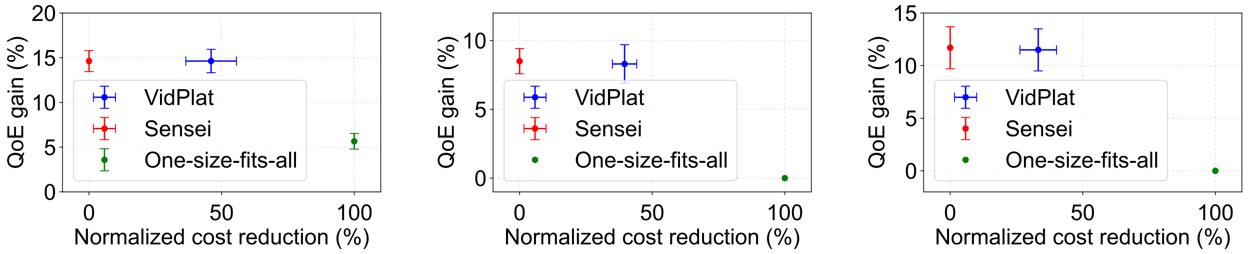


Figure 4.9: On-demand video streaming

Figure 4.10: Online gaming

Figure 4.11: Web server resource allocation

Figure 4.12: QoE gain vs Cost reduction

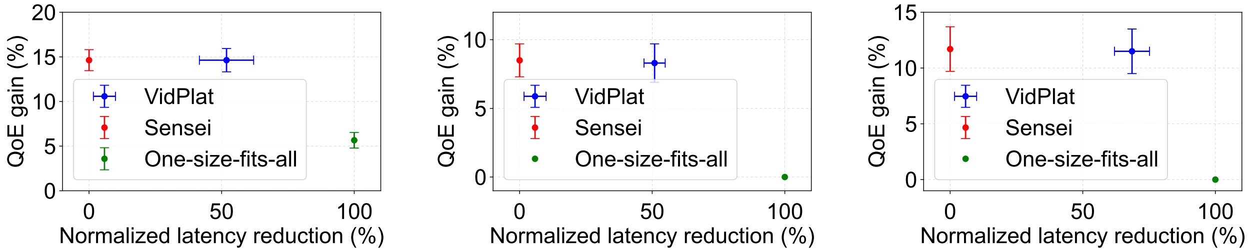


Figure 4.13: On-demand video streaming

Figure 4.14: Online gaming

Figure 4.15: Web server resource allocation

Figure 4.16: QoE gain vs Latency reduction

in §4.6, indicate that the omission of offline techniques does not significantly compromise the accuracy of QoE measurements.

Overhead of demo generation: Different from conventional tools, VidPlat dynamically generates demos for raters during the user study, as opposed to doing so in advance. Consequently, the time taken for demo generation must be factored into the overall latency. This generation time encompasses the video compression duration (essentially, the period required to generate videos with low-quality events) and the execution time of the demo-generation logic. However, the time taken for demo generation can be amortized as more raters evaluate a given demo. If a rater finishes rating a demo and VidPlat already has other generated demos ready for rating, the rater does not need to wait for new demos to be generated by the logic. Nonetheless, in extreme scenarios where video generation time becomes the primary source of latency, VidPlat might struggle to mitigate the overall delay in the user study.

4.6 Evaluation

We evaluate VidPlat in building QoE models for the three use cases in §4.4. Our key finding is that VidPlat can save 50.9% — 68.8% latency and 31.8% — 46.0% QoE ratings that need to be collected (indicating the same amount of monetary cost saving) without degrading QoE by applying the QoE models trained by VidPlat.

4.6.1 Experimental setup

We evaluate VidPlat by real experiments on Amazon Mechanical Turk followed by trace-driven simulation in which the traces are collected from our real experiments.

Use case setup: We list the experimental settings of the three use cases in §4.4.

- **On-demand video streaming:** We build per-video QoE models for the source videos in the dataset [172], and the dataset has 16 source videos across different genres. By the per-video QoE models, we use a video streaming system, Pensieve [117], to stream those videos over different networks whose throughputs are randomly selected from two public network traces, FCC [52] and 3G/HSDPA [135] as in [172] by the per-video QoE models and the one-size-fits-all models, KSQI [60].
- **Online video gaming:** We build QoE models for the videos in the online gaming dataset [32], and this dataset has 10 games including Multiplayer Online Battle Arena (MOBA), First-person Shooter (FPS) and Massive Multiplayer Online Role-Playing Game (MMORPG) game types. We use Nebula [24] to stream the videos frame by frame over the network traces in [172] by per-video QoE models described in §4.4 and a one-size-fits-all model in [24].
- **Resource allocation on web servers:** We train per-page QoE models for three types of webpages from an open-source dataset from Microsoft [174]. We apply those per-page QoE models and a one-size-fits-all QoE model, Speed Index [83], to web service backend systems for allocating computing resources as the settings in [174].

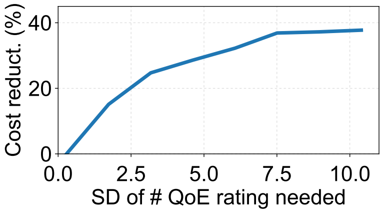


Figure 4.17: Cost reduction vs Standard deviation in the number of required QoE ratings across web pages with different page load times.

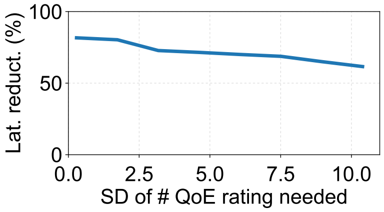


Figure 4.18: Latency reduction vs Variance in the number of required QoE ratings

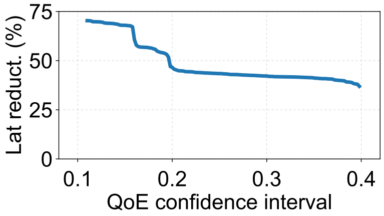


Figure 4.19: QoE confidence interval vs Latency reduction

Crowdsourcing setup: In this evaluation, we conduct user studies on Amazon Mechanical Turk. The crowdsourcing raters are identified as "Master Turkers", which means their historical response acceptance rate is $> 99\%$. We use a fixed payment structure for all the workers. The rater payment is \$10 per hour, which aligns with the lawful minimal wage.

Metrics: We use three metrics. Given a network trace and a source content (video or web page), the *QoE gain* is measured by $\frac{QoE_f - QoE_b}{QoE_b}$, where QoE_f is the QoE by using fine-granular QoE models built by Sensei and VidPlat and QoE_b is the QoE by using one-size-fits-all models. The QoE of those application demos is measured by average ratings (*i.e.,* Mean opinion score, MOS [133]) collected from workers. We also calculate the *latency* and *cost reductions* in building fine-granular QoE models compared with Sensei, *i.e.,* $\frac{L_{Sensei} - L_{VidPlat}}{L_{Sensei}}$ and $\frac{C_{Sensei} - C_{VidPlat}}{C_{VidPlat}}$, where $L_{Sensei}, L_{VidPlat}, C_{Sensei}, C_{VidPlat}$ represent latency and cost of training QoE models by Sensei and VidPlat. Latency is measured by the length of the timespan of the user study. Cost is measured by the number of responses we need to collect from the workers, since we use a fixed payment structure, it can be regarded as a monetary cost in our evaluation.

4.6.2 End-to-end evaluation

We evaluate the cost and latency reductions in building QoE models compared with Sensei. We can see that we can save 31.8% — 46.0% cost and 50.9% — 68.8% latency compared with Sensei as shown in Figures 4.12 and 4.16. For the cost reduction, VidPlat and Sensei show the same number of application demos, so the cost reduction is from the savings in the number of responses we need to collect for the demos, because VidPlat can immediately stop QoE rating collections as we have collected enough ratings. For latency, we can see online gaming and web service have more reductions than on-demand video streaming. It is because in online gaming and web services, we conduct more sequential tasks than on-demand video streaming which only has two tasks as in [172]. The cost and latency reductions of VidPlat are not at a cost of lower QoE improvement. From Figures 4.12 and 4.16, the QoE models built by Sensei and VidPlat have similar QoE improvement over one-size-fits-all QoE models. It is because that we have built QoE models with enough accuracy confirmed by the assignment generator provided by requesters.

4.6.3 In-depth analysis

We use trace-driven simulation to study where VidPlat's improvement comes from. We simulate the user study of using crowdsourcing to build QoE models for webpages. The pattern of signing-up time of workers is from the traces we collected from end-to-end evaluation.

Variance in QoE ratings needed for every application demo: We investigate the cost and latency improvement of VidPlat over Sensei, when the variance of number of ratings needed for webpage load events changes. The number of ratings needed for each page load event is from a uniform distribution in the range $[a, b]$ where we fix a as 10 and change b from 10 to 50. Figure 4.17 shows the cost reduction increase with the variance of number of ratings needed for webpage load events. In Sensei, we use each page load event is rated by b times for avoiding an event presenting in two tasks which increases latency. Thus, the cost reduction will finally converge to $1 - \frac{(a+b)/2}{b}$ where $(a + b)/2$ is the average QoE ratings needed of page load events in VidPlat under this setting, and b is the average QoE ratings needed of page load events in Sensei under this setting. Figure 4.18 shows the latency reduction with the variance. The reduction drops slowly with the variance since VidPlat can dynamically recruit more workers to fully parallel QoE rating collections instead of recruiting a fixed number of raters using traditional tools.

QoE model accuracy: Figure 4.19 shows the latency reduction with 90%-confidence intervals of QoE of the webpage load events. The latency reduction decreases with larger confidence interval as a “step” shape. The shape is because that if we want to use Sensei to achieve a larger confidence level, we might use fewer sequential tasks, which significantly reduces Sensei’s latency and thus lowers our latency reduction.

4.7 Related Work

We briefly survey the most relevant work in QoE modeling and crowdsourcing user study.

QoE of Internet Applications: Quality of Experience (QoE) reflects the degree of user satisfaction concerning the perceived application quality. QoE is evaluated in two ways: subjective and objective. Subjective measurement directly collects user feedback about perceived application quality, typically using subjective assessment scores such as the Mean Opinion Score (MOS) which averages scores collected from users. To ensure these scores are valid and repeatable, common practices [81, 80, 36] are proposed. These works include detailed descriptions of QoE measurement settings [36], testing environments [179], data processing [106], and more.

Objective measurements build models to predict QoE based on perceived application quality such as buffering stalls and bitrate in video streaming, or page load time in web services. Apart from perceived application quality, several other factors significantly influence user QoE, like application content [172, 55, 42], user context (*e.g.,* user location [82], end-user environment [71]), and human factors (*e.g.,* user quality preference [89, 173], emotional status [179]). Consequently, users might experience different QoE levels for the same perceived application quality due to their unique QoE influencing factors.

Prior work leverages these QoE influencing factors to optimize Internet applications.

Examples include:

- Web services: Metrics such as Above-The-Fold [83] (time spent until objects in the user viewport are fully loaded) and Eyetrack [96] (monitoring motion of users' eyeballs to determine the webpage area) reflect more accurately the subjective page load time. Similarly, algorithms that consider the order of loading web objects have been proposed [39].
- Video streaming: Early systems [168, 117] use a one-size-fits-all model to optimize QoE. More recent optimizations consider additional dimensions of QoE influencing factors. For instance, Sensei [172] customizes the bitrate selection strategy according to the video content, and other studies [89, 182] account for the heterogeneity of quality preferences across users to personalize video streaming strategy.

User Study: QoE optimization success highly depends on user studies which directly measure QoE with real human users. Traditional lab-based studies are costly and time-consuming, because the user-study participants must physically present in the place for user study (*e.g.,* a research lab). Therefore, crowdsourcing has gained popularity due to its flexibility in recruiting participants and task allocation. Nevertheless, crowdsourcing still incurs ineligible costs and requires significant time. Several techniques have been proposed to reduce these, including:

- Cost-saving: Active learning [43, 111] is used to minimize the number of data samples requiring human annotations for training machine-learning models. Domain-specific knowledge is also leveraged to prune the samples whose human annotations can be inferred by the annotations that have already been collected.
- Saving latency: Techniques to reduce crowdsourcing latency focus on user recruitment and per-task worker latency. Dynamic pricing [151] and maintaining workers' speed by phasing out slow workers are some of the strategies used [76]. Furthermore, overlapping human annotation data processing and human annotation data collection for the applications can reduce latency [86].

Compared to lab-based user studies, the inevitable problem is that crowdsourcing may collect lower-quality responses (*e.g.,* random responses) due to malicious workers who want to get their payment faster or varying degrees of expertise among workers. Therefore, methods for collecting high-quality responses have been developed, such as modeling each user's response quality [43, 106], implementing a qualification questionnaire [179], or using golden tasks [157, 172]. Given the methods, crowdsourcing can eliminate workers unable to offer high-quality responses.

A number of tools are proposed to assist crowdsourcing-based user study by integrating cost and latency techniques. Quadrant of Euphoria and Eyeorg [46, 47, 157] provides a web-based platform that automatically recruits crowd workers and assigns tasks. It takes a set of videos or audio with different quality as input, and then outputs their QoE ranking. Sensei is a platform that takes videos as input and outputs reliable QoE ratings. Other tools passively monitor QoEs [41, 120, 173] on platforms like YouTube, including video quality and user engagement time, and then process the user study data offline. Its passive manner does not involve extra human efforts except installing those monitors in the beginning, making it collect a large amount of user data with a low cost.

CHAPTER 5
CONCLUSION

5.1 Contributions

This thesis contributes by proposing a new user-centric approach to optimizing the QoE of Internet applications.

The key challenges in user-centric QoE optimizations are 1) high cost of QoE measurements of different application contexts and 2) large system overhead of applying fine-grained optimizations by leveraging the per-context QoE models.

To solve the above challenges, we have proposed three systems, VidPlat, E2E and Sensei. VidPlat enables dynamic application demo pruning to complete in a single user study task, reducing 31.8% – 46.0% monetary cost and 50.9% – 68.8% latency of QoE measurements for web services, on-demand video streaming and online video gaming. E2E can efficiently allocate web servers to the web requests with different server-side delay sensitivity, improving QoE of web services by 28%. Sensei first trains per-video-content QoE models in a cost-efficient way and then allocates network resources to download video segments by their QoE sensitivity. Sensei can improve 15.1% QoE without using more network bandwidth, and maintain the same QoE with 26.8% less bandwidth.

5.2 Future work

Three immediate directions in user-centric QoE optimizations can be explored.

Joint optimization between application and human perception: Every service provider (*e.g.,* Internet service providers, content providers, etc.) that supports Internet applications should integrate human perception into its optimization. Though previous QoE-optimization work can set service-level objectives (SLOs) for each system provider and application session to maximize QoE, it ignores the fact that a user's QoE preference might change within a session and some service providers might fail to achieve their original SLOs. My research will explore a logically centralized controller that takes real-time QoE preferences and system performance as input, and uses them to set and change SLOs for service providers at any time during a session. The controller should also incur low system overhead so as not to become a performance bottleneck. Intuitively, these service providers have an incentive to cooperate with each other since such a controller can help improve user QoE without additional system resources.

Performance fairness and privacy protection in QoE-driven systems: It is suboptimal to serve each user with equal system-level performance due to their heterogeneous sensitivity to system-level performance. However, this raises a fairness issue because we might always prioritize some users over others to achieve higher overall QoE across all users, because some users might have more marginal QoE improvement when given the same amount of extra system resources than other users. Moreover, to build QoE models, application providers need user-perception data that might be regarded as private by users and cannot be shared. Thus, our goals are to 1) disconnect user-perception data from specific users in order to protect their privacy, and 2) maintain QoE fairness across the users of an application. To achieve these goals, my research will integrate emerging techniques, such as federated learning and differential privacy, to keep user-perception data locally and train QoE models while preserving user anonymity, in QoE-driven systems.

Expanding user interfaces for QoE-driven systems: To collect user-perception data, previous work relied on user studies to directly ask users for their perceived QoE (*e.g.,* user ratings in exit surveys) of Internet applications and system-level performance (*e.g.,* page load time and video resolution). However, this is not sufficient, because QoE is also strongly correlated to users' physical and environmental status. For example, a user would not be sensitive to video resolution when walking or jogging, and the user's status can be detected through vital signs like heartbeat and respiration. I will conduct systematic research about what additional data we need to collect from users for QoE optimization.

BIBLIOGRAPHY

[1] Amazon Mechanical Turk. https://www.mturk.com/.

[2] Apache Cassandra. https://cassandra.apache.org.

[3] Cisco Annual Internet Report (2018–2023) White Paper. https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html/.

[4] DASH.js. https://github.com/Dash-Industry-Forum/dash.js/wiki.

[5] How Netflix Pays for Movie and TV Show Licensing. https://www.investopedia.com/articles/investing/062515/how-netflix-pays-movie-and-tv-show-licensing.asp.

[6] Live stream on YouTube, See your live stream's metrics. https://support.google.com/youtube/answer/2853833.

[7] Media Source Extensions. https://www.w3.org/TR/media-source/.

[8] Microworkers. https://www.microworkers.com/.

[9] MongoDB. https://www.mongodb.com/.

[10] RabbitMQ. https://www.rabbitmq.com/.

[11] Reuters: YouTube, Amazon Prime forgo streaming quality to relieve European networks. https://uk.reuters.com/article/us-health-coronavirus-youtube-exclusive/exclusive-youtube-to-reduce-streaming-quality-in-europe-due-to-coronavirus-idUKKBN2170OP.

[12] Video Quality of Experience: Requirements and Considerations for Meaningful Insight. https://www.sandvine.com/hubfs/downloads/archive/whitepaper-video-quality-of-experience.pdf.

[13] VMAF - Video Multi-Method Assessment Fusion. https://github.com/Netflix/vmaf.

[14] YouTube joins Netflix in reducing video quality in Europe. https://www.theverge.com/2020/3/20/21187930/youtube-reduces-streaming-quality-european-union-coronavirus-bandwidth-internet-traffic.

[15] Marissa Mayer at Web 2.0. http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html, 2006.

[16] Akamai online retail performance report: Milliseconds are critical. https://www.akamai.com/uk/en/about/news/press/2017-press/akamai-releases-spring-2017-state-of-online-retail-performance-report.jsp, 2017.

[17] Find out how you stack up to new industry benchmarks for mobile page speed. https://www.thinkwithgoogle.com/marketing-resources/data-measurement/mobile-page-speed-new-industry-benchmarks/, 2018.

[18] Golnaz Abdollahian, Cuneyt M Taskiran, Zygmunt Pizlo, and Edward J Delp. Camera motion-based analysis of user generated video. *IEEE Transactions on Multimedia*, 12(1):28–41, 2009.

[19] Victor Agababov, Michael Buettner, Victor Chudnovsky, Mark Cogan, Ben Greenstein, Shane McDaniel, Michael Piatek, Colin Scott, Matt Welsh, and Bolian Yin. Flywheel: Google's data compression proxy for the mobile web. In *NSDI*, volume 15, pages 367–380, 2015.

[20] Vaneet Aggarwal, Emir Halepovic, Jeffrey Pang, Shobha Venkataraman, and He Yan. Prometheus: Toward quality-of-experience estimation for mobile apps from passive network measurements. In *Proceedings of the 15th Workshop on Mobile Computing Systems and Applications*, page 18, 2014.

[21] Hasti Ahlehagh and Sujit Dey. Adaptive bit rate capable video caching and scheduling. In *2013 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1357–1362. IEEE, 2013.

[22] Hasti Ahlehagh and Sujit Dey. Video-aware scheduling and caching in the radio access network. *IEEE/ACM Transactions on Networking*, 22(5):1444–1462, 2014.

[23] 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 *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 44–58, 2018.

[24] Ahmad Alhilal, Tristan Braud, Bo Han, and Pan Hui. Nebula: Reliable low-latency video transmission for mobile cloud gaming. In *Proceedings of the ACM Web Conference 2022*, pages 3407–3417, 2022.

[25] Amazon. Amazon mechanical turk. https://www.mturk.com/, 2022.

[26] Ramon Aparicio-Pardo, Karine Pires, Alberto Blanc, and Gwendal Simon. Transcoding live adaptive video streams at a massive scale in the cloud. In *Proceedings of the 6th ACM Multimedia Systems Conference*, pages 49–60, 2015.

[27] Athula Balachandran, Vaneet Aggarwal, Emir Halepovic, Jeffrey Pang, Srinivasan Seshan, Shobha Venkataraman, and He Yan. Modeling web quality-of-experience on cellular networks. In *Proceedings of the 20th annual international conference on Mobile computing and networking*, pages 213–224, 2014.

[28] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. Developing a predictive model of quality of experience for internet video. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 339–350. ACM, 2013.

[29] Christos G Bampis and Alan C Bovik. An augmented autoregressive approach to http video stream quality prediction. *arXiv preprint arXiv:1707.02709*, 2017.

[30] Christos G Bampis, Zhi Li, and Alan C Bovik. Continuous prediction of streaming video qoe using dynamic networks. *IEEE Signal Processing Letters*, 24(7):1083–1087, 2017.

[31] Christos G Bampis, Zhi Li, Ioannis Katsavounidis, Te-Yuan Huang, Chaitanya Ekanadham, and Alan C Bovik. Towards perceptually optimized end-to-end adaptive video streaming. *arXiv preprint arXiv:1808.03898*, 2018.

[32] Nabajeet Barman, Emmanuel Jammeh, Seyed Ali Ghorashi, and Maria G Martini. No-reference video quality estimation based on machine learning for passive gaming video streaming applications. *IEEE Access*, 7:74511–74527, 2019.

[33] Daniel S Berger, Benjamin Berg, Timothy Zhu, Siddhartha Sen, and Mor Harchol-Balter. Robinhood: Tail latency aware caching–dynamic reallocation from cache-rich to cache-poor. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 195–212, 2018.

[34] Daniel S Berger, Ramesh K Sitaraman, and Mor Harchol-Balter. Adaptsize: Orchestrating the hot object memory cache in a content delivery network. In *NSDI*, pages 483–498, 2017.

[35] Enrico Bocchi, Luca De Cicco, and Dario Rossi. Measuring the quality of experience of web users. *ACM SIGCOMM Computer Communication Review*, 46(4):8–13, 2016.

[36] Kjell Brunnström, Sergio Ariel Beker, Katrien De Moor, Ann Dooms, Sebastian Egger, Marie-Neige Garcia, Tobias Hossfeld, Satu Jumisko-Pyykkö, Christian Keimel, Mohamed-Chaker Larabi, et al. Qualinet white paper on definitions of quality of experience. 2013.

[37] Mike Burrows. The Chubby lock service for loosely-coupled distributed systems. In *OSDI*, 2006.

[38] Michael Butkiewicz, Daimeng Wang, Zhe Wu, Harsha V Madhyastha, and Vyas Sekar. Klotski: Reprioritizing web content to improve user experience on mobile devices. In *NSDI*, volume 1, pages 2–3, 2015.

[39] Michael Butkiewicz, Daimeng Wang, Zhe Wu, Harsha V Madhyastha, and Vyas Sekar. Klotski: Reprioritizing web content to improve user experience on mobile devices. In *12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15)*, pages 439–453, 2015.

[40] Matt Calder, Manuel Schröder, Ryan Stewart Ryan Gao, Jitendra Padhye, Ratul Mahajan, Ganesh Ananthanarayanan, and Ethan Katz-Bassett. Odin: Microsoft's scalable fault-tolerant CDN measurement system. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, 2018.

[41] Pedro Casas, Michael Seufert, and Raimund Schatz. Youqmon: A system for on-line monitoring of youtube qoe in operational 3g networks. *ACM SIGMETRICS Performance Evaluation Review*, 41(2):44–46, 2013.

[42] Pedro Casas, Sarah Wassermann, Nikolas Wehner, Michael Seufert, and Tobias Hossfeld. Not all web pages are born the same content tailored learning for web qoe inference. In *2022 IEEE International Symposium on Measurements & Networking (M&N)*, pages 1–6. IEEE, 2022.

[43] Haw-Shiuan Chang, Chih-Fan Hsu, Tobias Hoßfeld, and Kuan-Ta Chen. Active learning for crowdsourced qoe modeling. *IEEE Transactions on Multimedia*, 20(12):3337–3352, 2018.

[44] Chao Chen, Lark Kwon Choi, Gustavo De Veciana, Constantine Caramanis, Robert W Heath, and Alan C Bovik. Modeling the time—varying subjective quality of http video streams with rate adaptations. *IEEE Transactions on Image Processing*, 23(5):2206–2221, 2014.

[45] Fangfei Chen, Ramesh K Sitaraman, and Marcelo Torres. End-user mapping: Next generation request routing for content delivery. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 167–181. ACM, 2015.

[46] Kuan-Ta Chen, Chi-Jui Chang, Chen-Chi Wu, Yu-Chun Chang, and Chin-Laung Lei. Quadrant of euphoria: a crowdsourcing platform for qoe assessment. *IEEE Network*, 24(2):28–35, 2010.

[47] Kuan-Ta Chen, Chen-Chi Wu, Yu-Chun Chang, and Chin-Laung Lei. A crowdsourceable qoe evaluation framework for multimedia content. In *Proceedings of the 17th ACM international conference on Multimedia*, pages 491–500, 2009.

[48] Yingying Chen, Ratul Mahajan, Baskar Sridharan, and Zhi-Li Zhang. A provider-side view of web search response time. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 243–254, 2013.

[49] Yihua Cheng, Anton Arapin, Ziyi Zhang, Qizheng Zhang, Hanchen Li, Nick Feamster, and Junchen Jiang. Grace++: Loss-resilient real-time video communication under high network latency. *arXiv preprint arXiv:2305.12333*, 2023.

[50] Michael Chow, David Meisner, Jason Flinn, Daniel Peek, and Thomas F Wenisch. The mystery machine: End-to-end performance analysis of large-scale internet services. In *OSDI*, pages 217–231, 2014.

[51] Michael Chow, Kaushik Veeraraghavan, Michael J Cafarella, and Jason Flinn. DQBarge: Improving data-quality tradeoffs in large-scale internet services. In *OSDI*, pages 771–786, 2016.

[52] Federal Communications Commission. Raw datameasuring broadband america. `https://www.fcc.gov/reportsresearch/reports`, 2016.

[53] Federal Communications Commission et al. Raw data-measuring broadband america. https://www.fcc.gov/reports-research/reports/. *Retrieved June*, 19:2018, 2016.

[54] Jingjing Cui, Yuanwei Liu, Zhiguo Ding, Pingzhi Fan, and Arumugam Nallanathan. Qoe-based resource allocation for multi-cell noma networks. *IEEE Transactions on Wireless Communications*, 17(9):6160–6176, 2018.

[55] Diego Neves da Hora, Alemnew Sheferaw Asrese, Vassilis Christophides, Renata Teixeira, and Dario Rossi. Narrowing the gap between QoS metrics and web QoE using above-the-fold metrics. In *International Conference on Passive and Active Network Measurement*, pages 31–43. Springer, 2018.

[56] Johan De Vriendt, Danny De Vleeschauwer, and David Robinson. Model for estimating qoe of video delivered using http adaptive streaming. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 1288–1293. IEEE, 2013.

[57] Djellel Difallah, Elena Filatova, and Panos Ipeirotis. Demographics and dynamics of mechanical turk workers. In *Proceedings of the eleventh ACM international conference on web search and data mining*, pages 135–143, 2018.

[58] Li Ding, Hua Huang, and Yu Zang. Image quality assessment using directional anisotropy structure measurement. *IEEE Transactions on Image Processing*, 26(4):1799–1809, 2017.

[59] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. Understanding the impact of video quality on user engagement. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 362–373, 2011.

[60] Zhengfang Duanmu, Wentao Liu, Diqi Chen, Zhuoran Li, Zhou Wang, Yizhou Wang, and Wen Gao. A knowledge-driven quality-of-experience model for adaptive streaming videos. *arXiv preprint arXiv:1911.07944*, 2019.

[61] Zhengfang Duanmu, Abdul Rehman, and Zhou Wang. A quality-of-experience database for adaptive video streaming. *IEEE Transactions on Broadcasting*, 64(2):474–487, 2018.

[62] Zhengfang Duanmu, Kai Zeng, Kede Ma, Abdul Rehman, and Zhou Wang. A quality-of-experience index for streaming video. *IEEE Journal of Selected Topics in Signal Processing*, 11(1):154–166, 2016.

[63] Sebastian Egger, Bruno Gardlo, Michael Seufert, and Raimund Schatz. The impact of adaptation strategies on perceived quality of http adaptive streaming. In *Proceedings of the 2014 Workshop on Design, Quality and Deployment of Adaptive Video Streaming*, pages 31–36, 2014.

[64] Mohammed S Elbamby, Cristina Perfecto, Mehdi Bennis, and Klaus Doppler. Toward low-latency and ultra-reliable virtual reality. *IEEE Network*, 32(2):78–84, 2018.

[65] Nagabhushan Eswara, S Ashique, Anand Panchbhai, Soumen Chakraborty, Hemanth P Sethuram, Kiran Kuchi, Abhinav Kumar, and Sumohana S Channappayya. Streaming video qoe modeling and prediction: A long short-term memory approach. *IEEE Transactions on Circuits and Systems for Video Technology*, 2019.

[66] Michael L Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

[67] Guanyu Gao, Linsen Dong, Huaizheng Zhang, Yonggang Wen, and Wenjun Zeng. Content-aware personalised rate adaptation for adaptive streaming via deep video analysis. In *ICC 2019-2019 IEEE International Conference on Communications (ICC)*, pages 1–8. IEEE, 2019.

[68] Guanyu Gao, Huaizheng Zhang, Han Hu, Yonggang Wen, Jianfei Cai, Chong Luo, and Wenjun Zeng. Optimizing quality of experience for adaptive bitrate streaming via viewer interest inference. *IEEE Transactions on Multimedia*, 20(12):3399–3413, 2018.

[69] Qingzhu Gao, Prasenjit Dey, and Parvez Ahammad. Perceived performance of top retail webpages in the wild: Insights from large-scale crowdsourcing of above-the-fold QoE. In *Proceedings of the Workshop on QoE-based Analysis and Management of Data Communication Networks*, pages 13–18, 2017.

[70] Bruno Gardlo, Sebastian Egger, and Tobias Hossfeld. Do scale-design and training matter for video qoe assessments through crowdsourcing? In *Proceedings of the Fourth International Workshop on Crowdsourcing for Multimedia*, pages 15–20, 2015.

[71] Bruno Gardlo, Michal Ries, and Tobias Hossfeld. Impact of screening technique on crowdsourcing qoe assessments. In *Proceedings of 22nd International Conference Radioelektronika 2012*, pages 1–4. IEEE, 2012.

[72] Deepti Ghadiyaram, Janice Pan, and Alan C Bovik. A subjective and objective study of stalling events in mobile streaming videos. *IEEE Transactions on Circuits and Systems for Video Technology*, 29(1):183–197, 2017.

[73] Deepti Ghadiyaram, Janice Pan, and Alan C Bovik. Learning a continuous-time streaming video qoe model. *IEEE Transactions on Image Processing*, 27(5):2257–2271, 2018.

[74] Rafael C Gonzalez, Richard Eugene Woods, and Steven L Eddins. *Digital image processing using MATLAB*. Pearson Education India, 2004.

[75] Michael Gygli, Yale Song, and Liangliang Cao. Video2gif: Automatic generation of animated gifs from video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1001–1009, 2016.

[76] Daniel Haas, Jiannan Wang, Eugene Wu, and Michael J Franklin. Clamshell: Speeding up crowds for low-latency data labeling. *Proceedings of the VLDB Endowment*, 9(4), 2015.

[77] Mingzhe Hao, Huaicheng Li, Michael Hao Tong, Chrisma Pakha, Riza O Suminto, Cesar A Stuardo, Andrew A Chien, and Haryadi S Gunawi. MittOS: Supporting millisecond tail tolerance with fast rejecting SLO-aware OS interface. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 168–183. ACM, 2017.

[78] Kotaro Hara, Abigail Adams, Kristy Milland, Saiph Savage, Chris Callison-Burch, and Jeffrey P Bigham. A data-driven analysis of workers' earnings on amazon mechanical turk. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pages 1–14, 2018.

[79] Tobias Hoßfeld, Sebastian Biedermann, Raimund Schatz, Alexander Platzer, Sebastian Egger, and Markus Fiedler. The memory effect and its implications on web qoe modeling. In *2011 23rd international teletraffic congress (ITC)*, pages 103–110. IEEE, 2011.

[80] Tobias Hoßfeld, Matthias Hirth, Judith Redi, Filippo Mazza, Pavel Korshunov, Babak Naderi, Michael Seufert, Bruno Gardlo, Sebastian Egger, and Christian Keimel. Best practices and recommendations for crowdsourced qoe-lessons learned from the qualinet task force" crowdsourcing". 2014.

[81] Tobias Hossfeld, Christian Keimel, Matthias Hirth, Bruno Gardlo, Julian Habigt, Klaus Diepold, and Phuoc Tran-Gia. Best practices for qoe crowdtesting: Qoe assessment with crowdsourcing. *IEEE Transactions on Multimedia*, 16(2):541–558, 2013.

[82] Tobias Hossfeld, Christian Keimel, and Christian Timmerer. Crowdsourcing quality-of-experience assessments. *Computer*, 47(9):98–102, 2014.

[83] Tobias Hoßfeld, Florian Metzger, and Dario Rossi. Speed index: Relating the industrial standard for user perceived web performance to web qoe. In *2018 Tenth International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1–6. IEEE, 2018.

[84] Tobias Hoßfeld, Michael Seufert, Matthias Hirth, Thomas Zinner, Phuoc Tran-Gia, and Raimund Schatz. Quantification of youtube qoe via crowdsourcing. In *2011 IEEE International Symposium on Multimedia*, pages 494–499. IEEE, 2011.

[85] Sudeng Hu, Lina Jin, Hanli Wang, Yun Zhang, Sam Kwong, and C-C Jay Kuo. Compressed image quality metric based on perceptually weighted distortion. *IEEE Transactions on Image Processing*, 24(12):5594–5608, 2015.

[86] Sheng-Jun Huang, Chen-Chen Zong, Kun-Peng Ning, and Hai-Bo Ye. Asynchronous active learning with distributed label querying. In *IJCAI*, pages 2570–2576, 2021.

[87] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 187–198, 2014.

[88] Patrick Hunt, Mahadev Konar, Flavio Paiva Junqueira, and Benjamin Reed. Zookeeper: Wait-free coordination for internet-scale systems. In *USENIX ATC*, 2010.

[89] L. Huo, Z. Wang, M. Xu, Y. Li, Z. Ding, and H. Wang. A meta-learning framework for learning multi-user preferences in QoE optimization of DASH. *IEEE Transactions on Circuits and Systems for Video Technology*, pages 1–1, 2020, To Appear.

[90] Television Business International. The where & why of netflix's original content spend, 2023.

[91] P ITU-T RECOMMENDATION. Subjective video quality assessment methods for multimedia applications. *International telecommunication union*, 1999.

[92] Virajith Jalaparti, Peter Bodik, Srikanth Kandula, Ishai Menache, Mikhail Rybalkin, and Chenyu Yan. Speeding up distributed request-response workflows. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 219–230, 2013.

[93] Junchen Jiang, Xi Liu, Vyas Sekar, Ion Stoica, and Hui Zhang. Eona: Experience-oriented network architecture. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, page 11. ACM, 2014.

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

[95] Roy Jonker and Anton Volgenant. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing*, 38(4):325–340, 1987.

[96] Conor Kelton, Jihoon Ryoo, Aruna Balasubramanian, and Samir R Das. Improving user perceived page load times using gaze. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 545–559, 2017.

[97] Jaehong Kim, Youngmok Jung, Hyunho Yeo, Juncheol Ye, and Dongsu Han. Neural-enhanced live streaming: Improving live video ingest via online learning. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 107–125, 2020.

[98] Woojae Kim, Jongyoo Kim, Sewoong Ahn, Jinwoo Kim, and Sanghoon Lee. Deep video quality assessor: From spatio-temporal visual sensitivity to a convolutional neural aggregation network. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 219–234, 2018.

[99] Rupa Krishnan, Harsha V Madhyastha, Sridhar Srinivasan, Sushant Jain, Arvind Krishnamurthy, Thomas Anderson, and Jie Gao. Moving beyond end-to-end path information to optimize CDN performance. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, pages 190–201. ACM, 2009.

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

[101] Gautam Kumar, Ganesh Ananthanarayanan, Sylvia Ratnasamy, and Ion Stoica. Hold'em or fold'em?: Aggregation queries under performance variations. In *Proceedings of the Eleventh European Conference on Computer Systems*, page 7, 2016.

[102] Stephan J Lemmer, Jean Y Song, and Jason J Corso. Crowdsourcing more effective initializations for single-target trackers through automatic re-querying. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2021.

[103] Peng Li, Binoy Ravindran, and E Douglas Jensen. Utility accrual resource access protocols with assured timeliness behavior for real-time embedded systems. In *ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems*, 2005.

[104] Peng Li, Haisang Wu, Binoy Ravindran, and E Douglas Jensen. A utility accrual scheduling algorithm for real-time activities with mutual exclusion resource constraints. *IEEE Transactions on Computers*, 55(4):454–469, 2006.

[105] Z. Li, A. Aaron, L. Katsavounidis, A. Moorthy, and M. Manohara. Toward a practical perceptual video quality metric. 2016.

[106] Zhi Li and Christos G Bampis. Recover subjective quality scores from noisy measurements. In *2017 Data compression conference (DCC)*, pages 52–61. IEEE, 2017.

[107] Zhi Li, Xiaoqing Zhu, Joshua Gahm, Rong Pan, Hao Hu, Ali C Begen, and David Oran. Probe and adapt: Rate adaptation for http video streaming at scale. *IEEE Journal on Selected Areas in Communications*, 32(4):719–733, 2014.

[108] Jan Lievens, Shahid M Satti, Nikos Deligiannis, Peter Schelkens, and Adrian Munteanu. Optimized segmentation of h. 264/avc video for http adaptive streaming. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 1312–1317. IEEE, 2013.

[109] Hongqiang Harry Liu, Raajay Viswanathan, Matt Calder, Aditya Akella, Ratul Mahajan, Jitendra Padhye, and Ming Zhang. Efficiently delivering online services over integrated infrastructure. In *NSDI*, volume 1, page 1, 2016.

[110] Xi Liu, Florin Dobrian, Henry Milner, Junchen Jiang, Vyas Sekar, Ion Stoica, and Hui Zhang. A case for a coordinated internet video control plane. *ACM SIGCOMM Computer Communication Review*, 42(4):359–370, 2012.

[111] Xiaochen Liu, Wei Song, Qi He, Mario Di Mauro, and Antonio Liotta. Speeding up subjective video quality assessment via hybrid active learning. *IEEE Transactions on Broadcasting*, 69(1):165–178, 2022.

[112] Yao Liu, Sujit Dey, Fatih Ulupinar, Michael Luby, and Yinian Mao. Deriving and validating user experience model for dash video streaming. *IEEE Transactions on Broadcasting*, 61(4):651–665, 2015.

[113] Matt Lovett, Saleh Bajaba, Myra Lovett, and Marcia J Simmering. Data quality from crowdsourced surveys: A mixed method inquiry into perceptions of amazon's mechanical turk masters. *Applied Psychology*, 67(2):339–366, 2018.

[114] Guo Lu, Wanli Ouyang, Dong Xu, Xiaoyun Zhang, Chunlei Cai, and Zhiyong Gao. Dvc: An end-to-end deep video compression framework. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11006–11015, 2019.

[115] Jianguo Lu and Dingding Li. Estimating deep web data source size by capture–recapture method. *Information retrieval*, 13(1):70–95, 2010.

[116] Christopher D Manning, Christopher D Manning, and Hinrich Schütze. *Foundations of statistical natural language processing*. MIT Press, 1999.

[117] H. Mao, R. Netravali, and M. Alizadeh. Neural adaptive video streaming with Pensieve. In *Proc. ACM SIGCOMM*, pages 197–210, Los Angeles, CA, USA, Aug. 2017. ACM.

[118] Florian Metzger, Stefan Geißler, Alexej Grigorjew, Frank Loh, Christian Moldovan, Michael Seufert, and Tobias Hoßfeld. An introduction to online video game qos and qoe influencing factors. *IEEE Communications Surveys & Tutorials*, 24(3):1894–1925, 2022.

[119] M Sajid Mushtaq, Brice Augustin, and Abdelhamid Mellouk. Crowd-sourcing framework to assess qoe. In *2014 IEEE International Conference on Communications (ICC)*, pages 1705–1710. IEEE, 2014.

[120] Hyunwoo Nam, Kyung-Hwa Kim, Doru Calin, and Henning Schulzrinne. Youslow: a performance analysis tool for adaptive bitrate video streaming. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 111–112, 2014.

[121] Vikram Nathan, Vibhaalakshmi Sivaraman, Ravichandra Addanki, Mehrdad Khani, Prateesh Goyal, and Mohammad Alizadeh. End-to-end transport for video qoe fairness. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 408–423. 2019.

[122] Ravi Netravali, Ameesh Goyal, James Mickens, and Hari Balakrishnan. Polaris: Faster page loads using fine-grained dependency tracking. In *NSDI*, pages 123–136, 2016.

[123] Ravi Netravali, Vikram Nathan, James Mickens, and Hari Balakrishnan. Vesper: Measuring time-to-interactivity for web pages. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2018.

[124] Anh Nguyen, Zhisheng Yan, and Klara Nahrstedt. Your attention is unique: Detecting 360-degree video saliency in head-mounted display for head movement prediction. In *Proceedings of the 26th ACM international conference on Multimedia*, pages 1190–1198, 2018.

[125] Braxton Osting, Jiechao Xiong, Qianqian Xu, and Yuan Yao. Analysis of crowdsourced sampling strategies for hodgerank with sparse random graphs. *Applied and Computational Harmonic Analysis*, 41(2):540–560, 2016.

[126] Cagri Ozcinar, Julian Cabrera, and Aljosa Smolic. Visual attention-aware omnidirectional video streaming using optimal tiles for virtual reality. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(1):217–230, 2019.

[127] Prolifc. `https://prolific.co/`, 2022.

[128] Benjamin Rainer, Stefan Petscharnig, Christian Timmerer, and Hermann Hellwagner. Statistically indifferent quality variation: An approach for reducing multimedia distribution cost for adaptive video streaming services. *IEEE Transactions on Multimedia*, 19(4):849–860, 2016.

[129] Benjamin Rainer, Markus Waltl, and Christian Timmerer. A web based subjective evaluation platform. In *2013 Fifth International Workshop on Quality of Multimedia Experience (QoMEX)*, pages 24–25. IEEE, 2013.

[130] Lenin Ravindranath, Jitendra Padhye, Sharad Agarwal, Ratul Mahajan, Ian Obermiller, and Shahin Shayandeh. AppInsight: Mobile app performance monitoring in the wild. In *OSDI*, volume 12, pages 107–120, 2012.

[131] Lenin Ravindranath, Jitendra Padhye, Ratul Mahajan, and Hari Balakrishnan. Timecard: Controlling user-perceived delays in server-based mobile applications. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 85–100, 2013.

[132] ITUT Rec. H. 264: Advanced video coding for generic audiovisual services. *ITU-T Rec. H. 264-ISO/IEC 14496-10 AVC*, 2005.

[133] ITUT Rec. P. 800.1, mean opinion score (mos) terminology. *International Telecommunication Union, Geneva*, 2006.

[134] Abdul Rehman, Kai Zeng, and Zhou Wang. Display device-adapted video quality-of-experience assessment. In *Human Vision and Electronic Imaging XX*, volume 9394, page 939406. International Society for Optics and Photonics, 2015.

[135] Haakon Riiser, Paul Vigmostad, Carsten Griwodz, and Pål Halvorsen. Commute path bandwidth traces from 3g networks: analysis and applications. In *Proceedings of the 4th ACM Multimedia Systems Conference*, pages 114–118, 2013.

[136] Werner Robitza, Marie-Neige Garcia, and Alexander Raake. A modular http adaptive streaming qoe model—candidate for itu-t p. 1203 ("p. nats"). In *2017 Ninth International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1–6. IEEE, 2017.

[137] Sanae Rosen, Bo Han, Shuai Hao, Z Morley Mao, and Feng Qian. Push or request: An investigation of HTTP/2 server push for improving mobile performance. In *Proceedings of the 26th International Conference on World Wide Web*, pages 459–468. International World Wide Web Conferences Steering Committee, 2017.

[138] Vaspol Ruamviboonsuk, Ravi Netravali, Muhammed Uluyol, and Harsha V Madhyastha. Vroom: Accelerating the mobile web with server-aided dependency resolution. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 390–403. ACM, 2017.

[139] Miha Rugelj, Urban Sedlar, Mojca Volk, Janez Sterle, Melita Hajdinjak, and Andrej Kos. Novel cross-layer qoe-aware radio resource allocation algorithms in multiuser ofdma systems. *IEEE transactions on communications*, 62(9):3196–3208, 2014.

[140] Brandon Schlinker, Hyojeong Kim, Timothy Cui, Ethan Katz-Bassett, Harsha V Madhyastha, Italo Cunha, James Quinn, Saif Hasan, Petr Lapukhov, and Hongyi Zeng. Engineering egress with edge fabric: Steering oceans of content to the world. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 418–431. ACM, 2017.

[141] Michael Seufert, Ondrej Zach, Tobias Hoßfeld, Martin Slanina, and Phuoc Tran-Gia. Impact of test condition selection in adaptive crowdsourcing studies on subjective quality. In *2016 Eighth International Conference on Quality of Multimedia Experience (QoMEX)*, pages 1–6. IEEE, 2016.

[142] Ivan Slivar, Lea Skorin-Kapov, and Mirko Suznjevic. Cloud gaming qoe models for deriving video encoding adaptation strategies. In *Proceedings of the 7th international conference on multimedia systems*, pages 1–12, 2016.

[143] Rajiv Soundararajan and Alan C Bovik. Video quality assessment by reduced reference spatio-temporal entropic differencing. *IEEE Transactions on Circuits and Systems for Video Technology*, 23(4):684–694, 2012.

[144] Kevin Spiteri, Ramesh Sitaraman, and Daniel Sparacio. From theory to practice: Improving bitrate adaptation in the dash reference player. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 15(2s):1–29, 2019.

[145] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, 2016.

[146] Neil Stewart, Christoph Ungemach, Adam JL Harris, Daniel M Bartels, Ben R Newell, Gabriele Paolacci, Jesse Chandler, et al. The average laboratory samples a population of 7,300 amazon mechanical turk workers. *Judgment and Decision making*, 10(5):479–491, 2015.

[147] Gary J Sullivan, Jens-Rainer Ohm, Woo-Jin Han, and Thomas Wiegand. Overview of the high efficiency video coding (hevc) standard. *IEEE Transactions on circuits and systems for video technology*, 22(12):1649–1668, 2012.

[148] Yi Sun, Xiaoqi Yin, Junchen Jiang, Vyas Sekar, Fuyuan Lin, Nanshu Wang, Tao Liu, and Bruno Sinopoli. Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction. In *Proceedings of the 2016 ACM SIGCOMM Conference*, pages 272–285, 2016.

[149] P Lalith Suresh, Marco Canini, Stefan Schmid, and Anja Feldmann. C3: Cutting tail latency in cloud data stores via adaptive replica selection. In *12th USENIX Symposium on Networked Systems Design and Implementation*, pages 513–527, 2015.

[150] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[151] Yongxin Tong, Libin Wang, Zimu Zhou, Lei Chen, Bowen Du, and Jieping Ye. Dynamic pricing in spatial crowdsourcing: A matching-based approach. In *Proceedings of the 2018 International Conference on Management of Data*, pages 773–788, 2018.

[152] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.

[153] Beth Trushkowsky, Tim Kraska, Michael J Franklin, and Purnamrita Sarkar. Answering enumeration queries with the crowd. *Communications of the ACM*, 59(1):118–127, 2015.

[154] Madeleine Udell and Stephen Boyd. Maximizing a sum of sigmoids. *Optimization and Engineering*, 2013.

[155] Balajee Vamanan, Jahangir Hasan, and TN Vijaykumar. Deadline-aware datacenter tcp (d2tcp). *ACM SIGCOMM Computer Communication Review*, 42(4):115–126, 2012.

[156] Matteo Varvello, Jeremy Blackburn, David Naylor, and Konstantina Papagiannaki. Eyeorg: A platform for crowdsourcing web quality of experience measurements. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 399–412, 2016.

[157] Matteo Varvello, Jeremy Blackburn, David Naylor, and Konstantina Papagiannaki. Eyeorg: A platform for crowdsourcing web quality of experience measurements. In *Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies*, pages 399–412, 2016.

[158] Yilin Wang, Sasi Inguva, and Balu Adsumilli. Youtube ugc dataset for video compression research. In *2019 IEEE 21st International Workshop on Multimedia Signal Processing (MMSP)*, pages 1–5. IEEE, 2019.

[159] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[160] Zhou Wang, Eero P Simoncelli, and Alan C Bovik. Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*, volume 2, pages 1398–1402. Ieee, 2003.

[161] Aoyu Wu, Liwenhan Xie, Bongshin Lee, Yun Wang, Weiwei Cui, and Huamin Qu. Learning to automate chart layout configurations using crowdsourced paired comparison. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2021.

[162] Chen-Chi Wu, Kuan-Ta Chen, Yu-Chun Chang, and Chin-Laung Lei. Crowdsourcing multimedia qoe evaluation: A trusted framework. *IEEE transactions on multimedia*, 15(5):1121–1137, 2013.

[163] Zhe Wu, Curtis Yu, and Harsha V Madhyastha. Costlo: Cost-effective redundancy for lower latency variance on cloud storage services. In *NSDI*, pages 543–557, 2015.

[164] Mai Xu, Ali Borji, Ce Zhu, Edward Delp, Marta Mrak, and Patrick Le Callet. Introduction to the issue on perception-driven 360 video processing. *IEEE Journal of Selected Topics in Signal Processing*, 14(1):2–4, 2020.

[165] Francis Y Yan, Hudson Ayers, Chenzhi Zhu, Sadjad Fouladi, James Hong, Keyi Zhang, Philip Levis, and Keith Winstein. Learning in situ: a randomized experiment in video streaming. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 495–511, 2020.

[166] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, Victor Lin, Colin Rice, Brian Rogan, Arjun Singh, Bert Tanaka, Manish Verma, Puneet Sood, Mukarram Tariq, Matt Tierney, Dzevad Trumic, Vytautas Valancius, Calvin Ying, Mahesh Kallahalla, Bikash Koley, and Amin Vahdat. Taking the edge off with Espresso: Scale, reliability and programmability for global internet peering. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 432–445. ACM, 2017.

[167] Hyunho Yeo, Youngmok Jung, Jaehong Kim, Jinwoo Shin, and Dongsu Han. Neural adaptive content-aware internet video delivery. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 645–661, 2018.

[168] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 325–338, 2015.

[169] Junyong You, Touradj Ebrahimi, and Andrew Perkis. Attention driven foveated video quality assessment. *IEEE Transactions on Image Processing*, 23(1):200–213, 2013.

[170] Minlan Yu, Wenjie Jiang, Haoyuan Li, and Ion Stoica. Tradeoffs in CDN designs for throughput oriented traffic. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 145–156. ACM, 2012.

[171] Ke Zhang, Wei-Lun Chao, Fei Sha, and Kristen Grauman. Video summarization with long short-term memory. In *European conference on computer vision*, pages 766–782. Springer, 2016.

[172] Xu Zhang, Yiyang Ou, Siddhartha Sen, and Junchen Jiang. Sensei: Aligning video streaming quality with dynamic user sensitivity. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, pages 303–320, 2021.

[173] Xu Zhang, Paul Schmitt, Marshini Chetty, Nick Feamster, and Junchen Jiang. Enabling personalized video quality optimization with vidhoc. *arXiv preprint arXiv:2211.15959*, 2022.

[174] Xu Zhang, Siddhartha Sen, Daniar Kurniawan, Haryadi Gunawi, and Junchen Jiang. E2e: embracing user heterogeneity to improve quality of experience on the web. In *Proceedings of the ACM Special Interest Group on Data Communication*, pages 289–302. 2019.

[175] Yulun Zhang, Yapeng Tian, Yu Kong, Bineng Zhong, and Yun Fu. Residual dense network for image super-resolution. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2472–2481, 2018.

[176] Tiesong Zhao, Qian Liu, and Chang Wen Chen. Qoe in video transmission: A user experience-driven strategy. *IEEE Communications Surveys & Tutorials*, 19(1):285–302, 2016.

[177] Yuanhuan Zheng, Di Wu, Yihao Ke, Can Yang, Min Chen, and Guoqing Zhang. Online cloud transcoding and distribution for crowdsourced live game video streaming. *IEEE Transactions on Circuits and Systems for Video Technology*, 27(8):1777–1789, 2016.

[178] Kaiyang Zhou, Yu Qiao, and Tao Xiang. Deep reinforcement learning for unsupervised video summarization with diversity-representativeness reward. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[179] Yi Zhu, Sharath Chandra Guntuku, Weisi Lin, Gheorghita Ghinea, and Judith A Redi. Measuring individual video qoe: A survey, and proposal for future directions using social media. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 14(2s):1–24, 2018.

[180] Yi Zhu, Ingrid Heynderickx, and Judith A Redi. Understanding the role of social context and user factors in video quality of experience. *Computers in Human Behavior*, 49:412–426, 2015.

[181] Thomas Zinner, Oliver Hohlfeld, Osama Abboud, and Tobias Hoßfeld. Impact of frame rate and resolution on objective qoe metrics. In *2010 second international workshop on quality of multimedia experience (QoMEX)*, pages 29–34. IEEE, 2010.

[182] Xutong Zuo, Jiayu Yang, Mowei Wang, and Yong Cui. Adaptive bitrate with user-level qoe preference for video streaming. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pages 1279–1288. IEEE, 2022.

Appendices

.1 Incentive to improve latency

We show that it is impossible to improve a group of users' QoE without reducing at least some of their external delays. Formally, this can be expressed by the following theorem.

Theorem 1. *For n requests $r_1, \ldots, r_n$, suppose S and S' are the server-side delay vector when the external delay vector is C and C', respectively. Then $\sum_i Q(c'_i + s'_i) > \sum_i Q(c'_i + s_i)$ only if there is an r_i such that $c'_i < c_i$.*

Proof. Our proof is by contradiction. Suppose that $\sum_i Q(c'_i + s'_i) > \sum_i Q(c_i + s_i)$ and for all i, $c'_i \geq c_i$ (*i.e.,* no request has a better external delay). Then

$$\begin{aligned} \sum_i Q(c_i + s_i) &\geq \sum_i Q(c_i + s'_i) && (S \text{ is better than } S \text{ given } C) \\ &\geq \sum_i Q(c'_i + s'_i) && (Q \text{ is monotonic}) \end{aligned}$$

which contradicts the assumption. □

.2 User Study on Web Quality of Experience

We provide more details about our user study, which measures the relationship between page load time (PLT) and quality of user experience. We conduct this study on participants hired through Amazon MTurk [1], a crowdsourcing marketplace.

.2.1 Test procedure

Before entering the study, participants have to fill out a questionnaire about their basic information, such as age group, nationality, gender, time spent online per day, and primary use of internet. Each participant is asked to rate their experience of the same web page when it is loaded with different PLTs. Since the actual PLT of a page may be affected by many factors—*e.g.,* the participants' browsers, operating systems, and network conditions—

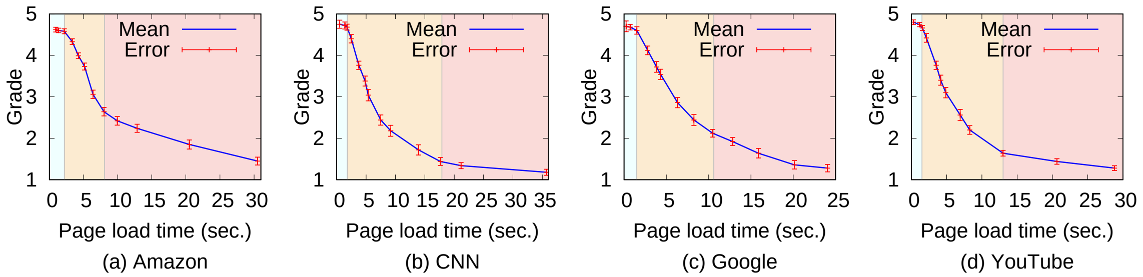
we show each participant a video recording of the web page being loaded at a certain speed, rather than letting them load the web page. This ensures that all the participants experience the same PLTs. The videos of different PLTs are played in a random order. After watching a video, the participants rate the video with a score ranging from 1 to 5 (with 1 being the least satisfactory and 5 being the most satisfactory), and this score is regarded as the QoE for the PLT shown in the video.

.2.2 Video recording

In our study, we need to show videos of certain PLTs. To avoid uncontrollable WAN and server-side delays, we first download all web page content on a local machine, and then load the pages on this machine. This reduces the factors affecting PLT to (1) the browser rendering time on the machine, which is a function of system configuration (*e.g.,* operating system, computer hardware, browser version, etc.) but remains fixed, and (2) the web data packet arrival rate. Since the data packets are loaded from the local machine itself, we can achieve the desired PLT by tuning the per-packet delay using a Chrome developer tool called NoThrottling. This allows us to load each web page at the desired PLT, and record a video of the loading process. These are the videos that are downloaded and shown to the participants during the study.

.2.3 Results

We ran the user study on the three page types in our traces (Table 2.1), as well as four other web pages: a Google search results page and the homepages of Amazon, CNN News, and YouTube. For each page, we use 50 MTurk participants. Figure 1 shows the results for the four web pages. We can see that although the websites have different PLT sensitivities, a sigmoid-like relationship between QoE and PLT exists for all of them.



(a) Amazon (b) CNN (c) Google (d) YouTube

Figure 1: The relationship between page load time and user rating in different websites.

.2.4 Response validation

A common problem in crowdsourcing is the validation of participants' responses. We filtered invalid responses in two ways:

- *Engagement:* Spending too long or too short on a video may indicate that the participant is distracted or unengaged. We set time thresholds for identifying such participants, and remove any response that takes more than 35 seconds or less than 2 seconds.
- *Outliers:* We view the average of all responses as the "ground truth". We drop responses from participants whose ratings consistently deviate from the ground truth by 3, across all videos.

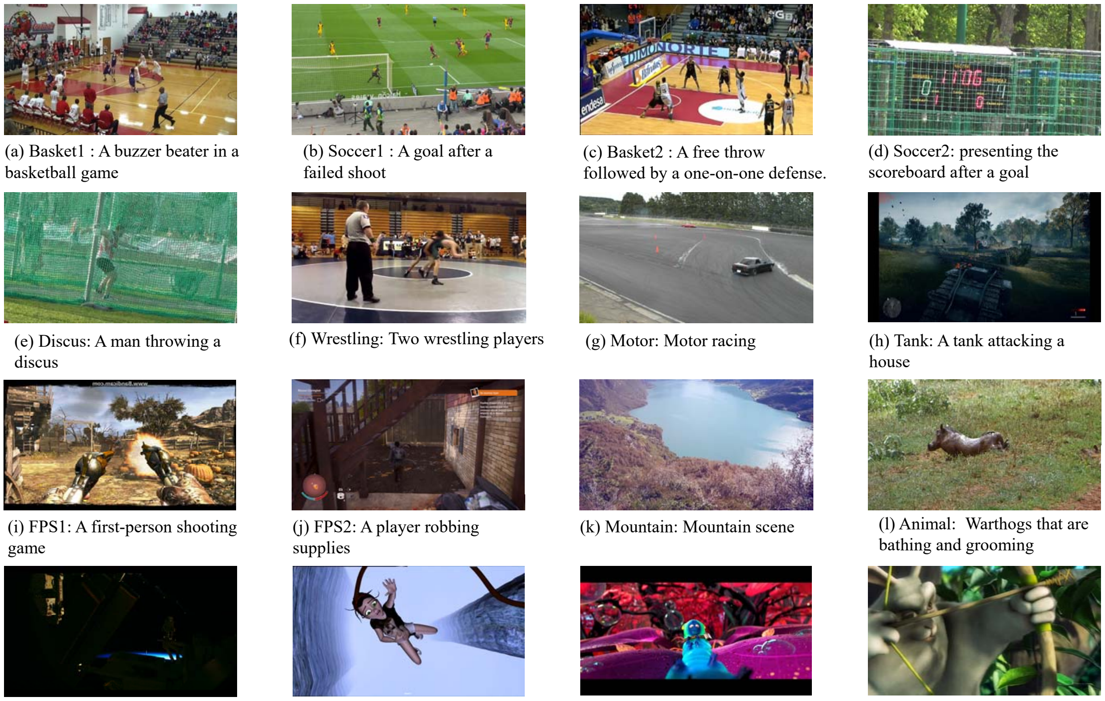
.3 Sensei's Dataset

Figure 2 provides screenshots and descriptions of the 16 source videos used in our dataset. Table 1 summarizes the test videos.

.4 Reliable QoE Crowdsourcing

We provide a few additional details on our crowdsourcing methodology.

Population bias of MTurk: As mentioned in §3.9, the per-chunk quality sensitivity could be biased by the population distribution of MTurkers. We confirm that about 43.8% (and



(a) Basket1 : A buzzer beater in a basketball game

(b) Soccer1 : A goal after a failed shoot

(c) Basket2 : A free throw followed by a one-on-one defense.

(d) Soccer2: presenting the scoreboard after a goal

(e) Discus: A man throwing a discus

(f) Wrestling: Two wrestling players

(g) Motor: Motor racing

(h) Tank: A tank attacking a house

(i) FPS1: A first-person shooting game

(j) FPS2: A player robbing supplies

(k) Mountain: Mountain scene

(l) Animal: Warthogs that are bathing and grooming

(m) Space: A satellite taking pictures for our Earth

(o) Girl: A girl falling off the cliff

(n) Lava: A lava is waking up

(p) BigBuckBunny: A rabbit dealing with three tiny bullies

Figure 2: Summary of source videos in our dataset. They span four genres: sports (a - g), gaming (h - j), natural (k - m), and animation (n - p). They are compiled randomly from public QoE datasets: LIVE-MOBILE [72](a,k), LIVE-NFLX-II [31] (b, n), YouTube-UGC [158] (c - j and l - o); and WaterlooSQOE-III [62] (p).

67.3%) of the received ratings come from MTurkers who participate in our survey only once (at most twice). This suggests that the pool of MTurkers is large enough to avoid small population bias, which corroborates our sanity-check results (§3.5) that on average MTurker quality ratings are strongly correlated with in-lab survey results.

Fast MTurker recruitment: While the MTurk platform cuts the overhead to publish our survey, if MTurkers sign up slowly, this can slow down the entire process. We take following steps to speedup the recruitment of MTurkers.

- *Competitive compensation:* We offer an hourly rate of \$10, a competitive compensation on the MTurk platform — only 4% MTurkers are paid more than \$7.25/hour [78], though we have not explored the impact of raising/lowering this rate. To prevent people from gaming the system by sitting on a job for too long, we pay each MTurker by the estimated amount

Name	Genre	Length	Source dataset
(a) Basket1	Sports	3:40	LIVE-MOBILE
(b) Soccer1	Sports	3:20	LIVE-NFLIX-II
(c) Basket2	Sports	3:40	YouTube-UGC
(d) Soccer2	Sports	3:40	YouTube-UGC
(e) Discus	Sports	3:40	YouTube-UGC
(f) Wrestling	Sports	3:40	YouTube-UGC
(g) Motor	Sports	3:40	YouTube-UGC
(h) Tank	Gaming	3:40	YouTube-UGC
(i) FPS1	Gaming	3:40	YouTube-UGC
(j) FPS2	Gaming	3:40	YouTube-UGC
(k) Mountain	Nature	1:24	LIVE-MOBILE
(l) Animal	Nature	3:40	YouTube-UGC
(m) Space	Nature	3:40	YouTube-UGC
(o) Girl	Animation	3:40	YouTube-UGC
(n) Lava	Animation	3:40	LIVE-NFLIX-II
(p) BigBuckBunny	Animation	9:56	WaterlooSQOE-III

Table 1: Summary of the test video set.

of time needed to finish a survey (which is proportional to the total length of the videos per MTurker), rather than by how much time the MTurker actually spends. In practice, this only weeds out MTurkers who spend too much time on a survey.

- *Maintaining good reputation:* The MTurkers' signing-up speed also depends largely on the reputation of the publisher (*i.e.,* us), because MTurkers tend to sign up if the publisher historically has a low rejection rate. Thus, it is critical to be clear upfront about our study's rejection criteria. In the meantime, to keep our rejection rate low, we try to target reliable MTurkers by restricting ourselves to Master MTurkers (a common practice for publishers on MTurk [113]).

.5 Sensei's Implementation

Automation of MTurk tests: We implement the pipeline shown in Figure 3.13 in Python (for the logic) and Javascript (for the video server). Given a source video, it first creates the rendered videos by adding specific low-quality incidents in the source video (via ffmpeg). It

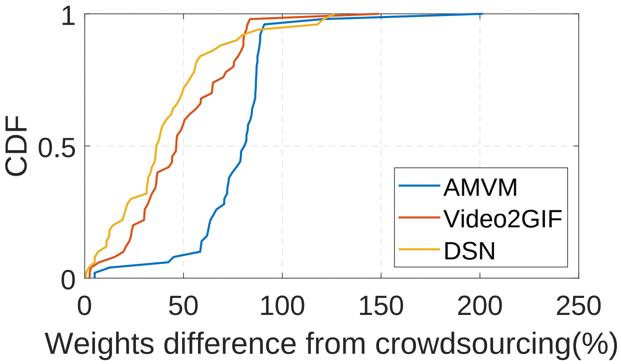
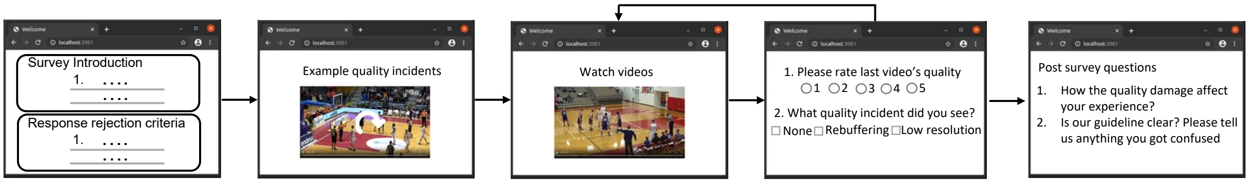


Figure 3: Chunk weight difference



Survey Introduction
1. ....
....
Response rejection criteria
1. ....
....

Example quality incidents

Watch videos

1. Please rate last video's quality
○1 ○2 ○3 ○4 ○5
2. What quality incident did you see?
□None □Rebuffering □Low resolution

Post survey questions
1. How the quality damage affect your experience?
2. Is our guideline clear? Please tell us anything you got confused

(a) Instruction page (b) Tutorial page (c) Video player page (d) Rating page (e) Post survey question page

Figure 4: A diagram of our QoE survey interface. In each survey, an MTurker is asked to rate K rendered videos; after watching each rendered video, an MTurker is asked to rate its quality on a scale of 1 (worst) to 5 (best).

then uploads these videos to a video server, from which MTurkers later download the video. After that, it generates a unique link for this campaign and posts it on the MTurk website (the only step that requires manual intervention). MTurkers can join the test by clicking the link, which redirects them to our video server. Once an MTurker has rated all assigned videos, the server logs the ratings and notifies us. Once enough ratings are received, the server trains the per-chunk weights as described in §3.4.2.

Single survey procedure: As shown in Figure 4:

- (a) Each survey starts with the instructions and rejection criteria under which ratings of an MTurker will be rejected. Each MTurker is expected to read the instructions carefully.
- (b) The MTurker then watches an example video that includes a quality incident so that they know what their ratings should be based on.
- (c, d) After that, the MTurker is asked to watch a sequence of rendered videos (determined by the scheduler) and, after each video, rate the quality on a scale of 1-5.

- (d) Finally, the MTurker does an exit survey.

.6 More discussion on saliency models

Saliency models used in §3.2.3: We test four models in total, a traditional motion-based heuristic (AMVM [112]), a highlight detection model (video2GIF [75]), and a video summarization model (DSN [178], dppLSTM [171]). We used the pretrained models of Video2GIF (`https://github.com/gyglim/video2gif_code`), DSN (`https://github.com/KaiyangZhou/pytorch-vsumm-reinforce`), and dppLSTM (`https://github.com/kezhang-cs/Video-Summarization-with-LSTM`).

Saliency score vs. quality sensitivity: Although a variety of saliency models have been proposed, we argue that these visual heuristics can be misaligned with video quality sensitivity. For example, in the soccer video (Figure 3.1), the scene right before the goal is most quality sensitive, but the highlight detection models and motion-based heuristics we evaluated believe the scenes showing the audience are the most important (probably because they show more human movements). Video summarization models pick all diverse moments of a video, but many of them may not be quality-sensitive. For example, in the same soccer video, the video summarization models identify every shot, rewind, and celebration clip as important, but the users pay more attention to shots that might score a goal.

We also acknowledge the potential use of viewership information to detect video highlights. While the popularity of a chunk is closely related to highlights (or high interestingness scenes) in a video, as we found in §3.2.3 and §3.7.3, these incidents may not perfectly align with users' sensitivity to video quality. Below are two examples specifically regarding the potential misalignment between content popularity and quality sensitivity.

- Example 1: A less popular part of a video can still be quality-sensitive. The "animal" video in our dataset is a part of a video about wildlife in Africa. Although a more "popular" or "interesting" scenes is one where the lions chase antelopes, we find that users are still

```
# Application demos and their ratings that have already been collected
appdemo_map_ratings: Dict[appdemo, QoE ratings (Set[int])]

# Application demos that we need to collect their ratings
demo_to_emit: Dict[appdemo, view_times (int)]

# Application demo (quality + content)
appdemo: Tuple[raw_video_id, List[quality_events]]

# Application quality event
quality_event:
 List[Tuple[quality_change_type, affected_time_range (Tuple[float, float]), \
    quality_to_change (float)]]

# Type of quality change
quality_change_type:
    Enum[’FreezeFrame’, ’ChangeBitrate’, ’ChangePlaybackRate’]

# Raw video identifier. Defined in the project manifest file.
raw_video_id: int
```

Figure 5: Data types used in writing assignment generator in VidPlat

```
demos_to_rate = initialize_assignments()
while demos_to_rate is not empty:
  demo_map_rating = call_crowd_tool(demos_to_rate)
  demos_to_rate = update_assignments(demo_map_rating)
```

Figure 6: Traditional interface

highly quality-sensitive in the scene where warthogs jump into a small pond for bathing.

- Example 2: A quality-sensitive part of a video may be a small fraction of a popular segment. One of the soccer videos in our dataset is a compilation of highlight moments from a long game, so all of its content is supposed to be “popular”. However, we still see that there is heterogeneity in the sensitivity of its chunks.

.7 Type definition in VidPlat

In detail, we define associated types of parameters in the above Python-style pseudocode for the use of requesters in Figure 5.

```
initial_demos = initialize_assignments()
call_VidPlat(initial_demos, update_assignments())
```

Figure 7: VidPlat's interface

.8 Why does not VidPlat require extra effort?

While VidPlat does introduce changes to the researcher interface, it does not result in a significant increase in effort for researchers compared to traditional tools. The rationale behind this is straightforward: even with conventional tools, researchers are required to script logic to determine subsequent assignments based on collected QoE ratings.

Figures 6 and 7 illustrate the contrasting interactions between researchers using traditional tools and those using VidPlat. With conventional tools, researchers must repeatedly invoke crowdsourcing tools until there are no remaining demos to be rated. This iterative process inherently leads to a sequence of tasks. On the other hand, with VidPlat, a single call is enough. More crucially, the logic used for initializing demos and subsequently updating them based on collected QoE ratings from traditional tools remains relevant. VidPlat can invoke these logic scripts seamlessly, without prematurely ending a user-study task. This efficiency significantly streamlines the initial setup phase.