The World is Too Big to Download: 3D Model Retrieval for World-Scale Augmented Reality

Yi-Zhen Tsai, James Luo, Yunshu Wang, Jiasi Chen
Department of Computer Science & Engineering
University of California, Riverside
Riverside, CA, USA
{ytsai036,jluo011,ywang1127}@ucr.edu,jiasi@cs.ucr.edu

Abstract

World-scale augmented reality (AR) is a form of AR where users move around the real world, viewing and interacting with 3D models at specific locations. However, given the geographical scale of world-scale AR, pre-fetching and storing numerous high-quality 3D models locally on the device is infeasible. For example, it would be impossible to download and store 3D ads from all the storefronts in a city onto a single device. A key challenge is thus deciding which remotely-stored 3D models should be fetched onto the AR device from an edge server, in order to render them in a timely fashion – yet with high visual quality – on the display. In this work, we propose a 3D model retrieval framework that makes intelligent decisions of which quality of 3D models to fetch, and when. The optimization decision is based on quality-compression tradeoffs, network bandwidth, and predictions of which 3D models the AR user is likely to view next. To support our framework, we collect real-world traces of AR users playing a world-scale AR game, and use this to drive our simulation and prediction modules. Our results show that the proposed framework can achieve higher visual quality of the 3D models while missing fewer display deadlines (by 20%) and wasting fewer bytes (by 10×), compared to a baseline approach of pre-fetching models within a fixed distance of the user.

CCS Concepts: • **Information systems** → **Multimedia streaming**; • **Human-centered computing** → **Ubiquitous and mobile computing**.

Keywords: Augmented reality, world-scale AR, 3D model retrieval

ACM Reference Format:
Yi-Zhen Tsai, James Luo, Yunshu Wang, Jiasi Chen. 2023. The World is Too Big to Download: 3D Model Retrieval for World-Scale Augmented Reality. In *Proceedings of the 14th ACM Multimedia Systems Conference (MMSys '23), June 7–10, 2023, Vancouver, BC, Canada.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3587819.3590970

MMSys '23, June 7–10, 2023, Vancouver, BC, Canada

ACM ISBN 979-8-4007-0148-1/23/06.
https://doi.org/10.1145/3587819.3590970

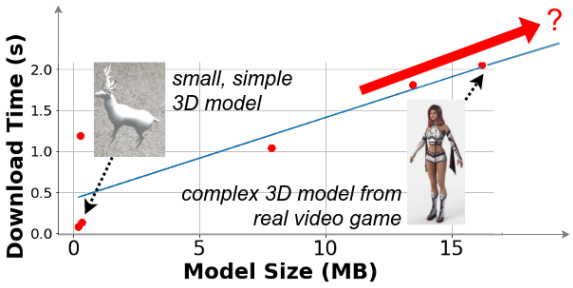


Figure 1. As 3D models become more complex, their retrieval time increases. Download times measured from an Android device with a 100 Mbps bandwidth.

1 Introduction

Augmented Reality (AR) overlays 3D models onto the physical world, and is one of the most promising "killer apps" for mobile and wearable devices. While many current AR apps are limited to small local areas (*e.g.*, furniture placement in a room), in the future we envision *world-scale AR*, which encompasses a much larger geographical area, becoming prevalent. Prime examples of world-scale AR include Pokemon Go, Niantic's Urban Legends demo app [5], and Google Maps Live View. As world-scale AR develops, we envision a rich ecosystem of AR apps populating the world with 3D models and users viewing these models seamlessly integrated with the real world. For example, as a user walks along a route, she may see 3D ads on top of real world store front, alongside virtual navigation directions and game creatures.

A key expectation of the user is that the 3D models pop up as soon as the user points her device in the appropriate direction. In the above example, when the user faces the relevant store front, the ads with high-quality 3D models should appear quickly in the display. Several contributors to the total latency: (a) determining where the 3D model should be placed on the display, (b) retrieving the 3D model data, and finally (c) rendering it onto the display. The former (a) has been optimized through object detection or image matching [17, 23], and the latter (c), is relatively fast (3-7% of end-to-end latency) without hardware accelerators [33, 35]. Our proposed work can reduce one component of end-to-end latency and are complementary to any rendering performance

optimizations. In this work, we focus on the retrieval of the 3D model data (b) as a critical contributor to the overall latency.

Why is retrieving 3D model data challenging? The main reason is that the 3D model is large, which can lead to long retrieval times. For example, a detailed dragon model [3] requires 113 MB, and even with compression techniques [1] still consumes 40 MB. While current 3D models in AR games are relatively simple (*e.g.*, a static Pokemon creature), we envision that in the future, more complex and animated 3D models will be needed. Figure 1 shows the sizes of different 3D models and their download times, from a simple deer model to a complex animated character from a modern video game. If multiple complex 3D models with an average size of 50 MB are present in a world-scale area, such as a campus of 5 km^2 with a 3D model every 10 m, this would require approximately 50 MB/10 $m^2 \times 5km^2$ = 25 GB of storage.

This 3D model data can either be stored locally or remotely. However, relying on local disk is undesirable, as it would require the dedicated 25 GB just for a single app.Remote storage and rendering [4] is an alternative but requires dedicated computing from edge infrastructure, as well as continuous network connectivity which may not be possible as a user moves around in world-scale AR. In this work, we focus on remote storage plus local rendering, as it requires minimal infrastructure support and is compatible with today's AR development platforms (*e.g.*, Unity, ARCore). However, retrieving 25 GB of data over the network is challenging, even with 5G download speeds of up to 1 Gbps [25], and it would take more than 3 minutes to pre-download all the models.A naive solution of retrieving all the 3D models in the world from a server leads to untenable latencies as the user waits for the models to be downloaded and shown on the display. Prior approaches in the multimedia community partition purely virtual spaces [7, 8, 15, 16] and retrieve nearby elements react to user movements without considering their future trajectory.

The main idea of this work is that as the world-scale AR user explores the real world, visiting locations where she expects to see/interact with 3D models, which should be retrieved *just in time* from the edge server for rendering locally. The challenge lies in determining which 3D model should be retrieved when, while respecting resource constraints. Retrieving model too early would waste network and storage resources, while too late would risk it not appearing on the display when she arrives at its location. Decisions made earlier on affect later decisions, as network resources are finite. An additional dimension that adds complexity is the choice of model quality. Overall, this problem is non-trivial to solve.

Two additional pieces of information could potentially aid this decision-making: (a) an estimate of a 3D model's visual quality, and (b) a prediction of which 3D models the user is likely to view next. Knowing the former allows for a graceful tradeoff of visual quality for latency; for example, a nearby 3D model could be retrieved more quickly at a lower quality, since it is likely to be viewed sooner. However, the visual quality of 3D models depends on myriad factors, such as the geometry and texture quality, and standard 3D model formats (*e.g.*, .glTF) do not typically include such information. Knowing the latter allows for better prioritization of model retrieval, based on past history of other AR users' behavior in the same geographical area. For example, a user located at an intersection may be more likely to turn to the right for a valuable 3D game treasure. However, no publicly available traces currently exist, to the best of our knowledge, of user mobility in world-scale AR (the closest is for synthetic room-scale AR [29]). AR user mobility patterns can differ from traditional walking patterns, so without AR-specific data, prediction is difficult.

The contributions of this work are summarized as follows:

- **Characterizing visual quality tradeoffs of 3D models:** We profile a set of 3D models to learn the tradeoffs between compression parameters (*e.g.*, texture quality, mesh quantization), perceptual quality (measured using SSIM [30]), viewpoint, and file size. Our main takeaway is that mesh quantization has the greatest impact on perceptual quality. We develop a prediction module that estimates the visual quality of a 3D model based on the relevant factors.
- **Predicting what an AR user will view:** We conducted a measurement campaign of real users playing a world-scale AR app. With this data, we designed a prediction module that predicts which 3D models the AR user will view next. It does this by leveraging information about user interactions with the AR app, such as what areas the user is looking at on the display, in order to better predict where a user will go, and hence which 3D models will be viewed.
- **Optimizing which 3D models are retrieved when**: With the aforementioned inputs, we formulate the problem of deciding which 3D models to request when, and at what quality. The resulting problem is a combination of a multiple-choice knapsack and an earliest-deadline first scheduler, which we solve optimally using dynamic programming.
- **Trace-driven simulations and prototype**: User study data as well as synthetic traces were used to evaluate overall framework. Our results show that compared to *Median* baseline, our method can retrieve 3D models with comparable quality while meeting retrieval deadlines (11% misses vs 57%) and wasting lesser data (1.4 GB vs 48 GB). We also implement an Android proof-of-concept prototype.

In the remainder of this paper, we discuss related work (§2), the overall system design (§3), scheduling and optimization of 3D model retrieval (§4), AR user behavior prediction (§5), and 3D model visual quality characterization (§6). Finally, we discuss the evaluation results (§7) and conclusions (§8).

2 Related Work

3D model quality characterization: Several works consider the impact of 3D object compression on perceptual quality. [24] considers texture adaptation, but not meshes. [32] considers both textures and meshes to generate rate-distortion curves. However, they do not consider a comprehensive range

of factors (*e.g.*, viewing angle, distance, mesh quantization, texture compression) in estimating quality, as we do.

3D model adaptation and world partitioning: Several works [12, 14] propose techniques for retrieving point clouds, which are a different type of 3D data than the mesh/texture-based 3D models that are often used in AR. [11] proposes extensions to DASH for VR, selecting the segments of the VR scene to download based on the predicted viewport. Furion [18] pre-renders different viewports, and chooses adjacent viewports for rendering. Modern video games split virtual world into chunks, and pre-load nearby chunks [7]. Other work on Networked Virtual Environments (NVE) [8, 15, 16] also partition virtual world and serve content to users based on their surrounding Area of Interest (AoI), which is typically circular. A detailed comparison is given in Table 1. None of these works incorporate prediction based on what is on the AR display to make proactive decisions, and instead react to the user's movements. Their method of choosing nearby chunks, partitions, or viewports for rendering is similar to our baseline that fetches nearby 3D models based on distance. Talaria [9] migrates parts of game state between edge servers, whereas we seek to retrieve objects from the server.

The closest to this work is perhaps [22], which also studies 3D model adaptation for AR; however, it considers a smaller-scale AR scene with only four 3D models in a single room, and assigns a joint deadline to all models under the assumption of fixed bandwidth. When scaling up to world-scale AR, bandwidth will be wasted since all models are downloaded. In contrast, we adjust each model's deadline based on when the user is predicted to view it.

Geography-based offloading: Several works in cloud/edge offloading consider user mobility when making offloading decisions [19, 27, 34]. Our framework differs in that we jointly optimize which 3D model to download and what version, unlike geography-based offloading, which typically optimizes which server to run a task on (ignoring what version of a task to run). Further, our user mobility predictor considers AR-specific aspects such as the visual display to help predict user behavior, unlike mobility prediction in these works.

AR user behavior: User movement prediction based on the history of user trajectories have been studied [28, 31] for non-AR user experience purposes. [17] predicts AR user movement for fast image matching, but only predicts time and rotation. Several works [26, 29] perform viewport prediction for users navigating a synthetic AR app in 27 m^2 (290 ft^2) indoor lab, rather than our dataset which is primarily outdoor, from a real-world AR application, covers larger areas (around 1 km^2) per trace, and contains traces from multiple US states.

3 System Design

The system design is shown in Fig. 2. On the server side, 3D models (*e.g.*, teapot, rabbit) with different quality levels are stored. Based on the 3D models' locations (provided by the AR ecosystem [17, 23]) and the user's predicted location, the client decides which version of 3D model should be fetched and when, downloads it from the server, and renders it on the display. There are three main modules to accomplish this:

	User prediction	Area of Interest	Content delivery	Optimal version selection
HyperVerse [8]	N	circular	P2P	N
VAST [15]	N	circular	P2P	N
FLoD [16]	N	circular	P2P	progressive mesh
DASH 3D [11]	Y	viewport based	centralized	N
Ours	Y	viewport based	centralized	Y

Table 1. Detailed comparison of related NVE works. They focus on partitioning the world and delivering content from adjacent regions, which is similar to our *Distance* baseline.

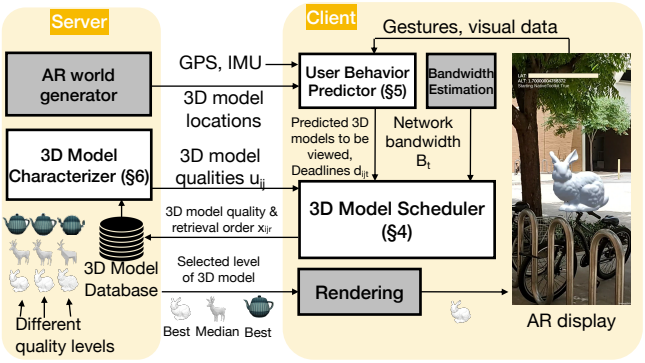


Figure 2. System architecture. Given a set of 3D models at various quality levels, our system retrieves the optimal 3D model versions from the server so that the 3D models are rendered in time on the display.

- **3D Model Scheduler (client)**: This module takes as input the 3D model quality/utility values, which 3D models the user is predicted to view next, time-varying estimates of network bandwidth, and the 3D model display deadlines (from the User Predictor module). It determines a retrieval order for each 3D model based on the deadlines, and runs a dynamic program that maximizes the sum utility of the downloaded 3D models, while meeting their retrieval deadlines within the available network bandwidth.
- **User Behavior Predictor (client)**: To predict which 3D models the user is likely to view next, this module takes inputs: the user's geolocation history (GPS, IMU), the location of 3D models of interest, and the user's interactions with the AR display (*e.g.*, viewing a map of nearby 3D models). The main insight is that by understanding these interactions, the predictions should outperform standard prediction that only relies on geolocation history.
- **3D Model Characterizer (server):** This module quantitatively evaluates the visual quality of compressed version of 3D model, based on compression parameters such as mesh quantization, view angle and distance, etc. The predicted visual quality/utility is used by the 3D Model Scheduler.

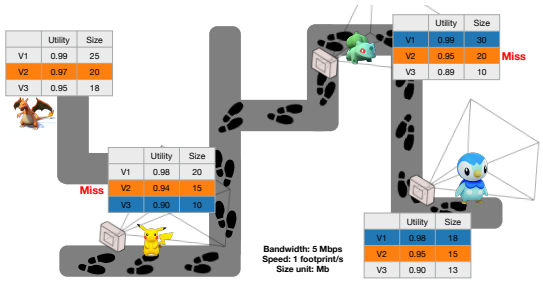


Figure 3. Toy example. A user encounters a set of 3D models, each available in 3 versions. Compared to the "Median" baseline, our method selects the correct models to achieve good visual quality (2.87 vs 0.95), with fewer misses (0 vs 2).

B_t	estimated network bandwidth at time slot t
c	confidence threshold for the User Predictor
d_{ijt}	download time of 3D model i, version j ending at time t
D_r	cumulative download duration of the first r 3D models
H	length of user history input to the User Predictor
M	number of versions of each 3D model available
N	number of 3D models
s_{ij}	size of 3D model i at version j
t_i	deadline of downloading 3D model i (derived from §5)
T	how far into the future the User Predictor should operate
u_{ij}	utility (*i.e.*, SSIM) of 3D model i in version j
$U(i,t)$	optimal utility of the first i 3D models after time t
x_{ijr}	decision variable of whether to download 3D model i in version j in the r^{th} place

Table 2. Table of Notation

These three modules work together in concert.The User Behavior Predictor runs on the client, due to privacy concerns (private geolocation and display information), and shares its result and deadlines with the 3D Model Scheduler. The 3D Model Scheduler also runs on the client, it obtains nearby 3D models information such as size and visual quality from the 3D Model Characterizer, running on the edge server, and requests the appropriate 3D model versions from server through HTTP request. The User Behavior Predictor and 3D Model Scheduler run continuously in the background as the user moves around the world.

4 3D Model Scheduler

The goal is to select which version of each 3D model should be retrieved, and in what order, which we formulate as an optimization problem. This decision is based on the next set of 3D models the user is likely to view (from the User Prediction module), as well as the set of available 3D models on the server (from the 3D Model Characterization module), and finally the available network bandwidth.

Toy Example: A toy example is shown in Fig. 3. The user walks along the path and encounters 3D models, each available in 3 versions (shown in the adjacent table). The "Median" baseline simply selects the median quality of all 3D models in the world (orange rows). It would retrieve unneeded 3D models (*e.g.*, the dragon) and request too high a quality for the first encountered 3D model (yellow Pikachu), causing it to miss its deadline, and resulting in a missed later 3D model (green Bulbasaur). A better solution would be to avoid downloading the dragon entirely, download a low-quality yellow Pikachu (V3) since it will soon need to be rendered, and download the remaining models at better quality (V1). Our solution, which accounts for the future, can retrieve all 3D models within deadline with good visual quality.

System model: More formally, there are a set of N 3D models $\{i\}$ available in the world, each with M possible versions indexed by j. Each model has a utility u_{ij} and a size s_{ij}, as well as a deadline t_i. There is a time-varying bandwidth $B(t)$ which we assume is known from a bandwidth predictor. The goal is to decide x_{ijr}, whether 3D model i should be downloaded at version j in the r^{th} place. In other words, we must both determine what quality models to download and in what order. The optimization problem is:

Problem 1. *3D Model Scheduling and Version Selector*

$$\max \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{r=1}^{N} u_{ij}x_{ijr} \tag{1}$$

$$s.t.\ s_{ij}x_{ijr} \le \sum_{t'=t-d_{ijt}}^{t} B_{t'}\ \forall i,j,r,t \tag{2}$$

$$D_r = \sum_{r'=1}^{r}\sum_{i=1}^{N}\sum_{j=1}^{M} d_{ijD_{r'}}x_{ijr'} \le \sum_{i=1}^{N}\sum_{j=1}^{M} t_i x_{ijr}\ \forall r \tag{3}$$

$$\sum_{j=1}^{M}\sum_{r=1}^{N} x_{ijr} = 1\ \forall i, \quad \sum_{i=1}^{N}\sum_{j=1}^{M} x_{ijr} = 1\ \forall r \tag{4}$$

$$variables\ x_{ijr} \in \{0,1\}, d_{ijt} \in N \tag{5}$$

The objective (1) is to maximize the total utility of selected 3D model versions (obtained from §6). Constraint (2) states that the selected 3D model's size must fit into the bandwidth.Constraint (3) defines the cumulative downloading duration D_r of the first r downloaded 3D models. Each 3D model must finish downloading before its retrieval deadline t_i (based on user prediction from §5). Constraint (4) says that only one version of each 3D model should be selected, and only one 3D model is downloaded at a time. Along with x_{ijr} as defined previously, d_{ijt} is another auxiliary variable that can be interpreted as the download time of model i, version j if it finished downloading in time slot t.

The above problem is a combination of a scheduling problem and an item selection problem: both the order of the 3D model and the version of each 3D model must be selected. The order matters because network resources consumed by earlier, high-quality model downloads could take away from available resources for later downloads. However, this problem can be simplified by sorting the 3D models by their deadlines t_i gives the optimal download order. This is given by the following Lemma. The proof is provided in Appendix A.

Lemma 1. *An optimal solution has the items downloaded in ascending order of their deadlines t_i.*

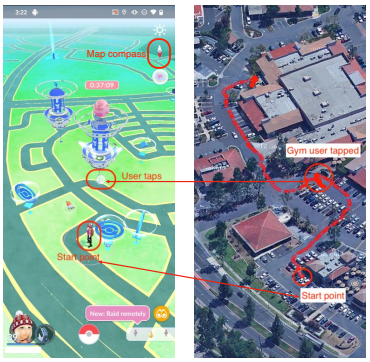


Figure 4. AR visual data. The user's tap on the AR display (left) can suggest user intent. The corresponding trace (right) shows that the user eventually walks toward the tapped point.

Hence this problem can be simplified by setting $r = i$, where the items i are sorted by ascending t_i, similar to an earliest-deadline first scheduling algorithm. The simplified problem formulation produces an optimal solution to the Problem 1 and is given by Problem 2 in Appendix A. Problem 2 can be solved optimally via dynamic programming. To do this, the d_{ijt} are pre-computed for all i, j, t as:

$$d_{ijt} = \min\{d : s_{ij} \leq \sum_{t'=t-d}^{t} B_{t'}\} \tag{6}$$

since we need to satisfy (9) while minimizing d_{ijt} for (10). The recurrence relation of the dynamic program is:

$$U(i, t + d_{ijt}) = \begin{cases} U(i, t-1) & \text{if } t > t_i \\ \max_j\{U(i-1, t-d_{ijt}) + u_{ij}\} & \text{if } t \leq t_i \end{cases} \tag{7}$$

where $U(i, t)$ is the optimal utility of the first i 3D models after time t. The first case represents when the time t has exceeded the deadline of item i, t_i, hence the optimal utility is simply the utility one time step ago. The second case represents when the deadline is not exceeded, and the regular recurrence relation for multiple choice knapsack holds. The final solution is found at $U(N, \max(t_i))$. The full algorithm is given in the Appendix (Alg. 1), with pseudo-polynomial runtime $O(NM \max(t_i) \max(s_{ij}))$. In particular, the algorithm is polynomial in N, the number of 3D models, so it should scale well in more densely packed AR scenarios in the future.

5 User Behavior Predictor

The goal is to predict what set of 3D models is likely to be viewed next and their deadlines t_i, based on history of a user's behavior. The deadlines are set based on the distance between user and the predicted interested 3D models and the user's average walking speed. We propose a data-driven approach to do this. We assume that the 3D models are fixed at certain locations, which are known to the predictor.

User study: We conducted a user study to measure the volunteers' behavior when playing an existing world-scale AR application (*i.e.*, Pokemon Go). Note that our framework is not specific to Pokemon Go, but we chose it because it is a widespread, popular app, from which we can collect realistic user behavior traces for this proof-of-concept research. Our framework applies to any future world-scale AR experience where 3D models are placed in around the world. We collected 7 users' data from 6 different zones (including parks, outdoor malls, and university campuses) across multiple US states for several weeks, resulting in a total of 289 minutes of traces.

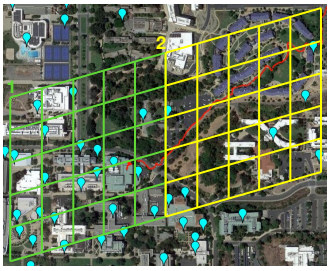


Figure 5. A zone consists 2 smaller sub-zones and each has 25 cells. The blue points of interest are the Poke-stops and Gyms, and the red lines show an example user trace.

Input features: We collected the following types of information to train the predictor:

- **AR visual data:** We recorded the user's screen and taps in order to understand the user's interaction in the AR app.We hypothesize that what is shown on the display or the user's taps should provide some hint of where she is likely to go, and hence improve the prediction of which 3D models are next needed. For example, in Fig. 4, after user taps on an object (the building), the user trace shows that user eventually walks to the location in the real world to view the 3D model. The map compass also provides a hint, as the user rotated the map to have the compass point south, and user eventually walks in a southerly direction.
- **Geolocation data:** The history of a user's movement from traditional sensors (*e.g.*, GPS, IMU) can help predict movement in the near future. Given knowledge of the locations of the 3D models, this can provide hints on which 3D model(s) the user is headed towards.

Design of the predictor: There were three main challenges we faced in the design of the predictor. We describe these challenges and our solutions briefly below.

How to generalize the predictor for world-scale AR? Ideally, we would like the predictor to work in all zones across the world, so that data collected from one zone can help train the predictor for other zones. The alternative would be to build a separate predictor for each zone, which would be impractical for world-scale AR as this would require lots of user data for training. For example, it would be easier for a global predictor to learn the general behavior "users move in straight lines" by seeing more examples from all zones, rather than a per-zone predictor that sees only few straight-line examples from its zone. To achieve this, we use relative position to represent the user location instead of absolute GPS coordinates; that way, each user trace is not tied to a specific real-world zone and can help train the overall model. Specifically, we

Figure 6. 3D models with different SSIM, created with combinations of mesh quant. levels and texture compression.

divide each zone into equal-sized sub-zones with 25 cells, give each sub-zone an ID, and convert the absolute longitude/latitude into sub-zone ID and cell ID. To the predictor, the input is a sequence of integer sub-zone and cell IDs representing the user's location, and the output is the the cell ID(s) that the user is likely to visit next, with its corresponding 3D model(s). This integer representation helps the predictor learn overall movement patterns that can generalize across zones; if instead the position were represented by GPS coordinates, the predictor would only learn patterns specific to each GPS area. An example of one such zone, sub-zones and cells is shown in Fig. 5.

How many 3D models to predict? A large output dimension (*e.g.*, predict the next 100 models) would lead to too many model parameters and possibly overfitting, while a low output dimension (*e.g.*, predict only the next 1 model) might lead to too few 3D models being retrieved, and hence missed deadlines. To address this, we set the output dimension to a relatively liberal value, and filter it based on its confidence. For example, if the predictor outputs three models $[A, B, C]$ with confidence values $[94\%, 5\%, 1\%]$ respectively, and we set a confidence threshold of $c = 5\%$, then only 2 models will remain after filtering. In this way, the output dimension of the predictor varies automatically.

How far into the future to predict? Ideally, a perfect prediction far into the future would be very useful to the 3D Model Scheduler. However, in practice, predictions far into the future tend to be mistaken, possibly making wrong decisions. To address this, we introduce a special *null* class as output and is equivalent to saying "I don't know". Beyond a time threshold T, the predictor should not try to predict, but rather return the *null* class. In this way, it avoids predicting too far into the future and returning mistaken predictions.

Predictor details: We use a gradient-boosted decision tree model in the XGBoost library [10] to perform multi-class classification, with 25 possible cells and the *null* class as output. We chose XGBoost as the ML model due to its good performance on a variety of application domains [13]; the modular design of our framework allows for other ML models to be tried in the future. The predictor is given the past H seconds of geolocation and AR visual data, and predicts the most likely 3D models that the user will view next, up to T seconds ahead. The number of 3D models output by the predictor is determined by a confidence threshold c. We experimentally evaluate the configuration settings in §7.3. Further details are in Appendix C.

6 3D Model Characterizer

The goal of this module is to understand how compression parameters and viewing configurations of the 3D models affect the visual quality and the file size. Based on this understanding, the 3D Model Scheduler (§4) can try to select 3D model version with reduced file size, without significantly degrading the visual quality. We considered the following compression parameters and viewing configurations:

- **Mesh quantization**: Mesh is the geometry, or shape, of a 3D model. It is a collection of vertices, edges, and faces, which usually consist of triangles. A more complex model has denser and smaller triangles.
- **Texture resolution**: A texture is a 2D image that is "wrapped" onto a 3D model to provide additional details/colors. The process of applying 2D images to 3D model requires mapping between 2D to 3D. A more complex model usually has more fine-grained and detailed texture files.
- **Distance and orientation**: We also examined the distance and orientation between the 3D model and the user. The intuition is that a distant 3D model at low quality may have similar visual quality as a nearby 3D model at high quality.
- **Vertices and faces counts**: The more complex 3D model, the larger of vertices counts and faces counts. For example, a cube model has 24/12 vertices/faces, while a relatively complex bunny model has 36,425/69,451 vertices/faces.

To measure visual quality, we use the structural similarity index measure (SSIM) [30], which has previously been used to assess AR/VR visual quality [12]. SSIM is a full-reference, perception-based quality assessment method to measure the similarity in structural information compared to the original. Unlike pixel-based quality metrics (*e.g.*, mean-squared error), SSIM takes the characteristics of human visual system into account and the inter-dependencies between pixels, and therefore can better reflect the user-perceived visual quality [30]. SSIM values range from 0-1, with the higher the better. A simple 3D bunny model with a detailed fur texture is shown in Fig. 6.The right-most version has a lower SSIM of 0.91 with noticeable visual artifacts in texture, while model with an SSIM of 0.98 appear reasonably close to the original.

7 Evaluation

In this section, we will first describe the setup and evaluation of the 3D Model Characterizer (§7.1), 3D Model Scheduler (§7.2), and User Behavior Predictor (§7.3) individually, then our end-to-end evaluation with all modules together (§7.4).

7.1 3D Model Characterizer Evaluation

7.1.1 Setup To create 3D models with different qualities, we used the Google Draco tool [1], to compress the mesh position coordinates with different quantization levels (where the level=number of bits). For texture files, we used the dimension resizing Mitchell algorithm [20] to downsize the texture file into different resolution levels. To create a blank slate for 3D model characterization, we created a Unity scene,

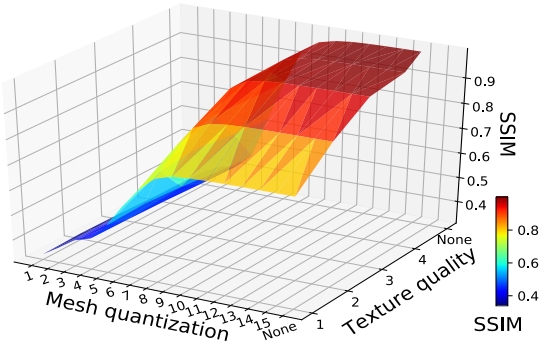


Figure 7. SSIM with different levels of mesh quantization and texture quality, of a bunny model. The mesh quantization level is positively correlated with SSIM up to level 8, after which the texture quality has more of an impact.

	SSIM correlation		
Compression Parameter	**All data**	**Mesh quant.<8**	**Mesh quant. >=8**
Mesh quantization	0.775	0.693	0.176
Texture resolution	0.023	-0.006	0.221
# Vertices	-0.076	-0.202	0.007
# Faces	-0.075	-0.202	-0.035
Distance	0.031	0.018	0.140
Orientation	0.003	0.009	-0.003

Table 3. Correlation coefficient between compression parameters and visual quality (SSIM). Mesh quantization has the highest correlation, then texture resolution.

placed the 3D model at different distances and orientations, captured screenshots, and computed the SSIM with reference to its undistorted model. For each 3D model, we evaluated 5 distances and 6 or 12 equally spaced angles between the 3D model and the user's viewpoint. We collected and characterized 14 popular 3D models [2] with 5 different textures and the aforementioned parameters to create our dataset. To predict the SSIM, we trained a regression model using XGBoost [10] (parameters n_estimators=10, max_depth=6), with a 75/25 train/test split. Further details are in Appendix B.

7.1.2 Results We first examined the Pearson correlation coefficient between the compression parameters (described in §6) and the visual quality (SSIM). The goal is to understand which parameters should be input to the visual quality predictor. The results in Table 3 shows that across all data (second column), the mesh quantization has the greatest correlation with SSIM. Hence we further divide the dataset into mesh quantization above/below 8 (third/fourth columns). The results indicate if the mesh quality is good enough, then texture is a second important factor. For mesh quantization level<8, texture is still correlated with SSIM, albeit to a lesser amount.

An illustration of the tradeoffs for the Stanford bunny model is shown in Fig. 7. The surface drops off sharply as a function of mesh quantization, and to a lesser extent with texture quality, consistent with Table 3. In fact, when the mesh quantization is below 8, it can only achieve SSIM up to 0.868, even with the best quality texture. The mesh file size ranges

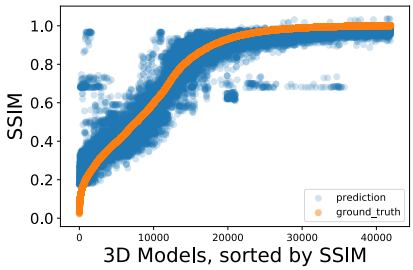


Figure 8. Regression result of the 3D Model Characterizer. The predicted SSIM (blue dots) follows closely with the true SSIM (orange line), with average error of 0.04.

from 521 to 690 KB as the mesh quantization level increases, showing increased visual quality requires larger file sizes.

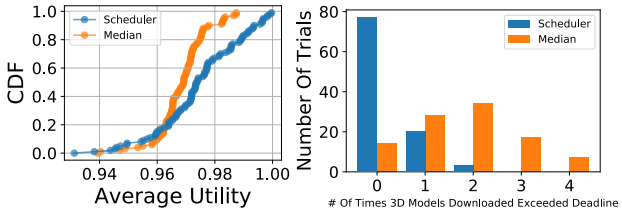
In Fig. 8, we show the results of the SSIM prediction with SSIM values on the y-axis, and the 3D models, sorted by true SSIM, on the x-axis. The orange line is the true SSIM, while the blue dots are the predictions. The blue dots generally follow the orange line, indicating reasonably good prediction performance, although there are a few isolated clusters of blue dots that suggest poor performance, especially for lower SSIMs. Upon closer examination, we found that these are very simple 3D models, such as a cube, that misled the characterizer. This is because there are fewer simple models in the training dataset – most are more complex, which we believe will be the case for future AR use cases. Overall, the predicted SSIM had a mean error of 0.04 compared to the ground truth with a Pearson correlation coefficient of 0.968.

7.2 3D Model Scheduler Evaluation

7.2.1 Setup To evaluate the scheduler, we created a simulation, where 4-10 3D models are placed at random locations, and a user performs a random walk for 30 seconds and 100 trials. We use a random walk model in order to stress test the scheduler with highly variable walking patterns and more traces compared to the real user data; the end-to-end evaluation later in §7.4 showcases real user traces. We define a user as viewing a 3D model when it is within 1 meter. The utilities are downsample from the dataset collected in §7.1 with 4 versions per 3D model, each with utility above 0.90. In expectation of more complex 3D models needed by future AR apps, we set the file sizes of each version between 5 to 40 MB, positively correlated with the utility. A network buffer keeps track of the progress, and flushes out unused 3D models that are more than 15 seconds old (similar to LRU cache). The network bandwidth is set to 5 MBps. We created a baseline *Median*, which retrieves all 3D models without considering deadlines, always fetching the median quality version in random order.

Metrics: The metrics include the average utility of the 3D models, as in Objective (1), and the number of times when user views a 3D model, but it has not finished downloading.

7.2.2 Results The CDF of the average utility of the 3D models over 100 trials in the simulation are shown in Fig. 9a. The histogram of the number of times the 3D models were



(a) 3D model utility **(b)** # of missed deadlines

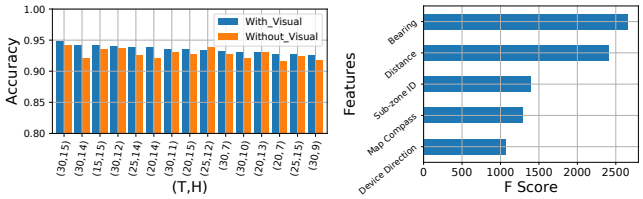
Figure 9. The 3D Model Scheduler achieves higher average utility across trials, fulfilling all the requests with fewer missed deadlines, compared to *Median*.

unable to be downloaded before their deadlines is shown in Fig. 9b. To summarize, our method (*Scheduler*) generally has higher utility and can successfully meet the retrieval deadlines. *Scheduler*'s utility is lower than the baseline *Median* for about 20% of cases, because we sometimes retrieve a lower-quality 3D model in order to meet the retrieval deadline. This is in line with our optimization formulation (Problem 1), which requires the deadline constraint to be satisfied before maximizing utility in the objective function. Digging deeper, in terms of download latency, both *Scheduler* and *Median* have median latency of 100 ms, with *Scheduler* being slightly higher since it chooses higher quality 3D models, while still meeting their deadlines (so the latency is invisible to the user).

7.3 User Behavior Predictor Evaluation

7.3.1 Setup The distance and GPS bearing (the direction from previous position to current position) of the user are calculated each timestep. We also manually labeled the direction of the application map compass as the AR visual data. These are input into the machine learning model. 75% of the data is used for training, 25% for testing. We define the Poke-stops and Gyms (fixed locations where users can collect items or battle with other users) as the 3D models viewed by the users. The AR visual data can be defined as the set of Poke-stops or Gyms that the user viewed on the application map (left screenshot in Fig. 4), the application map compass (upper right corner in the same screenshot), or a combination of both. We chose the application map compass as the main form of visual data, as it is the most helpful for prediction. We define a sub-zone as 25 level-17 S2 cells [6], which is a geographical region used by Google and Pokemon Go, and covers about 5000 m^2 per cell (level number corresponds to the cell size). Each zone consists of 1-23 sub-zones. For example, an outdoor shopping mall had 1 sub-zone, a park had 9 sub-zones, and a university campus had 23 sub-zones.

7.3.2 Results To systematically find the best hyperparameter values, we performed a grid search across $T = [5, 10, \ldots, 30]$ and $H = [1, 2, \ldots, 15]$, and evaluated the accuracy on the validation set, as shown in Fig. 10a. For clarity, we only plot the accuracy for those data samples where a 3D model was actually viewed (results are similar either way). The results shows the top 15 accurate combinations of (T, H). "With_Visual" means the predictor takes geolocation and AR visual data as inputs, while "Without_Visual" only uses geolocation data.



(a) Accuracy for various look-ahead times T and history H. **(b)** The top five features for user behavior prediction.

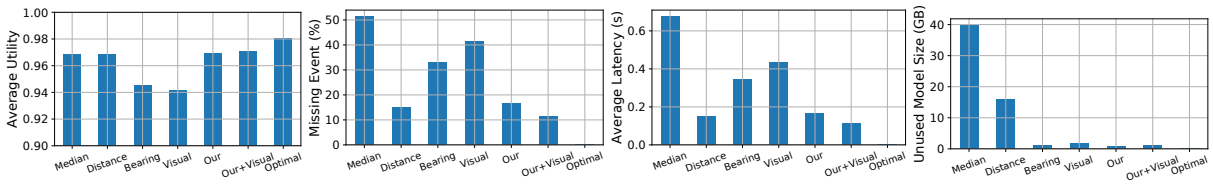
Figure 10. User Behavior Predictor evaluation. Using AR visual and geolocation data (*With_Visual*) yields higher accuracy than without AR visual data (*Without_Visual*).

The combination of $T = 30s$ and $H = 15s$ yields the highest accuracy, more than 90%. A higher H value improving prediction, since more history should help. A higher T value means predicting further into the future, which may be harder, but also more samples to train the classifier, so it overall increases accuracy. Moreover, having AR visual data improves accuracy. We also plot the top 5 important features for the predictor in Fig. 10b. The most important 3 features are the geolocation data, followed by the "Map Compass", which is the AR visual feature and was weighted more than the "Device Direction". In other words, the user's interactions with the screen (*e.g.*, rotating the AR app's world map reflecting in the "Map Compass") mattered more than the direction that the device was physically pointing, validating our hypothesis in §5. In summary, the trained predictor can accurately predict which 3D model is about to be viewed (with the best (T, H)), and we use it in the end-to-end evaluation (§7.4).

7.4 End-to-End Evaluation

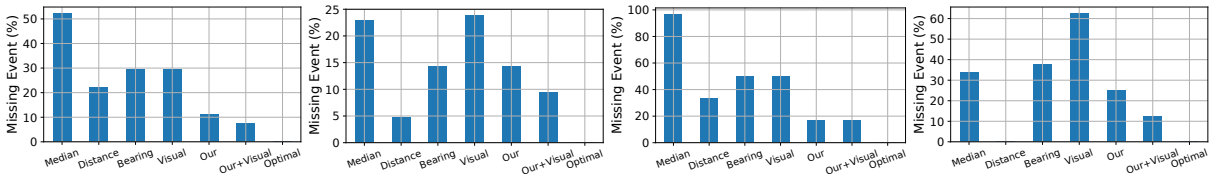
7.4.1 Setup Finally, we created an end-to-end trace-based simulation that incorporates the preceding modules. The simulation uses 4 test traces (23/37/4/8 minutes each) from 4 different locations and users, which have not been seen before by the User Behavior Predictor. An LRU cache of size 100 MB is used, which about half the total size of all the median-quality 3D models in one zone. The deadlines t_i are calculated as the distance divided by user's walking speed. Only those 3D models that are predicted to be viewed by the User Predictor are sorted and fetched by the 3D Model Scheduler. The network bandwidth is sampled from [21] to simulate outdoor 5G network, with mean throughput of 74.5 MB/s and standard deviation of 51.3 MB/s.

Baselines: In addition to *Median* (§7.2), we also create *Distance*, which retrieves the 3D models within a 160 meters radius of the user, and in increasing order of distance between the user and the 3D model. Other baselines include *Bearing*, which retrieves 3D models in neighboring cells as pointed to by the GPS bearing during the last $H = 15$ seconds; *Visual*, which uses the same mechanism as *Bearing*, except for the directional information is derived from the AR visual data (the map compass), and *Our*, uses a simplified User Behavior Predictor that does not include AR visual data. Our method, which uses geolocation data and AR visual data as input features, is labeled *Our+Visual*. We also implement an *Optimal*



(a) 3D model utility **(b)** % of missed deadlines **(c)** Missed model delay **(d)** Data waste

Figure 11. *Our* method achieves high average utility, few missed deadlines, low missed model delay, and much lower data waste compared to the baseline. With AR visual data included (*Our+Visual*), further gains are possible.



(a) Outdoor shopping mall **(b)** Outdoor park **(c)** University campus **(d)** Outdoor smaller park

Figure 12. *Our* and *Our+Visual* generally outperform the baseline methods at different locations, particularly in challenging environments such as a university campus with diversity of user behaviors.

oracle where the future behavior is perfectly known, and the 3D Model Scheduler picks exactly the models on time.

Metrics: In addition to previous metrics (see §7.2), we also show the data waste, (how much unused data is downloaded), and missed model delay, (how much delay is experienced by users waiting for a missed model to be retrieved).

7.4.2 Results In Fig. 11, we plot previous mentioned metrics and we also provide an *Optimal* solution that has the best average utility, zero miss, zero delay, and zero data waste.

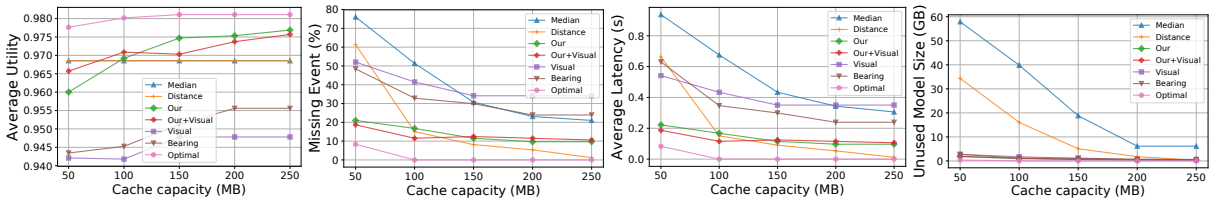
The results show that although the *Distance* baseline is better than the other baselines (*Median*, *Bearing*, and *Visual*) in terms of utility, fraction of missed deadlines, and missed model delay, *Our* also has few missed deadlines and saves on data (by 10×). With the help of the AR visual data in the user behavior prediction module, *Our+Visual* can maintain comparable utility to *Optimal* and the fewest missed deadlines among all methods (23% less than *Distance*). Moreover, *Our+Visual* has the lowest missed model delay, compared to all baselines. In terms of data waste, since the *Distance* method downloads models around the user, it wastes bandwidth retrieving models not in view compared to *Our* and *Our+Visual* methods. For *Bearing* and *Visual* baselines, although directional information is considered and can save data, their mechanisms are too simple to predict correctly, resulting in more missed events and lower utility.

Different zones: Delving deeper into the results, individual results for missed deadlines at four of our test zones are in Fig. 12. Our methods *Our* and *Our+Visual* outperform others, but the *Distance* baseline does reasonably well in second place and is a good alternative if training data for our methods are not available. Also, the percentage of missed events can vary quite widely among zones, with the outdoor parks being easier zone, and the university campus being the most difficult. The university campus is challenging because it is a huge zone with 23 sub-zones and there are several outlier 3D models at the edges of the sub-zones, which are hard to predict. Most players stay close to an important campus building, so when they do venture to the outlying areas, the User Behavior Predictor is not as capable of predicting those 3D model viewing events, and hence the 3D Model Scheduler does not retrieve them in time.

On the other hand, the outdoor park, although a medium-sized zone with 9 sub-zones, has relatively few Poke-stops, so user prediction is easier. A smaller park with 4 sub-zones, also supports this statement. In such easy zones, *Distance* performs well, since the Poke-stops in park are sparse and the directional information used by *Our+Visual* is not that useful. In summary, not only the zone size matters, but also the distribution of 3D models within the zone, with our methods doing well particularly in more challenging environments.

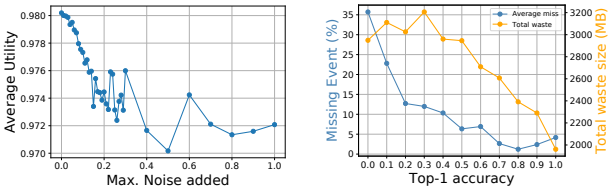
Impact of cache size: We also show how cache capacity can affect the performance in Fig. 13. As the cache capacity grows, *Our* and *Our+Visual* methods are more stable in terms of missed deadlines and missed model delay since we only retrieve necessary models compared to *Distance* baseline. Moreover, *Our* and *Our+Visual* methods are able to utilize the increasing cache capacity and retrieve models with higher utility, gradually approaching the *Optimal*. For the data waste, noted that a cache capacity of 200 MB can accommodate all the median versions 3D models in a single zone, and Fig. 13d again proves that *Our+Visual* method can better utilize network resources compared to *Distance*.

Robustness to noisy predictions: Finally, we examine how noisy predictions by the 3D model characterizer and user behavior predictor modules can impact overall performance. For the model characterizer, we add noise to its predicted SSIM values, with the noise chosen uniformly between $[-x, +x]$ where $x \in [0, 1]$. These noisy SSIM values are fed into the end-to-end framework. To isolate the impact of this



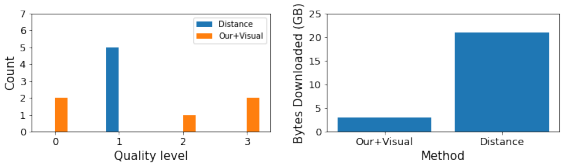
(a) 3D model utility **(b)** % of missed deadlines **(c)** Missed model delay **(d)** Data waste

Figure 13. *Our* and *Our+Visual* methods are more stable in terms of missed deadlines and extra delay of those missed models, as cache capacity changes, compared to other baselines.



(a) Model characterizer **(b)** User behavior predictor

Figure 14. With more noisy 3D model characterizer predictions, the average utility drops as well. For the user behavior predictor, as the top-1 accuracy increases, the missed deadlines decreases as does the data waste.



(a) Retrieved model quality **(b)** Data downloaded

Figure 15. Results of our prototype. *Our+Visual* saves data while retrieving higher quality models on average.

noise, we temporarily assume perfect user prediction. Intuitively, the added noise could reverse the ordering of different versions of a 3D model, resulting in a lower quality version having a higher predicted SSIM and vice versa, misleading the overall framework. In Fig. 14a, the average utility decreases as the maximum amount of noise (x) increases. Note that the SSIM do not drop significantly and still remain above 0.97, which is higher than the baselines in Fig. 11a.

We conducted a similar robustness evaluation on the user behavior predictor. We temporarily assumed perfect SSIM prediction, and manually manipulated the percentage of the correct cells predictions that were output by the user behavior predictor. In Fig. 14b, we plot the top-1 accuracy (that we manipulated) with respect to the missing events and total data waste. *Our+Visual*'s missing event rate decreases as top-1 accuracy increases and maintains reasonable values, *e.g.*, $< 10\%$ missed when the top-1 accuracy is higher than 0.4. The data waste also decreases with increasing accuracy, and even with low top-1 accuracy our method still wastes less data compared to the *Distance* baseline in Fig. 11d.

7.5 Android prototype

Finally, we developed an Android proof-of-concept prototype, to complement the above trace-driven simulations.The locations of the 3D models are hardcoded to be similar to Pokemon Go, and we followed a similar trajectory as the previous university campus testing trace. When approaching a 3D model, the application sends a request, pre-determined by the 3D Model Scheduler, to the server to retrieve the model at the determined quality, which is then rendered on the app as shown in the AR display part in Fig. 2. We evaluate the *Our+Visual* and *Distance* methods as those are the most promising from the preceding simulations.

Fig. 15a shows the results of the retrieved model quality level, with *Our+Visual* retrieving higher quality models on average. Fig. 15b shows the data downloaded. These values are quite large, because this zone is a very challenging environment as we mentioned in §7. But, *Our+Visual* still achieves significant data savings compared to *Distance*. Moreover, in terms of fraction of models missed, *Our+Visual* retrieves all on time, while *Distance* misses one model.

We also measured how much time ahead of the deadline that each model was retrieved (higher is better). The average time ahead for *Distance* was 14.44 s, while for *Our+Visual* was 45 s. Overall, these results help validate the simulations and demonstrate a proof-of-concept prototype.

8 Conclusions

In this paper, we explored the scenario of world-scale AR with multiple 3D models geographically dispersed throughout the real world. Under the premise that local storage is insufficient to store all possible 3D models, and remote rendering requires cloud/edge compute infrastructure, we explored challenges in retrieving remotely stored 3D models from an edge server for local rendering on the AR device. We propose a framework to optimize which 3D models to download and when, by characterizing 3D model quality-compression tradeoffs and predicting AR user behavior. To drive the user predictions and evaluations, we recorded the behavior of real users playing a world-scale AR app (Pokemon Go). The results show that combining AR user behavior predictions with principled 3D model pre-fetching can improve user experience by enabling more models to be retrieved before they are viewed, at a higher visual quality. Future work includes further data collection and exploring other visual data types to help the AR user behavior prediction. *This work does not raise ethical issues, and the user study was conducted with IRB approval.*

References

[1] Google draco 3d data compression. https://google.github.io/draco/.

[2] The stanford 3d scanning repository. http://www-graphics.stanford.edu/data/3Dscanrep/, 1994.

[3] Dragon free 3d model. https://www.cgtrader.com/free-3d-models/animals/other/dragon-3d-model, 2017.

[4] Azure remote rendering. https://azure.microsoft.com/en-us/services/remote-rendering/, 2021.

[5] Niantic planet-scale ar alliance accelerates social ar future in codename: Urban legends. https://nianticlabs.com/blog/niantic-planet-scale-ar-5g-urban-legends/, 2021.

[6] S2 geometry. https://s2geometry.io/, 2021.

[7] AVARD, A. The secret art of the video game loading screen, and why they won't be going away anytime soon. https://www.gamesradar.com/the-secret-art-of-the-video-game-loading-screen-and-why-they-wont-be-going-away-anytime-soon/, March 2019.

[8] BOTEV, J., HOHFELD, A., SCHLOSS, H., SCHOLTES, I., STURM, P., AND ESCH, M. The hyperverse: Concepts for a federated and torrent-based '3d web'. *Int. J. Adv. Media Commun. 2*, 4 (2008), 331–350.

[9] BRAUD, T., ALHILAL, A., AND HUI, P. Talaria: In-engine synchronisation for seamless migration of mobile edge gaming instances. In *ACM CoNEXT* (2021), pp. 375–381.

[10] CHEN, T., AND GUESTRIN, C. Xgboost: A scalable tree boosting system. In *ACM KDD* (2016).

[11] FORGIONE, T., CARLIER, A., MORIN, G., OOI, W. T., CHARVILLAT, V., AND YADAV, P. K. Dash for 3d networked virtual environment. In *ACM Multimedia* (2018).

[12] HAN, B., LIU, Y., AND QIAN, F. Vivo: visibility-aware mobile volumetric video streaming. In *ACM MobiCom* (2020).

[13] HARASYMIV, V. Lessons from 2 million machine learning models on kaggle. https://www.kdnuggets.com/2015/12/harasymiv-lessons-kaggle-machine-learning.html, 2015.

[14] HOSSEINI, M., AND TIMMERER, C. Dynamic adaptive point cloud streaming. *Packet Video Workshop* (2018).

[15] HU, S.-Y., CHEN, J.-F., AND CHEN, T.-H. Von: a scalable peer-to-peer network for virtual environments. *IEEE Network 20*, 4 (2006), 22–31.

[16] HU, S.-Y., HUANG, T.-H., CHANG, S.-C., SUNG, W.-L., JIANG, J.-R., AND CHEN, B.-Y. Flod: A framework for peer-to-peer 3d streaming. In *IEEE INFOCOM 2008-The 27th Conference on Computer Communications* (2008), IEEE, pp. 1373–1381.

[17] JAIN, P., MANWEILER, J., AND ROY CHOUDHURY, R. Overlay: Practical mobile augmented reality. *ACM MobiSys* (2015).

[18] LAI, Z., HU, Y. C., CUI, Y., SUN, L., DAI, N., AND LEE, H.-S. Furion: Engineering high-quality immersive virtual reality on today's mobile devices. *IEEE Transactions on Mobile Computing 19*, 7 (2019), 1586–1602.

[19] LIN, T.-Y., LIN, T.-A., HSU, C.-H., AND KING, C.-T. Context-aware decision engine for mobile cloud offloading. In *2013 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)* (2013), pp. 111–116.

[20] MITCHELL, D. P., AND NETRAVALI, A. N. Reconstruction filters in computer graphics. In *ACM SIGGRAPH* (1988).

[21] NARAYANAN, A., RAMADAN, E., MEHTA, R., HU, X., LIU, Q., FEZEU, R. A. K., DAYALAN, U. K., VERMA, S., JI, P., LI, T., QIAN, F., AND ZHANG, Z.-L. Lumos5g: Mapping and predicting commercial mmwave 5g throughput. In *ACM IMC* (New York, NY, USA, 2020), IMC '20, p. 176–193.

[22] PETRANGELI, S., SIMON, G., WANG, H., AND SWAMINATHAN, V. Dynamic adaptive streaming for augmented reality applications. In *IEEE ISM* (2019).

[23] RAN, X., CHEN, H., LIU, Z., AND CHEN, J. Deepdecision: A mobile deep learning framework for edge video analytics. *IEEE INFOCOM* (2018).

[24] SIMON, G., PETRANGELI, S., CARR, N., AND SWAMINATHAN, V. Streaming a sequence of textures for adaptive 3d scene delivery. In *IEEE VR* (2019).

[25] VERIZON. 5g speed: How fast is 5g? https://www.verizon.com/about/our-company/5g/5g-speed-how-fast-is-5g, 2020.

[26] VIOLA, A., SHARMA, S., BISHNOI, P., GADELHA, M., PETRANGELI, S., WANG, H., AND SWAMINATHAN, V. Trace match & merge: Long-term field-of-view prediction for ar applications. In *IEEE AIVR* (2021), IEEE, pp. 1–9.

[27] WANG, D., TIAN, X., CUI, H., AND LIU, Z. Reinforcement learning-based joint task offloading and migration schemes optimization in mobility-aware mec network. *China Communications 17*, 8 (2020), 31–44.

[28] WANG, H., TERROVITIS, M., AND MAMOULIS, N. Location recommendation in location-based social networks using user check-in data. In *ACM SIGSPATIAL* (2013).

[29] WANG, N., WANG, H., PETRANGELI, S., SWAMINATHAN, V., LI, F., AND CHEN, S. Towards field-of-view prediction for augmented reality applications on mobile devices. In *ACM International Workshop on Immersive Mixed and Virtual Environment Systems* (2020).

[30] WANG, Z., BOVIK, A., SHEIKH, H., AND SIMONCELLI, E. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing 13*, 4 (2004), 600–612.

[31] XUE, H., HUYNH, D. Q., AND REYNOLDS, M. Bi-prediction: Pedestrian trajectory prediction based on bidirectional lstm classification. In *IEEE International Conference on Digital Image Computing: Techniques and Applications (DICTA)* (2017).

[32] YANG, S., LEE, C.-H., AND KUO, C.-C. J. Optimized mesh and texture multiplexing for progressive textured model transmission. In *ACM Multimedia* (2004).

[33] YOUNIS, A., QIU, B., AND POMPILI, D. Latency-aware hybrid edge cloud framework for mobile augmented reality applications. In *2020 17th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)* (2020), pp. 1–9.

[34] YU, F., CHEN, H., AND XU, J. Dmpo: Dynamic mobility-aware partial offloading in mobile edge computing. *Future Generation Computer Systems 89* (2018), 722–735.

[35] ZHANG, W., HAN, B., AND HUI, P. On the networking challenges of mobile augmented reality. *ACM SIGCOMM Workshop on Virtual Reality and Augmented Reality Network* (2017).

APPENDIX

A Details of 3D Model Scheduler

Lemma 2. *An optimal solution of Prob. 1 has the items downloaded in ascending order of their deadlines t_i.*

Proof. Suppose there is an optimal solution without items downloaded in order of t_i. Then we claim that an equivalent solution exists with the items in sorted order. Let r_i be the order of the i^{th} item. Then there exists at least one pair of items (a, b) such that $r_a = r_b + 1$ despite $t_a < t_b$. We claim that we could swap their orders r_a, r_b to get new orders $\tilde{r}_a \leftarrow r_b, \tilde{r}_b \leftarrow r_a$ and still achieve the same utility, since the item versions are unchanged. Constraint (2) still holds since the same number of bits is still downloaded within the time period $[D_{r_b-1}, D_{r_a}]$, with the same available bandwidth. Constraint (3) still holds for item a at order $\tilde{r}_a$, since $D_{\tilde{r}_a} = \sum_{r'=1}^{r_b} \sum_{i,j} d_{ijD_{r'}} x_{ijr'} < \sum_{r'=1}^{r_a} \sum_{i,j} d_{ijD_{r'}} x_{ijr'} \leq t_a = \sum_{i,j} t_i x_{ij\tilde{r}_a}$. Constraint (3) still holds for item b at order $\tilde{r}_b$, since $D_{\tilde{r}_b} = \sum_{r'=1}^{r_a} \sum_{i,j} d_{ijD_{r'}} x_{ijr'} \leq t_a < t_b = \sum_{i,j} t_i x_{ij\tilde{r}_b}$. Constraint (3) holds for all other items since the only the two swapped items' download times are changed. The constraints

in (4) are satisfied by construction. Hence items (a, b) can be swapped and keep the same utility, without violating any constraints. We can swap additional item pairs until the items are sorted. □

This Lemma is used to transform Problem 1 to the following simplified problem formulation:

Problem 2. *Equivalent formulation of Problem 1*

$$\max \sum_{i=1}^{N} \sum_{j=1}^{M} u_{ij} x_{ij} \tag{8}$$

$$s.t.\ s_{ij} x_{ij} \leq \sum_{t'=t-d_{ijt}}^{t} B_{t'}\ \forall i, j, t \tag{9}$$

$$D_i = \sum_{i'=1}^{i} \sum_{j=1}^{M} d_{ijD_{i'}} x_{ij} \leq t_i\ \forall i \tag{10}$$

$$\sum_{j=1}^{M} x_{ij} = 1\ \forall i \tag{11}$$

$$variables\ x_{ij} \in \{0, 1\}, \quad d_{ijt} \in N \tag{12}$$

The items are the 3D models with choice of version, item value u_{ij}, and item weight d_{ijt}. The knapsack capacities are the deadlines t_i. However, there are key differences from the classical knapsack problem: (a) there are multiple knapsack constraints t_i, and (b) the individual item weights d_{ijt} are not fixed, but depend on the previous $i - 1$ chosen items.

Finally, Algorithm 1 presents the optimal dynamic program to solve Problem 2. The input of this algorithm includes utility, size, of all versions of the 3D models, retrieval deadlines for each 3D model, and estimated bandwidth at each timestamp. The output is the optimal version of each 3D model. For each version, we first pre-compute the downloading time of each 3D model in line 6, and the cumulative utility in line 8. Then, we check if the cumulative downloading time up to this version meets the retrieval deadline in line 9 and determine the optimal utility of each 3D model i up to time t. The final optimal solution is found at $U(N, \max(t_i))$.

Algorithm 1: Dynamic program to solve Problem 2

Input: Utility of version j of 3D model i u_{ij}, size of version j of 3D model i s_{ij}, deadline of 3D model i t_i, bandwidth at time t $B(t)$.
Output: Optimal version j of each 3D model i.

1 **for** $i \in [1, N-1]$ **do**
2 **for** $t \in [0, \max_i\{t_i\} - 1]$ **do**
3 **for** $j \in [0, M-1]$ **do**
4 $d_{ijt} \leftarrow 0$;
5 **for** $s = s_{ij}; s > 0; s \leftarrow s - B(t - d_{ijt})$ **do** ▷ (6)
6 $d_{ijt}++$
7 **end for**
8 $new\_utility \leftarrow U(i-1, t - d_{ijt}) + u_{ij}$
9 **if** $D_i \leq t_i$ *and* $new\_utility > U(i, t)$ **then** ▷ (7)
10 $U(i, t) \leftarrow new\_utility$
11 **else**
12 $U(i, t) \leftarrow U(i, t-1)$
13 **end if**
14 **end for**
15 **end for**
16 **end for**
17 **return** item versions corresponding to $U(i, t_{N-1})$

Input features for our model		
Name	**Description**	**Values**
Mesh quantization level	Draco mesh compression level	1-15
Texture quality level	texture file downsizing level	1-5
# of Vertices	vertices count	real values
# of Faces	faces count	real values
Distance	distance from user to model	1-5 m
Orientation	orientation around the vertical axis	In $30, 60°$ increments
Regression model parameters		
Name	**Values**	
# Estimators	10	
Max depth	6	
Training set	75% dataset	
Testing set	25% dataset	

Table 4. Parameters for 3D model characterizer.

B Details of 3D Model Characterizer

14 popular 3D models range from simple cube with 12 faces and 24 vertices count to complex video game asset with 263,146 faces and 757,378 vertices count are included in the dataset with each model textured with 5 detailed texture files with 4-5 quality levels ranging from 256x256 to 4096x4096 resolution and 128KB to 32 MB file size. Each model then placed in 5 different distances (from 1 to 5 meters) and 6 or 12 equally spaced angles (30, 60,.. 360 degrees around the vertical axis) to create different view to users. For each view, a screenshot is taken to evaluate its SSIM by comparing to its undistorted, best quality reference model view. Total dataset consists of 167,670 samples.

C Details of User Behavior Predictor

To train our classifier, geolocation features such as our converted sub-zone ID, and cell ID, user's distance and bearing from previous position to current position, device direction derived from device IMU sensor and hand annotated AR visual data such as application map compass (*i.e.*, north, northeast, etc.) are used as input features. Besides, $H = 15$ seconds of features are used to predict the next timestamp. The output of the classifier is the cell IDs (1-25) that the user is likely to view next or the *null* class. We use a confidence value $c = 6\%$ to dynamically control the output dimension.

Input features for our model		
Name	**Description**	**Values**
Zone ID	IDs assigned to our test sites	1-6
Sub-zone ID	IDs assigned to equal-sized sub-zones in each zone	1-44
Cell ID	user's current location within the cell in a sub-zone	1-25
Distance	displacement from previous position to current position	real values
Bearing	direction from previous position to current position	[-180,180]
Device direction	device compass derived from IMU sensor	0-359°
Map compass	annotated AR visual data	(N, NW, W, SW, W, SE, E, NE)
Multi-class classifier parameters		
Name	**Values**	
# Estimators	100	
Max depth	6	
Training set	75% dataset	
Testing set	25% dataset	

Table 5. Parameters for user behavior predictor.