Bespoke Experiences: Personalized Augmented Reality Via Imitation Learning on the Edge

Surin Ahn, *Student Member, IEEE*, Maria Gorlatova, *Member, IEEE*
Parinaz Naghizadeh, *Member, IEEE*, Mung Chiang, *Fellow, IEEE*.

***Abstract*—Augmented reality (AR) technologies are rapidly gaining momentum in society and are expected to play a critical role in the future of cities and transportation. In such dynamic settings with a heterogeneous population of AR users, it is important for holograms to be placed in the surrounding environment with regard to the user's preferences. However, the development of general mechanisms for making AR experiences more personalized to the user is currently largely unexplored. This paper proposes to use behavioral cloning, an autonomous imitation learning technique, as a means of automatically generating policies that capture user preferences of hologram positioning. We use a reduced feature-based state vector, as opposed to the complete camera feed, to make behavioral cloning practical for this application. Through empirical results obtained with a Unity-based AR simulator we designed and developed, we demonstrate that user-specific policies can be learned both quickly and accurately, even in the presence of noisy training examples. We demonstrate that our approach adheres to the strict latency and frame rate constraints of AR systems. By minimizing the volume of data sent to the cloud, we help to preserve user privacy and increase communication and learning efficiency.**

***Index Terms*—Augmented reality, edge computing, behavioral cloning, privacy.**

I. Introduction

In augmented reality (AR), a layer of virtual content such as text, video, and holograms is superimposed onto a user's view of the real world [1]. AR is seeing increasingly widespread adoption across numerous domains and is projected to generate billions of dollars in revenue within the next decade [2]. It is expected by many to become the next big computing platform, potentially as revolutionary as the personal computer and as ubiquitous as the smartphone. Not only is AR permeating the space of mobile devices [3], but fully immersive AR systems – whether in the form of head-mounted displays (HMDs) [4], [5], or AR-enhanced windshields for vehicles [6] – are also becoming commercially viable. A grand vision for AR is a smart city in which users can perceive real-time annotations of roads, buildings, transportation systems, and even people [7], [8]; see Figs. 1 and 2.

Challenges in AR. Current and future AR deployments face a number of key challenges, which we summarize here as three interconnected areas:

S. Ahn is with the Department of Electrical Engineering, Stanford University, Stanford, CA 94305, USA (e-mail: surinahn@stanford.edu). M. Gorlatova is with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708, USA (e-mail: maria.gorlatova@duke.edu). P. Naghizadeh and M. Chiang are with the School of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907, USA (e-mails: parinaz@purdue.edu, chiang@purdue.edu).

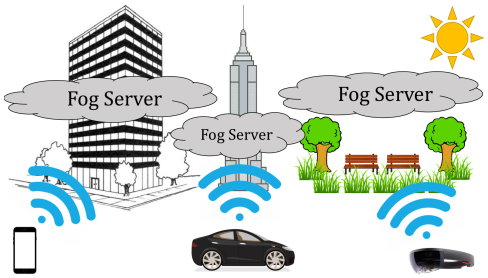


Fig. 1: Geographically dispersed local servers (such as fog nodes) can help enable ubiquitous AR experiences in a smart city setting, offering low-latency data processing capabilities to a heterogeneous set of devices and applications.

1) *Power and delay constraints.* The graphical demands of AR applications require significant power consumption. AR experiences are also very sensitive to delay, and the presence of lag in the processing of user interactions or virtual content could have negative real-world implications [9], [10]. For example, users could experience motion sickness in more immersive scenarios, and in AR-enhanced driving situations, delays in visual markers could result in navigation errors or even accidents. Thus, enabling seamless and pervasive AR experiences will require a networking infrastructure, such as fog and edge computing architectures, that supports ultra low-latency and resource-efficient data processing [11].

2) *Security and privacy.* A relatively new but growing area of interest is the security and privacy of AR systems. As these devices collect and analyze continuous streams of fine-grained sensor data, including video, audio, accelerometer readings, and user interactions with the environment, there is a pressing need to regulate what can be seen and manipulated by third parties such as cloud computing platforms [12]. From a security standpoint, we also need to ensure that holograms are presented to the user in a safe, non-distracting, and unobstructive way [13], [14], [15]. These safety mechanisms will become essential as AR-equipped vehicles and glasses scale to wide-spread common-place commercial deployments.

3) *Personalization.* By virtue of their complex arrays of sensors, actuators, and information processing capabilities, AR systems are in many ways more intimately connected to users than any devices before. To the best of our knowledge, there is no existing literature on *personalizing hologram placements based on user preferences*; the aforementioned works on security and privacy do not consider ways to incorporate user feedback into the resulting policies. We expect this

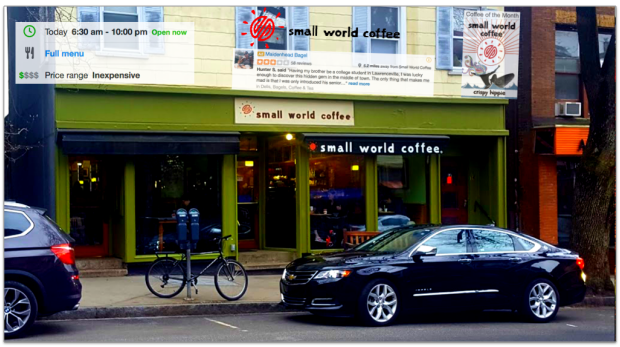


Fig. 2: Concept image illustrating the potential of AR to merge digital information with the real world.

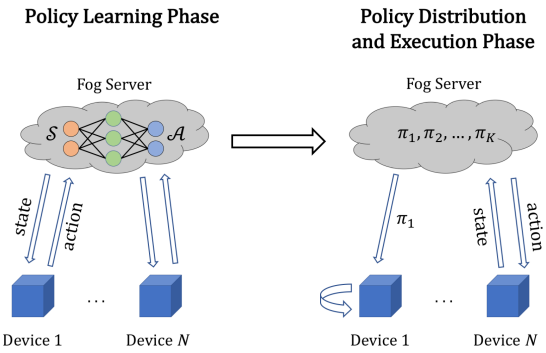


Fig. 3: System diagram illustrating how a local server first trains several policies using state-action pairs from nearby devices, then either distributes these policies to powerful devices or executes them on behalf of resource-poor devices.

kind of personalization to be necessary to accommodate a heterogeneous population of AR users, especially those using HMDs. Such users may differ in attributes such as height, eyesight, or hand dominance, all of which may influence their preferences for where holograms should be placed.

Our contributions. This paper advances the personalization dimension of AR systems by developing a method to automatically learn user preferences of hologram placements using *behavioral cloning*, an autonomous imitation learning technique, which has been employed successfully in several different scenarios where a human expert's demonstrations can serve as valuable training data (e.g., autonomous vehicles [16]). Moreover, we simultaneously address the first two challenges by employing a light-weight architecture that leverages the availability of local computational resources. Our architecture is in particular consistent with *fog computing* [17], a paradigm in which traditionally cloud-based data storage, computation, communication and control can exist anywhere along the continuum from the cloud to the end devices that produce the data. By keeping the data processing closer to the devices, we improve network latency, increase control over who gets to see the information, and reduce the chance of successful eavesdropping by an adversary [18].

The overall system, illustrated in Fig. 3, works as follows: in the *policy learning phase*, a local server (such as a fog server) sends simulated states of the environment to AR devices in the vicinity. The users then respond with the actions they would take in the given states. Using these traces of state-action pairs, the local server generates a set of personalized policies via the behavioral cloning algorithm. In the *policy distribution and execution phase*, devices that are capable of running neural networks can download policies from the local server. More resource-poor devices can have the local server run policies for them, by sending their real-world state and receiving the recommended action from the server at each time step.

We make the following contributions in this paper:

- We **propose a formalization of AR personalization** as a supervised learning problem, in which a set of hologram placement policies are learned directly from user demonstrations. We then present an algorithm to learn, distribute, and execute these personalized policies across a heterogeneous population of AR users and devices.
- We **present an implementation of the AR personalization algorithm** in a Unity-based simulator we designed and developed. Our results demonstrate the ability of our algorithm to quickly and successfully learn users' hologram placement preferences, for both content-dependent and -independent hologram placements, as well as its robustness to noise and unreliability in user demonstrations.
- We **examine system architecture considerations** for the proposed approach, and contrast it with some alternatives. We estimate that even resource-poor AR devices that rely on local servers to perform the feedforward function of the policies at runtime are likely to achieve at least 60 frames per second when using our method. We also elaborate on the **privacy benefits** of our proposed architecture.

In Sect. III, we describe our method for learning user preferences in AR. Sect. IV presents our experimental results. Sect. V provides a discussion of system architecture considerations. Sect. VI concludes the paper.

II. Related Work

Personalizing content – e.g., news articles, ads – is the norm in web applications, and has been studied in mobile systems contexts as well [19], [20]. These approaches personalize the content of the digital experiences. In contrast, in this work we personalize *where* holographic content is placed. On cell phones and other small mobile devices, the location of digital experiences is of limited importance. The immersiveness of AR displays, on the other hand, brings about multiple paradigm shifts in user experiences [15]. We argue that in immersive experiences, personalizations are important for user-friendliness, quality of experience, and safety of diverse users.

Traditionally, hologram placements in AR are pre-programmed and are not customized for the different users. Broadly, our work can be seen as an example of using advanced machine learning algorithms to automatically generate policies for decisions traditionally done via heuristics [21], [22], [23]. For such techniques, it is important to find state space representation that keep the problem tractable [22]. Our approach to state space reduction is inspired by AR-specific system abstractions developed in [14]. That work, however,

focused on a different problem – securing AR outputs – and on the development of fixed rules, rather than personalized techniques.

The specific machine learning technique, behavioral cloning, which we employ in this work, has seen great success in several recent applications involving autonomous agents, including quadrotors [24] and self-driving cars [16]. We are unaware of previous work on using behavioral cloning for enhancing AR experiences.

III. Personalized Hologram Placement

Toward the goal of making AR experiences more personalized, we propose using behavioral cloning to directly learn a hologram placement policy based on user guidance. This personalization can occur during the setup phase (or in the device settings), when a user is first configuring or updating his/her AR device. We begin with a brief overview of imitation learning. We then formalize the AR personalization problem, and present our solution based on behavioral cloning.

A. Imitation Learning Overview

In the standard reinforcement learning (RL) problem, an agent is tasked with learning a desired behavior through trial and error with the aid of a given reward function for evaluating performance. However, what if we instead want the agent to learn by observing an expert perform that task? This is the main idea of *imitation learning* (also known as *apprenticeship learning* or *learning from demonstration* [25]), which is particularly useful if the reward function itself is difficult to specify. More formally: given execution traces from an expert, can we imitate the expert's policy?

Behavioral cloning [26] is an approach to imitation learning in which policies are estimated directly from a trace (i.e., a set of demonstrations) of the expert's policy $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$ that maps states to actions, where $\mathcal{S}$ is the state space and $\mathcal{A}$ is the action space. This trace is viewed as the "training set," and is represented as a sequence of state-action pairs $\{(s_0, a_0), (s_1, a_1), (s_2, a_2), \ldots\}$. Any supervised learning method of choice can be applied to fit a model to the data, e.g., neural networks, SVMs, decision trees, etc.[1]

B. Problem Formalization

We consider a scenario in which N users in close proximity to one another are employing AR devices, which may be mobile phones, HMDs, heads-up displays, or others. At each time step t, suppose a nearby server (such as a fog server) presents user i with a random, simulated environmental state denoted by $s_t^i \in \mathcal{S}$, which may consist of the locations and rotations of real-world and virtual objects, the sizes of their bounding boxes, and features of the application content (e.g., category). Additionally, the local server may ask the user for some specific attributes, such as height or eyesight, which can be incorporated into the state. In general, the server can generate these simulated states based on environmental context, by incorporating statistics about the real-world environment (e.g., locations, sizes, and frequencies of real-world objects based on sensors or video feeds). These statistics can help the local server create more realistic, context-specific simulations, which can then improve the efficiency of training and accuracy of the resulting policy.

User i then sends back its chosen action $a_t^i \in \mathcal{A}$, which encodes how the user displaced the holograms from their original positions. Thus, at each time step, the local server collects N state-action pairs $(s_t^i, a_t^i),\ i = 1, \ldots, N$. Over some period of time T, the local server collects T samples per user, which forms a training set with $N \times T$ total examples:

$$\begin{bmatrix} (s_0^1, a_0^1) & (s_1^1, a_1^1) & (s_2^1, a_2^1) & \cdots & (s_T^1, a_T^1) \\ (s_0^2, a_0^2) & (s_1^2, a_1^2) & (s_2^2, a_2^2) & \cdots & (s_T^2, a_T^2) \\ (s_0^3, a_0^3) & (s_1^3, a_1^3) & (s_2^3, a_2^3) & \cdots & (s_T^3, a_T^3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (s_0^N, a_0^N) & (s_1^N, a_1^N) & (s_2^N, a_2^N) & \cdots & (s_T^N, a_T^N) \end{bmatrix} \quad (1)$$

Here, each column of the training set gives the state-action pairs of all users at a particular time step, and each row gives the state-action pairs of a specific user during the entire duration of training. In practice, the size of N depends on the number of users typically in the vicinity of the local server (ranging from one to hundreds). The number of demonstrations per user on the other hand would depend on the complexity of the environments/state spaces; for our experiments, we find that $T = 10 - 20$ suffices for good performance.

From this training data, the local server wishes to generate a set of policies that closely mimic the users' demonstrations.

C. Learning Policies with Behavioral Cloning

At this stage, the local server can do one of two things:

1) Assume that each user has sufficiently different placement preferences, such that each row of the training set can be viewed as a distinct trace $d_i,\ i = 1, \ldots, N$. Then, for each user, the local server can estimate a placement policy π_i based solely on d_i. This would generate a total of N different policies.
2) Identify similar users, and merge their training data to produce a joint policy. This approach has the benefit of more efficient training due to distributed data collection and aggregation. After this merging process, the local server has K different training sets $d_1, d_2, \ldots, d_K,\ K < N$, which lead to K different policies $\pi_1, \pi_2, \ldots, \pi_K$.

Regardless of which approach is used, each policy itself is learned using supervised learning. In our work, we employ a multi-layer neural network for function approximation, where the input and output layers correspond to states and actions, respectively. If approach (2) is used, we must identify a notion of "similarity" and how it can be estimated by the local server. A potential approach is for the local server to present the users with identical states $s_i^t = s^t, \forall i$, for a subset of the samples, and measure similarity based on the distance between the users' displaced holograms in these states.

[1] Another approach to imitation learning is *inverse reinforcement learning* (IRL) [27], where the reward function R is learned based on samples of the expert's policy, and then used to generate an estimated policy. We focus on behavioral cloning in this paper, and leave IRL to be explored in future work.

Algorithm 1 Learning Personalized Hologram Placements

procedure LEARNHOLOGRAMS
Input: Number of users N, horizon T
Output: Policies $\pi_1, \pi_2, \ldots, \pi_K$, where $K \leq N$
Initialize: $t \leftarrow 0$
While $t < T$:
    Send out simulated states s_t^i to users $i = 1, \ldots, N$
    Collect actions a_t^i from users
    $t \leftarrow t + 1$
End while
Generate K different policies from user demonstrations (s_t^i, a_t^i) using behavioral cloning
Return $\pi_1, \pi_2, \ldots, \pi_K$

Suppose the local server adopts the first approach. Then each user i generates an individual training trace d_i by providing a set of actions in response to the simulated states sent from the server. A multi-layer neural network is used to approximate a policy π_i based on d_i. The learning method is summarized in Algorithm 1. Note that the local server can re-run this algorithm whenever new users enter the vicinity.

D. Distributing and Executing Policies

After the policies are learned, they are cached in the local server and either distributed to users with more powerful AR devices or run on behalf of resource-poor devices.

Powerful Devices. First, we consider the case where the user's AR device is powerful enough to run a policy (i.e., a multi-layer neural network) itself. For example, the next iteration of Microsoft's HoloLens will contain a dedicated processor for implementing deep neural networks [28]. These devices can simply download and run policies themselves, thereby reducing reliance on (and communication costs from) the local server. This approach is likely to improve frame rate, as the device does not need to wait the round-trip time of sending the environmental state and receiving an action from a server in order to run a policy.

Recall that in the policy learning phase, the local server sends the user a set of *simulated* states meant to capture the dynamics of the local environment. In the execution phase, however, the AR device must compute its own state based on what it observes *in real time*. From the device's raw video stream, it must extract the relevant features of the state using real-time image processing techniques such as the popular YOLO object detection algorithm [29] or the HoloLens's spatial mapping functionality.

Resource-Constrained Devices. If resource constraints prevent an AR device from running the policy itself (e.g., smartphones or low-power HMDs), then the local server can compute the policy output based on the state sent from the device. As a consequence, the device must communicate much more frequently with the local server to send states and receive corresponding actions over the network. As with powerful AR devices, image processing techniques will likely be required to compute the state. However, for especially resource-poor devices, raw images will have to be offloaded to the local server or another edge node to be processed first. Edge-based offloading of deep learning for video applications has been explored in [11] and similar works.

TABLE I: Behavioral cloning hyperparameters in the experiments.

Parameter	Value
Batch size	16
Batches per epoch	5
β	5×10^{-3}
ϵ	0.2
Number of hidden layers	4
Hidden units	64
Learning rate	3×10^{-4}
Max steps	3×10^{4}
Number of epochs	3
Time horizon	64

IV. EVALUATION

Using an AR simulator that we implemented, we first demonstrate that relatively few user demonstrations suffice to generate a policy which accurately captures the user's preferences. We next show that, by incorporating additional features about the application content into the policy state, the local server can learn where users prefer to place certain categories of holograms. Even in the presence of noise and unreliability in the user demonstrations, the learning process remains fairly robust. Finally, by reducing the dimensionality of the state space from a full image to carefully chosen features, we preserve user privacy, reduce training time, and minimize communication costs.

A. Evaluation Setup

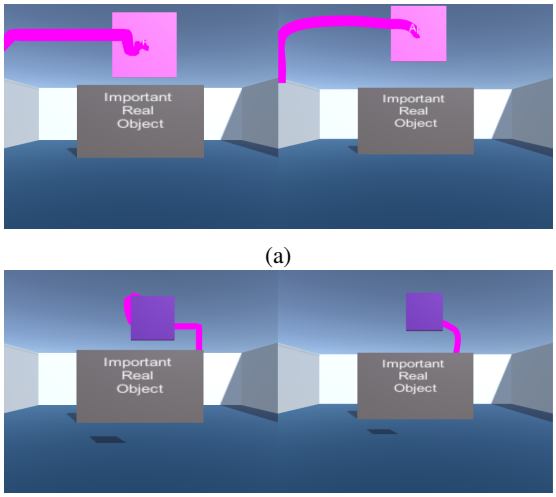
Our local server has an Intel Core i7-7700K CPU @ 4.20 GHz, 16 GB of RAM, and an NVIDIA GeForce GTX 1070 graphics card with 8 GB of RAM and 1920 CUDA cores.

Using the Unity game engine, a platform commonly used to create AR experiences, we built an AR simulator with C# that randomly generates holograms (represented as colored tiles to resemble applications) in a virtual HMD-like experience. A large sign reading "Important Real Object" is placed in the center of the environment both to serve as part of the state and to remind the viewer that anything other than the holograms constitutes the "simulated real world."

On top of the AR simulator, we implemented the behavioral cloning algorithm with a multi-layer neural network using the Unity Machine Learning Agents Toolkit [30]. There are two key components to behavioral cloning in this implementation:

1) *Teacher agent*: The agent controlled by the user, who provides a state-action pair at every video frame, which is then broadcast to the student agent. The complete trace of state-action pairs serves as the training set (full demonstration).
2) *Student agent*: The completely autonomous agent that observes and tries to learn from the teacher agent's (user's) demonstrations. The student agent updates its policy as new demonstrations are provided.

We modified our AR simulator to be split-screen, such that during the learning phase, the user can provide demonstrations



(a) (b)

Fig. 4: Two examples of behavioral cloning. In each example, the left screen shows the user-controlled hologram, while the right screen shows the student agent-controlled hologram. The bright pink trail is the hologram's trajectory.

(a) (b)

Fig. 5: Birds-eye view of the examples given in Fig. 4. The agent learns both the preferred final position of the holograms, as well as the trajectory taken by the user.

on the left screen while simultaneously viewing the student agent's progress on the right.

At the beginning of each trial (i.e., user demonstration) in the learning phase, a hologram of random size and color is placed in a random initial location in the simulated environment and presented to the user as though he/she is looking through an HMD. The user uses the keyboard to reposition the hologram as desired. From these demonstrations, the agent estimates a policy (represented as a neural network) that best estimates the user's behavior. After 500 frames have elapsed, the environment is reset and a new trial begins.

We employed a standard neural network architecture for behavioral cloning, in which an input layer takes a feature vector representing the state of the environment, feeds this through four hidden layers, and produces at the output layer an action vector indicating how the holograms should be displaced from their current positions. Table I lists other hyperparameters that were used in training. Since the "max steps" parameter was set to 3×10^4 frames, and each user demonstration lasts 500 frames, a complete training set comprises a total of 60 user demonstrations. In each of the following experiments, we ran 50 full training sessions, which is equivalent to training 50 independent and identical agents in the same environment.

B. Content-Independent Hologram Placement

We first show that the student agent is able to learn the user's placement preference of a single hologram, independent of the application's content. That is, the agent's goal is simply to mimic the way in which the user moves around a hologram from arbitrary initial positions in the environment. The state of the environment at every video frame is given by the coordinates of the hologram and real-world object, and the size of their bounding boxes. Including features of real-world objects in the state is important because users may have different placement preferences depending on how real objects are configured. The action vector indicates how much to move the hologram along the x, y, and z directions.

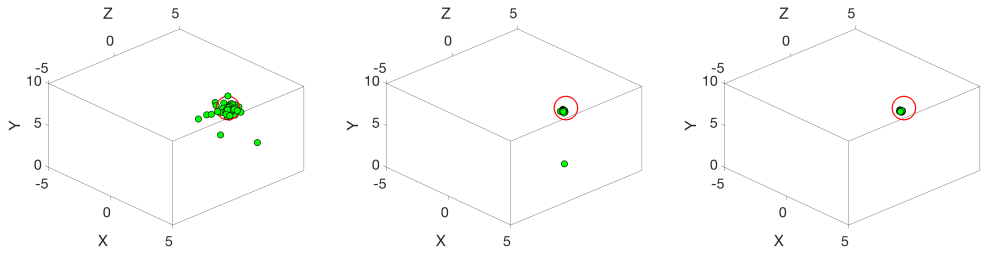
Qualitative comparisons between the user and agent hologram placements are given in Fig. 4. Observe that the agent is not only able to learn the preferred final position of the holograms, but also the *physical trajectory* taken by the user. The full training session was $\approx$ 7 minutes; but the agent's actions began to closely match those of the user in much less than the allotted time. Furthermore, we see in Fig. 5, which depicts the birds-eye view of the examples given in Fig. 4, how the size of the hologram bounding box comes into play: the agent learns that the user prefers smaller holograms to be placed closer to them, to maintain a similar level of visibility as a larger hologram. A particularly useful benefit of this is that *the agent can adapt to users with different strengths of eyesight*.

To provide a more quantitative analysis of learning ability, we defined a deterministic (noiseless) function for controlling the behavior of the teacher agent. Given an initial hologram position selected by the AR content generator, and a target destination representing the user's preferred final position, the teacher agent moves the hologram in a perfectly straight line from start to finish.

We collected data on 50 training sessions, each with 60 demonstrations from the teacher agent. At the end of each demonstration, we recorded the final destinations of both the teacher agent's and student agent's holograms. Fig. 6 shows these destinations for all 50 training sessions, at the beginning, middle, and end of training. The policies estimated by the student agent early in training occasionally yield actions that deviate widely from the target. However, as more user demonstrations from the teacher are provided, the student agent's accuracy and precision improve.

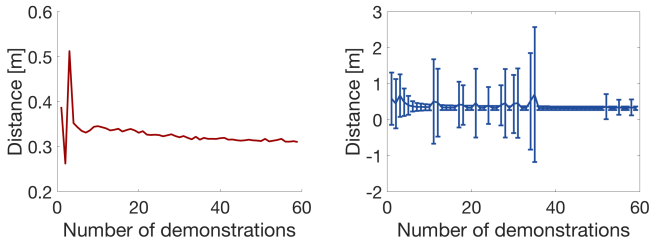
The metric we use to assess the performance of the student agent is the distance between its hologram and the teacher agent's hologram at the end of each trial. Fig. 7 shows the median and mean (with standard error) of this metric over the duration of training. Overall, the agent's actions converge quickly to the target, but with high variability near the beginning of training. A couple of outliers around the 30-40 demonstration mark result in high standard error.

Fig. 8 shows the empirical cumulative distribution function (CDF) of the number of demonstrations needed for the student agent's hologram to converge to the target destination. Here, we define convergence to have occurred when the agent moves the hologram to a specified radius threshold (in meters)



(a) 1 demo (b) 20 demos (c) 40 demos

Fig. 6: Content-independent hologram destinations at the beginning, middle, and end of training, shown for 50 training sessions. The green circles represent the hologram destinations chosen by the agent, while the red circle represents the target destination. As more user demonstrations from the teacher are provided, the student agent's accuracy and precision improve.



(a) Median (b) Mean

Fig. 7: Median and mean distances between the agent-positioned hologram and the target in the content-independent case, over 50 training sessions. The agent's actions converge quickly, but with high variability in the beginning of training.

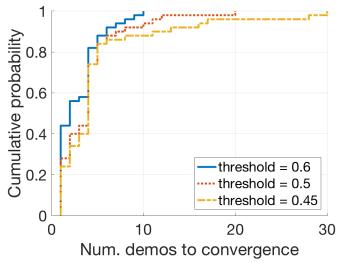


Fig. 8: Content-independent empirical CDF of user demos required for the agent to converge to the target. Convergence occurs in ≤ 10 demos w.p. $\geq 80\%$ for all three thresholds.

from the target, three consecutive times. For the strictest threshold shown (0.45 meters), the policy converges in under 10 demonstrations more than 80% of the time. Successively looser thresholds lead to faster convergence times.

C. Content-Dependent Hologram Placement

We next test the student agent's learning ability when features of the hologram content are included in the state. The motivation is that users may prefer having different types of applications (e.g., email or navigation) in different positions in the environment. In our AR simulator, we use the hologram's color to represent its application category. This color is then encoded as an integer in the environmental state. Our experiment uses two different categories, represented by red and blue, and we apply a deterministic function that moves holograms in a straight line to one of two separate target destinations (depending on the hologram's color).

We generate a similar set of figures to those in the previous experiments. Fig. 9 shows the student agent-driven destinations of the blue and red holograms at three different points in the training process. Early on, the student agent drives most of the holograms to the correct target, with a few misclassifications. However, even as early as 13 demos, the agent achieves full accuracy. Policy convergence is demonstrated in Figs. 10 and 11. Convergence is again defined with respect to a distance threshold from the correct target.

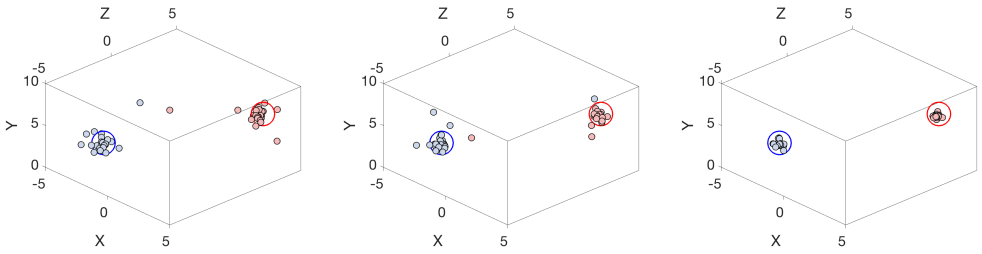
D. Learning in the Presence of Noise

In practice, slightly inconsistent or noisy user actions are inevitable. We showed qualitatively in Figs. 4 and 5 that the agent is able to closely match the overall trajectory of the user's hologram. In those experiments, the user was an actual person whose demonstrations were inherently noisy.

We further investigate learnability in the presence of noisy demonstrations by modifying the deterministic function used in the previous experiments. The teacher agent first computes the straight-line trajectory from the hologram's initial position to the target destination (as usual), but uniformly random noise in the range $[-0.05, 0.05]$ meters is added along the x, y and z directions to the original action vector in each video frame. Examples from training in the content-independent and content-dependent cases are shown in Figs. 12 and 13, respectively. In both cases, the agent's hologram placements at the beginning of training exhibit noticeably higher variability. Ultimately, however, the destinations converge to the correct target (with the exception of one data point in each figure, which lies slightly outside of the target).

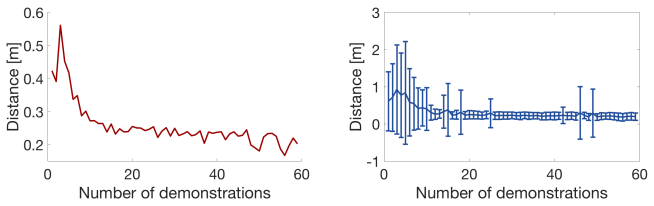
E. Benefits of Dimensionally Reduced State Space

Many current AR and video applications require entire image frames to be offloaded to an external cloud service. Our behavioral cloning approach, in contrast, uses a reduced, feature-based state vector that constitutes a small portion of the environment's overall complexity, but still captures an adequate amount of information for the student agent to learn properly (as supported by the previous experiments). We sought to investigate the effect of using the full image as the state on the training time and policy accuracy.



(a) 4 demos (b) 6 demos (c) 13 demos

Fig. 9: Content-dependent hologram destinations at three different stages of the learning process. The color of the circles and dots corresponds to the application category. Even as early as 13 demos, the agent achieves full accuracy.



(a) Median (b) Mean

Fig. 10: Median and mean distances between the agent-positioned hologram and the target in the content-dependent case. Again, learning converges quickly, but has high variability in the beginning of training.

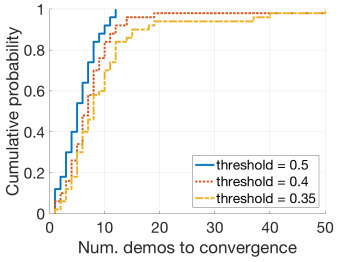
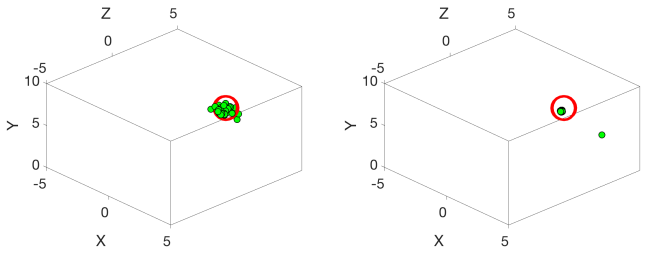
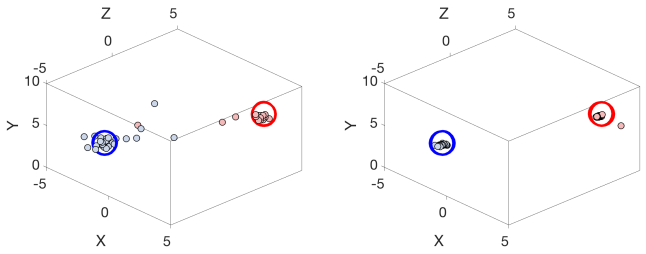


Fig. 11: Content-dependent empirical CDF. Convergence occurs in ≤ 15 demos w.p. $\geq 80\%$ for all three thresholds.



(a) 1 demo (b) 60 demos

Fig. 12: Content-independent learning with noise.



(a) 2 demos (b) 60 demos

Fig. 13: Content-dependent learning with noise.

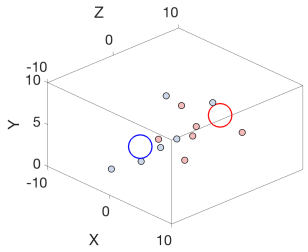


Fig. 14: Hologram destinations after 12 trials when using the full AR image as the state, in lieu of select features. Compared to Fig. 9 there is a clear degradation in accuracy.

We configured the Unity AR simulator to obtain the state in each frame by converting what is seen by the agent's camera into a `Texture2D` object (essentially an image) of size 460×315. The student agent was tasked with learning content-dependent hologram placements from noiseless demonstrations. Everything except for the state representation was kept identical to the previous content-dependent experiments.

Over the course of 3 hours, only 12 demonstrations were completed for a single agent. This rate of training is prohibitively slow – about 8 times longer than using the reduced state space. After 12 trials were completed, the training system crashed from a memory error. Fig. 14 plots all of the destination data that was collected. When compared to Figs. 9 and 13, there is a clear degradation in hologram accuracy and precision – none of the agent's 12 attempts landed within one of the target circles, let alone within the correct one for the application category. Thus, a reduced state representation we have employed in this work can help reduce training time, minimize communication costs (with respect to both time and energy), and facilitate privacy preservation, all without loss of policy performance.

V. System Architecture Considerations

This section discusses the use of local servers, the expected communication costs, and privacy considerations for personalized AR experiences.

A. Local servers and AR

Local servers, e.g., fog and edge computing nodes, have been proposed as an aid to AR by many researchers [11], [31]. In particular, the latency demands of AR make fog/edge computing an attractive alternative to more geographically

distant cloud services [9]. Our methods can be integrated within a fog computing architecture.

Specifically, the Microsoft HoloLens developer guidelines state that a steady 60 FPS must be maintained to create an ideal AR experience [32]. In terms of latency, this is equivalent to the operating system receiving a new video frame (i.e., image) every 16 ms. For games and other immersive applications, it is crucial to have latencies under 20 ms [10]. We argue that our approach to personalized AR experiences adheres to these strict performance requirements, as detailed below.

B. Communication Costs

1) Policy Learning Phase: In the learning phase, communication costs are incurred by the transfer of state and action information between the user and the local server. Suppose each state and action is encoded as an n- and m-dimensional vector of 32-bit floats, respectively. Then at each time step, the local server sends $32 \cdot n \cdot N$ bits for the N total states sent to the users. The server also receives $32 \cdot m \cdot N$ bits for the actions returned by the users. Thus, in total, the communication cost of the learning phase is $32 \cdot N \cdot (n + m)$ bits per time step. Assuming the dimensions of the state and action spaces are similar (i.e., $n \approx m$), then there is roughly symmetrical utilization of the downlink and uplink data rates. This is in contrast to the asymmetric link utilization that is typical of most AR scenarios (in which more data is offloaded rather than downloaded by the AR device). The learning phase communications are not delay-sensitive. Traditional content transfer protocols fully satisfy the requirements of this phase.

2) Policy Distribution and Execution Phase: After the learning phase is complete, the degree of subsequent communication costs depends on the computational resources available to the AR device.

To gauge the complexity of policies generated with our approach, we draw a comparison to YOLO [29]. The standard YOLO architecture uses 24 convolutional layers and 2 fully connected layers. A smaller Fast YOLO architecture uses a neural network with fewer convolutional layers (9 rather than 24). YOLO maintains 45 FPS on average, while Fast YOLO maintains 155 FPS. In contrast, our neural network architecture – with only an input layer, 4 hidden layers, and an output layer – is much simpler than YOLO's. Moreover, we use select features of the environmental state as the input vector, rather than the entire image (as YOLO does). From our experiments in Sect. IV, we also find that the typical policy file size is 58 KB for a simple environment. Thus, the main limiting factor of our approach in most cases will be communication, rather than computation.

Powerful Devices. In this case, the device needs to download the policy from the local server only once, at initialization; so the communication cost is limited to the size of the policy. This is akin to installing an app on a device.

Resource-Constrained Devices. We analyze the communication costs of having the local server perform the feedforward function of the policies at runtime, on behalf of the devices.

We measured ICMP ping round-trip time between nodes located in different places with respect to one another, with nodes connected over WiFi. The median round-trip latencies ranged from 0.9 ms for the devices located in the same room to 2.3 ms for the devices located in the same building to 16 ms between a fog node located on campus and a device located a couple of miles away. Similar latencies were observed in prior studies as well [31].

In a smart city setting, it is therefore reasonable to assume that the round-trip latency between AR devices and a nearby server is $\approx$ between 2 ms and 16 ms. The remaining latency is due to running the policy itself on the local server. Using the neural network policies generated in our experiments, as described in Sect. IV, we found the policy execution time to be negligible (on the order of microseconds) for a relatively sparse environment. We can then estimate the total latency for an AR device to send its state and receive an action from the server to be 2 ms - 16 ms. Thus, even a resource-poor AR device is likely to achieve at least 60 FPS. For more complex environments with high-dimensional state and action spaces, the overall latency is still expected to be well within that required for a seamless AR experience.

However, the overall frame rate could be rate-limited by image processing techniques that may exist earlier in the pipeline. For example, if the full version of YOLO is used to extract features from the AR device's video stream, then the frame rate may be limited to roughly 45 FPS. This can be mitigated by using Fast YOLO or another simpler algorithm.

C. Privacy Considerations

Our behavioral cloning approach to hologram personalization applies the principles of *focused collection* and *data minimization* proposed by the White House in 2012 [33]. The main idea is that *only what needs to be collected should be collected.* This notion is especially pertinent in AR, where the common practice is to offload raw video streams and other sensor data to the cloud for processing, which may compromise the privacy of both the user and nearby people who are unknowingly tracked by these sensors. For the purposes of personalized hologram placement, we argue that less is more: by limiting the state space to only what is necessary to generate policies, we preserve user privacy, improve learning speed, and minimize storage and communication costs.

There is also an inherent tradeoff between personalization and privacy preservation, as fitting a policy to a user's preferences requires access to potentially private information.

Our approach helps preserve privacy by keeping the sensitive data closer to users (i.e., either on a local server or on their personal devices). In this setting, if an adversary wants to perform an attack, it has to target specific local servers or devices, each of which contains only a small portion of the overall data. In contrast, using a centralized cloud service would place all users in a vulnerable position. So, even if certain applications call for entire images to be offloaded to enable learning, the data would be distributed across multiple nodes in our approach, limiting privacy loss compared to centralized data collection. Similar edge-based learning solutions have been recently proposed for personalized recommendation [20], and activity recognition and topic identification tasks [34].

A potential drawback of this approach is the limited amount of data available locally per user. In these scenarios, our batch-training (similarity-based) approach provides a *collaborative* personalized model training solution: a trusted shared server can be used by a group of users with similar interests for privacy-preserving training, acting as a middle ground between the cloud-level and device-level training alternatives.

VI. Conclusions

We proposed an (edge-based) behavioral cloning approach to the new area of AR personalization. By using a reduced state space that captures only the necessary features of the user's perspective to send to a local server for learning, the approach offers many benefits such as privacy preservation, faster training time, and low-cost communication. Our empirical results demonstrate that an autonomous agent can quickly and accurately learn a user's preferences for hologram placements (in both the content-independent and -dependent cases), even in the presence of noisy demonstrations.

References

[1] D. Schmalstieg and T. Höllerer, *Augmented Reality: Principles and Practice*. Boston, MA: Addison-Wesley, 2016.

[2] G. Sachs. (2016, January) Virutal & augmented reality. [Online]. Available: http://www.goldmansachs.com/our-thinking/pages/technology-driving-innovation-folder/virtual-and-augmented-reality/report.pdf

[3] Google. (2017) Arcore. [Online]. Available: https://developers.google.com/ar/

[4] Microsoft. (2016) Hololens. [Online]. Available: https://www.microsoft.com/en-us/hololens

[5] M. Gurman. (2017, November) Apple is ramping up work on AR headset to succeed iPhone. [Online]. Available: https://www.bloomberg.com/news/articles/2017-11-08/apple-is-said-to-ramp-up-work-on-augmented-reality-headset

[6] M. May. (2015, August) Augmented reality in the car industry. [Online]. Available: https://www.linkedin.com/pulse/augmented-reality-car-industry-melanie-may/

[7] H. Schaffers, N. Komninos, M. Pallot, B. Trousse, M. Nilsson, and A. Oliveira, "Smart cities and the future internet: Towards cooperation frameworks for open innovation," in *The future Internet assembly*. Springer, 2011, pp. 431–446.

[8] L. Sanchez, L. Muñoz, J. A. Galache, P. Sotres, J. R. Santana, V. Gutierrez, R. Ramdhany, A. Gluhak, S. Krco, E. Theodoridis *et al.*, "Smartsantander: IoT experimentation over a smart city testbed," *Computer Networks*, vol. 61, pp. 217–238, 2014.

[9] C. Westphal. (2017, March) Challenges in networking to support augmented reality and virtual reality. [Online]. Available: https://www.ietf.org/proceedings/98/slides/slides-98-icnrg-challenges-in-networking-to-support-augmented-reality-and-virtual-reality-cedric-westphal-00.pdf

[10] T. Braud, F. H. Bijarbooneh, D. Chatzopoulos, and P. Hui, "Future networking challenges: The case of mobile augmented reality," in *Proc. IEEE ICDCS'17*, 2017.

[11] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "DeepDecision: A mobile deep learning framework for edge video analytics," in *Proc. IEEE INFOCOM'18*, 2018.

[12] F. Roesner, T. Kohno, and D. Molnar, "Security and privacy for augmented reality systems," *Communications of the ACM*, vol. 57, no. 4, pp. 88–96, 2014.

[13] S. Ahn, M. Gorlatova, P. Naghizadeh, M. Chiang, and P. Mittal, "Adaptive fog-based output security for augmented reality," in *Proc. ACM SIGCOMM VR/AR Network Workshop*, 2018.

[14] K. Lebeck, K. Ruth, T. Kohno, and F. Roesner, "Arya: Operating system support for securely augmenting reality," in *Proc. IEEE Symposium on Security and Privacy*, January 2018.

[15] ——, "Towards security and privacy for multi-user augmented reality: Foundations with end users," in *Proc. IEEE Symposium on Security and Privacy*, 2018.

[16] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv:1604.07316*, 2016.

[17] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the internet of things," in *Proc. MCC'12*, 2012.

[18] M. Chiang and T. Zhang, "Fog and IoT: An overview of research opportunities," *IEEE Internet of Things J.*, vol. 3, no. 6, pp. 854–864, December 2016.

[19] M. Hardt and S. Nath, "Privacy-aware personalization for mobile advertising," in *Proc. ACM CCS'12*, 2012.

[20] S. Jain, V. Tiwari, A. Balasubramanian, N. Balasubramanian, and S. Chakraborty, "PrIA: A private intelligent assistant," in *Proc. ACM HotMobile'17*, 2017.

[21] H. Mao, M. Alizadeh, I. Menache, and S. Kandula, "Resource management with deep reinforcement learning," in *Proc. ACM HotNets'16*, 2016.

[22] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang, "Machine learning for networking: Workflow, advances and opportunities," in *IEEE Network*, Nov. 2017.

[23] A. Mirhoseini, H. Pham, Q. V. Le, B. Steiner, R. Larsen, Y. Zhou, N. Kumar, M. Norouzi, S. Bengio, and J. Dean, "Device placement optimization with reinforcement learning," in *Proc. ICML'17*, 2017.

[24] A. Giusti, J. Guzzi, D. C. Ciresan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro *et al.*, "A machine learning approach to visual perception of forest trails for mobile robots." *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 661–667, 2016.

[25] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and autonomous systems*, vol. 57, no. 5, pp. 469–483, 2009.

[26] M. Bain and C. Sommut, "A framework for behavioural cloning," *Machine intelligence*, vol. 15, no. 15, p. 103, 1999.

[27] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proc. ICML'04*, 2004.

[28] Microsoft. (2017) Second version of HoloLens HPU will incorporate AI coprocessor for implementing DNNs. [Online]. Available: https://www.microsoft.com/en-us/research/blog/second-version-hololens-hpu-will-incorporate-ai-coprocessor-implementing-dnns/

[29] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *arXiv:1506.02640*, 2015.

[30] Unity. (2018) Unity ml-agents toolkit. [Online]. Available: https://github.com/Unity-Technologies/ml-agents

[31] Z. Chen, W. Hu, J. Wang, S. Zhao, B. Amos, G. Wu, K. Ha, K. Elgazzar, P. Pillai, R. Klatzky, D. Siewiorek, and M. Satyanarayanan, "An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance," in *Proc. ACM SEC'17*, Oct. 2017.

[32] Microsoft, "Hologram stability," 2018. [Online]. Available: https://docs.microsoft.com/en-us/windows/mixed-reality/hologram-stability

[33] White House Report, "Consumer data privacy in a networked world: A framework for protecting privacy and promoting innovation in the global digital economy," *Journal of Privacy and Confidentiality*, 2012.

[34] S. Servia-Rodríguez, L. Wang, J. R. Zhao, R. Mortier, and H. Haddadi, "Privacy-preserving personal model training," in *Proc. IEEE IoTDI'18*, 2018.