ARGUS: Visualization of AI-Assisted Task Guidance in AR

Sonia Castelo*, Joao Rulff*, Erin McGowan*, Bea Steers, Guande Wu, Shaoyu Chen, Iran Roman, Roque Lopez, Ethan Brewer, Chen Zhao, Jing Qian, Kyunghyun Cho, He He, Qi Sun, Huy Vo, Juan Bello, Michael Krone, and Claudio Silva

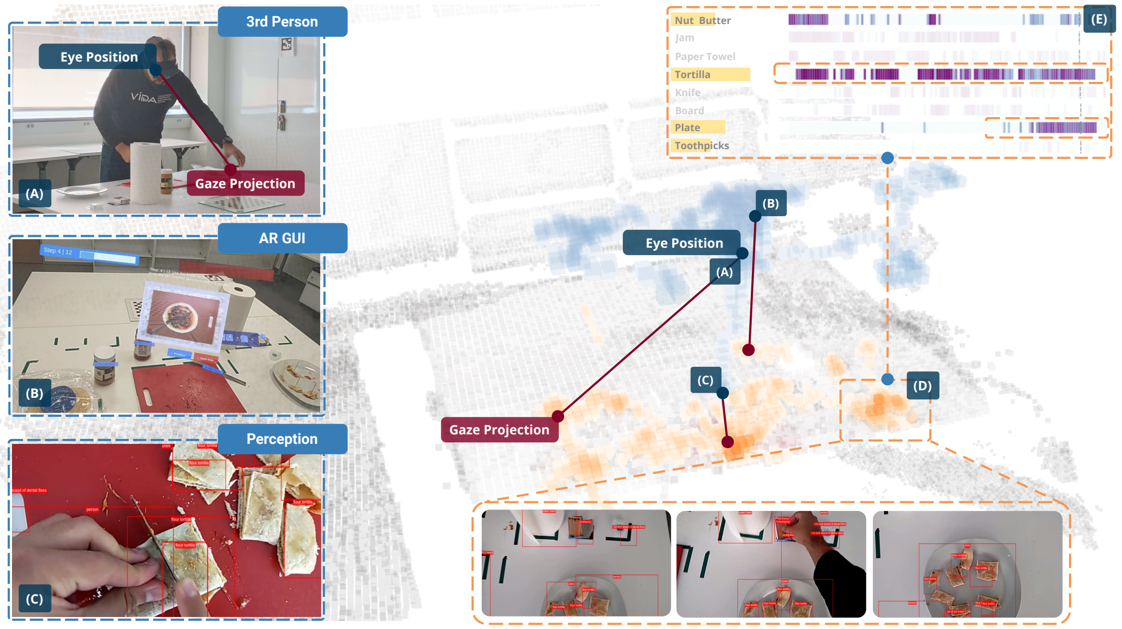


Fig. 1: *ARGUS* is a visual analytics tool for real-time and historical evaluation of sensor and model outputs of AR task assistants. Shown here are (A) 3rd-person perspective of a human (*performer*) performing a task with AR headset guidance, (B) the AR GUI from the perspective of the performer, (C) a snapshot of a visual perception model analyzing data from the headset camera. For each time step, detailed information highlights the performer's gaze direction and the corresponding frame recorded by the headset. (D) heatmaps of the performer's gaze projection onto the world point cloud allow for inspection and understanding of their attention over time. The cluster on the left shows the performer finalizing a recipe. (E) object detection information from *ARGUS* Temporal View illustrating that *Plate* was only detected towards the end of the task while *Tortilla* was detected throughout the whole session.

Abstract—The concept of augmented reality (AR) assistants has captured the human imagination for decades, becoming a staple of modern science fiction. To pursue this goal, it is necessary to develop artificial intelligence (AI)-based methods that simultaneously perceive the 3D environment, reason about physical tasks, and model the performer, all in real-time. Within this framework, a wide variety of sensors are needed to generate data across different modalities, such as audio, video, depth, speech, and time-of-flight. The required sensors are typically part of the AR headset, providing performer sensing and interaction through visual, audio, and haptic feedback. AI assistants not only record the performer as they perform activities, but also require machine learning (ML) models to understand and assist the performer as they interact with the physical world. Therefore, developing such assistants is a challenging task. We propose ARGUS, a visual analytics system to support the development of intelligent AR assistants. Our system was designed as part of a multi-year-long collaboration between visualization researchers and ML and AR experts. This co-design process has led to advances in the visualization of ML in AR. Our system allows for online visualization of object, action, and step detection as well as offline analysis of previously recorded AR sessions. It visualizes not only the multimodal sensor data streams but also the output of the ML models. This allows developers to gain insights into the performer activities as well as the ML models, helping them troubleshoot, improve, and fine-tune the components of the AR assistant.

Index Terms—Data Models; Image and Video Data; Temporal Data; Application Motivated Visualization; AR/VR/Immersive.

1 Introduction

The concept of an augmented reality (AR) assistant has captured the human imagination for years, becoming a staple of modern science fiction through popular franchises such as the *Marvel Cinematic Universe*, *Star Trek*, and *Terminator*. The applications of such a system are seemingly endless. Humans, even those with domain expertise, are fallible creatures with imperfect memories whose skills deteriorate over time, especially during repetitive tasks or under stress. An AR assistant could help experts and novices alike in performing both familiar and new tasks. For instance, an AR assistant could aid a surgeon performing a

*All authors are with the New York University. E-mails: {s.castelo, jlrulff, erin.mcgowan, bs3639, guandewu, sc6439, irr2020, rlopez, ethan.brewer, cz1285, jq2267, kyunghyun.cho, hh2291, qs2053, huy.vo, jpbello, mk8949, csilva}@nyu.edu. * These authors contributed equally and should be regarded as joint first authors.*

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org. Digital Object Identifier: xx.xxxx/TVCG.201x.xxxxxxx

familiar yet complex procedure, who could benefit from a second set of "eyes" due to the high-stakes nature of their task. Equally, it could walk an amateur chef through the steps of an unfamiliar recipe. In an ideal scenario, the AR assistant would become "invisible" in the sense that it is seamlessly integrated into the task procedure, providing well-timed audio and visual feedback to guide uncertain performers and correct human errors while otherwise fading into the background. Overall, the AR assistant would be able to reduce human error via correction, improve performance by reducing cognitive load, and introduce new tasks across a wide variety of applications.

While aspects of this vision are currently still aspirational, we are finally beginning to develop the technology that allows concepts once relegated to the world of science fiction to become reality. With respect to machine perception, the recent explosion of research on machine learning (ML), especially deep neural networks, has given way to powerful models able to detect objects, actions, and speech in real time with high accuracy. Ever-evolving implementations of Bayesian neural networks, reinforcement learning, and dialog systems (e.g., conversational agents) allow for task modeling and transactional question answering. A rise in AR technology, especially the commercial availability of headsets such as Microsoft HoloLens 2, Magic Leap, Google Glass, or Meta Quest Pro (and soon, Apple Vision Pro) has provided the hardware necessary for task guidance. The time is ripe for the development of assistive AR systems.

Challenges in perceptually-enabled task guidance. Developing an AR assistant, however, comes with a host of challenges. Such a system requires several moving parts to work in tandem to perceive the performer's environment and actions, reason through the consequences of a given action, and interact with both the performer and the user (for the sake of clarity, we will refer to subjects using the AR system to perform tasks during a session as "performers" and subjects using *ARGUS* to collect and analyze data as "users"). Creating these parts is a complex, time and computational resource-consuming process. The challenges include collecting, storing, and accessing a large volume of annotated data for model training, real-time sensor data processing for action and object recognition (or reasoning), and performer behavior modeling based on first-person perspective data collected by the AR headset (see Sec. 4 for a more detailed discussion of tasks and requirements).

Our Approach. We propose ***ARGUS*: Augmented Reality Guidance and User-modeling System**, a visual analytics tool that facilitates multimodal data collection, enables modeling of the physical state of the environment and performer behavior, and allows for retrospective analysis and debugging of historical data generated by the AR sensors and ML models that support task guidance. Our tool operates in two main modes. The online mode (see Sec. 5.1) supports real-time monitoring of model behavior and data acquisition during task execution time. This mode displays tailored visuals of real-time model outputs, which allows users of *ARGUS* to monitor the system during live sessions and facilitates online debugging. Data is saved incrementally. Once finalized, all data and associated metadata collected during the task is seamlessly stored to permanent data store with analytical capabilities able to handle both structured data generated by ML models and multimedia data (e.g. video, depth, and audio streams) collected by the headset. Our system can be used to explore and analyze historical session data by interacting with visualizations that summarize spatiotemporal information as well as highlight detailed information regarding model performance, performer behavior, and the physical environment (see Sec. 5.2).

Our design was inspired by requirements from developers of AR systems and experts that create and evaluate these systems in the context of the Defense Advanced Research Projects Agency's (DARPA) Perceptually-enabled Task Guidance (PTG) program [10]. These experts use *ARGUS* and have provided feedback throughout its development. In summary, **our main contributions are:**

- *ARGUS*, a visual analytics tool tailored to the development and debugging of intelligent assistive AR systems. It supports online monitoring during task execution as well as retrospective analysis and debugging of historical data by coupling a scalable data management framework with a novel multimodal visualization interface capable of uncovering interaction patterns between performer actions and model outputs.
- The design of novel visual representations to support complex spatiotemporal analysis of heterogeneous, multi-resolution data (i.e., data streams with different frame rates). *ARGUS*not only supports the visualization of internal AR assistant ML states in the context of the actions of the performer, but also the visualization of the interactions of the performer with the physical environment.
- We demonstrate the usefulness of *ARGUS* by a set of case studies that demonstrate real-world use of *ARGUS*, exhibiting how AR assistant developers leverage the tool to improve their systems.

This paper is organized as follows: Sec. 2 reviews the relevant literature on assistive AR systems and visualization of related data. Sec. 3 provides background and context for *ARGUS*, including the AR personal assistant framework and architecture it is designed upon. Sec. 4 specifies the requirements we aim to achieve. Sec. 5 describes *ARGUS* in detail, including all components of its online real-time debugging mode and offline data analytics mode. Sec. 6 explores two case studies in which *ARGUS* proves useful to AR task assistant developers, ending with user feedback and limitations of our system. Finally, we offer concluding remarks and future work in Sec. 7.

2 Related Work

2.1 Assistive AR Systems

The idea of using AR technologies to build assistive systems that have an internal model of the real world and are able to augment what a performer sees with virtual content dates back more than three decades [9]. Yet only recent advances in AR display technologies and artificial intelligence (AI), combined with the processing power to run the necessary computations in real time, have enabled us to start building such systems. Referring to the terminology introduced by Milgram and Kishino [34] in their seminal paper on Mixed Reality, this not only requires a *class 3 display*—a head-mounted display (HMD) equipped with see-through capability that can overlay virtual content on top of the real world—but also a great *extent of world knowledge*. That is, the environment should be modeled as completely as possible so that the assistive system can react to objects and actions in the real world. Simultaneously, the *reproduction fidelity* and the *extent of presence* of an assistive system should be minimal, since the performer needs to focus on the real world, not be immersed in virtual content. In addition, the in-situ instructions help to reduce errors and facilitate procedural tasks. To date, results are mixed for task completion time using an assistive AR system versus not, with several studies finding longer times with assistive AR systems [50,56] whereas others find the opposite [18]. Nevertheless, most studies agree that AR helps to reduce errors and overall cognitive load as it provides in-situ instruction and guidance.

AR can be enabled by a multitude of different display technologies, ranging from handheld devices like smartphones and tablets to projector-based solutions and heads-up displays found in airplanes or modern cars. We, however, focus on see-through AR HMDs for assistive AR systems, since these do not significantly encumber the performer. These headset displays do not restrict performers to a limited space and leave their hands free to execute situated tasks in the real world. Furthermore, they usually offer a wider range of built-in sensors for modeling the environment and performer such as cameras, microphones, or IMUs. See-through AR headset displays available today include Microsoft HoloLens 2 (the hardware platform used in our work) and Magic Leap 2.

As was proposed by Caudell and Mizell [9], a common use case for such systems is to support performers in repair and maintenance tasks [15,20]. Similarly, AR assistants were proposed for manufacturing, e.g., training [27] or live monitoring of production lines [5]. Another prominent area for AR assistive systems is healthcare and medicine [4], e.g., to assist surgery [40] or other procedures [22,49]. Furthermore, digital assistants can also make use of AR to enable a virtual embodiment of the assistant [25,38,44]. Most of the modern systems mentioned above integrate ML methods for specific tasks, e.g., for object or voice command recognition. However, they are mostly tailored to specific tasks and only have limited support for situated performer modeling and perceptual grounding. Integrating more complex AI methods will make the development and testing of such systems also more challenging.

To support the development of AR assistants, software toolkits have been proposed, for example, RagRug [16], which is designed for situated analysis, or Data visualizations in eXtended Reality (DXR) [46], which is specifically designed to build immersive analytics [30] applications. However, while such toolkits make it easier to develop feature-rich assistive systems that use data from the multiple sensors provided by the AR headset display and integrate AI methods, they do not offer explicit tools for external debugging of the required ML models and sensor streams. Our goal is to fill this gap with *ARGUS*. This requires visualizing the multiple data streams from the sensors as well as the output of the models.

2.2 Visualization of Multivariate Temporal Data

The visualization of multivariate temporal data is a very active field of research. A plethora of different methods and tools have been proposed which, for example, use multiple views, aggregation, and level-of-detail visualizations to represent the data efficiently. A review of these methods is beyond the scope of this paper, therefore, we refer to a number of comprehensive surveys [1, 24, 28].

There have been recent attempts to develop visualization systems to debug and understand the data acquired by multimodal, integrative-AI applications. PSI Studio, a platform to support the visualization of multimodal data streams [6] is able to provide useful visualization of sets of recorded sessions. However, it requires the user to not only compose their own visual interfaces by organizing predefined elements in a visualization canvas, but also to structure the streaming data in a predefined format, *psi-store*. Built with a similar goal, Foxglove [17] requires developers to organize their data into a Robot Operating System (ROS) environment. Moreover, these tools focus on supporting the visualization of the data streams and are not able to summarize long periods of recordings with visualizations. To the best of our knowledge, existing tools also lack the ability to debug associated ML models. Other visualization tools, such as Manifold [54], are tailored to the interpretation and debugging of general ML models. In our case, we are interested in a narrower set of ML models, those that pertain to the understanding the behavior of AI assistants, which have different requirements than other visualization systems.

3 Background: Building The TIM Personal Assistant

In this section we describe the context of the development of *ARGUS*. This includes the ecosystem of components needed to support intelligent AR assistant systems, ranging from software running on the headset device to data management modules able to ingest data in real-time.

3.1 Motivating Context

The development of *ARGUS* is driven in large part by the requirements of the DARPA PTG program [10]. PTG aims to develop AI technologies that help users perform complex physical tasks while making them both more versatile by expanding their skillset and more proficient by reducing their errors. Specifically, the program seeks to develop methods, techniques, and technology for AI assistants that provide just-in-time visual and audio feedback to help with task execution. The goal is to utilize wearable sensors (head-mounted cameras and microphones) that allow the AR assistant to see what the performer sees and hear what they hear, so that the assistant can provide helpful feedback to the performer through speech and aligned graphics. The assistants learn about tasks by ingesting knowledge from checklists, illustrated manuals, training videos, and other sources of information (e.g., making a meal from a recipe, applying a tourniquet from directions, conducting a preflight check from a checklist). They then combine this task knowledge with a perceptual model of the environment to support mixed-initiative and task-focused performer dialogs. The dialogs may assist a performer in completing a task, identify and correct errors during a task, and instruct them through a new task, taking into consideration the performer's level of expertise. As part of PTG, our team has been building TIM, the Transparent, Interpretable, and Multimodal AR Personal Assistant, which is described below.

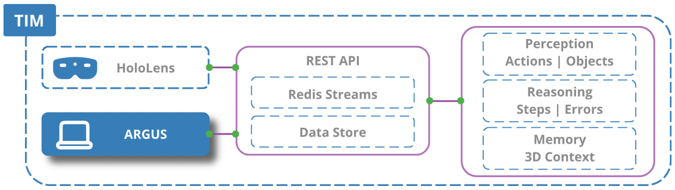


Fig. 2: TIM's architecture proposes a data communication service between the system components: the Hololens, AI modules, and *ARGUS*.

3.2 Overview of the TIM Personal Assistant

Our assistive AR framework (TIM) integrates perceptual grounding, attention, and user modeling during real-time AR tasks, and is composed of multiple software and hardware components. TIM perceives the environment, including the state of the human performer, by using a variety of data streams (details below) which are the input to the task guidance system. TIM communicates with the performer through the HoloLens 2 headset display.

The task guidance system is primarily composed of three AI components that interpret the incoming data streams: (1) *Perceptual Grounding* utilizes information from historical instances of actions from similar tasks and makes its best prediction of what the current action and objects are. (2) *Perceptual Attention* takes the objects and transforms them into 3D coordinates and contextualizes objects over time in the 3D environment. (3) *Reasoning* then uses the objects and actions returned by perception to identify which step of the task the user is in and to understand whether or not they are performing the task correctly. Any of these data can be ingested and displayed on our platform, *ARGUS*.

3.3 System Architecture

Since the computational resources on HoloLens 2 are limited, TIM is implemented as a client-server architecture. To enable data streaming capabilities, the system utilizes server-side infrastructure that provides a centralized data communication hub and real-time ML inference to facilitate ingesting, operating over, and contextualizing the produced data streams. A system diagram can be seen in Fig. 2.

Data Orchestration and Storage. The core of our architecture is Redis Streams, which we use as our data message queue. A REST + Websocket API provides a uniform abstraction layer for components to interact with. The HoloLens streams its sensor data to the API where it is made available to all other components in the system. The user is able to record data streaming sessions, which will listen and copy all data streams to disk. Later, users can selectively replay that data in the system as if the HoloLens was running, for easy offline testing.

Communication. TIM uses the REST API to stream onboard sensor data (i.e., gaze, hand tracking; see details in Section 3.4) in real-time. This allows us to shift the computation-heavy tasks to the server while keeping the essential tasks on the HoloLens to improve responsiveness. TIM also collects the ML prediction results from the server and updates the AR interaction and interface accordingly. The AR client running on the HoloLens ingests two streams to support contextual interaction: a *perception* stream that recognizes objects in the scene and a *reasoning* stream that recognizes performer actions. On average, these streams take about 100 ms to complete one update cycle to the AR client.

3.4 Data

The HoloLens 2 can provide various data from multiple sensors. With *Research Mode* enabled [52], we stream data from the main RGB camera, 4 grayscale cameras, an infrared depth camera, and an IMU that contains an accelerometer, gyroscope, and magnetometer. Details of the camera data can be found in Table 1. Although it is theoretically possible to stream some of the data at higher resolution or frame rate, the need to run a user interface on the HoloLens creates a practical limit.

Table 1: Description of streamed data from HoloLens 2 visual sensors.

Sensor	Resolution	Format	Framerate
RGB camera	760 × 428	RGB8	7.5 fps
Grayscale camera	640 × 480	Grayscale8	1 fps
Infrared active brightness	320 × 288	Grayscale8	5 fps
Infrared depth	320 × 288	Depth16	5 fps

Not only are the computational resources limited, but streaming extra data consumes more energy and may result in headset overheating.

The streamed frame rate in practice may be lower due to the packet drop during streaming. Hand tracking and eye tracking data are also streamed. The eye tracking data consists of 3D gaze origin positions and directions. The hand tracking data consists of 26 joint points for each hand. Each joint point contains a 3D position and a quaternion that indicates the orientation. In our system, the per-frame point cloud which consists of RGB and depth frames can be integrated into a holistic 3D environment. Performer sessions can vary in size. For instance, the recording of a simple recipe (preparing pinwheels [35]) usually takes ∼6 min and results in ∼600 MB data without the point cloud data, but 3 GB with the point cloud data.

Privacy and Ethical Considerations. While AR provides incredible opportunities, performer privacy must be protected during data collection and utilization [39]. Our experiment protocol is approved by an Institutional Review Board (IRB). It ensures data is never directly linked to an individual identity, code numbers, rather than names or other identifying information, are used for video recordings in *ARGUS*. Names or any other identifiable information are not collected and do not appear in any part of the system. Despite these efforts, it is theoretically possible to re-identify performers, see, e.g., [37], where it is shown that motion data can be used for identification. Another path to re-identification is the audio produced by the voice interactions.

3.5 AI Task Guidance System

Perceptual Grounding. To connect what the HoloLens sees and hears to task knowledge, the AR assistant needs to be equipped with models to recognize physical objects, actions, sounds, and contexts needed to complete a specific task. TIM uses multimodal machine-sensing models to detect human-object interactions in the environment. The output is real-time estimations with model confidence levels of three environmental elements: object categories, object localizations, and human action detections. We modulate object outputs via text instructions, allowing us to selectively detect objects and actions that are part of a particular procedure (e.g., recipe) and disregard everything else. To achieve this, our models generate and compare text and sensor representations. The models have the following main features:

Object detection and localization. We use "Detector with image classes" (Detic) [58] to generate these estimations, since it is a model that produces RGB frames and free-form text descriptions of objects of interest (e.g., "the blue cup") with bounding-box and object mask estimations for the regions in the frame where the objects are detected. Its direct comparison of RGB and text modalities is enabled via Contrastive Language-Image Pretraining (CLIP) [42].

Action recognition. TIM supports three action-recognition models: Omnivore [19], SlowFast, [13, 23, 53] and EgoVLP [41]. These models process video streams to output verb-noun tuples that describe actions. Each model has its benefits and limitations. While Omnivore is considered state-of-the-art for action recognition, it is a classification model with a fixed vocabulary. EgoVLP has a joint RGB-text representation that, similar to Detic and CLIP, allows for the detection of free-form text descriptions of actions. SlowFast integrates audio and RGB information, potentially allowing for the detection of actions outside the RGB field of view. Therefore, the optimal model to use is dependent on the deployment conditions.

Reasoning and Knowledge Transfer. Reasoning and knowledge transfer first preprocess the input task description and create the corresponding objects and actions needed for each step [26, 55]. In each frame, it takes the object and action outputs from the perceptual component, along with the processed input task description, and makes two decisions. First, the reasoning module performs *error detection*, in which it attempts to determine if the performer has made an error in the current frame based on how much the objects and actions detected through perception align with the preprogrammed knowledge of the step. Second, it performs *step prediction*, in which the system predicts whether the current step is complete and should move to the next step. This decision is governed by a hidden Markov model (HMM) [3]-like approach that primarily uses the probability of each action to appear in a given step of the task. These probabilities are calculated beforehand on a training dataset.

4 Tasks & Requirements

ARGUS was developed to support the development and operation of the AR personal assistant outlined in Sec. 3. On top of the obvious need to visualize the multitude of raw data streams, *ARGUS* was designed to enable the real-time and post-hoc visualization of ML models and performer interactions in the context of the physical environment, all in a time-synchronized fashion. To summarize, such a system should have the following design requirements (R1-R5). These requirements were created by working side-by-side with the developers of the AR assistant components described in Section 3 (i.e., perception, reasoning), and with end users through interviews and feedback sessions during and after use of TIM and *ARGUS*. For context, the AR-enabled tasks that *ARGUS* aims to support include, but are not limited to: making a meal from a recipe, applying a tourniquet, repairing an engine, and completing an aircraft preflight check.

[R1] **Live monitoring**: The ability to visualize the output of the various components of the system during task execution. This is crucial to understand possible system failures before completing recording sessions and gaining real-time insights about model outputs.

[R2] **Seamless provenance acquisition**: The future availability of the multimodal dataset produced during a recording session. This supports developers in improving algorithms and debugging system outputs and researchers in retrospectively investigating user-generated data. Therefore, automatically storing the acquired data (and metadata) into databases is important for such a system.

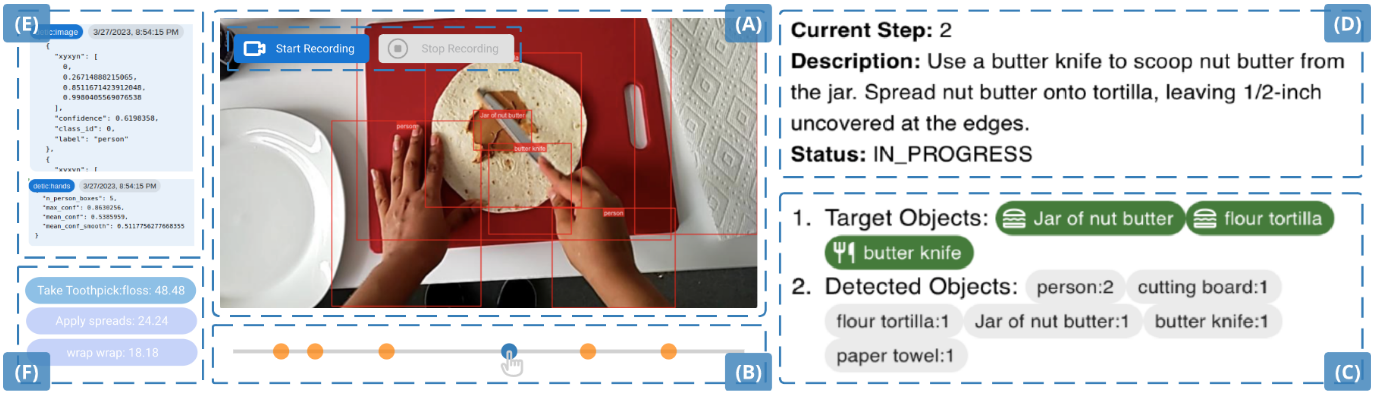
[R3] **Retrospective analysis of model performance**: The ability to visualize and inspect large chunks of the acquired data and model outputs to uncover relevant spatial and temporal trends.

[R4] **Physical environment representation**: A representation of the physical environment where the performance occurs. This representation should support data exploration tasks by explaining most of the observed user-generated data (e.g., performer movement patterns limited by physical constraints).

[R5] **Aggregated and detailed visualization performer behavior**: A summary of the global interaction patterns of the user with the environment. This is key in analyzing general performer behavior. Aggregating large chunks of data temporally and spatially can hide important details, thus, the system should provide both global and local perspectives of performer behavior data.

5 *ARGUS*: Augmented Reality Guidance and User-modeling System

As described in Sec. 4, we developed *ARGUS* concomitantly with TIM to meet the development needs of building an effective AR task assistant. In total, *ARGUS* enables the interactive exploration and debugging of all components of the data ecosystem needed to support intelligent task guidance. This ecosystem contains the data captured by the HoloLens's sensors and the outputs of the perception and reasoning models outlined in Sec. 3. *ARGUS* has two operation modes: "Online" (during task performance), and "Offline" (after performance). Users can use these two modes separately if needed, for instance, to perform real-time debugging through the online mode. In another usage scenario, users may start by using the online mode to record a session and then explore and analysis the data in detail using the offline mode. We describe additional usage scenarios in Section 6 through two case studies.



{
  "xyxyn": [
    0,
    0.26714888215065,
    0.8511671423912048,
    0.9980405569076538
  ],
  "confidence": 0.6198358,
  "class_id": 0,
  "label": "person"
},
{
  "xyxyn": [

"n_person_boxes": 5,
"max_conf": 0.8630256,
"mean_conf": 0.5385959,
"mean_conf_smooth": 0.5117756277668355
}

Current Step: 2
Description: Use a butter knife to scoop nut butter from the jar. Spread nut butter onto tortilla, leaving 1/2-inch uncovered at the edges.
Status: IN_PROGRESS

1. Target Objects: Jar of nut butter, flour tortilla, butter knife
2. Detected Objects: person:2, cutting board:1, flour tortilla:1, Jar of nut butter:1, butter knife:1, paper towel:1

Fig. 3: The online component of *ARGUS* for real-time debugging. (A) Streaming Video Player: users can inspect the output of the headset's camera overlayed with bounding boxes representing the detected objects. Users have the option to record any session. (B) Confidence Controller: a slider that allows the user to control the threshold model confidence. (C) Perception model outputs, including target and detected objects. "Target Objects" represent the objects needed in the current step (from recipe instructions) while "Detected Objects" shows all the objects identified by the perception models and their corresponding number of instances (e.g., multiple knives may be detected in a frame). (D) Reasoning model outputs, including the step and error predictions, step description, and the performer's status. (E) Raw data views, showing the raw data collected by the system. (F) Widgets showing the predicted actions with their probabilities. In this example, the model predicts that the current action is "Take Toothpick" with 48% likelihood, followed by "Apply spreads" with 24% and "wrap wrap" with 18%.

5.1 *ARGUS* Online: Real-time Debugger

Real-Time Debugging. The *ARGUS* architecture allows streaming data collection and processing in real-time, which makes instantaneous debugging and data validation possible **[R1]**. As depicted in Fig. 3, the online mode provides information on the outputs of the reasoning and perception models using custom visual widgets. The caption of Fig. 3 describes each component. Since what the HoloLens main camera sees (and thus what is analyzed by the models) is not the same as what the performer sees (due to different fields of view), having a real-time viewer such as (A) can help ensure the HoloLens is capturing what the performer and user wish to capture. Additionally, components (C) & (F) provide information that can help validate the objects and actions identified by the models in real-time (as opposed to having to do so post hoc). We note that these features are primarily intended to aid a user in analyzing performer behavior and model performance in real time, rather than to aid the performer as they complete a task.

Data collection. Users of *ARGUS* can decide when to save the recording for future analysis. By clicking the *Start Recording* button, all data captured and generated by the sensors and models from that point on are redirected from the online streaming database to the historical database until the user clicks *Stop Recording*. The data migration process is transparent to the user **[R2]**.

5.2 *ARGUS* Offline: Visualizing Historical Data

The offline mode's main goal is to enable analysis of historical data generated by the models and performer actions in the physical environment **[R3]**. To allow for easy exploration of this large and heterogeneous data, *ARGUS* provides a visual user interface that enables querying, filtering, and exploration of the data. Due to the spatiotemporal characteristics of the data, we provide both spatial and temporal visualization widgets to allow users to analyze the data from different perspectives. Fig. 4 shows the components composing *ARGUS* in offline mode. In the following, we describe the main components of the offline mode: the Data Manager, the Temporal View, and the Spatial View. We highlight the interaction flow a user is likely to follow, and for each component, we describe the visualizations, the interactions provided, and their goals.

5.2.1 Data Manager

Users start the exploration by using the Data Manager shown in Fig. 4(A) to filter the set of sessions available in the data store. Our data is organized as sessions (each session contains all recordings, data streams, and model outputs for a performer executing a task). The Data Manager enables data retrieval by allowing users to specify filters and select specific sessions from a list of results.

Data Querying. Users can query the data by specifying various filters, as shown in Fig. 4(A1). Filters are presented in the form of histograms the users can brush to select the desired range.

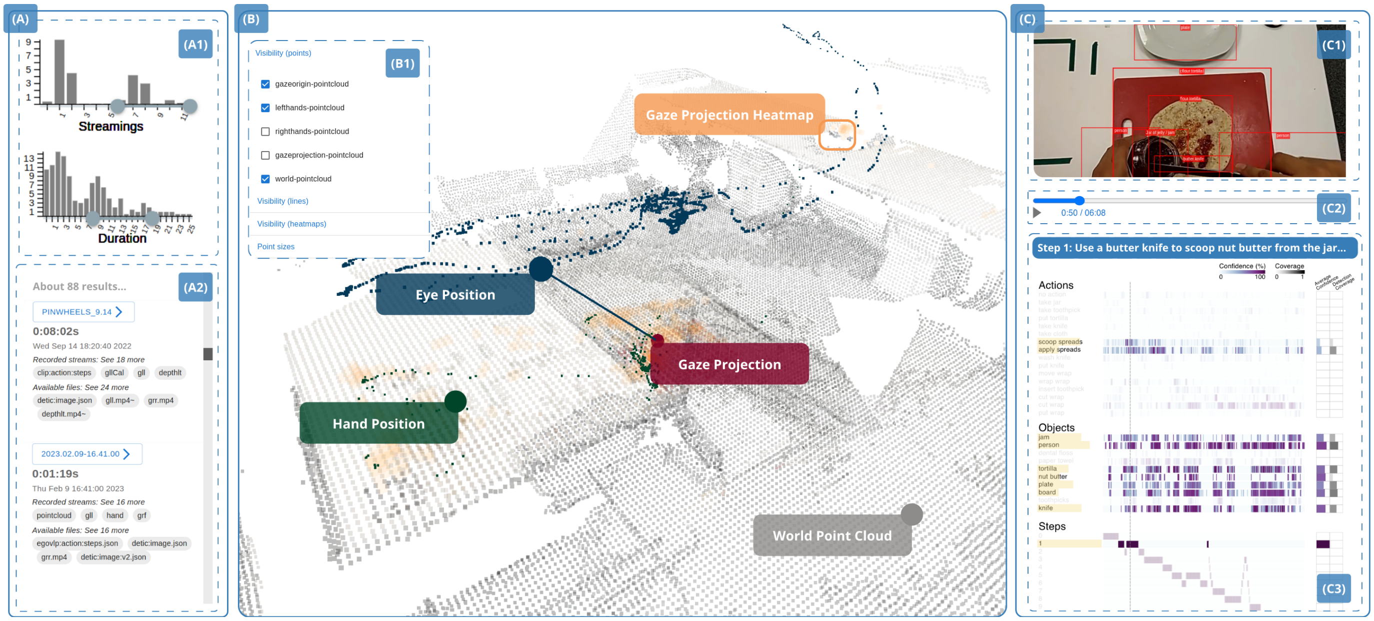
Query Results. The results component displays the retrieved sessions in a list format. Fig. 4(A2) shows the results for a given query specified by the user. Each element represents a session showing key features, including name, duration, date, recorded streams, and available model outputs. Once an element of the list is selected, the corresponding data will be loaded into the views of the system.

5.2.2 Spatial View

As described in Section 3.4, the spatial nature of some of the streamed data demands a 3D visualization to allow users to meaningfully explore the data. For this, *ARGUS* provides a Spatial View shown in Fig. 4(B) that allows users to analyze how performers interact with the physical environment in conjunction with the spatial distribution of model outputs. The Spatial View can help resolve where performers were located, where they were looking during specific task steps, where objects were located in the scene, etc. Below, we describe the elements of the Spatial View and its interaction mechanisms tailored to support the analysis of the spatial data following well-established visualization guidelines [45] to provide both overview and detailed information.

The basis of the Spatial View is a 3D point cloud (or *world point cloud*) as shown in Fig. 4(B) representing the physical environment where the performer is operating **[R4]**. This representation helps us interpret different aspects of the space, such as physical constraints imposed by the environmental layout. However, the point clouds generated based on the data acquired by the headset cameras can easily contain millions of points, making it unfeasible to transfer them over the web and render them within most web browsers. To give an overview of the whole task, all point clouds of one recording are merged to obtain a temporal aggregation. Hololens 2 generates approximately one point cloud per half second, which creates redundancy. This redundancy can be removed by creating a union of all point clouds and then downsampling it using voxelization. However, selecting imprecise parameters can lead to a subrepresentation of the physical space, losing important information and, consequently, hindering analysis. Thus, we parameterize the voxel-based downsampling to create voxels at 1 cm resolution, providing enough detail for the purposes of our tool. In our experiments, the downsampled point clouds had less than 100,000 points even in the worst case, leading to reasonable transfer and rendering times. With the world point cloud representing the physical environment, we are able to visualize performer activity in context.

As illustrated in Fig. 4, eye position, hand position, and other data streams can be represented as 3D points in the same scene. The blue



PINWHEELS_9.14
0:08:02s
Wed Sep 14 18:20:40 2022
Recorded streams: See 18 more
clip:action:steps gllCal gll depthlt
Available files: See 24 more
detic:image.json gll.mp4~ grr.mp4 depthlt.mp4~

2023.02.09-16.41.00
0:01:19s
Thu Feb 9 16:41:00 2023
Recorded streams: See 16 more
pointcloud gll hand grf
Available files: See 16 more
egovlp:action:steps.json detic:image.json grr.mp4 detic:image:v2.json

Fig. 4: Overview of the user interface and components of *ARGUS* Offline. (A) The **Data Manager** shows the applied filters (A1) and the list of retrieved sessions (A2). (B) The **Spatial View** shows the world point cloud representing the physical environment, 3D points for eye and hand positions, and gaze projections and heatmaps. (B1) Render Controls allow the user to select the elements of the Spatial View they desire to see. (C) **Temporal View:** (C1) The Video Player is the main camera video output of the current timestamp selected by the user. (C2) The Temporal Controller controls the video player and updates the model output viewer as well. (C3) The Model Output Viewer displays the output of the machine learning models (reasoning and perception) used during execution time.

dots show the eye position of the performer during a session, while the green dots show the hand position. For each collection of 3D points representing a data stream, users can retrieve more detailed information by interacting with the points. For example, if the user hovers their mouse over the points representing the eye position, a line representing the gaze direction will automatically be rendered in the scene, representing what point in space the performer was looking at from their current position at a specific timestamp. This is possible by calculating the intersection of the gaze direction vector with the world point cloud. This gaze information can also be represented as a 3D point cloud to provide a visual summary of the areas the performer was focused on **[R5]**. This interaction also updates the corresponding video frame in Fig. 4(C1) and highlights the models' outputs in Fig. 4(C3).

Although the point cloud provides a summarization of the spatial distribution of these data streams, this representation fails to convey aggregated statistics of the data, such as the density of points in a given region which is proportional to the amount of time the performer spent in a given location of the scene. For this purpose, a 3D heatmap is a more suitable visualization. The Spatial View can create 3D heatmaps of each data stream. The heatmap in Fig. 4(B) shows the distribution of the gaze data during the session. We leverage the voxel information created during the downsampling to calculate the density of points within voxels. Using an appropriate color scale, we render cells with non-negligible densities to create the 3D heatmap. Every data stream containing spatial information can be incorporated into the Spatial View as 3D point clouds or heatmaps in *ARGUS*. Information regarding perception and reasoning models are also available in this view. By combining bounding boxes generated by perception models and depth information captured by the headset, we reconstruct the center point of each detected object, helping users understand the spatial distribution of objects. Also, occupancy maps representing the density of objects in different regions can be derived as presented in Fig. 8. Moreover, the Spatial View provides a summarization of gaze information by rendering sets of vectors representing gaze directions over time. Users can control the style (e.g., size and opacity of points) and visibility of all data streams, choosing what data should be visible for analysis. Lastly, point clouds can also be filtered based on timestamp ranges, allowing for focused analysis of specific task steps ("Visibility" in Fig. 4(B1)).

5.2.3 Temporal View

ML models are a core component of an AI assistant system. While the field of ML has seen many recent advancements to support assistive AR applications [5, 15, 40], the need for tools to improve them remains. Model debuggers are powerful tools used to analyze, understand, and improve these models by identifying issues and probing ML response functions and decision boundaries. This helps developers make models more accurate, fair, and secure, promoting trust and enabling understanding which is highly desirable in intelligent AR assistants. *ARGUS* provides a model debugger based on temporal visualizations to debug the ML models used in AI assistant systems **[R5]**. We describe the different temporal components in detail in the following subsections.

Video Player. The object detection model not only recognizes all objects in an image but also their positions. To inspect these outputs, *ARGUS* contains a video player component that identifies the spatial location of detected objects over time, as shown in Fig. 4C. This component allows the user to toggle between two views: 1) the raw main camera video stream and 2) a panoramic mosaic view which consists of a sequence of panoramic mosaics generated from this main camera stream. We highlight all detected objects with bounding boxes, which are provided by the object detection model.

The first view of the video player, which displays the raw main camera video stream collected by HoloLens, enables a highly granular level of model debugging. This allows the user to note specific frames where object detection failed or yielded unexpected results. However, the main camera of the HoloLens has a limited field of view. Often objects that the performer sees at a given timestep cannot be seen in the frame of the raw main camera video at that timestep. Therefore, we aggregate frames into a series of panoramic mosaics in the second view of the video player component, capturing a broader scope of what the performer sees at each timestep. We generate these panoramic mosaics by sampling frames from a temporal window centered around the current timestep. We then compute SIFT features for each frame [29], match them using a Fast Library for Approximate Nearest Neighbors (FLANN)-based matcher [36], and filter for valid matches by Lowe's ratio test [29] before warping and compositing the frames into a panoramic mosaic. We observe that these panoramic mosaics expand the view of the scene significantly, revealing objects

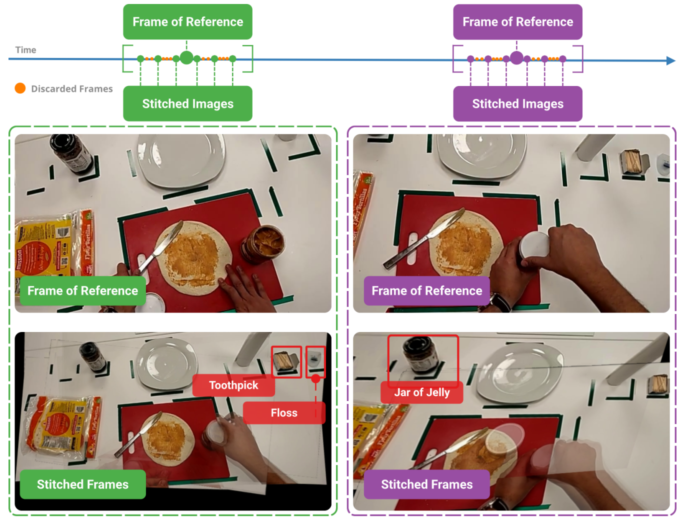


Fig. 5: A visual representation of frame selection for the panoramic mosaic view of the video player (top) and comparison of these panoramic mosaics with corresponding frames from the same timestep of the raw main camera video for reference (bottom). Each panoramic mosaic is composed of several frames sampled from a window around the current timestep of the raw main camera video. In both examples, we highlight objects that are visible in the panoramic mosaic but not in the raw main camera video (toothpick, floss, and jar of jelly, respectively) in red.

within the field of view of the performer at a given timestep that were not captured by the main camera at that same timestep (see Fig. 5).

We note that in much of the existing literature on panoramic mosaics, the goal is to capture a seamless wider view of an (often static) scene at a single point in time. In these cases, previous works have endeavored to work around both in-scene and camera motion by excluding moving objects within the scene [21] or only addressing simple, slow camera panning motions [48]. When capturing video from an AR headset of a performer completing a task, however, unpredictable and rapid motion is not only unavoidable, but a valuable indicator of performer behavior. Therefore, our goal extends beyond the typical spatial expansion provided by a panoramic mosaic; we also aim to show how objects move around the complete scene over time, and how the object detection model performs over the given time range in order to facilitate both temporal and spatial analysis of a scene. We note that in our example task shown in Fig. 5 (cooking), the performer will often remain in the same position for many consecutive timesteps, consequently, the panoramic mosaic may not significantly expand the field of view at every timestep. Nevertheless, for tasks where the performer traverses a larger area or turns their head in a wider range (and at timesteps where that behavior occurs in this task), the panoramic mosaic will significantly increase the portion of the scene shown at a given timestep.

Model Output Viewer. During the debugging process of AR assistant models, the need for model output summaries is key to starting an analysis or evaluation. However, the temporal aspect inherent to these kinds of models makes this task more challenging since they often need to manage the sequence of actions or events chronologically. The Model Output Viewer provides a summarization of the temporal distribution of the ML models outputs across the whole session (see Fig. 4(C3)). This visualization is especially useful to find salient patterns, such as quick transitions between steps in step detection models, or to evaluate prediction consistency across time, allowing users to quickly have a global picture of the model behavior, something that could not be achieved by analyzing specific time frames.

As mentioned in Section 3.5, for AR assistive systems, the most relevant model outputs are the objects, actions, and steps. Once these model outputs are available, they are used to create the matrix visual representations for temporal model analysis. Fig. 6 illustrates the Model Output Viewer, where three main components are highlighted: the model outputs, the confidence matrix, and the global summaries. The *Model output* view presents all the model outputs grouped by category. For example, as shown in Fig. 4(C3), there are three categories listed vertically: Objects, Actions, and Steps. The object, action, and step sections have multiple rows, each of them listing the model outputs for each category, e.g., the detected objects identified by the perception model. *Confidence matrix*: The x-axis (or columns) indicates the time, from 0 to the total duration of the session (video). Each cell of the matrix is colored according to the confidence score of the detected item at time t (0% ▨ 100%). If no action, object, or step is present, the matrix cell is left blank (white). The total number of cells is proportional to the size of the session (seconds), and all cells are equal in height. Users can hover over the cells to see additional details. *Global summaries*: The Model Output Viewer also provides summaries of the average confidence and detection coverage for each row on the right side of the view so users can quickly evaluate them. The average confidence only takes the confidence value of detected objects, actions, or steps into account. Detection coverage refers to the total number of detections available for each model output (objects, actions, and steps).

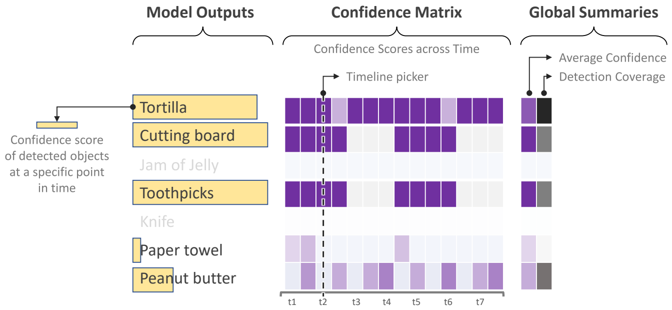


Fig. 6: Illustration of the Model Output Viewer applied to the analysis of a cooking recipe session. To the left, the model outputs are listed vertically. The bars depict the confidence score of the detected outputs' labels at the specific time picked on the timeline. In the middle, the temporal distribution of ML model output confidences across the whole session is displayed. To the right, two summaries are shown: the average confidence and detection coverage for each output across the entire session. Color darkness is proportional to confidence value: ▨.

Even though the Temporal View representation can provide a visual summary of the temporal distribution of the model output, details-on-demand functionalities remain crucial for debugging. The Model Output Viewer allows users to do a focused analysis by letting them explore the model output results at specific points in time for further analysis. The user can use the temporal controller or the 3D viewer to do this selection. After this, all the objects, actions, and steps detected for that specific point in time that meets the confidence threshold are highlighted, as shown in Fig. 4(C3). Users can adjust the confidence threshold value using the slider to investigate the object detection results. We also display object and action labels with bars depicting the confidence value for each label following guidelines of Felix et al. [14] (see Fig. 1).

5.3 Implementation and Performance Details

The implementation of *ARGUS* follows a set of constraints to allow for interactive query and rendering times. The backend supporting the rest API was written using Python and FastAPI [43]. The ML models were trained and/or fine-tuned using PyTorch and serve predictions in real-time utilizing the same streaming protocol used by *ARGUS*. The interface was structured as a dashboard-like single-page application built with React [47] and TypeScript [31]. The visualization of 3D components uses Three.js [51] and D3 [7]. All the data consumed by *ARGUS* online mode comes from querying our Redis database, while the data available in the offline mode comes from the data store in JSON format. All the code is open source and hosted on GitHub [2].

We have measured the latency of Microsoft's Windows Device Portal (part of their mixed reality capture [32]) at ∼1.3 s for streaming the main Hololens 2 camera, while ARGUS has a lower latency of ∼300 ms. Currently, during online use, we save the various data streams at they get off the device. For the session in Section 6.2, which takes 1:42 min, the streamed point cloud has more than 10 M points, and it is highly

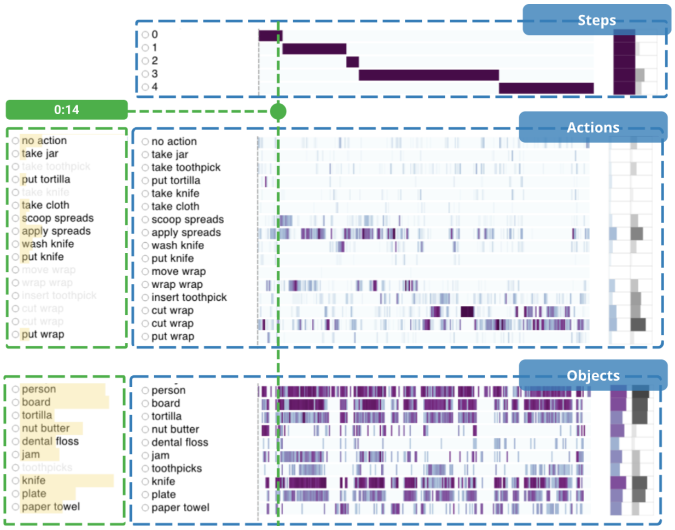


Fig. 7: Analysis of actions, objects, and steps in the Model Output Viewer. Color darkness is proportional to confidence value (0% ▬ 100%). The confidence matrix and the average confidence views show that the confidence scores for objects are higher than actions. The arrows show the confidence scores for actions and objects at minute 0:14 of the video. The detection coverage view shows that some actions (e.g., *take jar*) are rarely identified during the video.

redundant, since the same geometry is sampled over and over again. After the performer finishes a recording, we merge and downsample this data into a consolidated point cloud (see Sec. 5.2.2), in this case with 70,000 points. We also create a voxel grid to generate the heat maps, which take 2.3 s. After loading, all data is rendered in real-time.

6 Case Studies & Discussion

In this section, we present two case studies describing how model developers have made use of some of the available features. The section ends with feedback from domain experts who have used *ARGUS* while developing AR task guidance software and a discussion of limitations.

6.1 Improving Step Transitions in Reasoning Module

To showcase how the Model Output Viewer supports the exploration and analysis of AI assistant model outputs (objects, actions, and steps), we describe how an ML engineer used this tool, the insights they gained, and how the reasoning module of the AI assistant, TIM, was improved through these insights. The ML engineer began by exploring the outputs of the reasoning and perception modules of a recorded session where a performer used TIM to follow a recipe [35].

Analyzing step transitions. The visualization of the entire cooking session can help users find salient patterns, e.g., how the transitions between steps were carried out. The first repeated pattern identified by the ML engineer while using *ARGUS*'s Model Output Viewer was the slow transition between steps (see Fig. 7). Investigating the "Steps" reveals that steps 1, 3, and 4 were performed over unexpectedly longer periods of time than steps 0 and 2. Also, the user noticed that the model only identified 5 out of 12 steps. Clearly, these two observations indicate that the reasoning module is making errors in identifying recipe steps. This visual summary of the Model Output Viewer allows developers to quickly possess a global picture of model performance and assess errors. This could not be achieved as easily without *ARGUS*.

Exploring detected objects. Under "Objects" in the Model Output Viewer (Fig. 7), the ML engineer noticed that the Detic model identified most of the objects for the entirety of a recipe video. This is apparent from the confidence matrix, where most rows are colored (meaning an object was detected). The user also analyzed the confidence values of each detected object. For instance, at the 0:14 mark of the video, objects like *board*, *nut butter*, and *knife* had high confidence values, indicated by the yellow background (see zoomed-in views in Fig. 7).

Table 2: Accuracy of the old and new version of the reasoning module for recognizing the steps of the recipe.

Version	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	Total
Old	1.0	0.9	0.3	0.9	0	0	0	0	0	0	0	0	0.35
New	1.0	1.0	0.5	1.0	0	0	0.4	0.9	0	1.0	1.0	0	0.73

They could also see this trend in the "Average Confidence" column, which provides an average of confidence values throughout the video.

Identifying missing actions. The ML engineer also analyzed the "Actions" section of the Model Output Viewer. They noticed that some actions were rarely detected by the EgoVLP model. As we can see in the "Detection Coverage" column, actions like *put knife*, *move wrap*, and *take cloth* were detected an unusually few number of times. This indicated that it was difficult for EgoVLP to detect those actions. They also noticed that the confidence values for the actions were much lower than the ones for objects. As is visible in Fig. 7, the predominant color during the whole session was light purple, which represents low confidence in detecting the actions. Also, at time 0:14 of the video, the confidence values for "scoop spreads" and "apply spreads" were low. In the "Average Confidence" column, we can see that actions such as "wash knife" and "insert toothpick" had approximately 30% confidence. This information led the ML engineer to hypothesize that a decrease in the confidence threshold might be necessary to recognize steps effectively. The visualizations provided by *ARGUS* also help to investigate whether lowering the threshold would lead to false positives in the step recognition.

Using insights to improve the reasoning module. After the analysis, the ML engineer modified the reasoning module to handle actions with low confidence. The reasoning module defaults to selecting actions with greater than 70% confidence. The ML engineer used the confidence slider of Model Output Viewer to tune this value. The most promising value they found was 30%. They reran the new version of the reasoning module for the same video. As shown in Table 2, the step estimation accuracy increased for every step and from 35% to 73% overall. Also, this new version was able to identify 8 out of 12 steps, while the previous version only identified 4 out of 12 steps.

Interpreting the new results. The Model Output Viewer was also useful for the ML engineer to understand why the reasoning module failed to recognize some steps. As we can see in Table 2, steps 4, 5, and 8 were not recognized. For instance, step 5 ("Roll the tortilla from one end to the other into a log shape") is directly related to the action "move wrap", and this action was not identified at all during the entire session (see "Detection Coverage" column). Since this action is necessary for step 5, it was not identified by the reasoning module.

6.2 Using Spatial Features to Explain Failures

Although the Temporal View can help users uncover undesired patterns in model performance, it does not paint the full picture of the situation, as model failures might be related to spatial characteristics or performer behavior. In this case study, we show how the Spatial View can provide deeper insights into both reasoning and perception models by assisting users in finding regions where the perception models underperform and to correlate performer behavior with reasoning outputs.

A very common way to assess the quality of perception models is by checking the spatial distribution of static objects. In other words, the physical objects captured by the headset cameras can generally be classified as either static objects (objects that will likely not move) or dynamic objects. This classification can help users quickly identify regions where the perception model fails by detecting objects not expected to move throughout a recording session. This case study highlights how this sanity check becomes trivial in *ARGUS*.

We start the exploration by first using the Data Manager to load the parts of a recording where the perception model underperformed. Once a recording from this period is loaded, we can use the Spatial View to find regions of the space where the performer was interacting. Fig. 4 shows points of performer positions (blue) and gaze projection

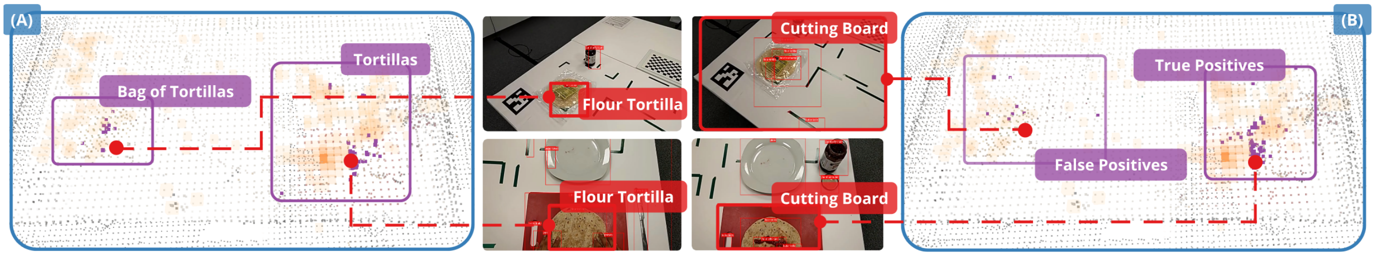


Fig. 8: Example of how the Spatial View can help users identify missing classes in the model's vocabulary or find clusters of false positives. (A) Points representing the 3D positions the object detection model identified as a "tortilla" during the session. Points on the left represent a bag of tortillas, while points on the right represent a single tortilla. (B) Points representing the 3D positions the object detection model identified as a "cutting board" during the session. The cluster on the left contains false positives, where the perception model generates wrong bounding boxes.

(orange). During an initial inspection, the user can quickly recognize three darker regions projected in the world's point cloud. The rightmost region represents the time during which the performer was interacting with *ARGUS* in online mode to start the recording, while the other two regions are on the desk. The user then hovers the mouse over the points on the 3D point cloud representing the gaze projection on the world point cloud to look at the corresponding video frames in Temporal View. This interaction reveals that the left region contains the ingredients for the recipe, while the actions (e.g., "spreading jelly on the tortilla") happen on the right side. By highlighting the heatmaps only, it becomes clear that the performer spent most of their time looking at the right side of the table (darker region), meaning the performer spent more time executing actions than selecting ingredients. With this understanding of the spatial distribution of the performer's attention, the user can infer that the model outputs high confidence values for *tortilla* and *cutting board* throughout the entire session as shown in Fig. 7, which makes the user question the validity of the output. Then, the user displays the 3D point cloud denoting the 3D positions of tortillas shown in Fig. 8(A). Two clusters show up, allowing the user to look at the corresponding frames, realizing that the left cluster represents a bag of tortillas while the other is the tortilla used for the recipe. This process highlights the need for a more comprehensive class vocabulary able to represent both tortillas and bags of tortillas. Following that, the same process is conducted for the *cutting board* class, and a similar pattern with two clusters arises (see Fig. 8(B)). Since the cutting board was a static object in the recording, the user can quickly realize that one of the clusters may be representing a model failure. The corresponding video frames selected interactively confirm that the left cluster contains only false negatives. Lastly, inspecting the bounding boxes rendered on the video frames (see Fig. 8) gives the user more detailed information about the error. In this case, the user sees that the model is generating bounding boxes covering almost the entire field of view of the performer.

6.3 Expert Feedback

The Model Output Viewer provides a visualization of object and action detections with model confidence levels. Even if a model performs very well on an offline evaluation dataset, when deployed in real-time, it will inevitably be presented with previously unseen conditions such as room lighting, skin pigmentation, or object angle. This is known as the "domain-shift" problem [57], where a model fails to perform when presented with data not well represented in its offline evaluation dataset. *ARGUS* streamlines real-time deployment, and its Model Output Viewer enables the evaluation of model confidence in a virtually unconstrained domain. This sheds light on which conditions the models perform best, and informs how model robustness could be enhanced by expanding data with new collection or augmentation strategies.

ARGUS is also useful for scenarios where multiple information sources must be analyzed at the same time. For TIM's reasoning module, which consumes multiple inputs in parallel (e.g., the detected objects/actions and its confidence scores), *ARGUS*'s visualizations allow the user to understand the reasons the system made the predictions and under what circumstances it succeeded/failed. As shown in Sec. 6.1, this tool helped the ML engineer to improve the system.

As ML models develop new capabilities and produce richer representations, it becomes increasingly important to develop scalable visualizations of those outputs. Conventionally, ML engineers either log outputs to the terminal or use drawing libraries to bake the predictions on top of the video. However, there is limited real estate when drawing on a video, and often the predictions and their associated text make it difficult to view the underlying image frames. In contrast, *ARGUS* provides a high level of interactivity, which allows it to selectively visualize relevant information while allowing the user to change the view and granularity of these information to suit their needs. Additionally, being able to contextualize and explore ML model outputs in 3D can lead to a better understanding of how model outputs can change based on the perspective, and spatially grounds the predictions for an entire recording in a single view. Overall, tools like *ARGUS* drastically lighten the visualization load placed on ML engineers and provide a convenient tool for understanding their models.

Limitations. While useful for exploration of spatiotemporal data captured by an intelligent assistant, *ARGUS*needs more robust data processing algorithms. For instance, in sessions where the performer's hands are recurrently in the field of view of the headset camera, the point cloud generation process captures and transforms it into points of the world space, resulting in potential noise that does not represent the physical environment. To overcome this problem, we review recordings with noisy point clouds and define bounding boxes representing regions where these noisy points must be excluded from the final rendering. We plan to explore methods [12, 33] to automatically remove point cloud noise during run-time acquisition.

7 Conclusion & Future Work

We presented *ARGUS*, an interactive visual analytics system that empowers developers of intelligent assistive AR systems to seamlessly analyze complex datasets, created by integrating multiple data streams at different scales, dimensions, and formats acquired during performance time. Furthermore, through interactive and well-integrated spatial and temporal visualization widgets, it allows for retrospective analysis and debugging of historical data generated by ML models for AI assistants.

We envision *ARGUS* to unlock several avenues for future research connecting human-computer interaction, visualization, and machine learning communities revolving around the goal of developing better and more reliable AR intelligent systems. In the future, we intend to conduct a deeper evaluation of our system's performance metrics (e.g. rendering times, stream latency). We also plan to explore how to extend the system to support the comparison of sessions of multiple performers. This includes the data and model outputs and will require registration of the point clouds. User-generated data acquisition (annotation) and integrated AI techniques during exploration time (segmentation and model training based on the annotated data) are other fronts we would like to cover. Since our Temporal and Spatial Views allow users to explore data and output models across the entire session, adding annotation capabilities is a natural next step. Furthermore, we want to investigate privacy-preserving methods for storing and streaming the collected data, similar to ones that have been proposed, e.g., for eye-tracking data [8, 11], to prevent performer identification.

Acknowledgments

We would like to thank Jay Jatin Sutaria and Fabio Felix Dias for help with the development of *ARGUS*. This work was supported by the DARPA PTG program. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA. Erin McGowan is funded by an NYU Tandon Future Leader Fellowship.

References

[1] W. Aigner, S. Miksch, H. Schumann, and C. Tominski. *Visualization of Time-Oriented Data*. Human–Computer Interaction Series. Springer-Verlag, London, 2011. doi: 10.1007/978-0-85729-079-3 3

[2] ARGUS. Augmented reality guidance and user-modeling system. `https://github.com/VIDA-NYU/ARGUS`, 2023. 7

[3] L. E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The annals of mathematical statistics*, 37(6):1554–1563, 1966. 4

[4] R. Beams, E. Brown, W.-C. Cheng, J. S. Joyner, A. S. Kim, K. Kontson, D. Amiras, T. Baeuerle, W. Greenleaf, R. J. Grossmann, A. Gupta, C. Hamilton, H. Hua, T. T. Huynh, C. Leuze, S. B. Murthi, J. Penczek, J. Silva, B. Spiegel, A. Varshney, and A. Badano. Evaluation Challenges for the Application of Extended Reality Devices in Medicine. *Journal of Digital Imaging*, 35(5):1409–1418, 2022. doi: 10.1007/s10278-022-00622-x 2

[5] M. Becher, D. Herr, C. Müller, K. Kurzhals, G. Reina, L. Wagner, T. Ertl, and D. Weiskopf. Situated Visual Analysis and Live Monitoring for Manufacturing. *IEEE Computer Graphics and Applications*, 42(2):33–44, 2022. doi: 10.1109/MCG.2022.3157961 2, 6

[6] D. Bohus, S. Andrist, A. Feniello, N. Saw, M. Jalobeanu, P. Sweeney, A. L. Thompson, and E. Horvitz. Platform for situated intelligence. *CoRR*, abs/2103.15975, 2021. 3

[7] M. Bostock. D3.js. `https://d3js.org/`. 7

[8] E. Bozkir, O. Günlü, W. Fuhl, R. F. Schaefer, and E. Kasneci. Differential privacy for eye tracking with temporal correlations. *PLOS ONE*, 16(8):1–22, 2021. doi: 10.1371/journal.pone.0255979 9

[9] T. Caudell and D. Mizell. Augmented reality: an application of heads-up display technology to manual manufacturing processes. In *Proceedings of the Twenty-Fifth Hawaii International Conference on System Sciences*, vol. ii, pp. 659–669, 1992. doi: 10.1109/HICSS.1992.183317 2

[10] DARPA. Perceptually-enabled task guidance (PTG). `https://www.darpa.mil/program/perceptually-enabled-task-guidance`. 2, 3

[11] B. David-John, D. Hosfelt, K. Butler, and E. Jain. A privacy-preserving approach to streaming eye-tracking data. *IEEE Transactions on Visualization and Computer Graphics*, 27(5):2555–2565, 2021. doi: 10.1109/TVCG.2021.3067787 9

[12] Y. Duan, C. Yang, H. Chen, W. Yan, and H. Li. Low-complexity point cloud denoising for lidar by pca-based dimension reduction. *Optics Communications*, 482:126567, 2021. 9

[13] C. Feichtenhofer, H. Fan, J. Malik, and K. He. Slowfast networks for video recognition. In *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6202–6211, 2019. 4

[14] C. Felix, S. L. Franconeri, and E. Bertini. Taking word clouds apart: An empirical investigation of the design space for keyword summaries. *IEEE Transactions on Visualization and Computer Graphics*, 24:657–666, 2018. 7

[15] I. Fernández del Amo, J. A. Erkoyuncu, R. Roy, and S. Wilding. Augmented Reality in Maintenance: An information-centred design framework. *Procedia Manufacturing*, 19:148–155, 2018. doi: 10.1016/j.promfg.2018.01.021 2, 6

[16] P. Fleck, A. Sousa Calepso, S. Hubenschmid, M. Sedlmair, and D. Schmalstieg. RagRug: A Toolkit for Situated Analytics. *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2022. doi: 10.1109/TVCG.2022.3157058 3

[17] Foxglove. Foxglove - Visualizing and debugging your robotics data. 3

[18] M. Funk, S. Mayer, and A. Schmidt. Using in-situ projection to support cognitively impaired workers at the workplace. In *Proceedings of the 17th international ACM SIGACCESS conference on Computers & accessibility*, pp. 185–192, 2015. 2

[19] R. Girdhar, M. Singh, N. Ravi, L. van der Maaten, A. Joulin, and I. Misra. Omnivore: A single model for many visual modalities. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16102–16112, 2022. 4

[20] S. Henderson and S. Feiner. Exploring the Benefits of Augmented Reality Documentation for Maintenance and Repair. *IEEE Transactions on Visualization and Computer Graphics*, 17(10):1355–1368, 2011. doi: 10.1109/TVCG.2010.245 2

[21] C.-T. Hsu and Y.-C. Tsan. Mosaics of video sequences with moving objects. *Signal Processing: Image Communication*, 19(1):81–98, 2004. doi: 10.1016/j.image.2003.10.001 7

[22] T. Jiang, D. Yu, Y. Wang, T. Zan, S. Wang, and Q. Li. HoloLens-Based Vascular Localization System. *Journal of Medical Internet Research*, 22(4):e16852, Apr. 2020. doi: 10.2196/16852 2

[23] E. Kazakos, A. Nagrani, A. Zisserman, and D. Damen. Slow-fast auditory streams for audio recognition. In *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 855–859. IEEE, 2021. 4

[24] J. Kehrer and H. Hauser. Visualization and Visual Analysis of Multifaceted Scientific Data: A Survey. *IEEE Transactions on Visualization and Computer Graphics*, 19(3):495–513, 2013. doi: 10.1109/TVCG.2012.110 3

[25] K. Kim, L. Boelling, S. Haesler, J. Bailenson, G. Bruder, and G. F. Welch. Does a Digital Assistant Need a Body? The Influence of Visual Embodiment and Social Behavior on the Perception of Intelligent Virtual Agents in AR. In *2018 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 105–114, 2018. ISSN: 1554-7868. doi: 10.1109/ISMAR.2018.00039 2

[26] A. Lin, S. Rao, A. Celikyilmaz, E. Nouri, C. Brockett, D. Dey, and W. B. Dolan. A recipe for creating multimodal aligned datasets for sequential tasks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 4871–4884, 2020. 4

[27] C.-F. Liu and P.-Y. Chiang. Smart glasses based intelligent trainer for factory new recruits. In *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services Adjunct*, MobileHCI '18, pp. 395–399. Association for Computing Machinery, 2018. doi: 10.1145/3236112.3236174 2

[28] S. Liu, D. Maljovec, B. Wang, P.-T. Bremer, and V. Pascucci. Visualizing High-Dimensional Data: Advances in the Past Decade. *IEEE Transactions on Visualization and Computer Graphics*, 23(3):1249–1268, 2017. doi: 10.1109/TVCG.2016.2640960 3

[29] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, Nov. 2004. doi: 10.1023/b:visi.0000029664.99615.94 6

[30] K. Marriott, F. Schreiber, T. Dwyer, K. Klein, N. H. Riche, T. Itoh, W. Stuerzlinger, and B. H. Thomas, eds. *Immersive Analytics*. Springer International Publishing, 2018. doi: 10.1007/978-3-030-01388-2 3

[31] Microsoft. Typescript. `https://www.typescriptlang.org/`. 7

[32] Microsoft. Using the windows device portal. `https://learn.microsoft.com/en-us/windows/mixed-reality/develop/advanced-concepts/using-the-windows-device-portal`, 2022. 7

[33] M. Miknis, R. Davies, P. Plassmann, and A. Ware. Near real-time point cloud processing using the pcl. In *2015 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 153–156. IEEE, 2015. 9

[34] P. Milgram and F. Kishino. A Taxonomy of Mixed Reality Visual Displays. *IEICE Transactions on Information Systems*, E77-D, 1994. 2

[35] MIT Lincoln Laboratory. PTG evaluation tasks vol. 1. `TBD`, 2022. 4, 8

[36] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Applications*, 2009. 6

[37] V. Nair, L. Rosenberg, J. F. O'Brien, and D. Song. Truth in motion: The unprecedented risks and opportunities of extended reality motion data. `https://arxiv.org/abs/2306.06459`, 2023. 4

[38] A. Nijholt. Towards Social Companions in Augmented Reality: Vision and Challenges. In N. A. Streitz and S. Konomi, eds., *Distributed, Ambient and Pervasive Interactions. Smart Living, Learning, Well-being and Health, Art and Creativity*, Lecture Notes in Computer Science, pp. 304–319. Springer International Publishing, 2022. doi: 10.1007/978-3-031-05431-0_21 2

[39] S. Pase. Ethical considerations in augmented reality applications. In *Proceedings of the International Conference on e-Learning, e-Business, Enterprise Information Systems, and e-Government (EEE)*, p. 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 2012. 4

[40] B. Puladi, M. Ooms, M. Bellgardt, M. Cesov, M. Lipprandt, S. Raith, F. Peters, S. C. Möhlhenrich, A. Prescher, F. Hölzle, T. W. Kuhlen, and A. Modabber. Augmented Reality-Based Surgery on the Human Cadaver Using a New Generation of Optical Head-Mounted Displays. *JMIR Serious Games*, 10(2):e34781, 2022. doi: 10.2196/34781 2, 6

[41] K. Qinghong Lin, A. Jinpeng Wang, M. Soldan, M. Wray, R. Yan, E. Zhongcong Xu, D. Gao, R. Tu, W. Zhao, W. Kong, et al. Egocentric video-language pretraining. *arXiv e-prints*, pp. arXiv–2206, 2022. 4

[42] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PMLR, 2021. 4

[43] S. Ramírez. Fastapi. https://fastapi.tiangolo.com/. 7

[44] A. Schmeil and W. Broll. MARA - A Mobile Augmented Reality-Based Virtual Assistant. In *2007 IEEE Virtual Reality Conference*, pp. 267–270, 2007. doi: 10.1109/VR.2007.352497 2

[45] B. Shneiderman. The eyes have it: A task by data type taxonomy for information visualizations. In *Proceedings 1996 IEEE symposium on visual languages*, pp. 336–343. IEEE, 1996. 5

[46] R. Sicat, J. Li, J. Choi, M. Cordeil, W.-K. Jeong, B. Bach, and H. Pfister. DXR: A Toolkit for Building Immersive Data Visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):715–725, 2019. Conference Name: IEEE Transactions on Visualization and Computer Graphics. doi: 10.1109/TVCG.2018.2865152 3

[47] M. O. Source. React. https://react.dev/. 7

[48] D. Steedly, C. Pal, and R. Szeliski. Efficiently registering video into panoramic mosaics. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, vol. 2, pp. 1300–1307 Vol. 2, 2005. doi: 10.1109/ICCV.2005.86 7

[49] X. Sun, S. B. Murthi, G. Schwartzbauer, and A. Varshney. High-Precision 5DoF Tracking and Visualization of Catheter Placement in EVD of the Brain Using AR. *ACM Transactions on Computing for Healthcare*, 1(2):9:1–9:18, 2020. doi: 10.1145/3365678 2

[50] A. Tang, C. Owen, F. Biocca, and W. Mou. Comparative effectiveness of augmented reality in object assembly. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 73–80, 2003. 2

[51] three.js. three.js. https://threejs.org/. 7

[52] D. Ungureanu, F. Bogo, S. Galliani, P. Sama, X. Duan, C. Meekhof, J. Stühmer, T. J. Cashman, B. Tekin, J. L. Schönberger, P. Olszta, and M. Pollefeys. Hololens 2 research mode as a tool for computer vision research. *CoRR*, abs/2008.11239, 2020. 3

[53] F. Xiao, Y. J. Lee, K. Grauman, J. Malik, and C. Feichtenhofer. Audiovisual slowfast networks for video recognition. *arXiv preprint arXiv:2001.08740*, 2020. 4

[54] J. Zhang, Y. Wang, P. Molino, L. Li, and D. S. Ebert. Manifold: A Model-Agnostic Framework for Interpretation and Diagnosis of Machine Learning Models. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):364–373, 2019. doi: 10.1109/TVCG.2018.2864499 3

[55] Z. Zhang, P. Webster, V. S. Uren, A. Varga, and F. Ciravegna. Automatically extracting procedural knowledge from instructional texts using natural language processing. In *LREC*, vol. 2012, pp. 520–527. Citeseer, 2012. 4

[56] X. S. Zheng, C. Foucault, P. Matos da Silva, S. Dasari, T. Yang, and S. Goose. Eye-wearable technology for machine maintenance: Effects of display position and hands-free operation. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pp. 2125–2134, 2015. 2

[57] K. Zhou, Z. Liu, Y. Qiao, T. Xiang, and C. C. Loy. Domain generalization in vision: A survey. *arXiv preprint arXiv:2103.02503*, 2021. 9

[58] X. Zhou, R. Girdhar, A. Joulin, P. Krähenbühl, and I. Misra. Detecting twenty-thousand classes using image-level supervision. In *Computer Vision–ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part IX*, pp. 350–368. Springer, 2022. 4