Università degli Studi di Milano - Bicocca
Dipartimento di Informatica, Sistemistica e Comunicazione
Corso di Laurea in Informatica

A Multimodal Pipeline for Automated Egocentric Dataset Creation and Annotation

Relatore: Prof. Simone Bianco

Correlatore: Prof. Raimondo Schettini

Correlatore: Dott. Danilo Pau

Tesi di Laurea di:
Matteo Breganni
Matricola 869549

Anno Accademico 2024-2025

Abstract

This work presents a multimodal pipeline for the automated creation and annotation of egocentric datasets using Meta's Project Aria glasses. By combining audio, visual, and motion data, the system allows users to annotate objects in their environment simply by looking and speaking. A user-friendly tool streamlines processing and parameter tuning, automatically generating datasets with object detections and pixel-level segmentations as annotations. The pipeline is evaluated for accuracy and efficiency, showing strong potential for real-world deployment. The work enables faster, more intuitive dataset creation for egocentric AI applications.

Contents

Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
AR	Augmented Reality
ASR	Automatic Speech Recognition
CLIP	Contrastive Language–Image Pre-training
COCO	Common Objects in Context
DFOV	Diagonal Field of View
FPS	Frames Per Second
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HRV	Heart Rate Variability
ID	IDentifier
IMU	Inertial Measurement Unit
IoU	Intersection over Union
ML	Machine Learning
MPS	Machine Perception Services
MRPT	Mobile Robot Programming Toolkit
NMS	Non-Maximal Suppression
PAN	Path Aggregation Network
POV	Point Of View
PPG	Photoplethysmography
RANSAC	RANdom SAmple Consensus
RepVL-PAN	Re-parameterizable Vision-Language Path Aggregation Network

RGB	Red Green Blue
RSSI	Received Signal Strength Indicator
SA-1B	Segment Anything - 1 Billion
SA-V	Segment Anything Video
SAM	Segment Anything Model
SAM2	Segment Anything Model 2
SIFT	Scale-Invariant Feature Transform
UI	User Interface
VAD	Voice Activity Detection
ViT	Vision Transformer
VOS	Video Object Segmentation
VR	Virtual Reality
VRS	Vision Record Stream
YOLO	You Only Look Once
YOLOv8	You Only Look Once version 8
YOLO-World	You Only Look Once - World

Introduction

Artificial Intelligence (AI), and in particular Machine Learning (ML), has undergone rapid advancements over the past decade, transforming fields such as healthcare, robotics, autonomous systems, and human-computer interaction. At the core of these developments lies data, specifically large, high-quality, and well-annotated datasets that enable models to learn meaningful patterns and generalize effectively to real-world scenarios. Whether in image classification, object detection, or semantic segmentation, the performance of modern machine learning systems is closely tied to the availability and quality of training data. However, the process of collecting and annotating such datasets remains a major bottleneck, often requiring significant human effort, time, and financial resources.

This challenge is especially pronounced in the domain of egocentric vision, a subfield of computer vision that focuses on understanding visual scenes from a first-person perspective. Unlike traditional third-person video recordings, egocentric data captures the world as seen through the eyes of the user, offering unique insights into human behavior, attention, and interaction with the environment. This perspective is particularly valuable for applications such as assistive technologies for the visually impaired, wearable AI assistants, and activity recognition. With the emergence of wearable devices like smart glasses, such as Meta's Project Aria [1], there is growing interest in leveraging continuous, multimodal sensory streams to build intelligent systems that understand and support human activities in real time.

Despite the promise of egocentric vision, progress in this area has been hindered by the scarcity of large-scale, densely annotated datasets. Existing datasets are often limited in size, diversity, or annotation richness, and creating new ones manually can be prohibitively expensive. Traditional annotation workflows involve watching hours of video and drawing bounding boxes or segmentation masks frame by frame, a process that is not only tedious but also prone to inconsistencies and errors.

This thesis addresses the critical need for efficient, scalable, and intuitive methods to create annotated egocentric datasets. This work explores how multimodal sensing can be leveraged to reduce human effort in the labeling process, by capturing natural interactions between users and their environment. The aim is to support the creation of high-quality datasets with significantly less manual intervention, enabling faster development and deployment of egocentric AI systems.

The purpose of this study is to design, implement, and evaluate a multimodal pipeline that enables efficient and intuitive creation of annotated egocentric datasets using data from wearable sensors, specifically using Meta's Project Aria glasses, with the goal of significantly reducing the manual effort traditionally

required in the labeling process.

The significance of this work lies in both its practical utility and methodological contribution. On the practical side, the pipeline enables researchers and developers to rapidly create custom datasets tailored to specific applications or environments, such as labeling household objects for a home assistant robot or identifying tools in a workshop, without relying on large annotation teams. The system includes a user-friendly interface that supports parameter customization, simultaneous processing of multiple recordings, and visualization of results and debug data, making it accessible for non-expert users. On the methodological side, this work investigates the integration of cutting-edge vision models such as You Only Look Once - World (YOLO-World) [2] for open-vocabulary object detection and Segment Anything Model 2 (SAM2) [3] for promptable segmentation within a real-world and multimodal framework, together with WhisperX [4] for efficient speech-to-text. The study evaluates how these models perform on egocentric data, highlights practical challenges, and identifies key limitations in their application.

The pipeline is evaluated for annotation accuracy, error sources, and computational cost. Annotation accuracy is measured using Intersection over Union (IoU) between the system's outputs and ground truth from Common Objects in Context (COCO) dataset images. To identify failure points, instances with poor classification are analyzed to determine which pipeline stage was the source of the error, revealing key weaknesses. Computational demands are assessed relatively, comparing processing time and resource use across pipeline components. This evaluation highlights trade-offs and guides improvements in scalability and robustness.

The document begins with a detailed examination of the state-of-the-art methods selected for integration into the pipeline. This sets the context for the core part of the work, which details the design choices and implementation of the multimodal pipeline, including the development of a dedicated benchmarking system used to test the pipeline on real datasets. This is followed by an analysis of the results, where the pipeline is evaluated in terms of annotation accuracy, robustness, and computational efficiency, along with a discussion of key limitations and practical challenges. The document concludes with suggestions for future improvements to the system.

Chapter 1

State of the Art Components

This chapter provides an in-depth examination of the key components that form the foundation of the proposed pipeline. It focuses on the state-of-the-art models and technologies leveraged in the system, detailing the core deep learning models used for speech processing, object detection, and segmentation, as well as Meta's Project Aria glasses, which serves as the data acquisition device. This analysis sets the technical groundwork for the pipeline's design and implementation choices presented in the following chapter.

1.1 Meta's Project Aria

Project Aria [1] is a research initiative developed by Meta Reality Labs Research aimed at advancing the development of intelligent, context-aware glasses that can understand human experiences from a first-person (egocentric) perspective. Unlike consumer products, Project Aria is designed as a research platform to accelerate innovation in areas such as scene understanding, human-computer interaction, and AI-driven assistance systems.

At its core, Project Aria provides a comprehensive ecosystem for egocentric perception research, combining hardware, software tools, and publicly available datasets. This ecosystem allows researchers to explore how AI models can interpret complex dynamic environments through synchronized audio, visual, and motion signals, all captured naturally from the user's point of view.

One of the key contributions of Project Aria is the Project Aria Tools [5], a comprehensive software suite designed to facilitate the access, visualization, and processing of egocentric recordings containing time-synchronized multimodal sensor streams. The suite includes cloud-based Application Programming Interfaces (APIs) [6] that expose advanced spatial AI and machine perception capabilities, powered by Meta's proprietary algorithms. These include hand tracking, eye-gaze estimation, 3D scene reconstruction, and user pose estimation. This integrated ecosystem significantly lowers the barrier for egocentric AI research while promoting reproducibility and methodological consistency across studies.

Meta has also released open datasets [7] collected through Project Aria,

providing researchers with real-world, egocentric data that supports the development of models for first-person perception. These datasets offer rich multimodal recordings from daily environments, enabling experimentation and validation in areas like scene understanding and human behavior analysis.

In addition to data, Meta has shared pre-trained models [8] derived from Project Aria. One notable example is the eye tracking model released in 2024 [9], which estimates 2D gaze direction from on-glasses eye cameras, providing gaze localization on the visual frame. A newer, more advanced eye gaze model is also available exclusively through Meta's cloud-based APIs. Unlike the older open version, this updated model goes beyond 2D direction by estimating gaze depth, predicting how far the user is looking into the scene by calculating the 3D convergence point of both eyes. This model also supports personalized calibration, resulting in significantly more accurate gaze estimation.

1.1.1 Meta's Project Aria Glasses

The Project Aria Glasses (Figure 1.1) are the device at the heart of the initiative and form a core part of the Aria Research Kit distributed to researchers.

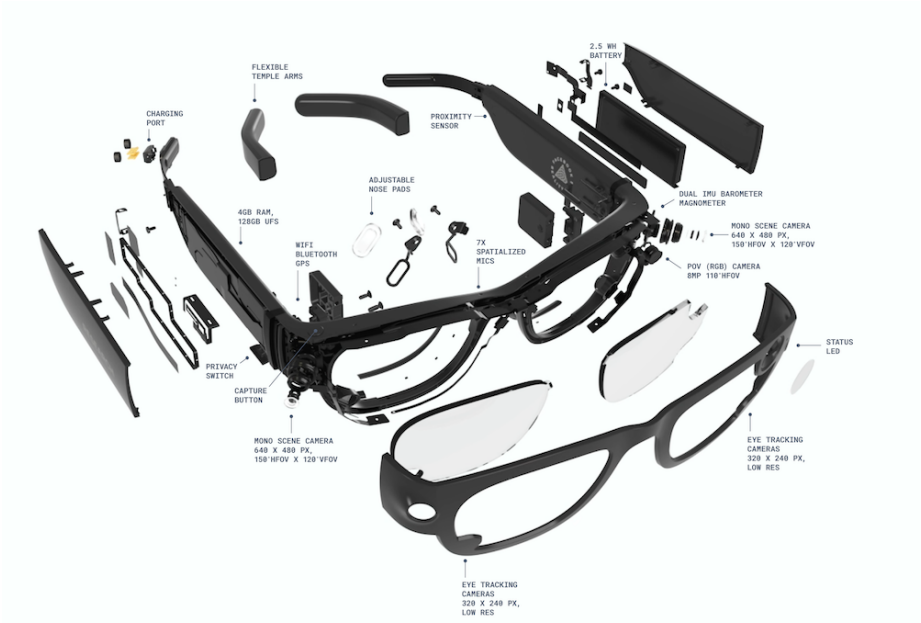


Figure 1.1: Architectural overview of Meta's Project Aria glasses.
Source: Project Aria: A New Tool for Egocentric Multi-Modal AI Research [1].

This non-display, sensor-rich wearable device is designed specifically for first-person perception research, enabling the collection of high-fidelity, multimodal data in real-world environments. It does not include a screen or speakers, reinforcing its role as a dedicated data capture device rather than a consumer-facing product. All sensor data is precisely time-synchronized, allowing for accurate cross-modal alignment. The device does not allow on-device custom computation or running third-party algorithms during data capture, as its

onboard processing capabilities are extremely limited. These constraints are primarily due to power and thermal limitations, which restrict the device from even simultaneously recording all sensor streams at their maximum resolutions. As a result, all data must be stored and transferred for offline processing, and researchers must rely on external systems for model inference and application development.

The Project Aria glasses include the following sensors:

- Point Of View (POV) Red Green Blue (RGB) Camera (Figure 1.2): A forward-facing, high-resolution rolling-shutter RGB camera with a 110° horizontal field of view, 2880×2880 pixel resolution, and a 4° downward tilt.

Figure 1.2: Aria glasses' front-facing RGB camera.
Source: Project Aria: A New Tool for Egocentric Multi-Modal AI Research [1].

- Mono Scene Cameras (Figure 1.3): Two monochrome global-shutter cameras mounted on the left and right sides of the glasses, each with a 150° horizontal field of view and an outward angle to maximize peripheral coverage while allowing some stereo overlap. These cameras operate at 640×480 resolution and use fisheye lenses to capture wide-angle scenes.

Figure 1.3: Aria glasses' side grayscale fisheye cameras.
Source: Project Aria: A New Tool for Egocentric Multi-Modal AI Research [1].

- Eye Tracking Cameras (Figure 1.4): Two monochrome, global-shutter, inward-facing cameras designed to capture eye movements and pupil position. Each camera has a Diagonal Field of View (DFOV) of 80° and typically operates at a resolution of 320×240 pixels.

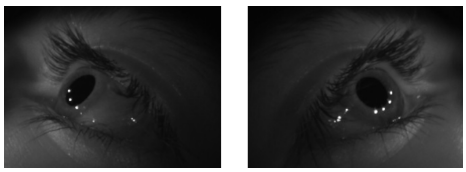


Figure 1.4: Aria glasses' grayscale eye cameras.

- Inertial Measurement Units (IMUs): Two IMUs, one on each side of the glasses, sampled at different rates and ranges to diversify sensor characteristics. The left IMU runs at 800 Hz (saturations: 4g, 500°/s), while the right one runs at 1000 Hz (saturations: 8g, 1000°/s). This helps ensure that their error behaviors are less correlated, improving robustness for sensor fusion and motion estimation.

- Microphones: A 7-microphone array distributed across the frame of the glasses: five positioned on the front and one on each side, enabling high-fidelity spatial audio capture. The system records at 24-bit depth with a configurable sample rate of up to 48 kHz.

- Magnetometer: Measures 3D ambient magnetic field with 0.1 µT resolution at 10 Hz, mounted on the rim to reduce interference.

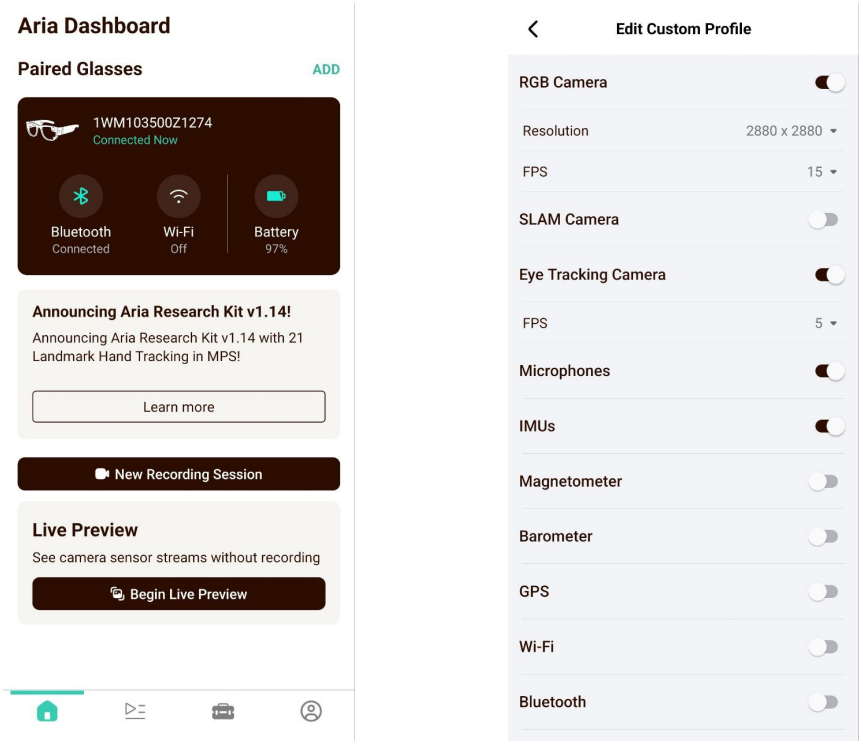
- Barometer & Thermometer: Captures air pressure (0.66 Pa) and temperature (0.005 °C) at 50 Hz.

- Global Navigation Satellite System (GNSS) Receiver: Provides location (latitude, longitude, height) and timing using Global Positioning System (GPS) and Galileo at 1 Hz.

- Wi-Fi & Bluetooth: Scans for nearby networks and beacons, recording signal strength (Received Signal Strength Indicator (RSSI)) at 0.1 Hz on both 2.4 GHz and 5 GHz bands.

The Project Aria Glasses are managed through the Aria app (Figure 1.5a), a dedicated mobile application used to control the device, configure recording sessions, and manage data. The app allows researchers to pair with the glasses via Bluetooth, initiate and stop recordings, and select predefined recording profiles that determine which sensors are active and at what settings.

Many of these sensor settings can be configured through recording profiles (Figure 1.5b), which are selected at the start of a recording and allow researchers to enable or disable specific sensors. This flexibility is essential because continuously recording all sensors at their maximum resolution and frame rate is not feasible due to the device's power, thermal, and onboard storage limitations.



(a) Aria app: device pairing and recording control.

(b) Aria app: sensor configuration via recording profiles.

Figure 1.5: Screenshots of the Meta Aria mobile app's interface.

All sensor data recorded by the Project Aria Glasses is stored in the Vision Record Stream (VRS) file format. VRS is an open, extensible container specifically designed for capturing and playback of time-synchronized streams of Augmented Reality (AR) sensor data. It supports very large file sizes and efficiently organizes data as time-ordered records for each sensor stream, preserving precise temporal alignment across modalities.

1.2 Whisper

The development of large-scale models has transformed the landscape of Automatic Speech Recognition (ASR), with approaches diverging along two main paths: unsupervised pre-training and supervised learning. Models like Wav2Vec 2.0 [10] rely on unsupervised pre-training, learning rich representations from raw audio without transcriptions, followed by fine-tuning on small labeled datasets to perform specific tasks. While effective, this paradigm decouples representation learning from task execution (the model learns representations without knowing what the downstream task is, the task is only introduced later during fine-tuning) and still depends on downstream supervision to produce usable outputs. This two-stage process limits end-to-end optimization and often requires task-specific labeled data for fine-tuning, which may not be available for low-resource languages or niche domains, making deployment complex and

less generalizable.

In contrast, fully supervised approaches train models directly on labeled audio-transcript pairs, enabling end-to-end learning of both acoustic and linguistic patterns without the need of fine-tuning. However, traditional supervised methods are constrained by the availability of high-quality annotated data, with most academic datasets limited to a few thousand hours of speech. This scale bottleneck has historically prevented supervised models from achieving the same level of generalization as their unsupervised counterparts, until the emergence of large-scale weakly supervised frameworks like Whisper [11].

Whisper, developed by OpenAI, represents a shift toward large-scale, end-to-end supervised training, but with a crucial distinction: it is trained on a massive dataset of 680,000 hours of audio-transcript pairs collected from web-based sources, rather than manually curated recordings. While this setup is supervised in the sense that input audio is paired with output text, the labels are not human-verified and may contain misalignments, transcription errors, or noise. As such, Whisper operates in a weakly supervised regime, leveraging scale and diversity to compensate for lower label fidelity, a strategy more akin to modern vision models trained on web-scraped data than to traditional ASR systems.

To improve data quality, OpenAI applied heuristic filtering to remove unreliable or low-fidelity transcriptions. A particular focus was on eliminating machine-generated captions, such as those produced by automated captioning systems like YouTube's auto-captioning, which tend to be error-prone and less suitable for training robust models. These were identified through patterns such as repetitive phrasing, unnatural punctuation, and characteristic error signatures. Additionally, audio-text alignment checks were used to discard misaligned or poorly synchronized pairs, and the language of the transcript was verified to match the spoken language in the audio. While the dataset remains weakly supervised, these preprocessing steps help ensure a higher proportion of meaningful, human-influenced transcriptions, improving the signal-to-noise ratio without sacrificing scale.

This weak supervision enables Whisper to learn robust mappings from speech to text across a wide range of accents, domains, recording conditions and recording qualities. The dataset includes not only transcriptions in English but also 117,000 hours of non-English speech spanning 96 languages, as well as 125,000 hours of translated content (speech in one language paired with English text), allowing the model to jointly learn multilingual transcription and speech translation.

Whisper simplifies the speech recognition pipeline by handling multiple tasks within a single model: transcription, translation, language identification, and Voice Activity Detection (VAD). Instead of relying on separate components for each function, Whisper uses a unified architecture that performs end-to-end speech processing, reducing system complexity.

Audio is split into 30-second segments paired with their corresponding transcripts. Non-speech segments are included in training with reduced frequency and serve as supervision for VAD. The model learns to output a special *nospeech* token when no speech is present, effectively integrating this capability directly into the core system.

Whisper uses a standard encoder-decoder Transformer architecture (Figure 1.6), avoiding novel design elements to focus on the impact of scale and data. The input is an 80-channel log-mel spectrogram, processed by a small

convolutional stem before being fed into the encoder. The decoder generates text autoregressively (the model produces the output text one token at a time, and each new token is predicted based on the previously generated tokens) and is conditioned on prior transcript context with some probability, helping it use the linguistic context to resolve ambiguities.

The encoder adds sinusoidal position embeddings (fixed signals that encode timing information) after the convolutional stem and processes the sequence through multiple Transformer blocks using pre-activation residual connections, a design where layer normalization and nonlinearities are applied before the main transformation to stabilize and accelerate training. The decoder generates text step by step, using self-attention to track what it has already output and cross-attention to access key information from the encoder's audio representation. This allows it to focus on the relevant parts of the speech signal when predicting each word. The encoder and decoder have the same size and structure.

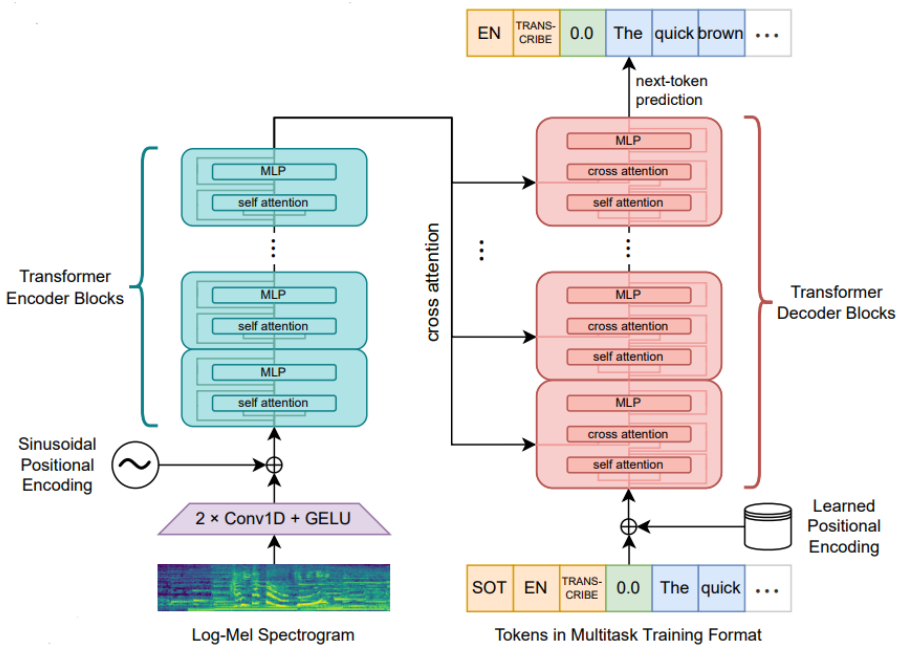


Figure 1.6: Whisper's architecture overview.

Source: Robust Speech Recognition via Large-Scale Weak Supervision [11].

Rather than depending on external modules for additional speech processing functions, Whisper uses a simple format (Figure 1.7) to specify all tasks and conditioning information as a sequence of input tokens to the decoder. The model uses special tokens to indicate task type, language, and formatting options, allowing it to perform different functions based on the input context. For example, before generating output, Whisper first predicts the spoken language using a language-specific token, followed by a task token (*transcribe* or *translate*) and a timestamp mode flag (*notimestamps* or *timestamped* output). When timestamps are enabled, the model emits them as paired markers indicating the start and end times of each text segment, effectively aligning the transcription with the original audio timeline, enabling coarse-grained temporal alignment. During

training, segments are prefixed with a *startoftranscript* token, and the model learns to generate structured outputs autoregressively, one token at a time.

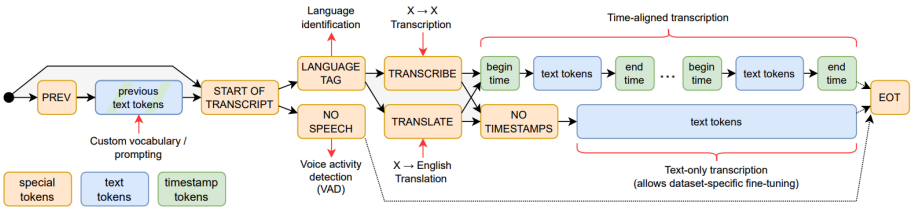


Figure 1.7: Whisper's multitask format.

Source: Robust Speech Recognition via Large-Scale Weak Supervision [11].

Whisper achieves state-of-the-art performance in ASR by leveraging massive scale and weak supervision, outperforming previous models, especially in multilingual, low-resource, and zero-shot settings. Although not always optimal for specialized tasks, Whisper sets a new benchmark for general-purpose speech recognition.

1.2.1 WhisperX

While Whisper achieves strong performance in a variety of speech recognition tasks, its design introduces practical limitations when applied to real-world, long-form audio such as podcasts, lectures, or meetings. Whisper processes audio in fixed 30-second segments, relying on predicted timestamps to stitch outputs together. However, this rigid segmentation does not respect natural speech boundaries: cuts can occur mid-word, mid-phrase, or during pauses, leading to fragmented context and incomplete utterances. As a result, the model may produce incomplete or grammatically inconsistent transcriptions, especially when words or phrases are split across segments. Furthermore, small timestamp inaccuracies in individual segments can accumulate over time, causing noticeable drift and temporal misalignment in the final output.

To address these limitations, WhisperX [4] enhances Whisper with a lightweight post-processing pipeline designed for accurate and efficient long-form audio transcription. Instead of using fixed 30-second segments, WhisperX first applies an external VAD model to identify speech regions and segment the audio at natural pause points, reducing the risk of cutting mid-word or mid-sentence. These segments are then processed in batches using Whisper, improving inference efficiency and contextual coherence. Finally, WhisperX employs a phoneme-based forced alignment model to refine the output, producing precise word-level timestamps that correct Whisper's coarse and often drifting time estimates.

Crucially, WhisperX does not require retraining or modifying Whisper. It enhances the frozen model with efficient external components for better segmentation, alignment, and overall transcription performance, making it significantly more effective for long-form audio.

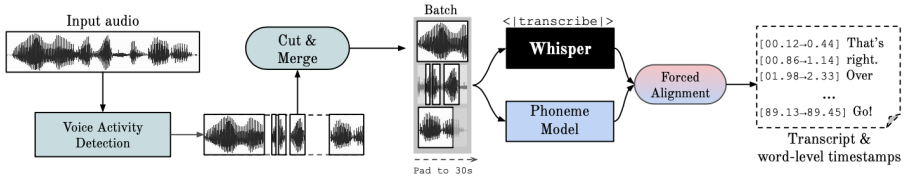


Figure 1.8: WhisperX's architecture.

Source: WhisperX: Time-Accurate Speech Transcription of Long-Form Audio [4].

A key component of WhisperX is the use of Voice Activity Detection (VAD) to pre-segment the input audio into regions of speech and non-speech. By identifying active speech segments upfront, VAD allows WhisperX to skip unnecessary ASR processing during silent periods, reducing computational load and inference time. Moreover, segmenting audio at low-activity boundaries helps avoid cutting through words or phrases, minimizing context fragmentation and improving transcription coherence. These well-defined speech segments can then be processed in parallel, enabling faster, batched inference.

After initial segmentation through VAD, speech regions can vary widely in length: some exceeding Whisper's 30-second input limit, others being too short to provide meaningful context. To ensure compatibility with Whisper's architecture and maintain transcription quality, WhisperX applies a two-step refinement process: cut and merge. Longer speech segments are split at points of minimal voice activity, typically near silent or low-energy regions, using the VAD model's activation scores. This min-cut strategy reduces the risk of interrupting words or phrases, preserving linguistic coherence while ensuring no segment exceeds the model's maximum input duration. On the other hand, very short segments are problematic: they offer limited context and increase computational overhead due to the higher number of individual inference calls. To address this, WhisperX merges adjacent segments whose combined duration falls below a threshold set empirically to match Whisper's training segment length (30 seconds). This maximizes contextual continuity and aligns the input distribution with what the model encountered during training, improving both accuracy and efficiency.

To achieve precise word-level timestamps, WhisperX performs forced alignment using a pre-trained phoneme recognition model such as Wav2Vec 2.0, which segments speech into fine-grained phonetic units. By matching the phoneme sequence derived from the audio with the expected phonetic transcription of the text, WhisperX computes accurate temporal boundaries for each word. This process corrects the coarse and often inconsistent timestamps produced by Whisper, delivering accurate word-level timestamps.

WhisperX was selected for this thesis work due to its ability to deliver highly accurate word-level timestamps, addressing a key limitation of Whisper, which only provides coarse, sequence-level timestamps that are often inconsistent or misaligned. In addition to improved temporal precision, the significant speed improvement (12 times faster than Whisper) achieved through parallel batch processing of intelligently segmented speech chunks was another key reason for choosing WhisperX for this project.

1.3 YOLO-World

In recent years, deep neural networks have driven substantial advancements in object detection, with frameworks like You Only Look Once (YOLO) [12] setting benchmarks for speed and accuracy in real-time applications. Despite these successes, conventional object detectors are trained to recognize only a predefined set of categories. This constraint limits their ability to generalize to new or user-defined categories, making them less suitable for dynamic, real-world environments where flexibility and scalability are essential.

YOLO-World [2] is a cutting-edge extension of the YOLO architecture designed for open-vocabulary object detection. Unlike traditional detectors, YOLO-World can recognize objects from a vast and flexible vocabulary described in natural language, enabling it to detect categories that were not seen during training. It leverages a pre-trained Contrastive Language–Image Pre-training (CLIP) [13] text encoder to interpret user-provided category names or phrases, aligning them with visual features extracted from images. To bridge the gap between language and vision efficiently, YOLO-World introduces a re-parameterizable architecture with a Re-parameterizable Vision-Language Path Aggregation Network (RepVL-PAN) that dynamically fuses textual and visual information during training. Crucially, after training, the text encoder can be detached, and the text embeddings are re-parameterized into the detection network's weights, enabling fast, end-to-end inference without sacrificing speed.

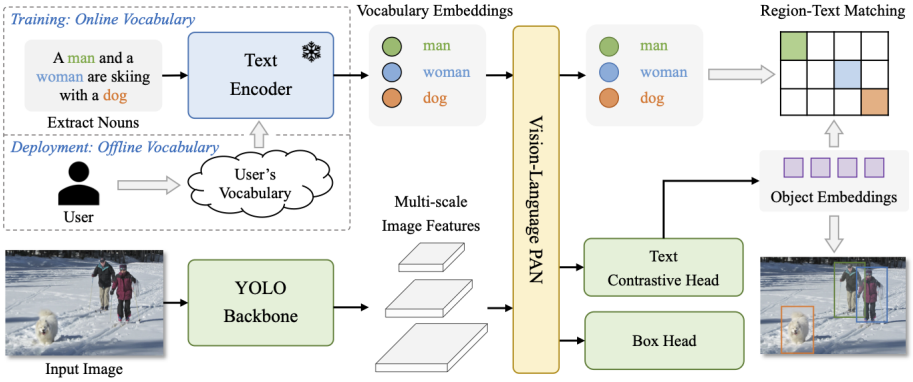


Figure 1.9: YOLO-World's architecture.

Source: YOLO-World: Real-Time Open-Vocabulary Object Detection [2].

The YOLO-World architecture (Figure 1.9) consists of three main components: a YOLO detector, a text encoder, and a RepVL-PAN. The text encoder converts input category names into embeddings, while the YOLO backbone extracts multi-scale image features. These are fused through RepVL-PAN, which aligns visual and textual information via cross-modality interaction, enabling fast and flexible open-vocabulary object detection while preserving the efficiency of the YOLO framework.

YOLO-World is based on You Only Look Once version 8 (YOLOv8), which uses a Darknet backbone to extract visual features from the input image. This backbone processes the image through a series of convolutional layers, generating rich, multi-scale feature maps.

YOLO-World incorporates a text encoder to enable language-driven detection. Given an input text prompt, it uses the pre-trained transformer-based text encoder from CLIP to generate text embeddings. This encoder excels at aligning language with visual concepts, offering stronger visual-semantic understanding compared to standard language models. For complex inputs like captions or referring expressions, a simple n-gram algorithm is used to extract relevant noun phrases, which are then encoded to represent the target object categories. An n-gram is a sequence of n consecutive words. The algorithm generates all possible n-grams from the input text and applies part-of-speech tagging to identify those that follow noun phrase patterns, such as determiner-adjective-noun sequences, filtering out less meaningful combinations. This allows YOLO-World to automatically detect and localize multiple relevant objects mentioned in a sentence without requiring manual annotation or prior knowledge of the categories.

After getting the multi-scale image features from the YOLO backbone and the text embeddings from the CLIP encoder, the next step is to effectively align and fuse these visual and linguistic modalities. This is where the RepVL-PAN comes in.

Feature pyramid fusion refers to the process of combining features from different layers of a neural network, each representing the image at a different scale, to enable robust detection of objects both large and small. In standard YOLO detectors, the Path Aggregation Network (PAN) strengthens this by fusing high-level semantic information (from deeper layers) with high-resolution details (from earlier layers), creating rich, multi-scale feature maps used for prediction.

RepVL-PAN extends this idea by injecting language knowledge into the fusion process. It uses cross-attention modules to dynamically adjust the visual features based on the text embeddings, meaning the network learns to focus on image regions that match the meaning of the given text. For instance, if the input text is "a red car", the model strengthens the features in areas containing red objects and car-like shapes, effectively aligning vision with language. This allows the detector to become conditionally aware of what it should look for, based on the provided description. This creates dynamic, language-conditioned feature pyramids that are more semantically aware.

After obtaining the enhanced multi-scale features from the RepVL-PAN, the detector uses them to perform object localization and classification. At each spatial location of the feature maps, the network predicts two main outputs: bounding boxes and object embeddings. The box head is a lightweight neural network head that regresses the coordinates of the bounding box for each potential object. Specifically, it predicts four values per anchor-free keypoint (reference point on the image): the distance from the current feature grid point to the left, top, right, and bottom edges of the object. These values are then used to compute the final bounding box coordinates in pixel space.

Alongside the box predictions, the text contrastive head generates an object embedding for each detected region: a feature vector that captures the semantic content of the region. This embedding is trained to be closely aligned with the text embedding of the correct category, using a region-text contrastive loss. Specifically, during training, the model compares each predicted object embedding with all text embeddings of candidate categories. The loss function encourages the embedding of a detected object to become more similar to its

matching text description while becoming less similar to others.

The final detection result is obtained by combining the predicted bounding boxes and the semantically aligned object embeddings. For each detected region, the object embedding is compared (through cosine similarity) to a set of pre-encoded text embeddings corresponding to the desired categories. This similarity score serves as the classification confidence for each category, effectively replacing the fixed classifier used in traditional detectors. The bounding box is then assigned to the category with the highest similarity score.

At inference time, YOLO-World operates in a highly efficient and streamlined manner. Once the desired categories or prompts are known, their corresponding text embeddings are pre-computed using the CLIP text encoder and cached offline. The key innovation enabling fast deployment is the re-parameterization of the RepVL-PAN: during training, the network uses cross-attention modules in RepVL-PAN to dynamically fuse text and visual features, allowing the model to learn how to align specific words (like "cat" or "red car") with relevant image regions. However, this fusion process involves the text encoder and attention operations, which would slow down inference if kept as-is. To eliminate this overhead, YOLO-World applies a structural transformation at inference: the influence of each pre-computed text embedding is re-parameterized into the weights of the convolutional layers within the RepVL-PAN. This means the language-aware feature fusion, originally achieved through attention, is converted into standard convolution operations that are hard-coded with the knowledge of the given categories. As a result, the entire detection pipeline becomes a single, compact feed-forward network that no longer depends on the text encoder or real-time text processing.

In summary, YOLO-World is fast, accurate, and supports open-vocabulary detection, making it ideal for real-world applications. Its ability to detect objects based on natural language prompts, rather than a fixed set of categories, allows it to generalize to unseen or user-defined classes. This is particularly crucial for this thesis work, as the target dataset may contain arbitrary objects depending on user input, requiring a flexible and scalable detection system that does not rely on predefined categories.

1.4 Segment Anything Model

Segment Anything Model (SAM) [14], introduced by Meta AI, represents a foundational shift in the field of image segmentation by redefining how models are trained, scaled, and applied across diverse visual tasks. At its core, SAM is designed around a novel, promptable segmentation framework, a paradigm that generalizes the traditional notion of segmentation by enabling the model to generate accurate masks in response to various forms of input prompts, such as points, bounding boxes, or text.

A key innovation behind SAM lies not only in its architecture but also in the development of a novel data engine designed to address the lack of large-scale, diverse segmentation annotations, which are not available at web scale.

To address this, a novel three-stage data engine was developed to scale mask annotation efficiently, combining human feedback with model-generated predictions, ultimately enabling the creation of Segment Anything - 1 Billion (SA-1B) (Figure 1.10), the largest segmentation dataset to date, featuring over

1.1 billion masks across 11 million different images.

Figure 1.10: SA-1B's dataset examples.

Source: Segment Anything [14].

The process evolved through the following stages:

1. Assisted-manual stage: Expert annotators labeled objects using an interactive tool powered by SAM, clicking on points to generate real-time mask predictions. Masks could be refined with brush and eraser tools. No semantic constraints were applied, annotators labeled any visible object in order of prominence (proceeding to the next image after annotation took more than 30 seconds). As the model improved through iterative retraining, average annotation time dropped from 34 to 14 seconds per mask, and the average number of masks per image increased from 20 to 44. This stage yielded 4.3 million masks across 120,000 images.

2. Semi-automatic stage: To capture less prominent objects, a detector trained on initial masks automatically filled in confident segments. Annotators then labeled only the remaining unsegmented objects, focusing effort on harder cases. This stage added 5.9 million masks across 180,000 images, increasing diversity and raising the average masks per image to 72.

3. Fully automatic stage: In the final stage, masks were generated entirely without human input. The model was prompted with a dense 32×32 grid of points across each image, producing multiple masks per point, including parts and wholes in ambiguous cases. To select high-quality outputs:

    - the model's Intersection over Union (IoU) predictor estimated mask confidence;
    - threshold consistency ensured stability by checking that small changes in mask probability thresholds produced similar shapes;
    - Non-Maximal Suppression (NMS) removed duplicate masks by keeping only the highest-confidence prediction among overlapping masks, effectively filtering out redundant outputs.

Zoomed image crops were also used to improve small-object accuracy. This stage was applied to 11 million images, generating the full 1.1 billion masks in SA-1B.

SAM adopts a modular architecture (Figure 1.11) composed of three main components: an image encoder, a flexible prompt encoder, and a lightweight mask decoder. This design enables the model to generalize across a wide range of segmentation tasks and input modalities while supporting real-time, interactive use.

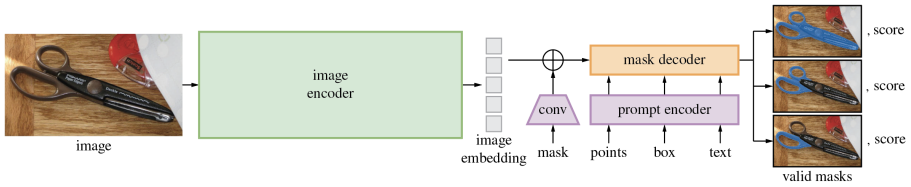


Figure 1.11: SAM's architecture.

Source: Segment Anything [14].

The image encoder is responsible for transforming the input image into a rich, compressed representation. To ensure strong visual priors and scalability, SAM employs a pre-trained Vision Transformer (ViT) [15] backbone, adapted to process high resolution images.

The prompt encoder allows SAM to interpret diverse forms of user input in a unified embedding space. Sparse prompts, such as points or bounding boxes, are encoded using positional embeddings and summed with learned embeddings for each prompt type. Dense inputs, such as previously generated masks, are embedded via a small convolutional network and then added element-wise to the image embedding. This flexibility in prompt representation is central to SAM's ability to support a wide variety of interaction modes.

Finally, the mask decoder fuses the image and prompt embeddings to produce segmentation masks. Inspired by Transformer-based detection architectures, it uses a modified Transformer decoder block with cross-attention mechanisms to integrate visual and prompt information. A dynamic mask head then generates the output mask based on the fused features. Notably, the decoder is designed for speed and runs efficiently on the embedded representations, enabling real-time mask prediction even on high-resolution inputs.

A key challenge in open-world segmentation arises from ambiguity: a single prompt (e.g., a point on an object) may correspond to multiple valid interpretations, such as a whole object, a part, or a subpart. For instance, clicking on a dog's ear could reasonably refer to the ear itself, the dog's head, or the entire animal. In such cases, a model constrained to produce a single output mask tends to generate an average or blended segmentation, which may not align with any of the user's intended interpretations. To address this, SAM is designed to predict multiple distinct masks for a single prompt, allowing it to disambiguate plausible segmentations rather than collapse them into a single, potentially incorrect result. The model outputs three masks (Figure 1.12) per prompt, each associated with a confidence score estimated by the IoU predictor. The choice of three outputs was empirically motivated: in most real-world scenarios,

hierarchical object structures are rarely deeper than three levels (e.g., whole, part, subpart). By generating a small set of diverse, high-quality candidates, SAM enables users or downstream systems to select the most appropriate mask interactively or programmatically.

Figure 1.12: SAM's output masks example.

Source: Segment Anything [14].

The overall design of SAM is strongly motivated by efficiency, enabling real-time, interactive segmentation across diverse platforms. Once the image embedding is precomputed, the prompt encoder and mask decoder operate lightweight computations that can run entirely on a web browser and even on CPU, with an average latency of approximately 50 milliseconds per prompt. This fast inference time ensures a seamless user experience, supporting immediate feedback during interactive segmentation, such as adding points or adjusting prompts on the fly.

1.4.1 Segment Anything Model 2

While the original SAM established a powerful foundation for promptable image segmentation, real-world visual data is often dynamic, with objects moving and changing over time. A static image provides only a limited snapshot of this complexity, and extending segmentation to video introduces significant challenges, particularly in handling motion, temporal consistency, and efficient processing across large sequences of frames.

To address these limitations, Meta AI introduced Segment Anything Model 2 (SAM2) [3], a unified framework that extends SAM's capabilities to video segmentation and improves the image segmentation. SAM2 takes prompts such as points, bounding boxes, or masks on any video frame to generate a masklet, defined as the segmentation of a target object across a portion of the video. These masklets can be progressively refined by providing additional prompts in subsequent frames, enabling interactive and iterative segmentation over time.

The key modification lies in the introduction of a temporal memory component that enables the model to retain and leverage information across video frames. While SAM processes each image independently, SAM2 maintains a dynamic memory of the target object and past interactions, allowing it to produce coherent time-aware segmentations.

The creation of SAM2's training dataset follows a similar philosophy to that of the original SAM, leveraging a human-in-the-loop data engine to scale annotation efficiently. However, instead of focusing on static images, the process

is extended to video, resulting in the Segment Anything Video (SA-V) dataset, which is significantly larger than any prior video segmentation dataset in terms of mask count.

Like SAM's data engine, the pipeline combines model predictions with interactive human refinement. The initial stage mirrors SAM's per-frame annotation approach, using the original model to generate masks frame-by-frame without temporal propagation, ensuring high spatial accuracy. In subsequent stages, SAM2 itself is integrated into the loop: first using a mask-only version for temporal propagation, and finally the full model, which leverages object memory to propagate segmentations across frames. This allows annotators to refine predictions interactively with minimal effort using points or masks rather than annotating each frame from scratch. As the model improves through iterative retraining, annotation speed increases substantially, enabling the efficient generation of high-quality spatio-temporal masks at scale.

The resulting SA-V dataset (Figure 1.13) comprises 35.5 million masks across 50,900 videos, exceeding the scale of existing video segmentation datasets by a factor of 53 in terms of annotated masks, making it the largest and most diverse video segmentation dataset to date.

SAM2 is trained jointly on image and video data, simulating real-world interactive segmentation scenarios. During training, the model processes sequences of 8 frames, with up to 2 frames selected for prompting. Prompts are sampled probabilistically, using ground-truth masks, points, or bounding boxes, and the model receives simulated corrective feedback based on its predictions, mimicking iterative user refinement. The objective is to predict the full spatio-temporal masklet sequentially, learning to integrate both visual context and temporal memory.

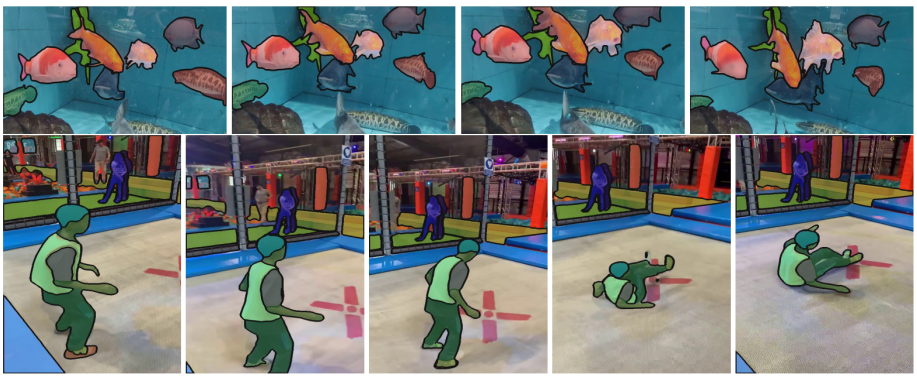


Figure 1.13: SA-V's dataset examples.

Source: SAM 2: Segment Anything in Images and Videos [3].

SAM2 retains the modular and promptable design philosophy of the original SAM, enabling flexible interaction through various input prompts such as points, boxes, and masks. However, it extends this framework to the video domain by incorporating temporal reasoning and memory to support spatio-temporal segmentation. Like SAM, the model decouples visual understanding from prompt interpretation, allowing efficient and interactive inference. It does so through a shared architecture (Figure 1.14) composed of an image encoder, a prompt

encoder, and a mask decoder, but each component is enhanced to handle the temporal dimension and streaming nature of video data.

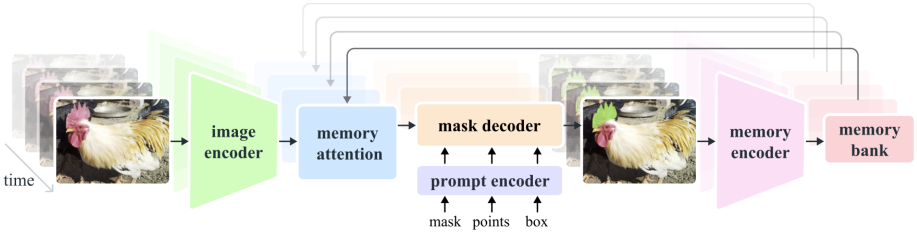


Figure 1.14: SAM2's architecture.

Source: SAM 2: Segment Anything in Images and Videos [3].

The image encoder extracts visual features from video frames in a streaming manner, processing each frame only once and caching its output for efficient reuse. SAM2 uses a pre-trained Hiera model [16]: a hierarchical vision transformer optimized for video that generates multiscale feature embeddings. This model provides unconditioned tokens for each frame.

Memory attention enables the model to incorporate information from past frames and previous interactions when processing the current frame. It consists of a stack of transformer blocks that first perform self-attention on the current frame's features, then cross-attend to a memory bank containing stored object representations and historical context. This allows the model to condition its predictions on both prior observations and new prompts, ensuring temporal consistency and effective propagation of object information across the video.

SAM2 retains the same prompt encoder as the original SAM, allowing it to process diverse inputs such as points, bounding boxes, or masks, enabling flexible and interactive segmentation at any frame.

The mask decoder follows SAM's architecture but introduces key adaptations for video. It predicts multiple masks per frame to account for temporal ambiguity, reflecting different plausible interpretations of the object. When multiple prompts are provided across frames, the model uses them to resolve this ambiguity by aligning the predicted mask sequence with the full set of user inputs, progressively refining its understanding of the target object over time. If no further prompts are given, only the mask with the highest predicted IoU is propagated forward. A new presence head determines whether the object is visible, allowing the model to handle occlusions gracefully. Additionally, skip connections from the hierarchical image encoder deliver high-resolution features directly to the decoder (bypassing the memory attention) to incorporate high-resolution embeddings for mask decoding.

To support temporal reasoning, SAM2 generates a compact memory representation for each frame by downsampling the predicted mask using a convolutional module and summing it element-wise with the image encoder's feature embedding.

These memories are stored in a memory bank, which maintains a fixed-size queue of the most recent N frames and a separate queue for up to M user-prompted frames. This allows the model to retain both continuous temporal context and key interaction points. In addition to spatial memories, the model

stores lightweight object pointers, derived from mask decoder outputs, that carry high-level semantic information about the target object, helping it stay consistent over time.

Temporal position embeddings are added to recent-frame memories to help the model capture short-term motion patterns by understanding the order and timing of consecutive frames. In contrast, prompted frames do not include temporal position information because user interactions can occur at arbitrary, irregular intervals. Encoding fixed time positions for these sparse, unpredictable prompts could harm generalization. By keeping them position-agnostic, the model remains robust and flexible, regardless of when prompts are given during inference.

Overall, SAM2 represents a significant advancement over its predecessor, offering faster, more accurate, and temporally coherent segmentations in video while maintaining strong performance on single images. When applied to images, the model naturally defaults to SAM-like behavior by simply initializing an empty memory, ensuring seamless compatibility across both domains. Due to its faster inference speed, SAM2 was the preferred choice for this project.

Chapter 2

Pipeline Design and Implementation

This chapter outlines the design and implementation of the multimodal pipeline for egocentric dataset creation, detailing how raw sensor data is transformed into structured annotations through a series of coordinated processing stages. This chapter focuses on its architectural components and practical realization.

The chapter begins with an overview of the custom User Interface (UI) tool developed to support pipeline operation, enabling intuitive visualization, parameter configuration, and workflow management. Following this, the full pipeline architecture is presented sequentially, going into detail into each module: audio processing and speech-to-text, frame preprocessing, eye-gaze integration, object detection with bounding box selection, and object segmentation.

The pipeline processes data sequentially, with each stage transforming the output of the previous one, guided by user-configurable parameters to adapt the behavior of individual components, ultimately generating an annotated dataset tailored to the user's needs.

2.1 Overview and Recording Procedure

To create a dataset, the user begins by recording a session using Meta's Project Aria Glasses. Table 2.1 outlines the recommended device configuration to be set through the Aria mobile app. While users are free to adjust camera and eye-tracking resolutions or frame rates, doing so may affect the pipeline's performance and is therefore discouraged unless necessary.

Table 2.1: Recommended device configuration.

Sensor	**Enabled**	**Rate/Resolution**
RGB Camera	Yes	2880 × 2880 @ 15fps
ET Camera	Yes	320 × 240 @ 15fps
IMU 1	Yes	1000 Hz
IMU 2	Yes	800 Hz
Magnetometer	No	Disabled
Barometer	No	Disabled
Audio	Yes	48 kHz, 7 channels
GPS	No	Disabled
BLE	No	Disabled
Wi-Fi	No	Disabled
SLAM Cameras	No	Disabled

During the recording, users are encouraged to perform the in-session eye gaze calibration through the Aria app to improve the accuracy of gaze data. As they move through their environment, users can naturally look at objects they wish to label and verbally state their names (Figure 2.1), either in isolation or within full sentences (e.g., "Here is an apple"). The pipeline does not require rigid, scripted speech. It is designed to handle natural, conversational input, reflecting a real-world scenario where users wear smart glasses in daily life without the explicit intent of data collection. This flexibility highlights the system's potential for passive, large-scale egocentric dataset creation by leveraging everyday interactions.

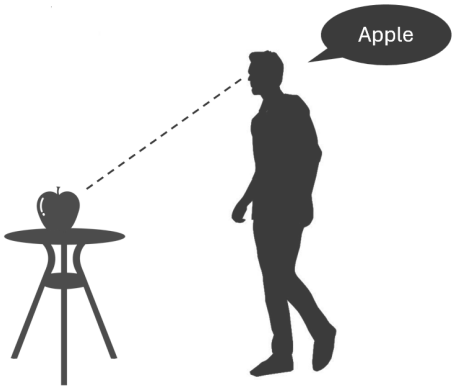


Figure 2.1: Recording procedure illustration.

The pipeline supports recordings in any language, ensuring broad applicability across linguistic contexts.

Due to the device's limited battery life and thermal constraints, long continuous recordings are impractical. To address this, the pipeline's logic and user interface are designed to process multiple recordings sequentially, applying the same set of parameters across sessions and consolidating the results into a unified output. This enables efficient, scalable dataset creation while avoiding the need for long, resource-intensive recording files.

2.2 Dataset Creation UI Tool and Configs

A dedicated user UI tool (Figure 2.2) was developed to facilitate configuration management and simplify interaction with the pipeline. The tool enables users to define and apply a desired set of parameters, such as model settings, dictionary words and processing options, across one or more recording sessions.

A key feature of the UI is the ability to load and save configuration files, allowing users to preserve and reuse parameter setups efficiently. This supports reproducibility and simplifies iterative experimentation or deployment across different datasets and sets of recordings.

The interface also includes a toggle between dark (default) and light mode, improving usability and user comfort during extended interaction.

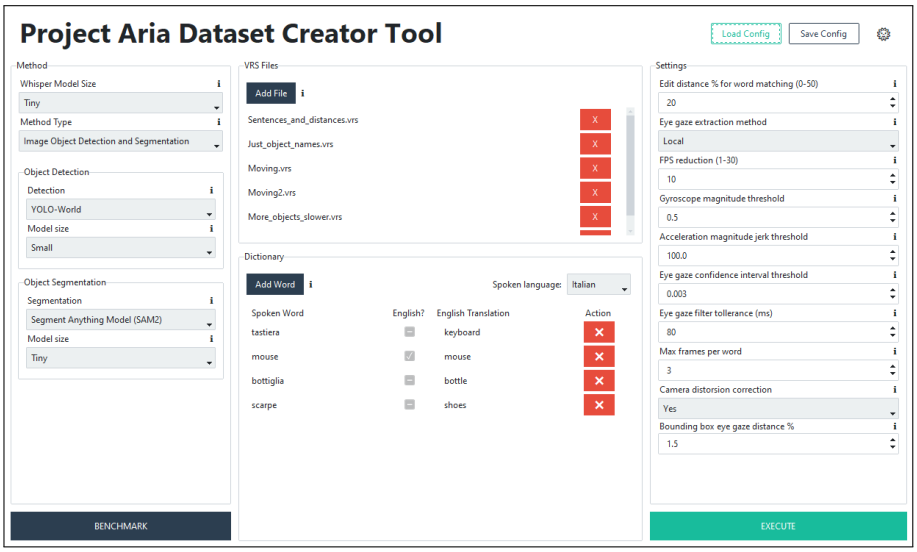


Figure 2.2: Dataset creation UI tool's main page.

In the *Method* panel, the user selects the processing mode and model configurations according to the desired output. The available method types are:

- *Image Object Detection*
- *Image Object Segmentation*
- *Video Object Segmentation*
- *Image Object Detection and Segmentation*

The *Image Object Detection and Segmentation* method is recommended to get the best possible results.

Depending on the selected method, the UI dynamically displays the relevant configurable models. The user can then choose the models and their sizes for each stage of the pipeline:

- WhisperX [4] for speech-to-text transcription,
- YOLO-World [2] for object detection,

- SAM [14] or SAM2 [3] for object segmentation.

This modular design allows users to balance speed, accuracy, and computational demands based on their specific needs. Smaller models process data more quickly but may yield lower accuracy or miss subtle references, while larger models offer a higher likelihood of capturing the intended objects and speech associations, at the cost of increased processing time and resource usage. For this reason, the UI tool is highly configurable, enabling users to experiment with different settings and iteratively refine the pipeline to find the optimal configuration for their use case.

The *VRS Files* panel allows users to add or remove recordings captured with the Project Aria Glasses by selecting the corresponding VRS files. All selected recordings are processed sequentially using the same set of parameters configured in the UI, enabling consistent and batch-wise execution across multiple sessions.

When a recording is selected via the file explorer, it is automatically verified and imported into the system. This process initializes the corresponding data provider, sets the devignetting mask (corrects uneven lighting and enhances image uniformity and clarity) and extracts the recording's audio track.

The *Dictionary* section allows the user to define the set of object categories that the generated dataset should include. If the language spoken by the user during the recordings is English, only the English category names need to be provided, as no translation is required. However, since the system supports the recognition of multiple spoken languages, the tool requires additional information when the input language is not English. In such cases, the user must provide the category names in the original (non-English) language and specify their corresponding English translations. The original language entries are used during speech-to-text transcription with WhisperX, while the English translations are used by YOLO-World for object detection, which operates exclusively in English.

Lastly, the *Settings* panel provides fine-grained control over various parameters that influence how each recording is processed through the pipeline. Some of these settings affect performance, others the final results, and several impact both. While this section presents the interface for configuring these options, their specific roles and technical implications are described in detail in subsequent sections of this document, where each component of the pipeline is discussed in depth.

When the configuration is ready, the user can press the *Execute* button to begin processing the recordings through the pipeline. Alternatively, the *Benchmark* button opens a dedicated UI page that can be used to benchmark the pipeline. The benchmarking framework is also described thoroughly in a subsequent dedicated chapter.

Every configurable option in the UI is accompanied by an information icon (i) that, when hovered, displays a tooltip with a concise explanation of the setting. These descriptions help guide users through the purpose and impact of each parameter, reducing the barrier to effective configuration, especially for those less familiar with the underlying models and processing steps. This built-in documentation enhances usability and supports informed decision-making during pipeline setup.

2.3 Proposed Pipeline

This chapter provides a detailed description of the multimodal pipeline's implementation, outlining the sequence of processing steps executed from the moment the user presses the *Execute* button in the UI to the final creation of the annotated dataset.

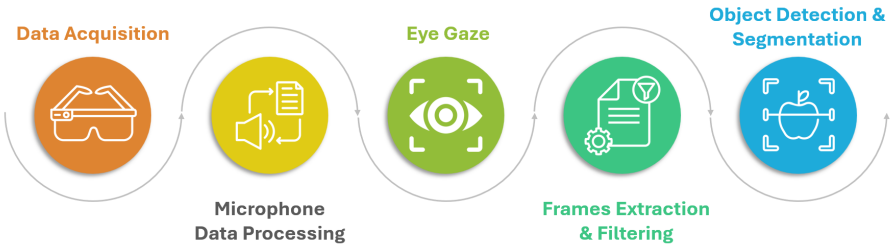


Figure 2.3: Pipeline's overview.

Figure 2.3 illustrates an overview of the pipeline. After a recording has been made with the Project Aria glasses, the data undergoes a series of processing stages. First, the audio stream is analyzed to detect speech segments and identify moments when keyword utterances occur. This is followed by the extraction and synchronization of eye-gaze data to determine where the user was looking during speech events. Subsequently, relevant video frames are extracted and filtered and processed to prepare them for input to the object detection and segmentation models.

In the system, dedicated classes are used to organize and simplify data management throughout the pipeline. Each key component is encapsulated in its own object to ensure clarity, reusability, and efficient data handling.

- *config*: Contains all user-defined settings (chosen in the main UI page) and provides methods to easily save and load them from files.
- *vrs_recording*: Holds the VRS recording data (for a single recording file) and accumulates processed results generated during pipeline execution.
- *benchmark*: Stores benchmarking information (settings and results) when the user chooses to execute a benchmark.

2.3.1 Speech Processing

The speech processing stage is responsible for analyzing the audio track of each recording to identify temporal segments where the user utters one of the keywords defined in the dictionary. To ensure support for multilingual input, the pipeline uses the original language entries from the dictionary during this stage, enabling accurate detection of spoken keywords regardless of the user's language.

The audio track is first extracted from the VRS file and temporarily saved to disk. It is then processed using the WhisperX model, with the model size and input language selected according to the user's configuration. WhisperX performs speech-to-text transcription, segmenting the audio into utterance-level segments and transcribing each word along with its associated confidence score,

start time, and end time. These temporal annotations are crucial for aligning spoken words with corresponding visual and gaze data.

The resulting timestamps are converted from audio time (relative) to the device's time domain (absolute) to ensure synchronization with other sensor streams. The transcription output, including words, timings, and confidence scores, is stored within the internal *recording* object for downstream use.

Next, the transcribed words are matched against the user-defined dictionary in the original language. To filter out low-quality transcriptions, a minimum confidence threshold is applied: only words with a confidence score above 0.15 are considered for further processing. This low threshold is chosen empirically to retain a broad set of potential matches, avoiding premature rejection due to minor transcription errors, while still discarding clearly unreliable outputs.

Because transcriptions may not be perfectly accurate due to variations in pronunciation, background noise, or grammatical form (e.g., "apple" vs. "apples"), exact string matching is insufficient. Instead, the system uses edit distance (Levenshtein distance) to measure the similarity between each transcribed word and the entries in the dictionary. The user can configure an edit distance threshold in the UI settings (Figure 2.2), expressed as a percentage of the transcribed word's length. This relative approach is preferred over a fixed threshold because it scales naturally with word length, reducing false positives on short words while maintaining sensitivity for longer ones.

If a transcribed word falls within the edit distance threshold of any dictionary entry, it is considered a match. These matches are recorded in a mask in the *recording* object, specifying which dictionary word was matched. If at least one valid match is found, the pipeline proceeds to the next stage, otherwise the recording is skipped. This two-stage filtering based on confidence and textual similarity enables robust keyword detection while placing control in the hands of the user, who can adjust the configuration to suit different languages, speaking styles, and acoustic environments, ensuring reliable speech-to-text alignment across a wide range of real-world scenarios.

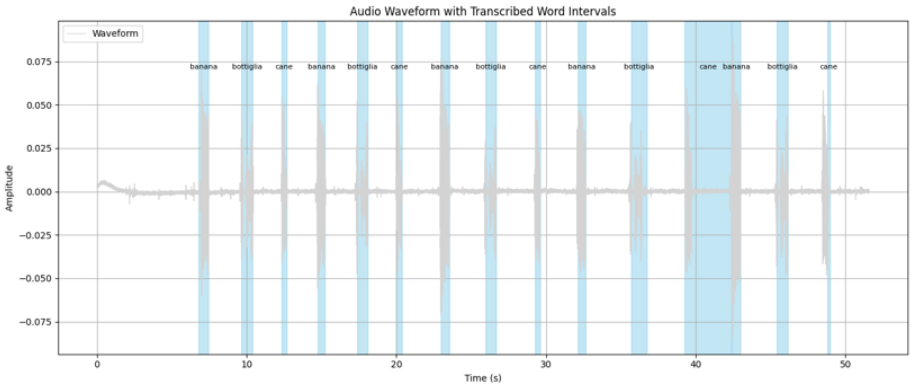


Figure 2.4: Example of the transcribed intervals on top of the audio waveform.

Figure 2.4 illustrates an example of a transcribed audio track, with word-level timestamps and their transcription overlaid on the corresponding audio waveform. The high temporal precision of WhisperX enables accurate alignment of individual words with their acoustic boundaries, a significant improvement

over previous models such as Whisper and FasterWhisper [17], which lack reliable word-level timing information.

In this example, the transcribed intervals closely match the waveform peaks and silences, demonstrating strong alignment. The recording used is part of a controlled benchmark in Italian, where only category names from the dictionary are spoken in isolation, making it easy to verify the correctness of the detected intervals. While the majority of the segments are accurately bounded, a minor error occurs at the 12th word, where the predicted end timestamp slightly deviates from the actual speech segment. While such errors can occasionally occur, they are rare in practice and generally negligible, as downstream stages such as frame selection can compensate by sampling across the interval.

2.3.2 Frame Extraction and Preprocessing

Before frame extraction begins, the system applies a set of precomputed devignetting masks provided by Meta for the Project Aria glasses. These masks are designed to correct the natural falloff in brightness toward the edges of the camera's field of view, a phenomenon known as vignetting, which is caused by lens geometry and optical characteristics. The provided masks correct uneven lighting, enhancing image uniformity and clarity. In this case, the masks are set for the front rgb camera, but they could also be used to correct the grayscale side cameras (which have not been used).

By applying these masks during preprocessing, the pipeline enhances image uniformity and improves visual quality across the entire frame, particularly in the periphery. This correction is critical for downstream tasks such as object detection and segmentation, where consistent illumination helps prevent artifacts and ensures reliable model performance. The devignetting step is applied to every extracted frame.

Figure 2.5 illustrates an example of the devignetting mask applied to a frame captured by the front-facing camera. Note that the frames stored in the VRS file are rotated 90° counterclockwise.

Before devignetting After devignetting

Figure 2.5: Before and after applying the devignetting mask.
Source: Project Aria Docs - Image Devignetting [18].

To extract the required frames, the system compares the original recording frame rate with the user-defined Frames Per Second (FPS) reduction value set via the UI's main page (Figure 2.2) and selects the lower of the two. This effective frame rate determines the temporal density of frames extracted for downstream processing, helping to reduce computational load while maintaining temporal coverage.

For each matched keyword, frames are extracted within the corresponding speech interval at regular intervals based on the reduced FPS value. Extracting multiple frames per interval increases robustness by providing visual diversity, helping mitigate issues such as poor lighting, motion blur, or object occlusion, since some frames may offer better visibility of the target object than others.

The extracted frames, along with their synchronized device-time timestamps, are rotated and stored in the internal *recording* object, ensuring precise alignment with other data for subsequent stages of the pipeline.

Camera Distortion Correction

Camera distortion correction can optionally be applied during frame extraction, depending on the user's configuration. This step is enabled either through a setting in the main UI (Figure 2.2) or automatically when running a benchmark.

Camera distortion correction aims to correct the radial and tangential distortions introduced by the camera lens, commonly perceived as a "fisheye" effect, particularly prominent in wide-angle egocentric cameras like those on the Project Aria Glasses. Whether to correct this distortion or preserve it depends on the intended use case: for dataset creation, retaining the original lens characteristics may be desirable for realism, while correcting it can improve object detection and segmentation performance by normalizing image geometry.

To model the distortion, an external calibration tool is used: *camera-calib* from Mobile Robot Programming Toolkit (MRPT) [19] (Figure 2.6). A custom Python notebook was developed to streamline the process. It requires a physical checkerboard of known dimensions to be printed and placed on a flat surface within the camera's field of view at different orientations and distances. The tool analyzes multiple selected frames, detects the checkerboard pattern, and leverages the known geometry to estimate intrinsic camera parameters such as focal length and distortion coefficients at different regions of the lens.

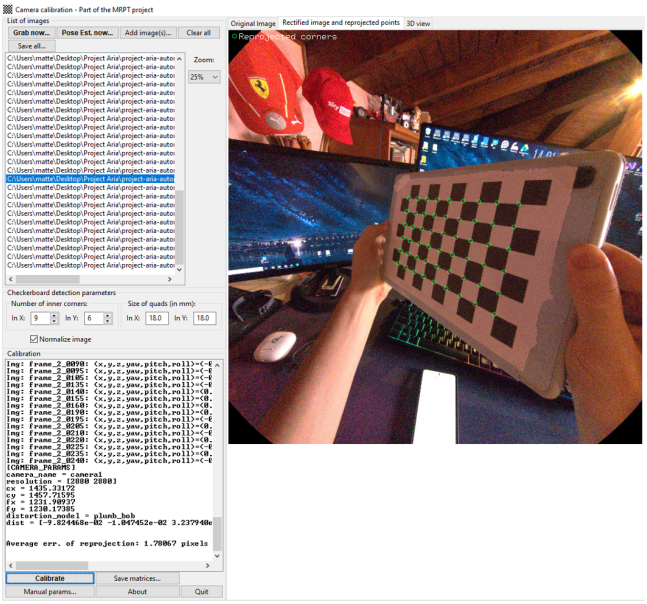
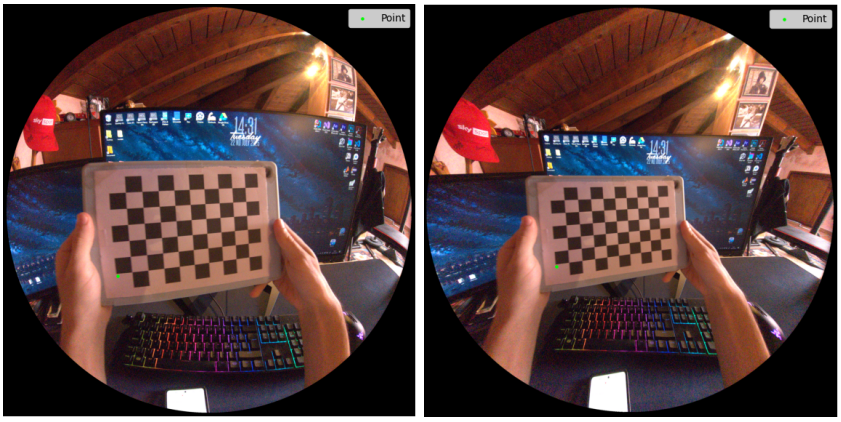


Figure 2.6: Camera-calib tool from the MRPT project.

The output consists of calibration matrices and distortion coefficients, which can then be applied to undistort frames during preprocessing, before storing the frames.

This calibrated correction provides an estimated undistortion that, while not perfect, improves geometric consistency across frames and enhances the quality of object localization and segmentation in subsequent stages of the pipeline.

Figure 2.7 shows an example frame before and after the camera lens distortion correction. The effect of the correction is particularly noticeable along the top edge of the monitor, where the pronounced curvature is visibly reduced.



Before lens distortion correction After lens distortion correction

Figure 2.7: Before and after applying the camera lens distortion correction.

The green point in the images illustrates how the distortion correction can also be applied to pixel coordinates, transforming their position to be aligned with the undistorted image geometry. This capability is essential for accurately reprojecting eye gaze coordinates into the corrected visual space, ensuring spatial consistency in later stages of the pipeline.

2.3.3 Eye Gaze Data Extraction

The Project Aria Glasses are equipped with two grayscale cameras dedicated to eye tracking, each capturing images of one eye (Figure 2.8). These images can be processed using either the Machine Perception Services (MPS) API [6] or locally [9].

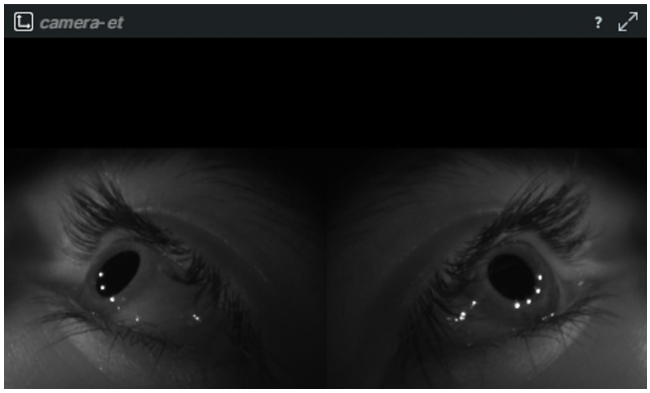


Figure 2.8: Frames captured by the grayscale eye cameras.

The local processing option uses an open model (pre-March 2024) provided by Meta, trained on data from over 1,200 individuals and 2 million frames. This model runs efficiently on consumer-grade hardware, for example, processing at roughly real-time speed on a Google Colab T4 GPU when handling 15 fps recordings made with the maximum resolution (320x240). While slightly outdated, it offers fast and reliable gaze estimation suitable for processing within the pipeline.

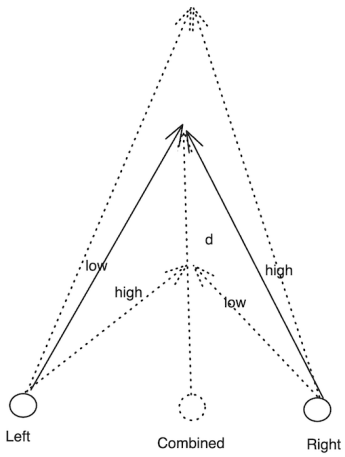


Figure 2.9: Eye gaze depth calculation illustration.
Source: Project Aria Docs - MPS Output - Eye Gaze [20].

In contrast, the MPS API employs a newer, proprietary model that delivers enhanced capabilities, including gaze depth estimation. This is achieved by triangulating the point of intersection between the two gaze rays from the left and right eyes (see Figure 2.9). However, this improved accuracy comes at a significant temporal cost: API-based processing can take hours for a single recording, orders of magnitude slower than the local model, even though it also enables personalized calibration for improved precision (the local model does not).

In this work, the user can choose between the local model and the MPS API directly through the settings panel in the main UI (Figure 2.2). This flexibility allows users to balance processing speed, accuracy, and access to advanced features based on their needs and resources. To maintain compatibility with both options and ensure consistent behavior across configurations, the pipeline does not utilize depth information, even when available from the API.

The model outputs the estimated yaw and pitch angles in radians, along with corresponding lower and upper bounds for each (Figure 2.10). These bounds act as uncertainty intervals: wider intervals indicate lower confidence in the respective angle estimation, typically occurring in situations such as eye occlusion, for example during blinking or when the eyelids are partially closed. This uncertainty information provides valuable context for downstream stages, helping to assess the reliability of the gaze direction at each time step.

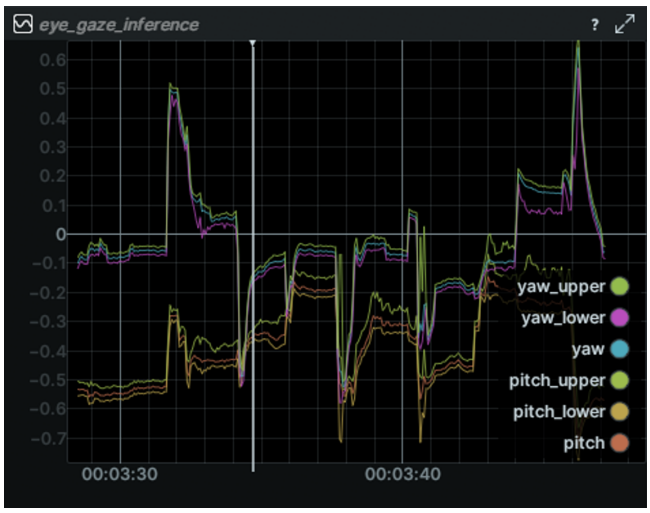


Figure 2.10: Eye gaze model's output example.

After the eye gaze data is retrieved and the corresponding gaze coordinates are projected onto the front-facing camera frame, a coordinate transformation is applied if camera distortion correction was enabled by the user. In such cases, the same calibration matrices used to undistort the image frames are applied to transform the gaze coordinates into the corrected image space. This ensures spatial alignment between the gaze point and the undistorted visual content. The processed gaze data is then stored in the recording object for use in subsequent stages of the pipeline.

Eye Gaze Calibration

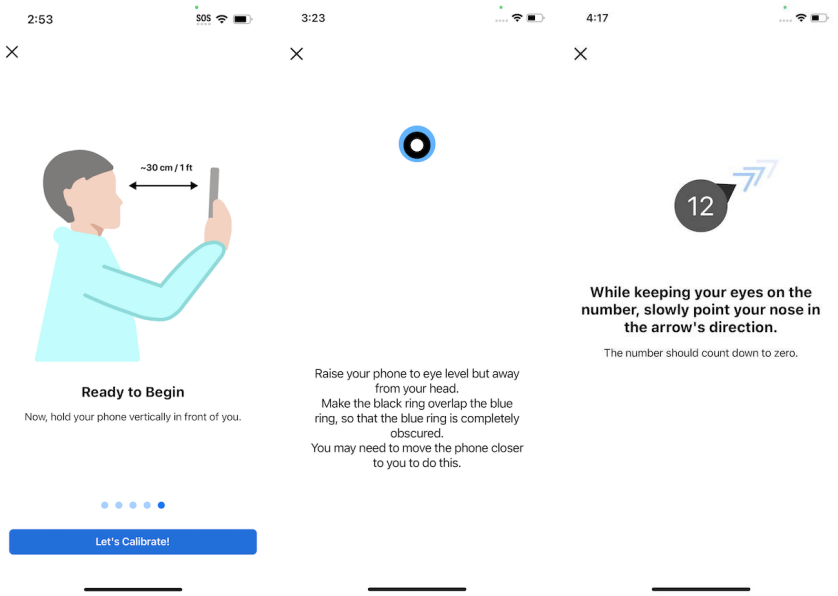
To improve the accuracy of eye gaze estimation, Project Aria supports an In-Session Eye Gaze Calibration procedure [21].

It is important to note that this calibration only affects the output when eye gaze inference is performed using the MPS API. It is not utilized when using the local inference model. Therefore, users seeking higher gaze accuracy must opt for MPS processing to benefit from the calibration.

The calibration process is initiated through the Aria mobile app during an active recording. Proper setup (Figure 2.11(a)) is crucial for successful calibration:

- The phone should be held steady, approximately 30 cm from the face, with elbows slightly bent.
- The device must be oriented vertically and aligned with the user's face, neither tilted nor angled up or down.
- The environment should have even lighting, without bright lights or reflective surfaces in the background.

The app first enters a *Leveler* stage (Figure 2.11(b)), guiding the user to adjust the phone's position until a stability check is passed. Once calibrated, the user must keep the phone in the same position for the remainder of the process.



(a) Initial instructions. (b) *Leveler* stage. (c) Eye calibration stage.

Figure 2.11: Aria app's in-session eye gaze calibration process.

Source: Project Aria Docs - In-Session Eye Gaze Calibration [21].

The actual calibration consists of 10 sequential stages (Figure 2.11(c)). In each, an arrow appears on screen indicating a direction. The user is instructed to move their nose (not just their eyes) toward that direction while keeping their gaze fixed on a central number on screen, which counts down as the movement progresses. This ensures both head motion and coordinated eye movement are captured, improving the personalization of the gaze model.

2.3.4 Frames Filtering and Selection

After frames are extracted at the user-defined reduced frame rate (see Section 2.3.2), a filtering stage is applied to exclude frames likely to be of poor quality due to motion artifacts such as blur, instability, or bad eye gaze data. This step improves the reliability of downstream object detection and segmentation by ensuring that only visually stable frames are processed.

IMU-Based Motion Filtering

The motion filtering leverages IMU data from the Project Aria glasses, specifically the gyroscope and accelerometer streams, to quantify motion dynamics during recording. For each timestamp, the following motion metrics are computed:

- Gyroscope Magnitude (Angular Velocity Magnitude): This measures the rate of rotational movement of the head-mounted device. High values indicate rapid head turns, which often result in motion-blurred images. It is computed as the Euclidean norm of the angular velocity vector

$\vec{w} = (w_x, w_y, w_z)$:

$$\|\vec{w}\| = \sqrt{w_x^2 + w_y^2 + w_z^2}$$

- Acceleration Magnitude: Quantifies the overall intensity of linear forces acting on the head-mounted device at each moment. It is from the acceleration vector $\vec{a} = (a_x, a_y, a_z)$:

  $$\|\vec{a}\| = \sqrt{a_x^2 + a_y^2 + a_z^2}$$

  While sustained high acceleration may not degrade image quality as it often corresponds to smooth, predictable motion, sharp spikes are strongly associated with abrupt head movements that induce motion blur or visual instability in the rendered feed.

- Jerk Magnitude: Jerk is the rate of change of acceleration, providing insight into how abruptly motion changes. It is approximated by computing the finite difference between consecutive acceleration samples:

  $$\vec{j}(t) = \frac{\vec{a}(t + \Delta t) - \vec{a}(t)}{\Delta t}, \quad \|\vec{j}(t)\| = \sqrt{j_x^2 + j_y^2 + j_z^2}$$

  High jerk values often correlate with impulsive movements that degrade image quality.

- Smoothed Jerk Magnitude: To reduce noise and transient spikes in the jerk signal, a symmetric (centered) moving average filter with a window size of 5 is applied:

  $$\|\vec{j}_{\text{smooth}}(t_i)\| = \frac{1}{5} \sum_{k=-2}^{2} \|\vec{j}(t_{i+k})\|$$

  This centered filter uses two past, the current, and two future jerk magnitude values to compute the smoothed value at time t_i. At the boundaries, the window is truncated to avoid out-of-range access. This smoothing improves robustness in thresholding by emphasizing sustained motion changes while reducing sensitivity to momentary fluctuations.

The computed motion metrics are stored in the *recording* object alongside their timestamps, enabling temporal alignment with video frames.

To identify unstable segments, two primary metrics are used for thresholding: the gyroscope magnitude and the smoothed jerk magnitude. The user can set sensitivity thresholds for each through the main UI (Figure 2.2), allowing adaptation to different recording conditions (e.g., walking vs.s. static scenarios).

Any frame whose timestamp falls within a time interval flagged by exceeding either threshold is marked as motion-degraded. These frames are recorded in a filtered-out mask within the recording object, signaling that they should be excluded from further processing. This selective filtering ensures that only visually stable, high-quality frames proceed to the subsequent selection of frames that will be used in object detection and segmentation.

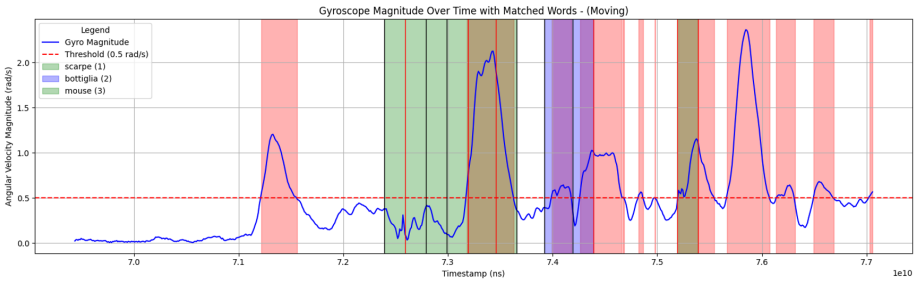


Figure 2.12: Gyroscope magnitude over time with matched speech intervals and filtered frames.

Figure 2.12 illustrates an example of gyroscope-based motion filtering. The blue solid line represents the magnitude of the angular velocity over time, while the horizontal red dotted line indicates the user-defined threshold (e.g., 0.5 rad/s). When the gyro magnitude exceeds this threshold, a red-shaded interval is displayed, marking the time periods during which frames have been flagged for filtering due to excessive motion.

Detected spoken word matches are overlaid as horizontal colored bands, alternating in green and blue, each representing the temporal extent of a matched word (based on the included frames). Within these word intervals, individual frames are visualized as vertical lines: red if the frame was filtered out (i.e. motion exceeded the threshold), and black if it was retained. In this example, three word matches are shown, demonstrating how speech events can coincide with periods of high head motion.

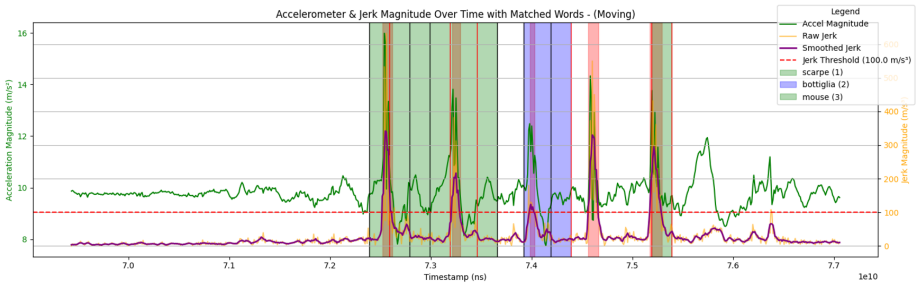


Figure 2.13: Acceleration and smoothed jerk magnitudes over time with matched speech intervals and filtered frames.

On the other hand, Figure 2.13 displays the acceleration magnitude (green), raw jerk magnitude (yellow), and smoothed jerk magnitude (purple) over time. While the smoothed jerk is the primary metric used for acceleration-based motion filtering, the other signals are included for comparison and to illustrate why smoothing is essential. The raw jerk exhibits high-frequency noise and transient spikes, making it less reliable for thresholding. In contrast, the smoothed jerk provides a more robust representation of sustained motion changes.

The horizontal red dotted line represents the user-defined jerk threshold (100.0 m/s^3). Whenever the smoothed jerk exceeds this threshold, a red-shaded interval is displayed, indicating that frames within this period are filtered out,

consistent with the motion filtering strategy applied on the gyroscopic data. As in the previous plot, matched spoken words are shown as alternating green and blue colored bands, with vertical lines marking individual frames: red if filtered, black if retained. This visualization demonstrates how abrupt motion events correlate with speech and how the smoothed jerk effectively captures meaningful motion dynamics while suppressing noise.

In this example, the first matched word interval contains 7 frames, of which the second, fifth, and sixth were filtered out due to exceeding the a threshold. The second word spans 3 frames, with the third frame being excluded based on motion criteria. The final word consists of only 2 frames, both of which were filtered out.

Note that the red vertical lines marking excluded frames are shared across all filtering metrics. Therefore, a frame may appear in red even if the current metric (e.g., smoothened jerk or acceleration) does not exceed its threshold. This occurs when the frame was removed by another filter.

Eye Gaze Filtering

To further improve the reliability of gaze data, a filtering step is applied to exclude frames where eye gaze estimates are likely inaccurate, such as during blinks, partial occlusions, or poor lighting conditions that can impair eye tracking accuracy.

The eye gaze model provides, for each timestamp, estimated yaw and pitch angles (in radians) along with their respective upper and lower confidence bounds. The width of these intervals reflects the model's uncertainty: a larger interval indicates lower confidence in the estimate.

These widths are analyzed independently.

Specifically, the confidence interval width for yaw and pitch is computed as:

$$\Delta_{\text{yaw}} = \text{yaw}_{\text{upper}} - \text{yaw}_{\text{lower}}, \quad \Delta_{\text{pitch}} = \text{pitch}_{\text{upper}} - \text{pitch}_{\text{lower}}$$

The user can set a threshold on the maximum allowed interval width through the main UI (Figure 2.2), allowing control over the strictness of the filtering process. Any time interval where either Δ_{yaw} or Δ_{pitch} exceeds the threshold is flagged as low confidence.

Additionally, a configurable temporal tolerance window (in milliseconds) extends the exclusion zone around each flagged interval. This ensures that not only frames with poor confidence, but also those immediately before and after (which might still be affected, since the eye gaze confidence can only be calculated at discrete times, based on the eye camera FPS), are excluded.

Frames whose timestamps fall within any of these filtered intervals are marked in a mask stored in the *recording* object. This mask is later used to discard frames with unreliable gaze data during the frame selection, ensuring that only higher quality frames can proceed to the object detection and segmentation stages.

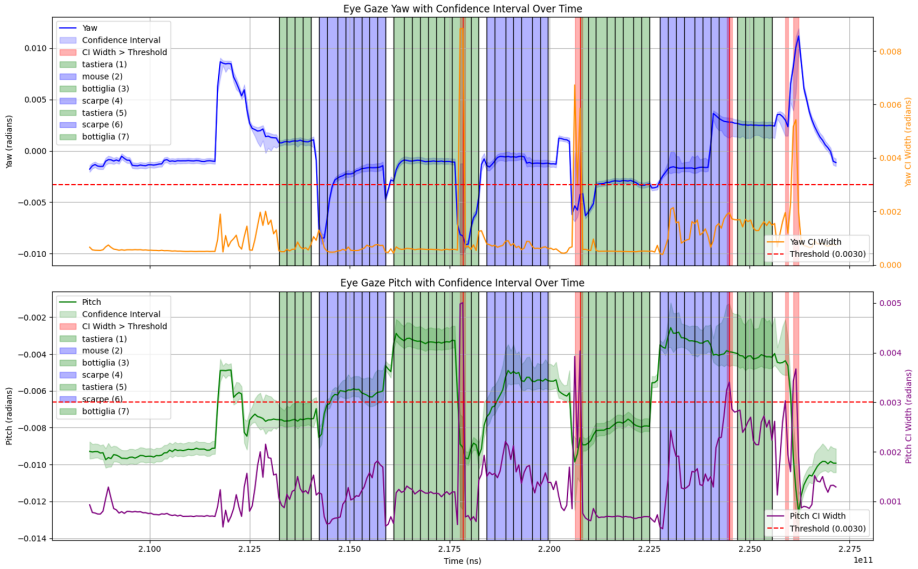


Figure 2.14: Eye gaze yaw and pitch over time with matched speech intervals and filtered frames.

Figure 2.14 illustrates the eye gaze filtering results applied to a different recording, with yaw and pitch confidence intervals independently analyzed and shown in two separate plots. The blue and green lines represent the estimated yaw and pitch angles over time, with their associated upper and lower confidence bounds shaded to indicate the uncertainty region.

The yellow and purple lines depict the width of the confidence intervals (the difference between upper and lower bounds) for yaw and pitch, respectively. When these widths exceed the user-defined threshold (shown as red dotted lines), the corresponding time intervals are flagged as low-confidence.

Matched speech words are colored alternately in green and blue for visual distinction. Their frames are represented as vertical lines, which are rendered in black if the corresponding frame passes the filter, and in red if it is excluded due to falling within a low-confidence interval.

Filtered intervals are overlaid in red across both plots to indicate time segments excluded from further processing. Note that these intervals appear globally, even on the opposite plot, meaning an interval flagged due to pitch uncertainty will still be shown in the yaw plot.

Frames Selection

After filtering out low-quality frames using IMU and eye gaze data (as described in Sections 2.3.4 and 2.3.4), the pipeline proceeds to the frame selection stage. The goal is to choose a subset of frames for each matched keyword to be processed by the object detection and segmentation models.

While only a single frame per matched word is strictly necessary to generate one annotation, selecting multiple candidate frames improves robustness. Processing several temporally close, high-quality frames increases the likelihood of

successful detection and segmentation, especially in cases where occlusion, poor lighting, or model failure might affect individual frames.

This redundancy allows the pipeline to later evaluate multiple results and select the most confident or visually consistent output, enhancing overall accuracy and reliability. This selection strategy ensures that even if detection fails on one frame, alternative candidates can provide a valid annotation.

The user can specify a maximum number of frames to select per matched word via the main UI's settings (Figure 2.2). This is a maximum limit. Actual selection may yield fewer frames, especially for short speech intervals (e.g., a brief utterance like "car") that naturally contain fewer frames.

From within the time interval of each matched word, the pipeline selects the best available frames (i.e., those that passed both IMU and eye gaze filtering) ensuring only high-quality, stable frames are considered. The selected frames are chosen to be as evenly spaced as possible across the original temporal sequence, taking into account the full frame timeline (including filtered-out frames) to maintain consistent coverage.

The spacing strategy is dynamic: it adapts based on the duration of the speech interval, the total number of original frames, the number of surviving (non-filtered) frames, and the user-defined maximum. This ensures robust temporal distribution while maximizing visual diversity, improving the chances of capturing a clear view of the target object under favorable conditions.

Choosing a lower user-defined maximum number of frames per matched word can be sufficient, especially under stable recording conditions, and offers the advantage of faster pipeline execution due to reduced computational load. However, it increases the risk of missing valid annotations.

Conversely, a higher maximum improves robustness by providing more candidate frames for each keyword, increasing the likelihood of capturing a clear, well-aligned view of the target object. This is especially beneficial in uncontrolled or dynamic environments where lighting, occlusion, or motion may affect frame quality. While this comes at the cost of longer processing times, the trade-off favors reliability and completeness, making it the preferred choice for real-world, egocentric recordings with varying conditions.

Another mask is generated to indicate which frames have passed all filtering and selection criteria. This final selection mask is stored in the recording object, preserving the identity of the chosen frames. These frames and their data will then be used in the subsequent stages of the pipeline for object detection and segmentation, ensuring that only the most relevant and reliable samples are processed by the vision models.

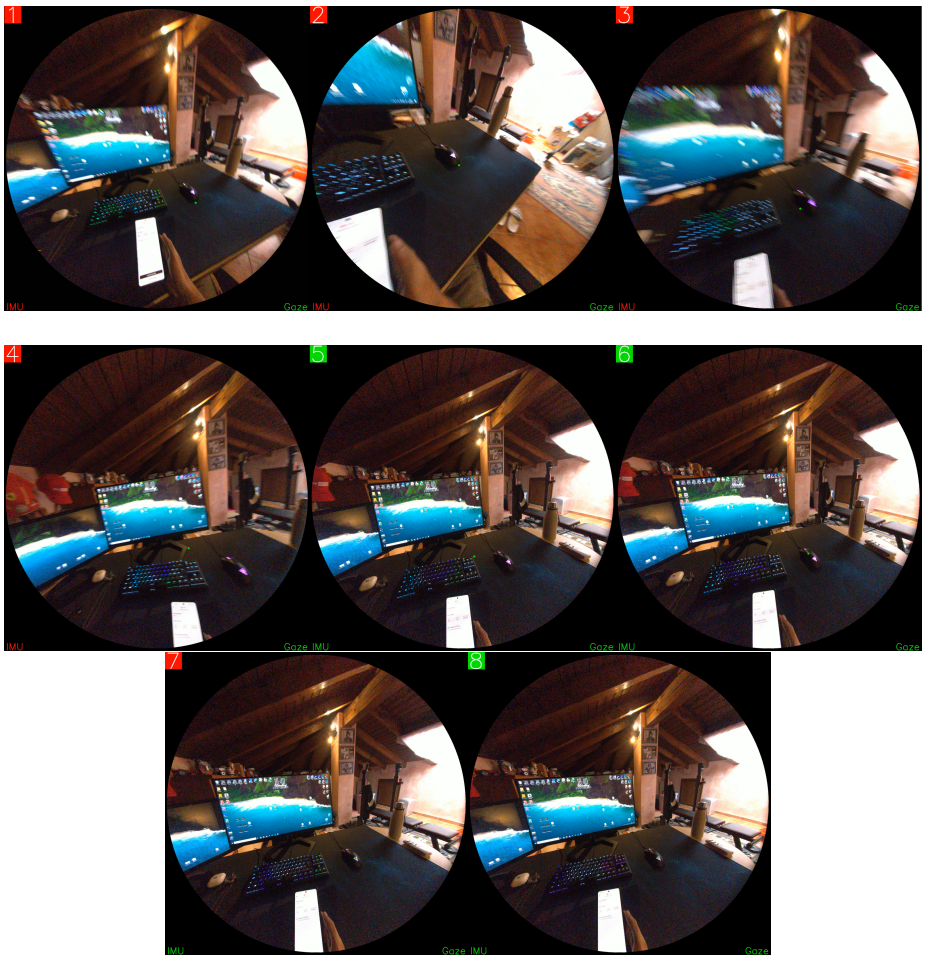


Figure 2.15: Motion frames filtering and selection examples.

Among the various visualization options available in the UI after pipeline execution, the user can generate a diagnostic *.mp4* video that visualizes the filtering and selection process for each frame. Figure 2.15 shows a composite frame from this video, where each frame is annotated with metadata to aid in debugging and analysis.

In the top-left corner, the frame number is displayed on a color-coded background: green if the frame was selected for processing, red if it was excluded during the final selection stage or during filtering. In the bottom corners, labels indicate the outcome of the two main filtering stages:

- *IMU* (bottom-left): colored green if the frame passed motion-based filtering, red if excluded due to excessive movement.
- *Gaze* (bottom-right): green if the eye gaze data was sufficiently confident, red if filtered out due to low gaze reliability.

This visualization helps users understand why certain frames were discarded and assess the effectiveness of the filtering pipeline. For instance, in Figure 2.15,

frames 1 to 4 are marked as excluded by the IMU filter, consistent with visible motion blur (though subtle in the reduced resolution images). Meanwhile, frames 5, 6 and 8 were selected, respecting the user-defined maximum of three frames per keyword. Their distribution shows they were spaced as evenly as possible across the interval, demonstrating the pipeline's strategy for maximizing temporal diversity while maintaining quality.

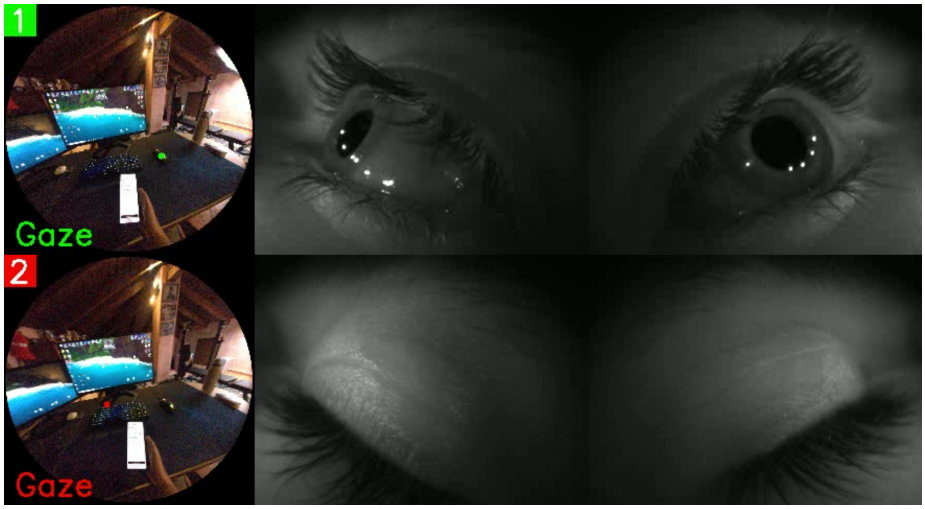


Figure 2.16: Eye gaze frames filtering examples.

The additional visualization in Figure 2.16 provides a focused view of the eye gaze filtering process, synchronized with the corresponding eye camera images. It helps verify the rationale behind filtering decisions, for example confirming that a frame was excluded because the user's eyes were closed or partially occluded, making the gaze estimate unreliable.

In the example shown, the second frame is filtered out by the eye gaze filter. The corresponding eye camera image clearly shows the eyelid closed, which explains the high uncertainty in the gaze estimation. The first frame's gaze is in fact correctly estimated on the *mouse* object, but the second frame's gaze is inaccurate because of the closed eyelid, so the frame is correctly filtered out. This correlation between visual evidence and filtering logic validates the effectiveness of the gaze quality check and demonstrates how the pipeline automatically discards frames where accurate gaze tracking is not possible.

2.3.5 Object Detection

Object detection involves identifying and localizing objects within an image by labeling them with bounding boxes. A bounding box is defined by its coordinates (e.g., top-left and bottom-right corners) and is used to precisely indicate the position of an object in the image. For example, in Figure 1, various bathroom objects such as the bathtub, mirror, and faucet are detected and labeled using colored bounding boxes, demonstrating how YOLO-World enables accurate localization and classification of items based on user-defined categories.

Figure 2.17: Example of the bounding boxes resulting from the object detection.
Source: YOLO-World: Real-Time Open-Vocabulary Object Detection [2].

This stage of the pipeline is executed when the user selects a processing method that includes object detection. The system currently employs YOLO-World as the sole detection model, leveraging its ability to perform open-vocabulary object detection based on user-defined category names. While no alternative models are integrated, the user can choose between two pre-trained variants to balance performance and accuracy:

- *Small*: *yolov8s-worldv2*, offering faster inference with lower computational demand.
- *Large*: *yolov8l-worldv2*, providing higher detection accuracy at the cost of increased processing time and resource usage.

This selection is made through the main UI's page (Figure 2.2) and allows users to tailor the pipeline to their specific requirements, whether they prioritize speed for large-scale processing or accuracy for detailed annotations.

For each matched word, the corresponding vocabulary entry is retrieved and used as the sole class prompt for the YOLO-World [2] model. This enables the model to perform conditional object detection, searching specifically for instances of the user-defined category within the associated time interval.

Each frame selected during the previous frame selection stage is then processed independently through the model. For every inference, YOLO-World returns a set of predicted bounding boxes, associated class labels, and confidence scores. Since the model is prompted with only the target class, the returned class labels are redundant; the focus instead lies on the bounding box coordinates and their confidence values.

All detection results are stored in the *recording* object for further processing. This structured output ensures full traceability and enables multimodal refinement.

Bounding Box Refinement

After object detection, the pipeline performs a refinement step to improve the quality of the detected bounding boxes. In some cases, multiple bounding boxes may be detected for the same object within a single frame because the model responds to different parts of an object. To address this, the pipeline analyzes the spatial relationships between detected boxes and merges them when appropriate.

Two main criteria are used to determine whether two bounding boxes should be unified: their degree of overlap and the presence of the user's eye gaze within the region. If two boxes significantly overlap, they are likely detecting the same object and are merged into a single, encompassing box. Furthermore, if the user's eye gaze falls within one of the boxes, this provides strong contextual evidence that the object of interest is being looked at, increasing confidence in the relevance of that detection.

For any pair of bounding boxes and a given eye gaze point, the boxes are unified into a single encompassing box if either of the following criteria is met:

1. Gaze Point Contained in Both Boxes:
   If the user's eye gaze point (x, y) lies inside both bounding boxes, they are merged. This condition indicates high confidence that both detections are aligned with the user's visual focus and likely refer to the same object.

2. Significant Overlap Relative to Larger Box:
   If the intersection area between the two boxes exceeds one-third of the area of the larger box, the boxes are merged:

$$\frac{\text{intersection}}{A_{\max}} > \frac{1}{3}$$

   This ensures only meaningfully overlapping detections are combined.

When either condition is met, the resulting unified bounding box is defined as the minimum enclosing rectangle:

$$\left[\min(x_1^{(1)}, x_1^{(2)}),\ \min(y_1^{(1)}, y_1^{(2)}),\ \max(x_2^{(1)}, x_2^{(2)}),\ \max(y_2^{(1)}, y_2^{(2)})\right]$$

This union captures the full spatial extent of both original detections. The confidence score of the new bounding box is computed as the average of the individual scores.

The merging process is applied iteratively: after a pair is merged, the updated list is re-evaluated to detect new potential merges. The loop continues until no further valid merges can be made, ensuring convergence to a stable, non-redundant set. This strategy balances attentional context (via gaze) and geometric consistency (via overlap), producing cleaner and semantically more accurate detections for downstream segmentation.

Best Bounding Box Selection

The final stage of the object detection stage selects the single most relevant bounding box for each matched keyword, chosen from all detections across the associated time interval. This selection is designed to identify the frame and detection that best represent the user's intended object, combining detection confidence with spatial alignment to the user's visual attention.

For each matched word, the system first identifies all bounding boxes from the selected (non-filtered) frames that meet at least one spatial criteria relative to the user's eye gaze:

- The eye gaze point lies inside the bounding box.
- The eye gaze point is within a distance threshold (configurable through the settings panel in the main UI's page) from the nearest edge of the box, measured as a percentage of the image diagonal. This allows for small misalignments between gaze and detection due to poor accuracy (possibly because of a missing in-session calibration).

All such qualifying boxes are collected as candidates for final selection. If no candidates are found, no bounding box is selected for that word, so there will be a missing annotation.

The choice of the best candidate depends on the pipeline's intended use:

- If the goal is object detection only (no object segmentation), the system selects the candidate with the highest confidence score, prioritizing model certainty regardless of gaze position.
- If the user also required object segmentation, a stricter policy is applied: the system first considers only candidates where the eye gaze point is inside the bounding box and with a confidence score of at least 0.5. If such candidates exist, the one with the highest confidence is chosen. If none meet these criteria, the system falls back to selecting the highest-confidence candidate among all qualifying boxes.

This dual strategy ensures that when segmentation is requested, the selected bounding box is not only confident but also spatially aligned with the user's gaze, increasing the likelihood that the correct object instance will be correctly segmented. The result is a single, high-quality bounding box per matched word, which is then marked in the *recording* object to be possibly used in the segmentation stage or exported in the created dataset.

Figure 2.18: Example of the bounding box selection after object detection.

Figure 2.18 illustrates an example of the bounding box selection process for the keyword "mouse". Three frames from the same speech interval were processed using the YOLO-World *small* model. While the second frame yielded no detection, the first and third frames each produced a valid bounding box.

In the visualization, bounding boxes are color-coded: green if the eye gaze point lies either inside the box or within the user-defined distance threshold (1.5% of the image diagonal in this case), and red otherwise. The final selected bounding box is highlighted with a thicker dotted green outline. Above each box, metadata is displayed, including the detection confidence score, whether the gaze was considered inside (or within threshold), and whether the box was ultimately selected. The eye gaze point is marked as a red dot on each image.

Notably, the bounding box in the first frame, despite having a slightly higher confidence score, was not selected because the user's gaze falls on the keyboard, outside the region of the detected mouse. In contrast, the third frame exhibits both a strong confidence score and precise gaze alignment, with the eye gaze point clearly within the bounding box. This strong multimodal agreement (high confidence and gaze concordance) led to its selection as the best representative instance. This example demonstrates how the pipeline leverages both visual and attentional cues to make semantically grounded decisions.

2.3.6 Object Segmentation

The object segmentation stage aims to generate pixel-accurate annotations of the target objects. This fine-grained labeling provides a significantly more precise representation than bounding boxes, enabling applications such as detailed scene understanding, instance-level analysis, and high-fidelity dataset creation. However, segmentation is a more complex and computationally intensive task compared to object detection.

To achieve this, the pipeline supports two state-of-the-art segmentation models: Segment Anything Model (SAM) and the newer Segment Anything Model 2 (SAM2). While both models are capable of zero-shot instance segmentation given a prompt (such as a bounding box), SAM2 is recommended for use in this pipeline. It offers improved accuracy and faster inference across all model variants.

For SAM, only the base variant (*sam_vit_b_01ec64*) is integrated. In contrast, SAM2 offers three selectable variants:

- *Tiny* (*sam2.1_hiera_tiny*): fastest inference, suitable for real-time or resource-constrained scenarios.
- *Base Plus* (*sam2.1_hiera_base_plus*): balanced performance and quality.
- *Large* (*sam2.1_hiera_large*): highest segmentation quality, at the cost of increased computational demand.

The choice of model allows users to balance processing speed and output fidelity based on their hardware and application requirements.

Segmentation is executed only when the user selects a processing method that includes segmentation, either *Image Object Segmentation* or *Image Object Detection and Segmentation*, ensuring that the computationally intensive segmentation stage is applied only when needed.

Standalone Gaze-Driven Object Segmentation

When object segmentation is performed independently of object detection the pipeline relies solely on the user's eye gaze coordinates as a point prompt for the

segmentation model. Each prompt is associated with a label indicating whether it is an inclusion (1) or exclusion (0) cue. However, since the pipeline operates autonomously and does not involve interactive refinement, only inclusion prompts are used, with a single eye gaze point serving as input.

This approach introduces two significant challenges:

- First, the accuracy of the segmentation is highly dependent on the precision of the eye gaze data. As shown in Figure 2.19, if the gaze point is misaligned due to imperfect eye tracking, lack of in-session calibration, natural gaze variability, or user behavior, the model may segment an incorrect object.

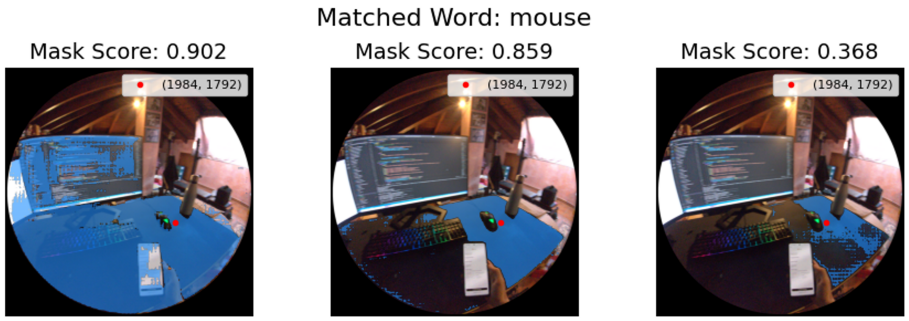


Figure 2.19: Example of segmentation results when the eye gaze point is not accurate (using *SAM2 tiny*).

- Second, even when the gaze point is accurate, the ambiguity inherent in a single-point prompt leads the model (SAM or SAM2) to generate multiple plausible masks (Figure 2.20), typically three per prompt representing different interpretations: the full object (whole), a constituent part (part), or a smaller subregion (subpart). This behavior is by design in SAM, as it aims to provide diverse segmentation hypotheses in interactive scenarios where the user can select the desired one. However, in an autonomous pipeline, this introduces a critical ambiguity: there is no human in the loop to choose the correct mask.

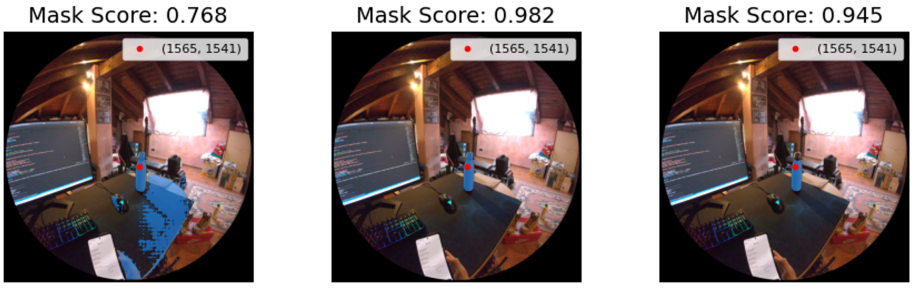


Figure 2.20: Example of the three masks returned by the segmentation process (whole, part, subpart) using *SAM2 tiny*.

In this case, the pipeline automatically selects the mask with the highest model-reported confidence score. While this heuristic works reasonably well

in some cases, it is not reliable. The most confident mask is not always the semantically correct or most complete one. For instance, the model might assign high confidence to a small, clearly defined part (e.g., a laptop's trackpad) while the user intended the entire device to be segmented.

Object Segmentation after Object Detection

It is strongly recommended to use the *Object Detection and Segmentation* method when the goal is high-quality object segmentation. This approach significantly improves segmentation accuracy by leveraging the bounding box from the detection stage as a spatial prompt for the segmentation model.

In this setup, the selected bounding box is provided as input to the segmentation model (SAM or SAM2), which uses it to better localize the target object. Additionally, the eye gaze coordinate is included as a secondary prompt, but only if it lies within the bounding box. This conditional inclusion ensures that gaze data is used to refine the segmentation only when it is spatially consistent with the detected object, avoiding misleading prompts when the gaze falls clearly outside the region of interest.

While this does not guarantee that the gaze point corresponds exactly to the intended object (e.g., in cases of occlusion or low gaze accuracy), it significantly reduces the risk of erroneous prompts and improves contextual alignment.

Crucially, the bounding box prompt helps resolve the ambiguity inherent in point-only segmentation. Although the model still generates three candidate masks, representing different interpretations such as whole, part, and subpart, the bounding box provides strong contextual constraints that guide the model toward more semantically meaningful outputs. As in the gaze-only case, the final mask is selected based on the highest confidence score reported by the model.

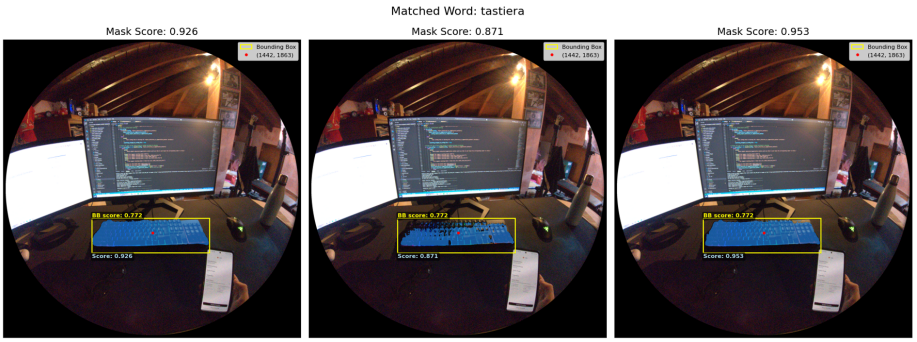


Figure 2.21: Example of the segmentation after detection results using *SAM2 tiny*.

Figure 2.21 illustrates the results: it shows the best bounding box selected during detection in yellow (with its confidence score overlaid), and the three generated segmentation masks in blue, each annotated with its corresponding confidence. The model used was *SAM2 Tiny*, showing that even the smallest variant can produce high-quality segmentations when guided by accurate prompts.

The generated segmentation masks are stored within the *recording* object,

preserving all the output masks for each frame as produced by SAM or SAM2. Alongside these, a selection mask is stored to indicate which of the three masks was chosen as the best candidate. This structured output ensures both flexibility and traceability, enabling the future dataset export.

Video Object Segmentation

Video object segmentation is supported exclusively by SAM2, thanks to its temporal propagation architecture, which enables consistent mask tracking across frames. This capability was explored early in the pipeline's development as a potential solution to the ambiguity issues inherent in single-frame, gaze-only segmentation, before the more effective detection-guided segmentation approach was implemented.

In this mode, the pipeline uses only the user's eye gaze coordinates as sparse prompts, applied at configurable intervals throughout the video segment. Through the main UI settings panel, the user can define the prompt interval (i.e., how frequently a new eye gaze point is provided to the model):

- A value of *0* means the prompt is given only on the first frame (similar to standard zero-shot Video Object Segmentation (VOS)).
- A value of *-1* applies the prompt to every frame.
- Any positive integer n applies the prompt every n frames.

This design allows control over how much guidance to give the model.

SAM2 outputs a set of masks per frame, each associated with a unique object IDentifier (ID), enabling multi-object tracking. However, in this use case, only one object is being referenced per keyword interval, so the pipeline selects and outputs a single mask per frame, corresponding to the tracked instance initiated by the first valid prompt.

Despite its advanced architecture, this method presents several limitations in the context of our pipeline:

1. Gaze-driven ambiguity persists: since prompts rely solely on eye gaze, the same accuracy issues remain: if the gaze point is misaligned (due to calibration quality, eye occlusion, or user behavior), the model may initialize on the wrong object, leading to incorrect tracking throughout the sequence. Figure 2.22 shows an example where a single key was wrongly segmented and tracked across frames instead of the whole keyboard.

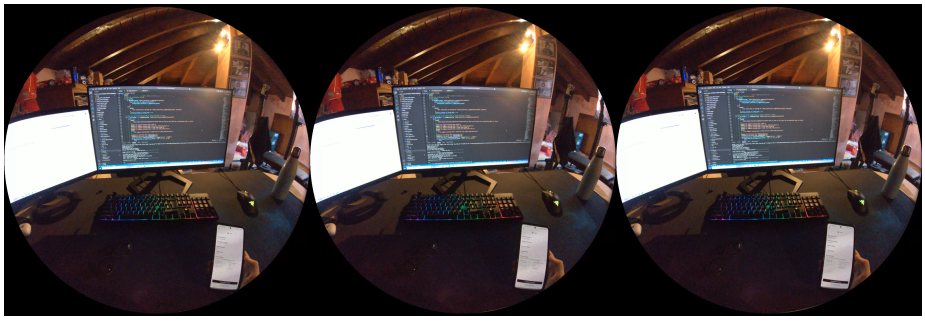


Figure 2.22: Example of the ambiguity issue with video segmentation.

2. No meaningful improvement over frame-by-frame processing: experimentation with different prompt intervals showed little to no benefit in mask consistency or accuracy. Providing prompts at every frame yields results nearly identical to processing each frame independently, suggesting that temporal coherence is not effectively leveraged in this setup.

3. Prompt overwriting behavior: when prompts are provided at intervals, the model often appears to reset or overwrite the previous state rather than refine it, undermining the intended benefit of sparse interactive prompting. So if any of the gaze points is inaccurate, the following portion of frames will be compromised, until a correct prompt is given. The use of multiple prompts does not seem to enhance performance by propagating a meaningful amount of information along the model. Figure 2.23 illustrates this issue, where every time a new pair of gaze coordinates is given, the segmentation seem to switch to tracking a different object (probably due to eye gaze inaccuracy, but this also shows how no meaningful information seems to be retained from previous prompts). Note that in the figure not all the frames are shown, but just the ones where a differnet object was tracked when a new prompt was given.

Figure 2.23: Example of the prompt overwriting behavior in video object segmentation. (Note: some frames are skipped.)

4. High computational cost with no gain: SAM2's video object segmentation mode is significantly slower than single-image segmentation. Processing times increase from seconds to minutes, even for short clips. Given the lack of improvement in output quality, this cost is unjustified for the pipeline's purpose.

While future work could explore using detection-generated bounding boxes as initialization or periodic prompts for video object segmentation, potentially improving robustness, such an approach would require a fundamentally different, video-centric pipeline design. In the current context, where the goal is to generate image-based annotations for egocentric datasets, the high computational overhead alone makes video segmentation impractical.

As a result, the video object segmentation mode was ultimately deprioritized in favor of the more efficient and accurate detection-then-segmentation workflow, which better aligns with the pipeline's objectives of scalability, precision, and real-world usability.

2.3.7 Results Interface and Dataset Export

After pressing the Execute button in the main UI (Figure 2.2), a new window opens to monitor the pipeline's execution (Figure 2.24). This results interface is divided into two main sections: real-time execution feedback on the right, and interactive debugging tools on the left.

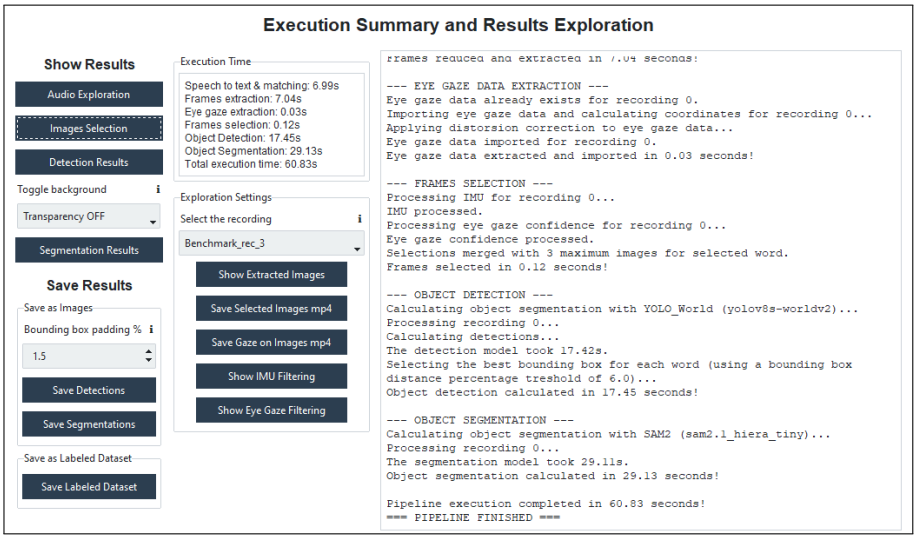


Figure 2.24: UI execution window after the pipeline has been processed.

The right panel displays live debug information, including the current processing stage, status messages, and overall progress. A dedicated Execution Time box logs the duration of key pipeline components such as eye gaze extraction, object detection, and object segmentation, providing insight into performance bottlenecks and computational load.

On the left side, a set of interactive buttons enables users to visualize intermediate results for debugging and analysis. For example, pressing the *Images Selection* button reveals a specific suite of tools under Exploration Settings (shown in Figure 2.24). A dropdown menu at the top lets the user select a specific recording when multiple recording sessions were processed, ensuring targeted review of individual results.

The *Show Extracted Images* button displays the frames extracted for each matched keyword, organized separately by speech interval (Figure 2.25). It also shows the timestamp of each frame, for debug purposes. This visualization allows users to inspect the temporal distribution of selected frames and verify their relevance to the corresponding utterance.

Figure 2.25: Plot showing the extracted images for a matched word.

The *Save Selected Images as mp4* button, previously illustrated in Figure 2.15, generates a diagnostic video showing the entire filtering and selection process over time. Similarly, the *Save Gaze on Images* button, illustrated in Figure 2.16, generates a video that overlays eye gaze coordinates onto each frame, displays the corresponding eye camera views at that moment, and indicates whether filtering was applied.

The Show IMU Filtering (Figure 2.12 and Figure 2.13) and Show Eye Gaze Filtering (Figure 2.14) buttons provide visual summaries of the frame filtering stages, highlighting time intervals excluded due to excessive motion or low gaze confidence, respectively.

If the *Audio Exploration* button is pressed, the *Exploration Settings* panel is populated with audio-specific tools, as shown in Figure 2.26.

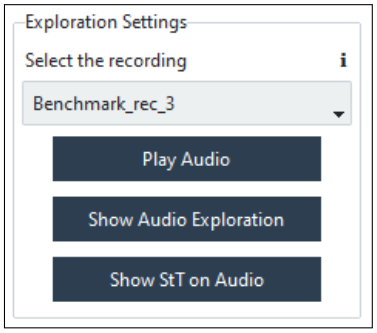


Figure 2.26: Exploration settings panel with audio buttons.

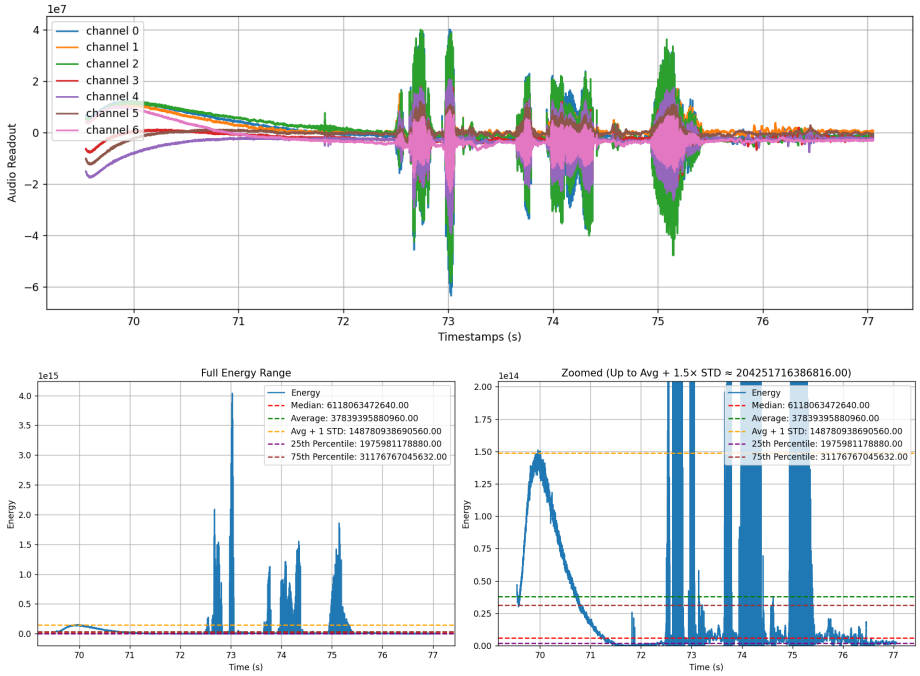


Figure 2.27: Audio waveform and energy plots example.

The *Play Audio* button allows the user to instantly listen to the extracted audio track, facilitating quick verification of recording quality and speech content. The *Show Audio Exploration* button displays the audio waveform and its energy envelope over time (Figure 2.27), helping to visually identify speech segments, pauses, and potential noise artifacts.

The *Show StT on Audio* button overlays the speech-to-text transcription onto the waveform, marking the start and end times of each transcribed word along with the matched dictionary entries, exactly as previously illustrated in Figure 2.4.

The *Detection Results* button becomes active only if the user has requested object detection. For each selected frame, it displays the detected bounding box (if any), along with the corresponding eye-gaze coordinates, the distance from the gaze point, and the bounding box identified as the best match, as illustrated in Figure 2.18.

If the user has selected a processing method that includes object segmentation, the *Segmentation Results* button becomes enabled, along with the *Toggle background* dropdown menu above it.

When transparency is set to off, the segmentation masks are overlaid directly onto the original frames. If object detection was also performed, the corresponding bounding box is displayed alongside the mask, as previously shown in Figure 2.20.

When transparency is turned on, the background is removed, and only the segmented region is shown on a transparent background. This mode effectively isolates the object of interest, making it easier to inspect the precise mask boundaries and evaluate segmentation accuracy, particularly around fine details or occluded areas, as demonstrated in Figure 2.28.

Figure 2.28: Example of the best segmentation mask for each matched word, with transparency (using *SAM2 tiny*).

The *Save Results* section allows users to export the processed output into a structured dataset on disk. The pipeline supports two distinct export modes, catering to different downstream use cases:

- Image-Centric Export (Cropped Dataset): In this mode, the detected or segmented regions are cropped from the original frames using the bounding box coordinates. Optional padding can be applied to include a small margin around the object, preserving contextual surroundings. Each category's images are saved in a separate folder, aggregated across all processed recordings. In the case of segmentation, the cropped regions are saved with transparent backgrounds. This format is ideal for training classification or few-shot learning models where clean, object-centered images are preferred.

- Full-Frame Export with Common Objects in Context (COCO)-Style Annotations: This mode preserves the full video frames and generates a comprehensive JSON file in a COCO-style format. The JSON includes annotations for both object detection (bounding boxes) and segmentation (polygons), along with metadata such as category labels, and file paths. This format is suitable for training modern object detection and segmentation models that require full-scene context and standardized annotation structures.

Users can select the desired export format through the UI, enabling flexibility in how the generated egocentric dataset is utilized in subsequent AI training and evaluation workflows.

Chapter 3

Evaluation and Benchmarking System

A custom benchmarking system was developed to rigorously evaluate the performance of the pipeline. While a straightforward approach involving manually annotating real-world recordings and comparing them to the pipeline's output was initially considered, this method presented significant drawbacks. Manual annotation is inherently time-consuming and labor-intensive, making it impractical for large-scale or iterative evaluations. Moreover, it introduces a high degree of subjectivity, as human annotators may interpret object boundaries, categories, or gaze associations differently, leading to inconsistencies across sessions or between annotators. These variations reduce reproducibility and compromise the reliability of the evaluation process, particularly when precise quantitative comparisons are required. To address these limitations, a more sophisticated and scalable solution was implemented, consisting of a benchmarking framework based on presenting curated images on a monitor during recording, which the user verbally labels while naturally looking at them.

This approach has some advantages:

- Comparison with established datasets: By displaying images from widely used benchmarks such as COCO [22], the system allows for quantitative evaluation against known ground truth annotations.
- Exploration of screen-based interaction scenarios: The setup demonstrates how the pipeline could be extended to applications involving monitors or displays, such as assisted annotation tools.
- Scalable and autonomous evaluation: Unlike manual annotation, this system enables fully automated, repeatable benchmarking across multiple users, sessions, and settings. This makes it possible to collect large-scale evaluation data with minimal human intervention, supporting robust statistical analysis and iterative pipeline improvement.

Despite its advantages, this approach introduces several technical challenges, including precise synchronization between image presentation and sensor data, accurate alignment of predicted annotations with ground truth labels, and ensuring the entire process is fully autonomous and reproducible. Addressing

these challenges was essential to building a robust and scalable evaluation framework.

3.1 Method Overview, Setup and Dataset

To initiate the benchmarking process, the user must first access the benchmarking interface by clicking the *BENCHMARK* button in the main UI (Figure 2.2) after configuring the desired pipeline settings. From this interface, the user can select a reference dataset by clicking the *Select Dataset* button located in the top-left corner, which opens a file dialog to load a pre-formatted dataset directory.

The benchmarking method is designed around a controlled setup in which images from a digital display are verbally annotated by the user while being recorded with the Project Aria Glasses. This approach requires precise alignment between ground truth labels and multimodal inputs (speech, gaze, and vision), and allows systematic evaluation of the pipeline under reproducible conditions.

3.1.1 Recording Setup

A monitor is required for this method. The user should position themselves close to the screen and align the front-facing camera of the Aria glasses with the center of the monitor. This minimizes background distractions and avoids issues caused by the camera's automatic exposure and contrast adjustments, which can otherwise render the screen as an overexposed white blur.

To improve visibility and exposure stability it is recommended to use the light mode of the UI tool, increasing the overall brightness of the displayed interface and making the bright part of the frame bigger, helping the auto-exposure focus on it.

The recommended device settings (shown in Table 3.1) in the Aria app should be used.

Table 3.1: Recommended device configuration (benchmark).

Sensor	**Enabled**	**Rate/Resolution**
RGB Camera	Yes	2880 × 2880 @ 5fps
ET Camera	Yes	320 × 240 @ 5fps
IMU 1	Yes	1000 Hz
IMU 2	Yes	800 Hz
Magnetometer	No	Disabled
Barometer	No	Disabled
Audio	Yes	48 kHz, 7 channels
GPS	No	Disabled
BLE	No	Disabled
Wi-Fi	No	Disabled
SLAM Cameras	No	Disabled

3.1.2 Reference Dataset Preparation

To facilitate benchmarking while maintaining manageable recording size and annotation effort, a custom Python notebook was developed to create a structured

subset of the COCO dataset, or any compatible image collection. This tool enables users to:

- Select specific object categories from the COCO dataset to focus the evaluation on relevant classes.
- Efficiently explore and hand-pick images from those categories.
- Generate a structured directory containing the chosen images and their corresponding annotations in a format compatible with the benchmarking system.

This preprocessing step ensures that only relevant, high-quality images are used, avoiding those with very low contrast or extremely small target objects, which could be challenging to recognize or label under real-world display conditions (e.g., glare, reflections, or screen resolution limits).

3.1.3 Annotation Procedure

During the session, the system displays one image at a time on the monitor, with the name of the target object category shown at the top of the screen (Figure 3.1).

Figure 3.1: Benchmarking interface displaying an image from the reference dataset along with the target object category to be annotated.

The user is instructed to:

- Look at the target object within the image,
- Verbally announce the name of the target object category,
- Press the *NEXT* button (on top) to proceed to the next image.

The controlled benchmarking setup introduces key challenges in evaluation accuracy. First, temporal synchronization between the displayed image and Aria sensor data must be ensured.

Second, establishing spatial correspondence between the front-facing camera's view and the ground truth image displayed on the monitor is a significant challenge. Due to perspective and lens distortion, the recorded frame captures a transformed and potentially degraded version of the screen content.

3.2 Camera-Display Synchronization System

After selecting the reference dataset, the user clicks the *RECORD BENCHMARK* red button, triggering on-screen instructions (Figure 3.2). Once recording is started on the Project Aria Glasses via the Aria App, the user presses *START*.

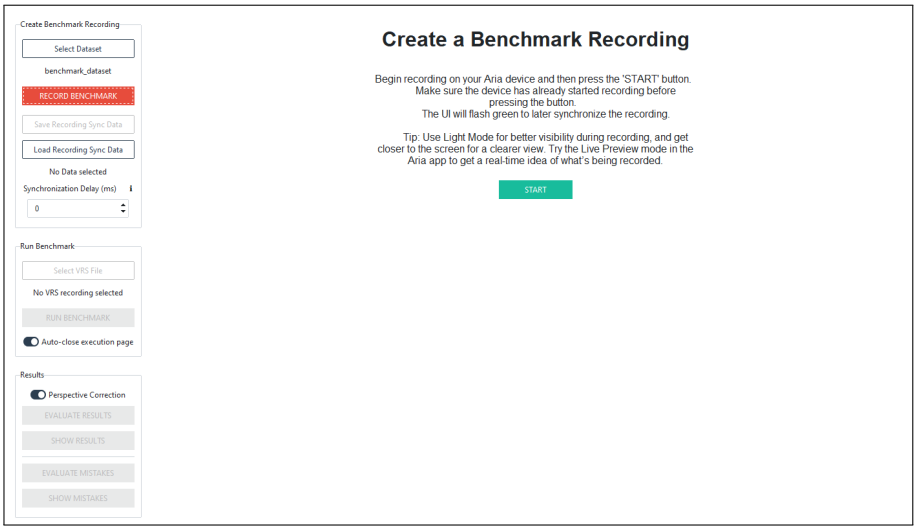


Figure 3.2: Benchmarking interface displaying the recording instruction after pressing the *RECORD BENCHMARK* button.

Just before the benchmark images begin displaying, the UI flashes a large green box for 1 second (Figure 3.3). At the moment this box appears, a local computer timestamp is recorded. This timestamp serves as a synchronization anchor, later used to align the on-screen image sequence with the camera feed by matching it to the corresponding frame in the device's recording.

Due to the controlled environment, synchronization is achieved robustly, regardless of UI light/dark mode, by detecting a sudden increase in green channel prominence in the front camera's video stream.

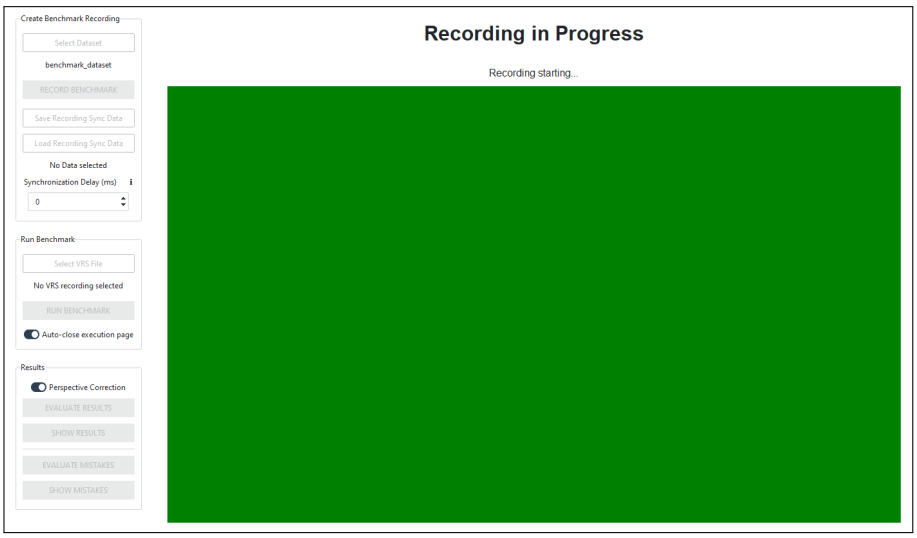


Figure 3.3: Benchmarking interface displaying the synchronization green box at the start of the benchmark.

The algorithm analyzes early frames of the recording (up to the first matched word index for efficiency), computing the average RGB values per frame (Figure 3.4) and, crucially, the change (delta) in each channel's average compared to the previous frame (Figure 3.5).

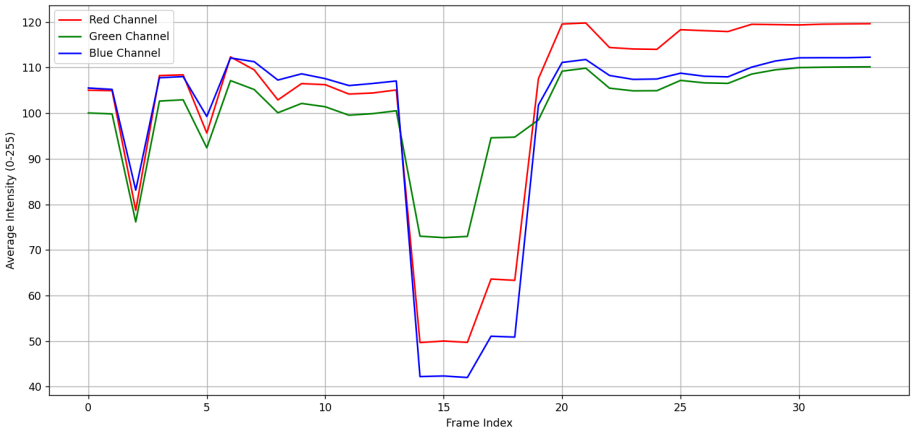


Figure 3.4: Plot of the average RGB values for the benchmark's recording.

Frames where:

- Red and blue deltas are negative (indicating a decrease),
- Green delta is less negative or positive compared to both red and blue

are considered candidates for the green flash event. This condition accounts for exposure adjustments: in dark mode, green typically increases. In light mode it may not increase but slightly decrease, less than red/blue, due to auto-exposure.

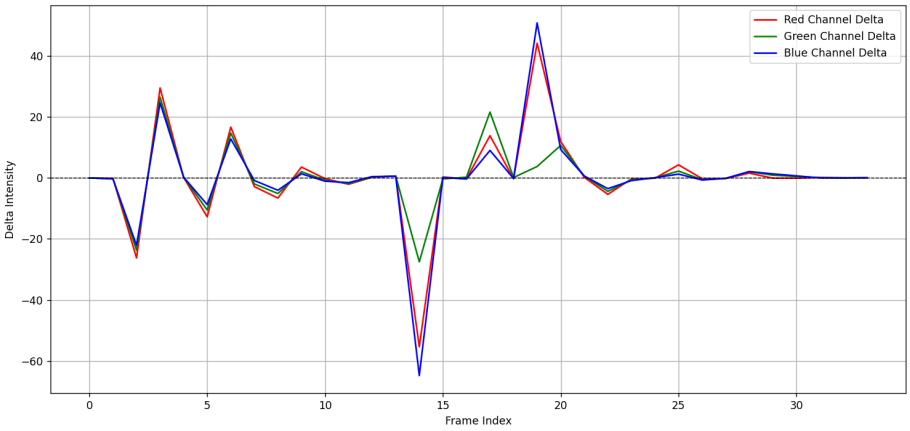


Figure 3.5: Plot of the delta of the average RGB values for the benchmark's recording.

For each candidate frame, a green score is computed:

$$\text{avg\_rb\_change} = \frac{|d_r| + |d_b|}{2} \qquad \text{(average change in red and blue)}$$

$$\text{green\_score} = d_g + \text{avg\_rb\_change} \qquad \text{(green score)}$$

where:

d_r, d_g, d_b : delta (change) in red, green, and blue averages from the previous frame.

The frame with the highest green score is selected as the moment the green box appears in the camera feed.

A timestamp sync delta is then calculated to convert between computer time and device time:

$$\text{sync\_delta} = t_{\text{vrs}} - t_{\text{start}} + \Delta_{\text{delay}}$$

where:

t_{vrs} : device timestamp of the detected green flash,

t_{start} : first timestamp in the recording,

Δ_{delay} : optional user-defined delay (in microseconds, default 0), compensating for rendering lag on slow systems.

This delta is stored within the *recording* object and serves as a global time offset to align timestamps across the computer and device throughout the benchmark.

Timestamps indicating when each image is displayed can be saved or loaded from disk, enabling the reproduction of the same benchmark sequence and timing across different experimental pipeline configurations.

This synchronization approach is both efficient and robust, leveraging a simple visual cue (the 1 second green flash) to establish precise temporal alignment between the on-screen stimulus and the Project Aria Glasses recording. This method provides a lightweight and computationally efficient solution for synchronization, requiring minimal processing and no external hardware or complex calibration.

3.3 Evaluation Against Ground Truth Annotations

Evaluation in this benchmarking system is not straightforward. We cannot directly compare annotations from a recorded frame to the corresponding ground truth image. The raw frames captured by the Project Aria Glasses contain more than just the displayed image. They include the surrounding monitor bezel, UI elements, ambient environment, and varying perspectives due to the user's position and orientation. Additionally, significant visual distortions are introduced by the camera's fisheye lens and the non-frontal viewing angle relative to the screen.

To enable meaningful comparison, these real-world discrepancies must be addressed. A robust transformation process is required to align the camera-captured frames with the ground truth image's reference frame. The following sections detail the preprocessing steps that make accurate evaluation possible, ensuring that predictions and ground truth are compared in a consistent and spatially aligned coordinate system.

Regarding the camera's fisheye distortion, a characteristic optical effect caused by the wide-angle lens that results in pronounced barrel distortion, where straight lines appear curved especially toward the edges of the image, this is corrected using a calibrated distortion model. The distortion is parameterized by intrinsic camera coefficients that can be used to estimate a correction, as it is detailed in the "Camera Distortion Correction 2.3.2" section of this document. This step is optional during normal pipeline execution but becomes mandatory during benchmark evaluations, as it is crucial for restoring geometric accuracy and ensuring reliable spatial alignment with the reference dataset image.

3.3.1 Image Localization in Frame

The first step in comparing predictions to ground truth is localizing the reference dataset image within the frame captured by the front-facing camera. Since the camera observes the display within a broader scene affected by perspective, scale, and position, a geometric alignment is required to bring both images into the same coordinate space.

Several approaches were explored, including the use of ArUco markers (Figure 3.6). The initial idea was to place a single marker in one corner of the displayed image. Knowing the marker size and the image dimensions would allow estimation of the other three corners. To improve robustness, a second marker was added in the opposite corner. However, marker detection was inconsistent, even with added borders to enhance visibility, and the resulting corner estimates suffered from poor accuracy, even when the markers were correctly detected.

Figure 3.6 illustrates the detected markers, the individual corner predictions from each marker, and the combined estimate. Due to unreliable detection and insufficient geometric precision, this method was ultimately discarded in favor of a more robust, marker-free approach.

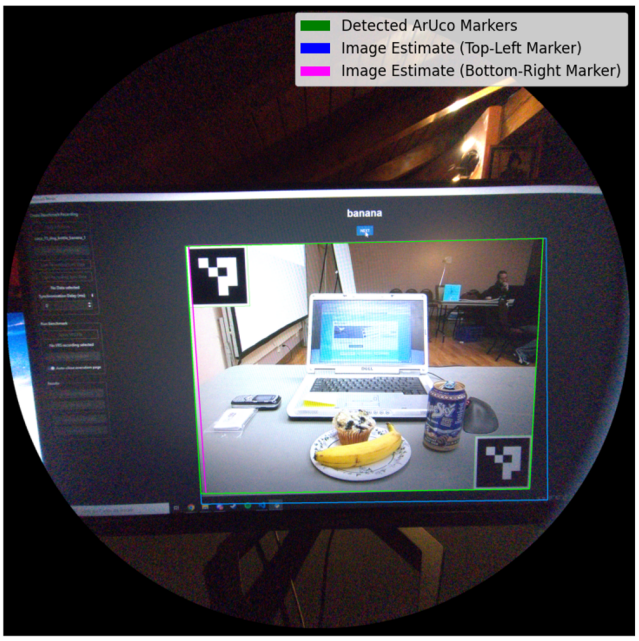


Figure 3.6: Benchmark's reference image with ArUco markers.

To reliably locate the reference image within the camera frame, a feature-based matching approach using the Scale-Invariant Feature Transform (SIFT) [23] was implemented. SIFT is a computer vision algorithm used to detect and describe distinctive keypoints in an image that are invariant to scale, rotation, and partially invariant to illumination and viewpoint changes.

The algorithm operates by identifying keypoints, which are locations in the image that exhibit stable, repeatable patterns across transformations. These are typically corners or regions with high gradient variation. For each keypoint, a descriptor is computed: a 128-dimensional vector encoding the local gradient structure around the point, normalized to be robust to changes in brightness and contrast.

Matching proceeds by comparing descriptors between the dataset's reference image and the camera-captured frame. A brute-force nearest-neighbor matcher is used to find potential correspondences, followed by a ratio test to filter ambiguous matches. Specifically, for each descriptor in the reference image, the two closest matches in the frame are identified. A match is retained only if the distance to the best match is less than 75% of the distance to the second-best match. This effectively eliminates false positives caused by repetitive textures or ambiguous patterns.

Only matches that pass this threshold are considered high-quality. A minimum of 10 such good matches is required to proceed with localization. Otherwise, the system concludes that the reference image is not sufficiently visible in the frame.

Given the set of matched keypoints, a geometric transformation (a homography) is estimated to map points from the reference image coordinate system to their corresponding locations in the camera frame. The homography accounts for perspective distortion and is computed using the RANdom SAmple Consen-

sus (RANSAC) algorithm, which iteratively selects random subsets of matches to estimate candidate transformations.

Once the optimal homography matrix is determined, it is applied to the four corners of the reference image, defined by its width and height, to compute their projected coordinates in the camera frame. These transformed corners define the precise location, orientation, and scale of the displayed image within the recording, enabling accurate spatial alignment.

Figure 3.7: Benchmark's reference image detected corners through SIFT.

This SIFT-based method proves highly robust across diverse viewing conditions, including varying distances, angles, and lighting, without requiring artificial markers or controlled environments. The magenta border was added around the reference image to help the SIFT algorithm accurately find the edges of the image, regardless of the background color.

3.3.2 Perspective Distortion Correction

Once the reference image is localized within the camera frame and its four corners are identified, the next step is to correct the perspective distortion introduced by the non-frontal viewing angle. The front-facing camera typically captures the display at an oblique angle, causing the rectangular image to appear as a general quadrilateral in the recorded frame. This effect, known as perspective foreshortening, distorts spatial relationships and prevents accurate comparison between annotations and ground truth.

This correction can be activated using the toggle in the benchmark's UI. Keeping it enabled is recommended to ensure precise alignment with the ground

truth. Note that it is only available during benchmark runs, as it relies on a reference image.

To restore the image to its canonical, front-parallel view, a homography-based perspective rectification is applied. A homography is a projective transformation that maps points from one plane to another under perspective projection. In this case, it transforms the quadrilateral region defined by the detected image corners in the camera frame to a corrected rectangle that preserves the original aspect ratio and internal geometry of the reference image.

The transformation is computed using the four detected corner points (top-left, top-right, bottom-right, and bottom-left) which define the distorted region in the source image. These points serve as the source coordinates in the homography estimation. The corresponding destination coordinates are defined in the reference benchmark's image space such that the resulting rectangle:

- Maintains the same aspect ratio as the original reference image,
- Is centered within a slightly padded canvas to avoid clipping and provide context.

To improve usability and provide visual context in downstream analysis, a 50% padding (relative to the size of the reference image) is added uniformly around the rectified image. This results in a final output image (Figure 3.8) that fully contains the undistorted image with sufficient border space, useful for visualization and when overlaying annotations or gaze data.

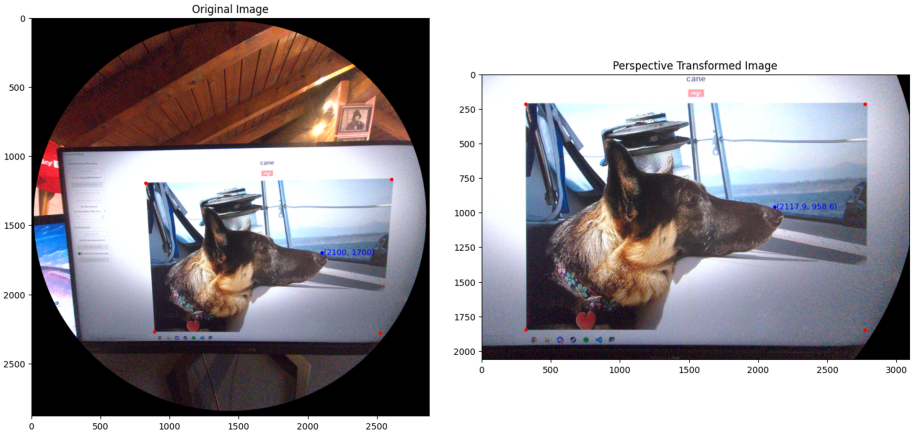


Figure 3.8: Camera frame before and after perspective distortion correction, with highlighted corners and eye gaze point.

Importantly, the same transformation is also applied to auxiliary data aligned with the original image space. The eye gaze coordinates are mapped to the corrected space. This preserves the spatial correspondence between gaze and content.

Similarly, any bounding boxes or segmentation masks (in COCO polygon format) associated with the original frame are transformed using the same homography, ensuring that annotated regions remain geometrically consistent after correction.

This approach effectively eliminates perspective skew, producing a rectified image in which straight lines are restored, proportions are preserved, and pixel locations align with the original ground truth layout. As a result, it enables pixel-accurate comparison of predicted annotations, such as object bounding boxes or segmentation masks, with their corresponding ground truth instances.

3.3.3 Evaluation Methods

After applying camera distortion correction, image localization, and perspective rectification, the captured frames are fully aligned with the reference coordinate system of the dataset's images. At this point, the transformed frames are ready for evaluation.

The system supports both automatic and manual evaluation modes (Figure 3.9). Automatic evaluation enables fast, consistent, and scalable assessment using predefined criteria, while manual evaluation provides the user with fine-grained control over the comparison process, allowing for subjective judgment particularly useful when assessing edge cases or ambiguous annotations.

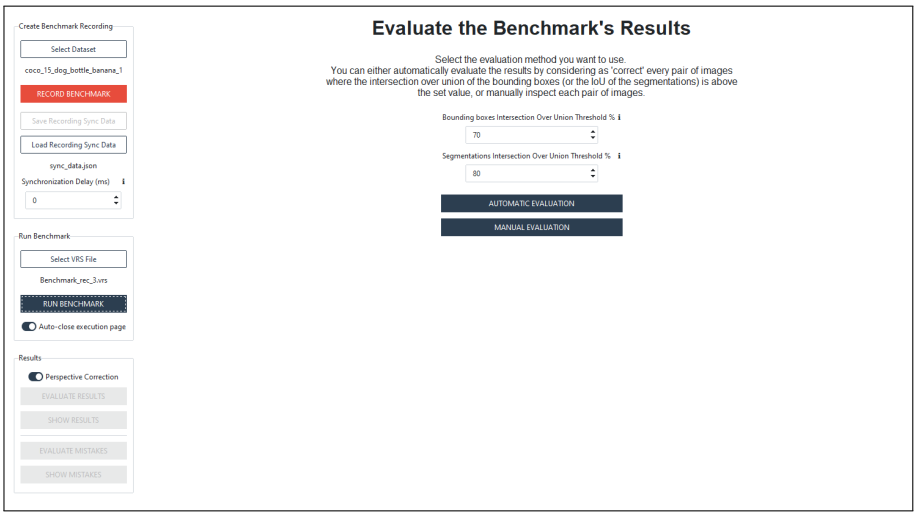


Figure 3.9: Benchmark evaluation mode (manual or automatic), with user-defined thresholds for the automatic mode.

Manual Evaluation

Manual evaluation allows the user to individually assess each predicted annotation and determine whether it should be classified as correct or incorrect. To initiate this process, the user clicks the *MANUAL EVALUATION* button (shown in Figure 3.9), which shows the manual review's interface.

As shown in Figure 3.10, results are displayed one frame at a time, with separate panels for detection (top) and segmentation (bottom) if both processing modes were enabled. For each task, the left side shows the dataset's reference image, overlaid with the ground truth annotation in red and the transformed predicted annotation in blue. For context, the right side displays the original

camera-captured frame with the predicted annotation in blue, allowing the user to verify how the annotation aligns with the real-world scene.

Figure 3.10: Benchmark manual evaluation's example.

To support informed decisions, the UI displays the Intersection over Union (IoU) value for each prediction relative to the ground truth. This quantitative measure is provided as a reference to help the user judge spatial accuracy, though the final decision remains subjective and can account for contextual factors beyond IoU.

The user can mark each prediction (detection, segmentation, or both) as correct or incorrect using the corresponding buttons at the bottom of the panel. Once judgments are made, pressing the "Next" button advances to the results of the subsequent frame, enabling systematic review across the entire sequence.

This mode is particularly valuable for validating edge cases, ambiguous predictions, or scenarios where automated metrics may not fully reflect correctness.

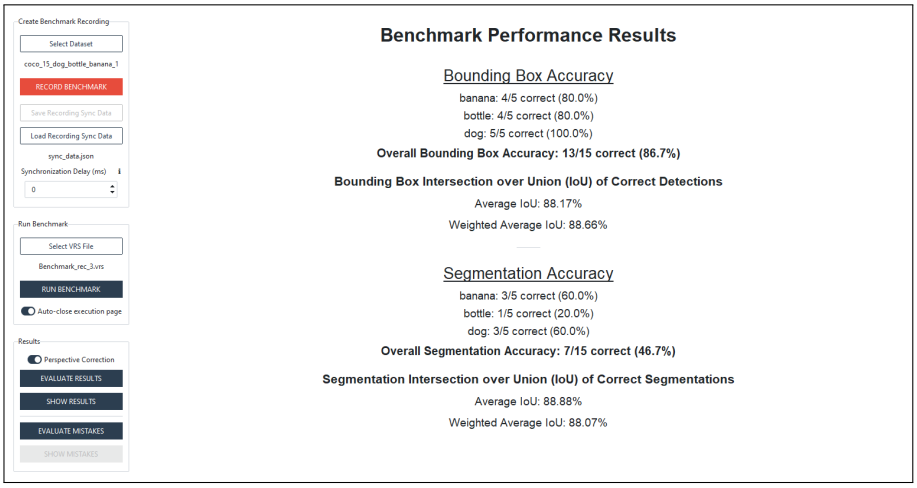


Figure 3.11: Benchmark evaluation's results example.

After all predictions have been reviewed, the evaluation results are displayed on screen (Figure 3.11). For each processing method (detection and segmentation) performance is broken down both per class and overall, providing detailed insight into model behavior across different object categories.

For each class, the system reports the percentage of images in which the annotation was judged as correct, along with the overall accuracy across all classes. In addition to binary correctness, quantitative metrics are provided to reflect spatial precision:

- The mean Intersection over Union (IoU) is computed over all correctly annotated predictions, giving a measure of average alignment quality.
- A size-weighted IoU is also calculated, where each IoU value is weighted by the relative area of the bounding box with respect to the entire image. This ensures that larger objects, whose annotations have greater visual or functional impact, contribute more significantly to the average.

This weighting scheme addresses a key limitation of standard IoU averaging: small objects can exhibit large IoU fluctuations due to minor pixel-level shifts, which may not reflect meaningful errors. By emphasizing larger regions, the weighted IoU provides a more perceptually relevant measure of performance, aligning better with real-world usability.

Automatic Evaluation

As illustrated in Figure 3.9, the system allows users to perform automatic evaluation by defining two threshold values: one for the bounding box Intersection over Union (IoU) and another for the segmentation mask IoU. Predictions that achieve an IoU value at or above the respective threshold are classified as correct; those below are marked as incorrect. This rule-based approach enables scalable, consistent, and objective assessment across the entire dataset.

The threshold values are user-configurable, allowing flexibility in defining what constitutes an acceptable prediction. Users may run the evaluation multiple times with different thresholds to analyze performance under varying strictness levels.

During execution, the interface displays real-time progress, indicating which frame is currently being processed. Once completed, the results are presented in the same structured format as in manual evaluation (Figure 3.11), including:

- Class-specific accuracy,
- Overall accuracy,
- Mean IoU for correct predictions,
- Size-weighted IoU.

This mode ensures full reproducibility, as the same thresholds applied to the same data will always yield identical outcomes, making it ideal for benchmarking and comparative analysis. By eliminating subjective judgment, automatic evaluation provides a scalable and standardized method for assessing model performance.

3.4 Mistakes Evaluation

After completing either the manual or automatic evaluation, the *EVALUATE MISTAKES* button becomes enabled. Clicking it opens the mistakes analysis interface (see Figure 3.12), designed to help users investigate and classify the root causes of incorrect predictions.

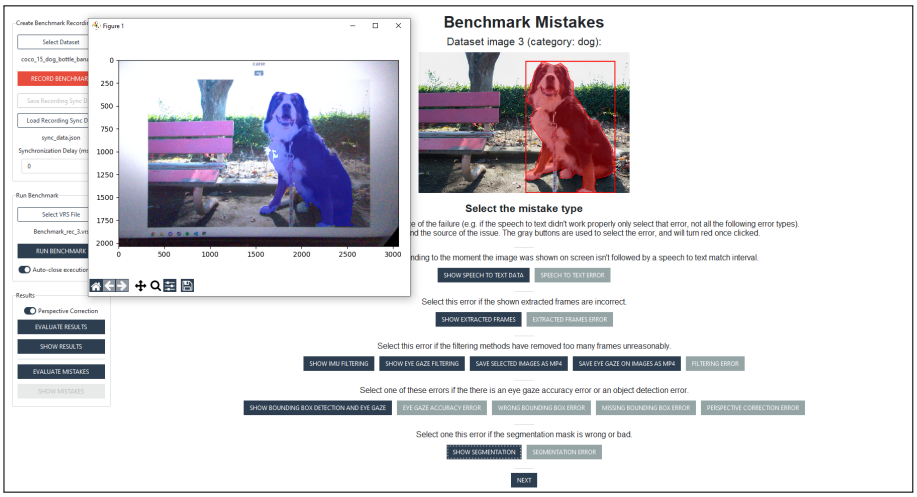


Figure 3.12: Benchmarking interface for the evaluation of pipeline mistakes.

This section enables users to systematically classify the types of errors made by the pipeline when annotations for a given benchmark image were not correctly predicted either in the detection, segmentation, or both stages. One image at a time, the interface displays each failed case, showing the reference image along with its ground truth annotations (bounding box and segmentation mask) for clear visual comparison.

To aid diagnosis, a set of interactive debug tools is provided in the form of blue buttons, each revealing a specific visualization such as intermediate resuts of parts of the pipeline. These plots help the user trace the error through the processing pipeline and identify where the failure occurred.

Based on this investigation, the user can assign blame to one or more components of the system (e.g., speech to text model, filtering, eye gaze accuracy) by clicking the corresponding gray selector buttons, which turn red upon selection to indicate a diagnosed failure point.

Once the source(s) of the error have been marked, the user proceeds to the next mispredicted frame by pressing the *NEXT* button, enabling a structured, frame-by-frame analysis of systematic weaknesses in the pipeline.

This diagnostic capability transforms the benchmark from a mere performance evaluator into a powerful debugging tool, helping developers understand not just how well the system performs, but why it fails, enabling targeted improvements and more robust system design.

The diagnostic buttons can visualize the following information:

- *SHOW SPEECH TO TEXT DATA*: Overlays the matched words and their time intervals on the audio waveform, as shown in Figure 2.4.

- *SHOW EXTRACTED FRAMES*: Displays the set of frames extracted from the audio-matched interval, sampled at the user-defined reduced frame rate. This visualization shows the frames *before* any filtering is applied.
- *SHOW IMU FILTERING*, *SHOW EYE GAZE FILTERING*, *SAVE SELECTED IMAGES AS MP4* and *SAVE EYE GAZE ON IMAGES AS MP4*: Useful to investigate issues within the frames filtering and selection process (see Figure 2.12, Figure 2.13, Figure 2.14, Figure 2.15 and Figure 2.16)
- *SHOW BOUNDING BOX SELECTION AND EYE GAZE*: Visualizes object detection results and eye gaze across selected frames, including all candidate bounding boxes and the one chosen as optimal based on gaze proximity and confidence. See Figure 2.18.
- *SHOW SEGMENTATION*: Displays the generated segmentation masks for the selected frames and highlights the final mask chosen by the pipeline, as shown in Figure 2.21.

After all mispredicted frames have been reviewed and the root causes of failure classified, the system generates a comprehensive error breakdown and displays it in the window (Figure 3.13).

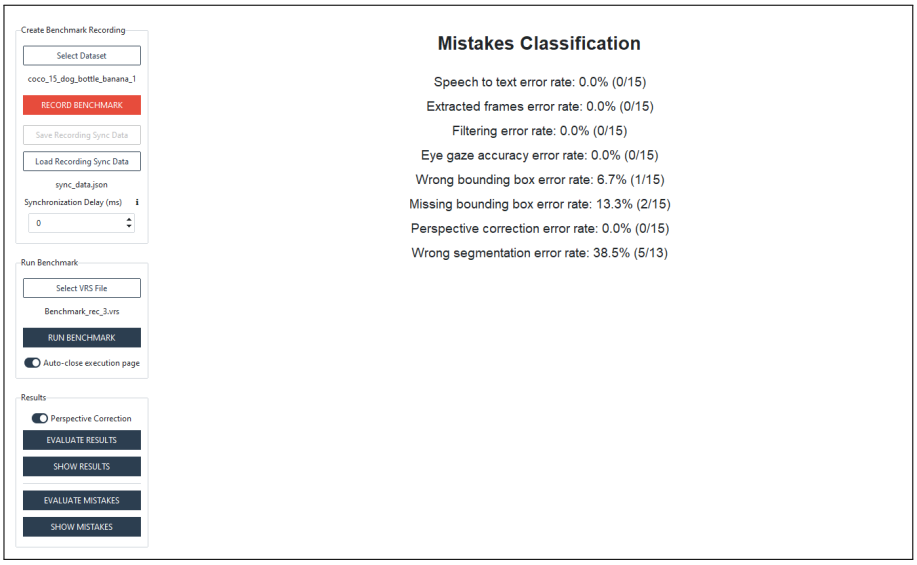


Figure 3.13: Example of benchmark's mistakes evaluation.

The results present the error rate for each stage of the pipeline. Crucially, these rates are computed relative to the number of frames that successfully passed all prior stages, enabling a per-stage failure analysis that reflects the actual flow of data through the system.

This conditional perspective reveals not only how often a component fails, but also where bottlenecks or weaknesses occur in the processing chain. For instance, a high error rate in segmentation may be less impactful if few frames even reach that stage due to earlier failures in detection or filtering.

By quantifying the contribution of each module to the overall error, this analysis enables prioritization of improvements where they will have the greatest impact on end-to-end performance. The result is a transparent, actionable diagnostic tool that goes beyond accuracy metrics to expose the underlying causes of system behavior.

Chapter 4

Evaluation Results

This chapter presents the quantitative evaluation of the proposed pipeline across two benchmarks executed on the same dataset: `benchmark_small` and `benchmark_large`. The assessment covers execution time, object detection and segmentation performance, as well as a classification of error types. All results are reported in an objective manner, based on empirical measurements and metrics such as accuracy, IoU, and error rate analysis. The tables in this chapter provide a direct comparison between the two benchmarks, enabling a clear understanding of system behavior under varying models sizes. Interpretation and qualitative discussion of these results are deferred to the next chapter.

Table 4.1: Execution time comparison for benchmarks.

Task	**benchmark_small**	**benchmark_large**
Speech to text & matching	43.43s	45.54s
Frames extraction	49.14s	48.87s
Eye gaze extraction	116.16s	0.04s
Frames selection	0.56s	0.68s
Object Detection	9.74s	17.57s
Object Segmentation	44.29s	230.22s
Total execution time	**263.41s**	**343.01s**

Table 4.2: Object detection results across benchmarks.

Object Class	**benchmark_small**	**benchmark_large**
banana	9/20 correct (45.00%)	14/20 correct (70.00%)
bird	20/20 correct (100.00%)	19/20 correct (95.00%)
bottle	16/20 correct (80.00%)	17/20 correct (85.00%)
dog	20/20 correct (100.00%)	20/20 correct (100.00%)
tv	18/20 correct (90.00%)	19/20 correct (95.00%)
Overall Accuracy	**83/100 (83.00%)**	**89/100 (89.00%)**
Average IoU	**87.60%**	**90.47%**
Weighted Avg IoU	**90.43%**	**92.33%**

Table 4.3: Segmentation results across benchmarks.

Object Class	**benchmark_small**	**benchmark_large**
banana	10/20 correct (50.00%)	14/20 correct (70.00%)
bird	19/20 correct (95.00%)	19/20 correct (95.00%)
bottle	15/20 correct (75.00%)	16/20 correct (80.00%)
dog	19/20 correct (95.00%)	19/20 correct (95.00%)
tv	15/20 correct (75.00%)	15/20 correct (75.00%)
Overall Accuracy	**78/100 (78.00%)**	**83/100 (83.00%)**
Average IoU	**85.36%**	**86.21%**
Weighted Avg IoU	**86.56%**	**85.75%**

Table 4.4: Error rate classification across benchmarks.

Error Type	**benchmark_small**	**benchmark_large**
Speech to text	0.00% (0/100)	0.00% (0/100)
Extracted frames	0.00% (0/100)	0.00% (0/100)
Filtering	1.00% (1/100)	1.00% (1/100)
Eye gaze accuracy	0.00% (0/99)	0.00% (0/99)
Wrong bounding box	3.03% (3/99)	3.03% (3/99)
Missing bounding box	13.13% (13/99)	7.07% (7/99)
Perspective correction	0.00% (0/99)	0.00% (0/99)
Wrong segmentation	6.97% (6/86)	6.52% (6/92)

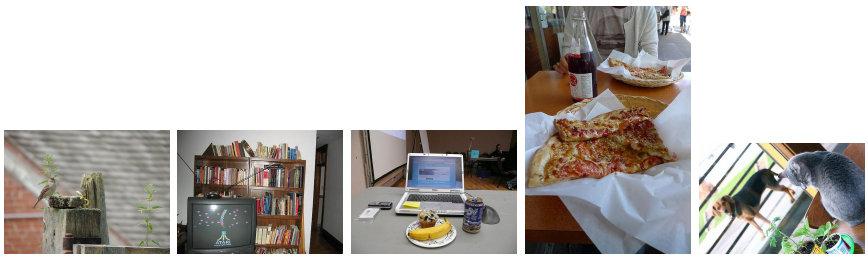
Chapter 5

Evaluation Discussion

This chapter provides a comprehensive discussion of the results presented in the previous chapter, analyzing the performance differences between the two benchmarks, exploring potential causes, and interpreting key observations through the lens of system design and dataset characteristics.

It is important to note that the results presented here most likely represent a lower-bound estimate of the pipeline's performance. Since the evaluation was conducted on the recording of a monitor displaying the dataset images, the recording's quality was inevitably affected by glare, reflections, and reduced contrast. Such artifacts are not representative of typical real-world recordings and would not normally be present during standard pipeline operation. Therefore, the reported accuracy values should be interpreted as conservative, with actual performance in practical scenarios likely exceeding these benchmarks.

The evaluation is based on a custom dataset composed of 100 images, evenly distributed across five object classes: *banana*, *bird*, *bottle*, *dog*, and *tv*. These images were carefully selected from the COCO 2017 validation set using a dedicated Python notebook to assist in filtering and organization. To ensure clarity and reduce ambiguity during verbal labeling and evaluation, only images containing a single instance of the target class were included. Additionally, images with objects that were too small, heavily occluded, or exhibited extremely low contrast relative to the background were excluded. This filtering strategy aimed to minimize confounding factors related to monitor-recording artifacts such as glare, reflection, or poor visibility, thereby improving the reliability and interpretability of the benchmark results.



(a) Bird (b) TV (c) Banana (d) Bottle (e) Dog

Figure 5.1: Examples from the curated evaluation dataset.

A representative sample of the selected images is shown in Figure 5.1, illustrating the visual diversity and quality of the dataset.

A recording session was conducted with this curated dataset, using Italian speech to evaluate the pipeline's multilingual capabilities. From this recording, two separate benchmarks were executed through the user interface, each applying a different model configuration while keeping all other parameters constant.

Both benchmarks used the *Image Object Detection and Segmentation* method, using the detection results as part of the segmentation prompt.

The results were then classified as correct or not correct using the automatic evaluation system described in Chapter 3.3.3, which compares predicted bounding boxes and masks against ground truth annotations based on defined IoU thresholds. Both the detection and segmentation thresholds were set to 30%, a deliberately low value that allows even relatively imprecise predictions to be counted as correct, provided that they overlap the ground truth by at least this amount. In this way, masks that are imperfect but still correspond to the intended object are included in the averages, while only clear failures, such as frames where the prediction does not overlap the target object or captures the wrong object entirely, are excluded.

The core pipeline configuration used in both benchmarks is defined by the following default settings:

Table 5.1: Default pipeline configuration for both benchmarks.

Parameter	**Value**
`words_dictionary`	bottiglia, cane, banana, televisione, uccello
`words_dictionary_english`	bottle, dog, banana, tv, bird
`spoken_language`	it
`edit_distance`	25
`max_frames_per_word`	3
`fps_reduction_value`	3
`eye_gaze_extraction_method`	1
`gyro_threshold_value`	0.5
`jerk_threshold_value`	100.0
`eye_gaze_confidence_interval_threshold_value`	0.003
`eye_gaze_filter_tolerance_value`	80
`camera_distorsion_correction`	1
`method_type`	3
`bounding_box_distance_percentage_threshold`	6.0

Both benchmarks use this identical configuration. The only difference lies in the size of the models used for object detection and segmentation, as detailed in Table 5.2. In the first benchmark (`benchmark_small`), smaller or base models were employed to assess baseline performance with lower computational cost. In the second benchmark (`benchmark_large`), larger models were used to evaluate whether increased model capacity leads to measurable improvements in detection and segmentation accuracy.

Notably, the WhisperX model for speech-to-text transcription was kept fixed at the *base* (74M parameters) variant in both benchmarks. This decision was based on preliminary testing: the *tiny* (38.9M) version produced insufficient transcription accuracy, particularly in handling background noise and speaker variability, while the *base* model achieved near-perfect transcription with minimal errors (within the edit distance allowance for error) rendering further increases in model size unnecessary for this task.

Table 5.2: Model configuration for benchmarks.

Model	**benchmark_small**	**benchmark_large**
WhisperX	Base (74M)	Base (74M)
YOLO_World	Small (77M)	Large (110M)
SAM2	Tiny (38.9M)	Large (224.4M)

The following sections will analyze how these configuration choices, particularly the model scaling, impact overall performance, error types, and execution time, providing insight into trade-offs between accuracy, speed, and resource requirements.

5.1 Accuracy Results

This section analyzes the accuracy of object detection and segmentation across the two benchmarks, focusing on overall performance, per-class trends, and the relationship between detection and segmentation outcomes.

5.1.1 Object Detection Performance

In `benchmark_small`, the system achieves an overall detection accuracy of 83.00%, which is solid but leaves room for improvement. Notably, performance varies significantly across classes: while *bird*, *dog*, and *tv* reach or exceed 90% accuracy, the *banana* class performs poorly at only 45.00% (9/20 correct). This suggests that certain visual or contextual characteristics of bananas such as variability, low contrast to other objects, or frequent partial occlusion, make them more challenging to detect reliably with the smaller model configuration.

Despite these detection gaps, the localization quality of successful detections is strong. The average IoU of 87.60% and weighted average IoU of 90.43% indicate that predicted bounding boxes are closely aligned with ground truth annotations. Achieving such high IoU values is particularly noteworthy, since even minor pixel-level differences, especially on small objects, can significantly lower the score. This suggests that the results are already near the upper bound of what can be expected given annotation subjectivity and interpretation differences.

In `benchmark_large`, where larger models are used for detection and segmentation, overall accuracy improves to 89.00% ($\sim$+6%). The most notable gain is in the *banana* class, which increases to 70.00% correct detections, indicating that model capacity plays a role in recognizing this more challenging category. However, this accuracy remains below that of other classes, warranting further

investigation into whether specific visual features contribute to the persistent difficulty (done in Section 5.4).

Localization accuracy also improves: average IoU rises to 90.47% (∼+3%), and weighted average IoU reaches 92.33% (∼+2%). This indicates not only better detection rates but also tighter, more precise bounding boxes when objects are found, suggesting that larger models provide both improved recognition and spatial refinement.

5.1.2 Segmentation Performance

Segmentation accuracy follows a similar trend but is naturally constrained by detection performance: if an object is not detected, no segmentation is attempted. As a result, segmentation accuracy is slightly lower than detection accuracy in both benchmarks: 78.00% for `benchmark_small` (vs. 83.00% detection) and 83.00% for `benchmark_large` (vs. 89.00% detection). The 5–6% drop implies that few additional errors are introduced during the segmentation stage itself, highlighting the robustness of the segmentation pipeline once detection succeeds.

Interestingly, segmentation quality, measured by average IoU (85.36% → 86.21%) and weighted average IoU (86.56% → 85.75%), shows only marginal improvement in `benchmark_large`.

This limited change may stem from several factors:

- The segmentation model (SAM2) might already be operating near its effective limit with the given input cues.
- There could be inconsistencies or inaccuracies in the ground truth masks, especially for ambiguous boundaries or small objects, which cap achievable IoU scores.
- The issues related to externally recording the screen (e.g., glare, reflections) may introduce noise that affects segmentation quality regardless of model size.

These aspects are further explored in Section 5.4.

5.2 Error Rate

The error classification results, summarized in Table 4.4, provide valuable insight into the failure modes of the pipeline and highlight which components benefit from model scaling. By analyzing the types and frequencies of errors across `benchmark_small` and `benchmark_large`, we can identify both model-related limitations and systemic issues in the processing chain.

5.2.1 Missing Bounding Boxes

The most significant source of error in `benchmark_small` is the *missing bounding box* category, with 13 out of 99 instances (13.13%) failing to be detected. While some of these misses may stem from edge cases such as ambiguous object boundaries, atypical poses, or category misinterpretations, the substantial improvement in `benchmark_large` (reduced to 7/99, or 7.1%) strongly suggests that model capacity plays a key role.

The fact that the larger YOLO-World model nearly halves the number of missed detections indicates that stronger feature extraction and context modeling improve robustness, especially for subtle or visually complex instances. This supports the conclusion that detection is a primary bottleneck in the smaller configuration and that investing in larger detection models yields meaningful gains in completeness.

5.2.2 Wrong Bounding Boxes

The *wrong bounding box* error rate remains unchanged between benchmarks at 3.03% (3/99), suggesting that these errors are not primarily due to model size but may instead reflect inherent ambiguities in the task. Potential causes include:

- Partial occlusion of the target object, reducing visible features,
- Uncommon orientations or poses that deviate from typical training examples,
- Visual similarity to other categories (e.g., a bottle resembling a vase),
- Ground truth labels that include borderline or loosely related instances.

The stability of this error across model sizes implies that addressing it may require improvements beyond simply increasing model capacity and have been investigated in the following sections.

5.2.3 Segmentation Errors

Segmentation errors remain numerically stable at 6 cases in both benchmarks (6/86 in `benchmark_small`, 6/92 in `benchmark_large`), but the error rate decreases slightly from 6.97% to 6.52% due to the higher number of valid bounding boxes coming from the object detection stage in the larger benchmark. This indicates that the SAM2 segmentation model performs consistently when provided with correct detections, and that most segmentation failures are not due to model degradation but rather to specific, possibly non-model-related factors.

These persistent errors may arise from:

- Inaccurate or noisy eye-gaze prompts used to condition the segmentation,
- Misalignments between the detected bounding box and the actual object,
- Ambiguities in mask interpretation, especially for objects with irregular, low contrast or transparent boundaries, or given by the degradation coming from the monitor's recording.

Further qualitative inspection of these cases is necessary to determine whether they represent fundamental model limitations or artifacts of the benchmark pipeline.

5.2.4 Filtering Error

A single error in both benchmarks is attributed to the *filtering process*, where a unique valid frame may have been incorrectly discarded. Given the low frame rate of the recording and the filtering thresholds (e.g., motion jerk, gaze stability), it is possible that the only suitable frame for a particular query was filtered out correctly due to violations of the criteria. Alternatively, the threshold values may be overly conservative, leading to unnecessary frame rejection.

This isolated but consistent error highlights the need for further investigation. In particular, it should be examined whether the thresholds are set too conservatively and might need to be lowered, since their optimal values cannot be determined *a priori* and depend strongly on the recording conditions. Alternatively, the error may point to an unexpected issue in the filtering process itself. These aspects will be further explored in the following sections.

5.3 Execution Time

The execution times reported in Table 4.1 reveal important trade-offs between computational efficiency and model performance across the two benchmarks. While both configurations follow the same processing pipeline, the use of larger models in `benchmark_large` leads to significant differences in runtime, particularly in the detection and segmentation stages.

Several pipeline components exhibit nearly identical execution times between the two benchmarks:

- Frames extraction: 49.14s (`small`) vs. 48.87s (`large`),
- Speech-to-text and matching: 43.43s vs. 45.54s,
- Frames selection: 0.56s vs. 0.68s.

These stages are unaffected by model scaling, as they rely on fixed algorithms or the same underlying models (e.g., WhisperX `base` for transcription). The minor variation in speech processing time ($\sim$+4.86%) falls within expected system variance due to background processes, disk I/O, or thermal throttling, and does not reflect algorithmic or model differences.

Notably, eye gaze extraction was only performed for `benchmark_small`, as the computed gaze data is saved to disk and reused across subsequent runs. This highlights an effective optimization: gaze processing is computationally lightweight but redundant if repeated, so one-time computation significantly improves reusability and efficiency. This can be really useful when executing the pipeline multiple times on the same recording, to test different configurations.

Object detection runtime increases from 9.74s to 17.57s, a rise of approximately 80.30% in `benchmark_large`. This is expected, as the switch from YOLO-World Small (77M) to Large (110M) involves deeper architectures and higher-resolution feature processing.

Despite the increased cost, this upgrade yields a substantial improvement: missing bounding boxes are reduced by nearly half (from 13 to 7), and overall detection accuracy rises from 83% to 89%. Given that detection failures propagate through the entire pipeline (preventing segmentation and increasing final error

rates), this performance gain may justify the additional time in accuracy-critical applications.

The most dramatic increase occurs in the object segmentation stage, where execution time jumps from 44.29s to 230.22s, an increase of over 5x, due to the switch from SAM2 Tiny (38.9M) to SAM2 Large (224.4M). However, this massive computational investment yields only marginal improvements in segmentation accuracy (78.00% → 83.00%) and IoU metrics (85.36% → 86.21%). The segmentation accuracy is also most likely boosted by the improved detection results, as more correct bounding boxes lead to more successful segmentations.

This suggests strong diminishing returns: while the larger model is more capable in theory, the actual benefit in this context is limited, possibly due to already-sufficient performance from the smaller model, or bottlenecks elsewhere (e.g., imprecise prompts from detection or gaze). For time-sensitive applications, such as real-time assistive systems, this cost-benefit ratio may be unacceptable.

It should be noted that reported times include model loading overhead, which can be significant for large models like SAM2 Large. In a real-world deployment with persistent inference (e.g., continuous interaction), models could remain loaded in memory, eliminating this one-time cost and reducing per-query latency.

Table 5.3: Execution time comparison as percentage of total.

Task	**benchmark_small**	**benchmark_large**
Speech to text & matching	16.5%	9.9%
Frames extraction	18.7%	10.6%
Eye gaze extraction	44.1%	25.3%
Frames selection	0.2%	0.1%
Object Detection	3.7%	3.8%
Object Segmentation	16.8%	50.2%

Table 5.3 presents the execution time of each pipeline stage as a percentage of the total, using the full eye gaze extraction processing time in both cases.

The results highlight that eye gaze extraction constitutes a significant portion of the overall pipeline on a relative basis, particularly in `benchmark_small`, where it accounts for nearly half of the total execution time (44.1%). Unfortunately, this stage offers limited room for optimization with current methods. While model quantization, applied to the Meta-provided model, could be a potential avenue for acceleration, it must be approached carefully to avoid unacceptable degradation in gaze estimation accuracy.

Another notable bottleneck is the frames extraction stage, which contributes 18.7% and 10.6% of the total time for `benchmark_small` and `benchmark_large`, respectively. Currently, frames are extracted at the user-defined reduced frame rate upfront. However, this process could be optimized by deferring frame extraction until after the filtering and selection phases, which operate solely on data coming from other sensors and does not require access to visual frames.

In the case of `benchmark_large`, object segmentation becomes the dominant stage (50.2%), largely due to the use of a more complex model. As discussed in Section 5.1.2, a lighter-weight model may suffice for the task, suggesting an opportunity for significant speedup without compromising functionality.

Lastly, it is important to note that all reported timings include model loading. In practical deployments, these costs can be mitigated by pre-loading models into memory, thereby eliminating initialization overhead during inference, a standard optimization in real-world applications.

5.4 Annotations Review

To better understand the sources of errors and evaluate the reliability of the ground truth, a qualitative review of the annotations was conducted. This section examines both correct and incorrect cases, focusing on detection and segmentation outcomes, with the aim of identifying patterns related to model performance, ambiguity in object labeling, and limitations in the input data or pipeline design choices. The results presented are drawn from `benchmark_large`.

5.4.1 Correct Annotations

To provide context for interpreting the reported IoU values, Figure 5.2 illustrates an example of a high-quality prediction for both object detection and segmentation. Despite nearly perfect alignment between the predicted (blue) and ground truth (red) annotations, small discrepancies, particularly along object edges with low contrast or ambiguous boundaries, result in IoU scores of 92.41% (bounding box) and 91.77% (segmentation mask), respectively.

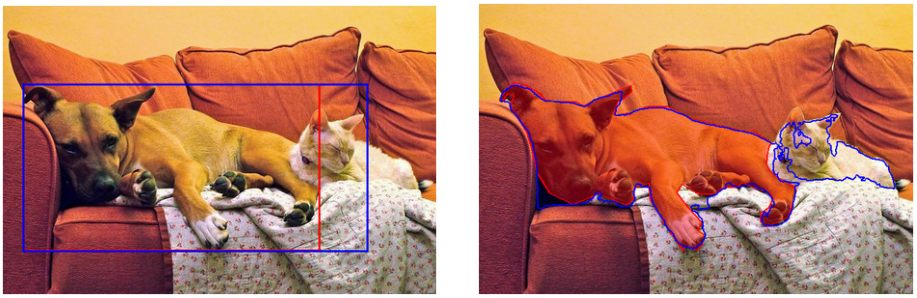
These minor differences often stem from subjective annotation choices and interpretations rather than actual model errors. This highlights the sensitivity of IoU to pixel-level variations, even when the prediction is visually accurate. As such, IoU values above 90% should be considered great, reflecting strong agreement rather than meaningful inaccuracies.

(a) Bounding boxes IoU: 92.41%. (b) Segmentation masks IoU: 91.77%.

Figure 5.2: Effect of small annotation differences on IoU (ground truth in red, prediction in blue).

Figure 5.3 demonstrates how inaccuracies in the object detection stage propagate into the segmentation results. In this example, the predicted bounding box for "dog" (blue) incorrectly includes part of an adjacent cat, resulting in a detection IoU of 85.20%. This spatial error is then carried forward: the segmentation mask, conditioned on the flawed bounding box, erroneously segments part of the cat as belonging to the dog, yielding a lower mask IoU of 76.91%.

This case highlights the strong dependency of segmentation quality on precise detection. Since segmentation models like SAM2 rely heavily on bounding box prompts to localize objects, even moderate detection errors can lead to significant mask inaccuracies. Consequently, improving detection accuracy through larger models, better training data, or refined cue integration, not only enhances detection performance but also boosts downstream segmentation results.

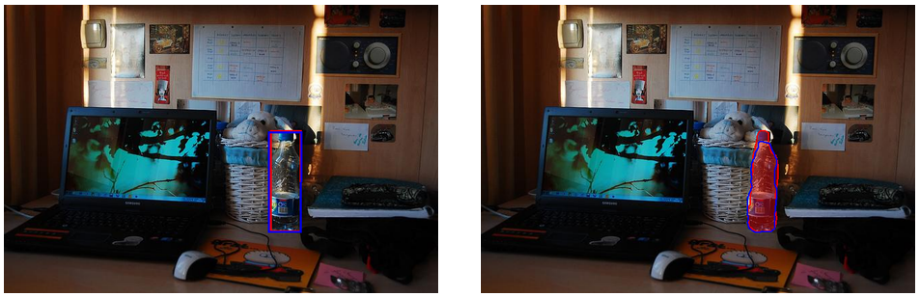


(a) Bounding boxes IoU: 85.20%. (b) Segmentation masks IoU: 76.91%.

Figure 5.3: Propagation of detection inaccuracies to segmentation quality (ground truth in red, prediction in blue).

Figure 5.4 shows a detection and segmentation result (blue) that closely matches the ground truth (red), with only minor pixel-level discrepancies along the object boundaries. Despite the high visual similarity, the IoU scores (89.28% for the bounding box and 84.98% for the segmentation mask) are lower than the visual agreement might suggest. This is due to the small absolute size of the object relative to the image: even a one or two pixel deviation, imperceptible to the human eye, represents a proportionally large overlap loss for tiny objects.

Such cases illustrate a well-known limitation of standard IoU as a performance metric: it treats all objects equally regardless of size, making it overly sensitive to minor errors on small instances. To address this, the weighted average IoU was introduced, which weights each object's IoU by its area relatively to the image's size. This reduces the disproportionate influence of small-object penalties on the overall score, leading to a more balanced and representative assessment of model performance.

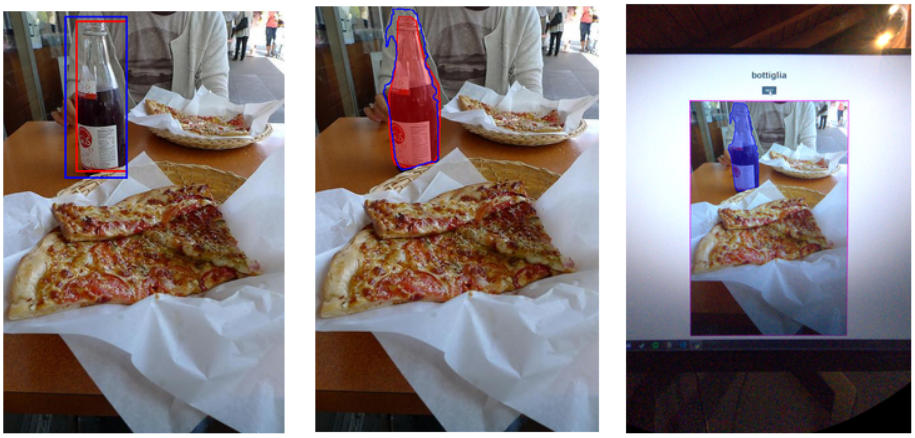


(a) Bounding boxes IoU: 89.28%. (b) Segmentation masks IoU: 84.98%.

Figure 5.4: Example of a small annotation with high similarity to the ground truth (ground truth in red, prediction in blue).

Figure 5.5 illustrates how low contrast along object boundaries, particularly at the top of the bottle in this case, makes precise detection and segmentation challenging. In such cases, both the model and human annotators may struggle to define the exact extent of the object, leading to slight misalignments between predicted (blue) and ground truth (red) annotations.

This uncertainty has a pronounced effect on the IoU, especially for smaller objects like the bottle in this example. Even minor deviations at the edges can significantly reduce the IoU score, despite the overall prediction being visually accurate.



(a) Bounding boxes Intersection over Union (IoU): 75.85%.

(b) Segmentation masks Intersection over Union (IoU): 83.74%.

(c) Processed image after applying the perspective correction.

Figure 5.5: Annotation inaccuracies near object boundaries under low contrast conditions (ground truth in red, prediction in blue).

Figure 5.6 presents a compelling case where the predicted segmentation (blue) is semantically more accurate than the ground truth (red), yet achieves a lower than expected IoU score. In this example, the model correctly segments only the bird, excluding part of the branch it is on, reflecting a more precise understanding of the object boundary. In contrast, the ground truth mask incorrectly includes the branch in the segmentation.

Despite this improved semantic accuracy, the predicted mask is penalized by the IoU metric for differing from the reference annotation. This illustrates a critical limitation: IoU measures geometric overlap, not semantic correctness. As a result, predictions that are better than the ground truth, due to noise, ambiguity, or human error in labeling, are not rewarded and may even be scored lower.

This case underscores the importance of interpreting quantitative metrics critically and recognizing that high-quality models can outperform their training labels.

(a) Bounding boxes IoU: 93.88%. (b) Segmentation masks IoU: 90.12%.

Figure 5.6: Case illustrating how segmentation can surpass the quality of the provided ground truth annotation (ground truth in red, prediction in blue).

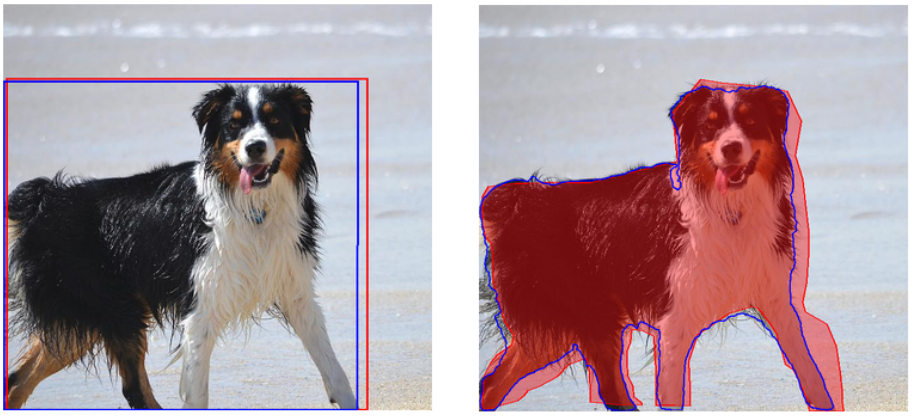
Figure 5.7 shows a case where the predicted bounding box (blue) includes the monitor's stand as part of the detected object, while the ground truth annotation (red) focuses only on the screen area. This difference reflects a divergent interpretation of what constitutes the *tv* object. Both are semantically reasonable, as the stand is physically attached to the monitor and may be perceived as part of the whole.

However, due to this discrepancy in object extent, the bounding box IoU is reduced to 75.10%, despite the detection being spatially well-aligned and contextually justified. This illustrates how IoU can be sensitive not only to localization errors but also to subjective differences in annotation scope. When models and human labelers interpret object boundaries differently, even reasonably so, the metric may penalize valid predictions, leading to an underestimation of detection quality.

Figure 5.7: Example of a predicted annotation that interprets the object differently from the ground truth (red: ground truth, blue: prediction). Bounding boxes IoU: 75.10%.

Figure 5.8 presents a case where the model's predicted segmentation (blue) is far more accurate than the provided ground truth annotation (red). While the bounding box alignment is excellent (IoU = 95.84%), the segmentation mask achieves a lower IoU of 86.58% due to a visible discrepancy between the two masks.

Crucially, in this instance, the model correctly captures a tighter, more precise outline of the object, whereas the ground truth appears to be sloppy, likely due to imprecise manual labeling. Despite being visually and semantically superior, the prediction is penalized by the IoU metric simply because it differs from the reference.



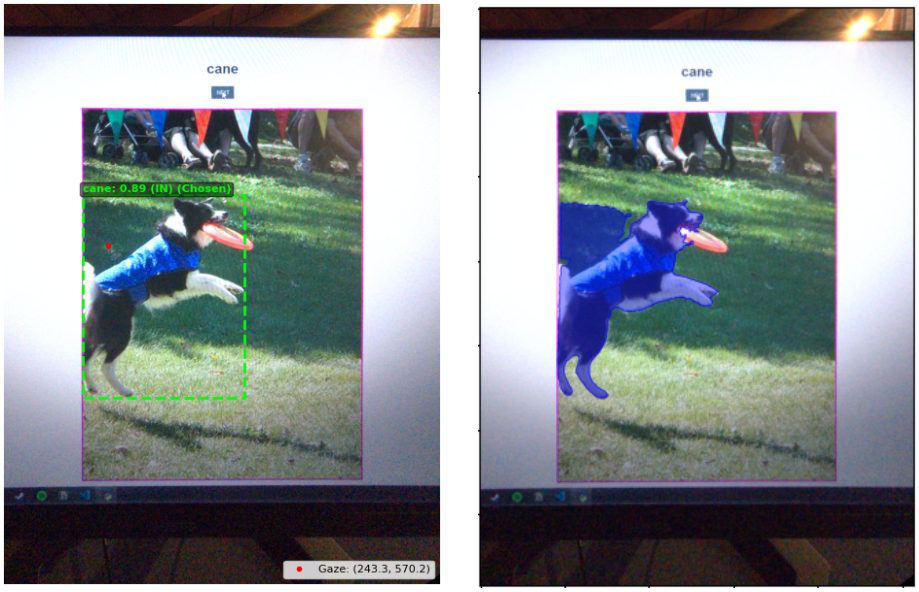
(a) Bounding boxes IoU: 95.84%. (b) Segmentation masks IoU: 86.58%.

Figure 5.8: Example where the ground truth segmentation is inaccurate, while the model prediction provides a more precise result (ground truth in red, prediction in blue).

Figure 5.9 demonstrates how eye gaze information, when used as a spatial prompt for segmentation, can inadvertently degrade mask quality if it does not precisely align with the target object. In this case, the detected bounding box for *dog* is accurate (Figure 5.9a), and the gaze point falls within it, satisfying the condition to also give to as prompt for the segmentation model, together with the bounding box. However, the gaze is located near the edge of the dog and overlaps with the adjacent grassy background.

As a result, the segmentation model (SAM2), conditioned on both the bounding box and the gaze location, interprets the point as indicating a region of interest and includes a portion of the background in the final mask (Figure 5.9b). This leads to an imprecise and semantically incorrect segmentation, despite a correct detection.

To address this issue, future improvements could involve running two parallel segmentation passes: one using both the bounding box and eye gaze as input, and another using only the bounding box. The confidence scores or mask quality outputs from both predictions could then be compared to determine the more reliable result. If the gaze-guided segmentation yields significantly lower confidence or coherence, the system could default to the gaze-agnostic version. This dual-path approach would preserve the benefits of gaze guidance when it improves accuracy, while reducing the risk of errors caused by misleading gaze points.



(a) Predicted bounding box and eye gaze. (b) Predicted segmentation mask.

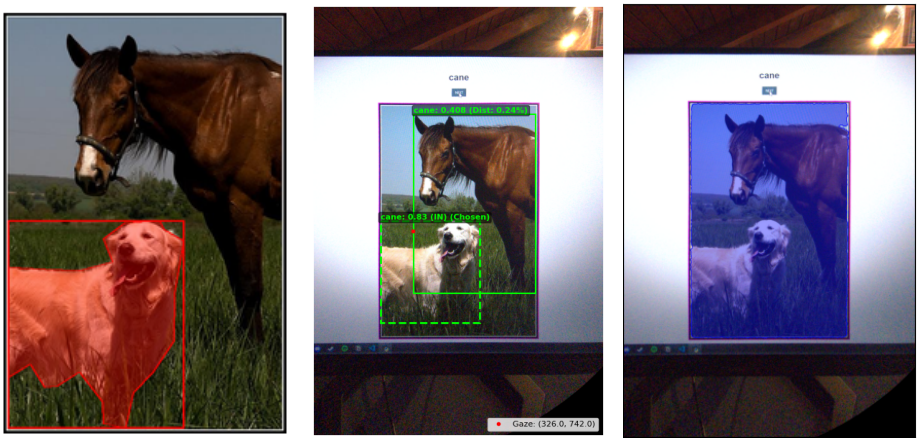
Figure 5.9: Example illustrating how an inaccurate eye gaze prediction can lead to an incorrect segmentation (ground truth in red, prediction in blue).

5.4.2 Incorrect Annotations

Figure 5.10 illustrates a case where two distinct objects, both labeled as *dog*, are incorrectly merged into a single bounding box due to significant spatial overlap (Figure 5.10b). While one of the individual detections was correct (and would've been chosen since it has a much higher confidence score), the post-processing stage that unites overlapping boxes applied merging logic inappropriately. This erroneous union directly propagates to the segmentation stage, resulting in a single, fused mask that segments the entire image (Figure 5.10c).

This error reveals a critical limitation in the current bounding box merging strategy: while merging may be beneficial for object categories that naturally appear in close proximity and are often interpreted as a group (e.g., *shoes*), it is inappropriate for instances like *dog*, where each detection should represent a separate entity, even when close together.

To address this, the merging logic should be made both class-aware and confidence-sensitive. Specifically, users should be able to define, via configuration, which object classes the merging rule applies to. Additionally, the merging decision could incorporate the detection confidence scores: only overlapping boxes with high confidence levels should be considered for union, reducing the risk of combining detections in ambiguous or complex scenes.



(a) Dataset's image and its ground truth.

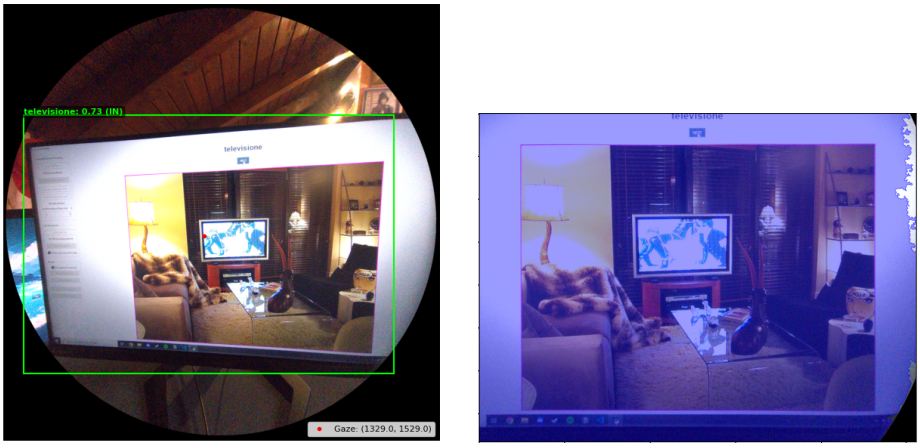
(b) Predicted object detection

(c) Predicted segmentation mask

Figure 5.10: Case illustrating how overlapping bounding boxes were wrongly united, causing the subsequent segmentation to be inaccurate.

Figure 5.11 illustrates a distinctive type of error in which the system detects and segments the physical monitor used to run the benchmark (Figure 5.11a) instead of the monitor depicted within the image content displayed on screen (Figure 5.11b).

This error is specific to the benchmarking setup, arising only when real-world display hardware overlaps semantically with on-screen content. In practical applications, such confusion would not occur, as the system would not be processing video of a monitor displaying similar content. Nevertheless, this case serves as a valuable illustration of how environmental context can influence perception in vision systems.



(a) Detection on the wrong object

(b) Segmentation on the wrong object

Figure 5.11: Example where the system detects and segments the physical monitor running the benchmark instead of the monitor displayed within the screen content.

Figure 5.12 shows a case where the system detects the wrong instance of a *bottle*, specifically the one on the right, while the intended target was the bottle on the left, closer to the user's eye gaze.

During dataset creation, images with multiple instances of the same class were intentionally excluded to minimize ambiguity and ensure clear correspondence between speech, gaze, and the target object. However, in this case, the second bottle-shaped object was either overlooked or not considered a valid instance during ground-truth annotation, allowing it to remain as a confounding element.

Figure 5.12: Example where the system detects the wrong instance of a *bottle*.

Figure 5.13 shows a failure case where the target object is not successfully segmented due to external visual disturbances, specifically glare and reflections on the screen surface. These artifacts significantly reduce the contrast and visibility of the content, making the object appear poorly defined to the camera.

While such issues are inherent to camera-based capture of screen content, potential mitigations include using anti-reflective screens, controlling lighting conditions, or applying pre-processing techniques to enhance contrast.

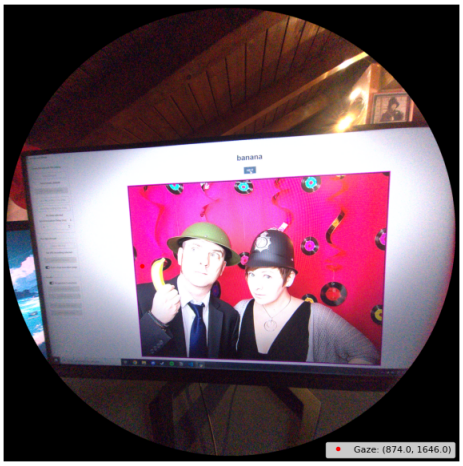
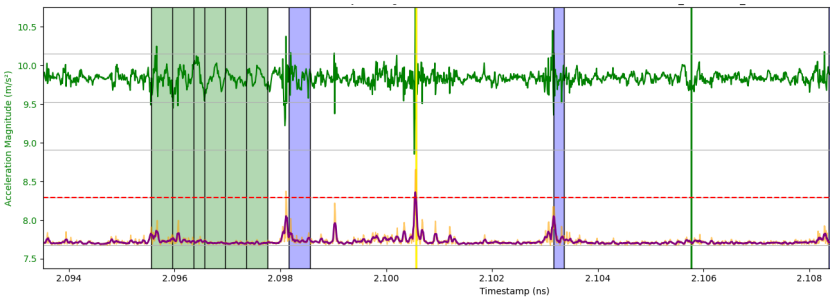


Figure 5.13: Example where glare and reflections reduce contrast in the monitor display, preventing the object (*banana*) from being detected.

Figure 5.14 illustrates a case where the current frame (highlighted with a vertical yellow line in Figure 5.14b) is excluded from processing because the smoothed jerk value slightly exceeds the user-defined threshold (dashed red line). As shown in the debug overlay (Figure 5.14a), this causes the frame to be discarded during the filtering stage, preventing downstream stages.

Visually, the frame does not appear degraded, suggesting that the jerk threshold may be set too conservatively. While the intent of the filter is to remove frames captured during rapid or unstable motion, in this case it eliminates a usable frame due to a minor threshold violation. This issue is exacerbated by the artificially low frame rate of the benchmark recording (originally *5 FPS*, reduced to *3FPS* when extracting frames). In normal operation with higher FPS, multiple frames would typically surround each keyword event, increasing the likelihood that at least one passes the filter. However, in this scenario, only a single frame was associated with the matched keyword and it was rejected, resulting in a complete processing failure for that instance.

(a) Filtering debug image showing the frame number and which filtering criteria were violated (red text at the bottom).



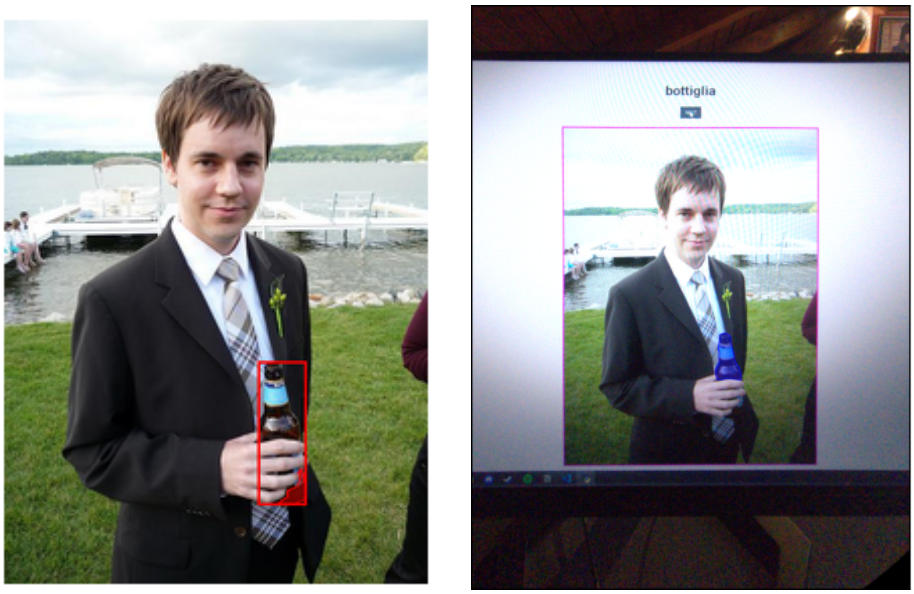
(b) Plot showing the acceleration (green) and smoothed jerk (purple) filtering data over time, the user-defined threshold (dashed red line), and the matched keyword intervals (shaded green/blue areas) with their corresponding frames (vertical black lines). The current image is shown as a vertical yellow line.

Figure 5.14: Example where the smoothed jerk exceeds the threshold.

Figure 5.15 reveals a case where the ground truth segmentation (Figure 5.15a) contains a clear annotation error: only the bottom portion of the bottle is masked, while the main, clearly visible body of the bottle is entirely omitted, indicating a mistake during manual labeling.

In contrast, the model's predicted mask (Figure 5.15b) captures the upper and central part of the bottle accurately, reflecting the most prominent and visually distinct region, while missing the lower portion. While not perfect, this prediction is semantically more reasonable and aligns better with the actual object extent.

Due to the lack of overlap between the two masks, this case is scored as a complete segmentation failure despite the model producing a more accurate result than the ground truth.



(a) Ground truth annotation. (b) Predicted segmentation mask.

Figure 5.15: Example illustrating an error in the ground truth segmentation, compared with the model's predicted mask.

Figure 5.16 illustrates a case where two instances of the same predicted object class fall within the eye gaze distance threshold, leading the system to consider both candidates and select the wrong detection as the target.

This ambiguity arises primarily due to the absence of in-session eye gaze calibration, which would allow for more precise alignment between gaze coordinates and screen content. Without calibration, the system must rely on a looser, more conservative distance threshold to ensure valid matches are not missed, increasing the likelihood of including false positives when multiple objects are present.

In this case, however, the threshold appears to be set too generously, allowing a clearly non-target object to be considered a valid candidate. While the dataset was designed to include only one instance of each target class per image, some images may include objects that are semantically similar or loosely belong to the

same category. These borderline cases can both appear plausible to the model and fall within the gaze proximity threshold, creating ambiguity in the selection of the correct bounding box.

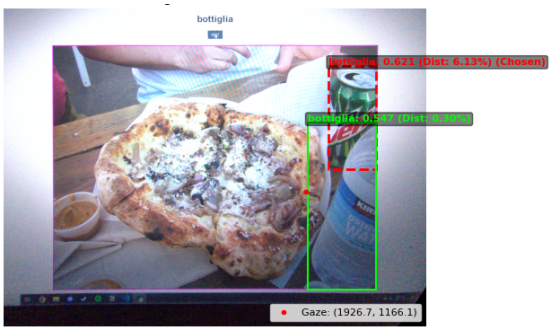


Figure 5.16: Illustration of an error caused by selecting the wrong candidate among two detections within the eye gaze distance threshold.

Chapter 6

Future Work

The current pipeline establishes a robust foundation for multimodal, egocentric data processing by tightly integrating speech, gaze, and vision to enable automatic object annotation in real-world scenarios. However, several avenues for improvement and expansion remain unexplored. This section outlines both specific enhancements to the system's performance and functionality, as well as broader, more visionary directions.

6.1 Performance and Functional Improvements

A crucial next step toward real-world applicability is evaluating and optimizing the pipeline for execution on resource-constrained, embedded hardware such as smart glasses, smartphones, or edge AI accelerators. This includes testing further model quantization, pruning, and distillation techniques to reduce memory footprint and latency while preserving accuracy. The models could also be compiled to accelerate them for specific hardware. By benchmarking end-to-end inference times and power consumption across different device tiers, it would be possible to determine how close the system is to real-time, on-glasses operation.

One of the most promising technical upgrades is the reintegration and refinement of video object segmentation using SAM2's temporal propagation architecture. While earlier attempts using gaze-only prompts proved unreliable due to misalignment and prompt overwriting behavior, future work could leverage the object detection stage to generate high-quality bounding boxes as prompt. By using these robust prompts at keyframes, either the first frame of a speech interval or at regular intervals, the system could achieve stable, long-term tracking of objects across video sequences. This would not only improve segmentation consistency but also reduce redundant computation by propagating masks over time rather than processing each frame independently. However, this still comes at a high computational cost, though the resulting accuracy could provide a strong foundation for the creation of temporally annotated datasets as well.

Furthermore, the pipeline could benefit from optimizing the current eye gaze estimation to run faster and more accurately. This could involve further quantization of the local model, improving efficiency while maintaining precision, and potentially extending it with depth estimation capabilities. If possible, the model could also be used to get the eye gaze data only within matched keyword

intervals, which would significantly reduce the computational cost. Additionally, if Meta were to release the model currently available only through the MPS API, it could be directly integrated into the system to achieve higher accuracy (through the in-session calibration) and to get the depth estimation data without relying on remote inference.

Given the high modularity of the pipeline, newer models for open-vocabulary object detection or segmentation could be seamlessly integrated as they become available, leading to progressively better results without major architectural changes. For the same reason, the pipeline could be adapted and used as the first stage for other tasks beyond annotation, such as activity recognition, where robust object tracking and gaze-informed segmentation could provide valuable insights.

Another promising avenue is to refine the integration of gaze information. While gaze-guided segmentation has clear benefits, it can also introduce errors when the point lies near object boundaries or overlaps with the background (Figure 5.9). Future systems could adopt a dual-path strategy, running segmentation both with and without gaze conditioning, then selecting the most coherent output based on confidence scores or mask quality heuristics.

The pipeline could also be enhanced through smarter handling of overlapping detections. Current merging strategies sometimes inappropriately fuse multiple objects into a single bounding box (Figure 5.10), propagating errors into the segmentation stage. A more adaptive, class-aware merging policy could be developed, where merging is applied selectively to categories that naturally appear in groups, and combined with confidence thresholds to prevent spurious unions. The system could allow users to choose the categories for which this logic should be applied. This would increase robustness in crowded or ambiguous visual scenes.

Another direction for future work is the integration of advanced frame pre-processing techniques to enhance visual quality before detection and segmentation. Methods such as contrast normalization, histogram equalization, denoising, or low-light enhancement could significantly reduce issues arising from poor illumination or low-contrast conditions. Incorporating these steps directly into the pipeline would not only improve the reliability of object detection and segmentation but also make the system more robust to variations in environmental lighting and recording quality.

Future improvements could also lie in moving beyond fixed model configurations toward adaptive, context-sensitive pipelines. As demonstrated by the disproportionate runtime increase in segmentation with minimal accuracy gains, uniformly scaling up models is inefficient. Instead, the system could employ a dynamic routing strategy, where lightweight models handle most inputs, and larger, more accurate models are invoked only when needed, such as when detection confidence is low or the frame's quality isn't high, for example due to poor contrast or excessive brightness (overexposure).

6.2 Project Aria Gen 2

While this project is built using the first-generation Project Aria glasses, the recently announced Project Aria Gen 2 introduces several advancements that could significantly enhance the scalability, responsiveness, and functionality

of future versions of this pipeline. The advancements open the door to more dynamic, context-aware applications and pave the way for deploying intelligent, user-adaptive systems in real-world environments beyond the constraints of tethered or offline setups.

Longer Battery Life One of the most significant improvements in Project Aria Gen 2 [24] is its substantially extended battery life compared to the first-generation device. This enhancement enables longer, uninterrupted recording sessions, critical for capturing natural, real-world interactions. The increased power capacity supports continuous operation of high-bandwidth sensors (including RGB cameras, eye trackers, and microphones) without frequent recharging, making the device far more practical for real-world deployments. This advancement also opens opportunities for always-on context awareness, where the system can passively monitor and process multimodal inputs over time.

On-Device Location, Hand and Eye Tracking, and Automatic Speech Recognition Project Aria Gen 2 integrates advanced AI capabilities that enable real-time processing of location, hand gestures, eye gaze, and speech, all computed locally on the glasses. This on-device intelligence reduces reliance on cloud connectivity, ensuring low-latency responses, improved privacy, and consistent performance even in offline environments. Performing these operations directly on the device also frees up external computational resources, allowing more processing power to be dedicated to demanding tasks such as detection or segmentation.

Open Air Speakers The inclusion of open-air speakers enables discreet, spatially aware audio feedback without blocking ambient sound, a crucial feature for maintaining situational awareness in dynamic environments. This capability supports hands-free, eyes-forward interaction, where the system can audibly respond to user queries (e.g. by saying "That's a coffee mug") or alert them to relevant objects or events in their field of view. For assistive applications, such as supporting individuals with visual impairments, these speakers could provide continuous auditory descriptions of the visual world, synchronized with gaze and speech. In general, they represent a way to provide real-time, contextually relevant feedback to the user.

Applications Can Be Deployed On-Device Project Aria Gen 2 transforms from a passive data collection platform into a fully programmable wearable computer by supporting the installation and execution of custom applications directly on the device. This shift enables developers to deploy real-time, interactive experiences such as live object labeling, gaze-driven navigation or environment description, without requiring constant data offloading or cloud processing. Applications can respond immediately to multimodal inputs, enabling responsive, low-latency user experiences. This on-device app ecosystem also enhances privacy, as sensitive data can be processed locally. It paves the way for personalized, user-tailored AIassistants that run natively on the glasses, adapting to individual needs and behaviors over time.

Cloud Streaming Unlike the first-generation Project Aria glasses, which only supported local streaming to a paired mobile device or nearby PC via USB or Wi-Fi, Project Aria Gen 2 introduces true cloud streaming capabilities, enabling seamless, long-range data transmission over the internet. This shift enables new use cases, including real-time remote assistance, live debugging of AI pipelines, and centralized aggregation of multimodal data across distributed users for collaborative research or model training.

PPG Sensors The integration of Photoplethysmography (PPG) sensors adds a physiological dimension to egocentric sensing, enabling the measurement of heart rate, Heart Rate Variability (HRV), and potentially other biometrics such as stress levels or cognitive load. When combined with gaze, speech, and visual context, this physiological data can provide deeper insights into user state, for instance detecting moments of fatigue, confusion, focus, or emotional response during object interaction. This multimodal fusion could enable adaptive systems that respond not only to what the user sees and says but also to how they feel, opening new frontiers in personalized, context-aware computing.

6.3 Broader Vision and System Extensions

Beyond performance gains, the pipeline opens the door to transformative applications in human-computer interaction and intelligent assistance. One such vision is the development of a context-aware egocentric assistant that leverages speech and gaze to understand user intent in real time. For example, when a user glances at a kitchen appliance and says, "How does this work?", the system could recognize the object, retrieve relevant information, and deliver a contextualized response via audio or augmented reality overlay. By combining multimodal grounding with retrieval-augmented language models, such a system could act as a personalized, hands-free guide in complex environments like laboratories, workshops, or medical settings.

Building on the current annotation pipeline, Project Aria Gen 2 could enable a new paradigm of "in-the-wild" dataset creation, where users naturally label their world through speech and gaze during everyday activities. Instead of manually annotating thousands of images post-hoc, users could generate high-quality annotations simply by living, saying "this is my laptop" while looking at it, or "I'm making coffee" while performing the task. These annotations could be used to train personalized models or contribute to open datasets, with privacy-preserving aggregation. This turns passive recording into active labeling, drastically reducing the cost and effort of dataset creation while capturing real-world diversity. This is technically already possible with the current pipeline, but the improvements in battery life, on-device processing, and cloud streaming in Project Aria Gen 2 would make it far more practical and scalable. An interpretation stage could also be added to the pipeline, where a large language model or a simple word-embedding based retrieval system could be used to match user utterances to known object categories even when the exact term is not used (e.g. "laptop" vs "notebook" or "computer").

By continuously learning from a user's gaze, speech, and behavior, the system could inform adaptive interfaces in smart homes, vehicles, or workplaces. For example, if a user frequently looks at a thermostat and says, "It's cold," the

system could learn to adjust the temperature automatically. Or, in a car, it could detect when a driver glances at the glove compartment and says "gloves," then suggest retrieving them, demonstrating a deeper understanding of intent. Over time, the environment itself could become anticipatory, responding not to explicit commands but to subtle, multimodal cues embedded in natural behavior.

With PPG sensors, extended battery life, and continuous multimodal sensing, the pipeline could be repurposed for long-term behavioral and mental health monitoring. Subtle changes in gaze patterns (e.g., reduced visual exploration), speech (e.g., slower pace, fewer utterances), or activity routines could serve as early indicators of depression, anxiety, or cognitive decline. When combined with physiological data, the system could provide clinicians with rich, objective insights into a patient's daily functioning, far beyond what periodic assessments can capture.

Building upon object detection, gaze tracking, and speech understanding, the pipeline can be extended to perform fine-grained activity recognition by modeling temporal sequences of multimodal events. Instead of merely identifying what objects are present, the system could infer higher-level actions, such as "making coffee," "reading a book," or "fixing a bike", by analyzing patterns in gaze transitions, hand-object interactions, spoken commentary, and object co-occurrence over time. For example, prolonged gaze on a kettle followed by hand motion toward a mug and the utterance "time for tea" would be fused into a coherent activity label. Over time, the system could learn personalized activity models, adapting to individual routines and providing contextual support, such as reminding a user they forgot to turn off the stove or suggesting a break after prolonged focus. This transforms the glasses from a passive recorder into an active observer of human behavior, unlocking rich semantic understanding of daily life.

Another promising direction for scalable, multimodal dataset creation lies in leveraging immersive 360° video recordings annotated through Virtual Reality (VR) headsets equipped with eye tracking. In this paradigm, users don't record their own environment passively. Instead, they enter curated or crowd-sourced 360° scenes captured in real-world settings and actively label them as if they were physically present. Using VR headsets with high-fidelity eye tracking, multiple annotators can collaboratively gaze at objects, utter natural speech (e.g. "That's a coffee mug"), or even gesture to define regions of interest, all while the system synchronously records their visual attention, vocal input, and temporal interaction patterns. This approach enables not only multi-user contribution, allowing diverse perspectives to refine annotations, but also real-time feedback and iterative refinement: an annotator might initially label an object as "cup," then correct it to "mug" upon closer inspection. The user could also receive real-time semantic segmentations overlaid on the 360° scene, which they can interactively refine through gaze-guided corrections, verbal feedback ("that's the table, not the shelf"), or manual adjustments. As users navigate through the scene, their iterative edits accumulate into coherent, temporally consistent annotations, effectively encoding not just what an object is, but how its identity and context evolve across time.

Conclusion

The multimodal pipeline developed successfully demonstrates that high-quality, annotated egocentric datasets can be generated automatically through natural human interaction. This capability fundamentally transforms the paradigm of dataset creation: instead of requiring hours of labor-intensive, frame-by-frame manual annotation, the system leverages everyday user behavior as a source of supervision, enabling the rapid, passive collection of richly annotated data at scale.

Evaluated on a controlled benchmark of 100 images from COCO, the system achieved an overall object detection accuracy of 89% with a size-weighted IoU of 92.33%, indicating highly precise spatial localization. Segmentation accuracy reached 83%, with a weighted IoU of 85.75%, showcasing the ability to generate pixel-accurate masks, with the majority of errors stemming not from model failure but from ambiguities in the input prompts, particularly when gaze was misaligned or object boundaries were visually indistinct due to screen glare or low contrast.

The stability of segmentation errors across model scales suggests that further improvements may require not just larger models, but better integration strategies such as dual-path segmentation or confidence-based bounding box fusion. Moreover, qualitative analysis revealed instances where the predicted segmentation masks surpassed the quality of the ground truth annotations, particularly in cases of ambiguous or imprecise manual labeling. For this reason, the quantitative metrics reported may underestimate the true performance of the pipeline.

The resulting pipeline is not only accurate but also highly configurable, allowing users to balance speed, precision, and computational load by selecting model sizes and adjusting parameters through an intuitive graphical interface. This modularity ensures adaptability across diverse use cases, from rapid predictions with lightweight models to high-fidelity annotation using larger, more precise versions.

The custom benchmarking framework developed in this work goes beyond conventional performance evaluation by introducing a structured, diagnostic-driven analysis of system failures. Rather than relying solely on aggregate metrics like accuracy or mean Intersection over Union (IoU), the evaluation system enables frame-by-frame error classification, allowing developers to trace failures back to specific stages, such as speech recognition inaccuracies, gaze misalignment, or object detection misses. This diagnostic capability transforms the pipeline from a static annotation tool into a dynamic debugging environment, facilitating targeted improvements and informed decision-making to fine-tune the pipeline's user-chosen parameters. Notably, the analysis revealed that while

a larger detection YOLO-World model significantly reduces missed detections, segmentation errors predominantly stem from the inherent limitations of the SAM2 model or the given prompts, particularly in complex scenes with occlusions or small objects.

Looking ahead, the pipeline lays the foundation for a new paradigm in continuous, lifelong dataset creation. Future extensions could incorporate on-device processing via Project Aria Gen 2 [24], enabling real-time, privacy-preserving annotation without reliance on cloud APIs. The framework could be extended to higher-level tasks such as activity recognition, where temporal modeling of gaze transitions, object interactions, and spoken language enables the inference of complex user intentions. Additionally, the inclusion of physiological signals like Photoplethysmography (PPG) opens avenues for affect-aware computing and behavioral monitoring. Ultimately, this work not only addresses the immediate challenge of dataset scarcity in egocentric AI but also envisions a future where intelligent systems learn continuously from natural, multimodal human behavior, transforming passive observation into active, contextual understanding.

References

[1] J. Engel et al., "Project aria: A new tool for egocentric multi-modal ai research," *arXiv preprint arXiv:2308.13561*, 2023.

[2] T. Cheng, L. Song, Y. Ge, W. Liu, X. Wang, and Y. Shan, "Yolo-world: Real-time open-vocabulary object detection," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2024, pp. 16 901–16 911.

[3] N. Ravi et al., "Sam 2: Segment anything in images and videos," *arXiv preprint arXiv:2408.00714*, 2024.

[4] M. Bain, J. Huh, T. Han, and A. Zisserman, "Whisperx: Time-accurate speech transcription of long-form audio," *arXiv preprint arXiv:2303.00747*, 2023.

[5] Meta Reality Labs Research, *Project aria docs: Project aria tools*, Accessed: 2025-07-08. [Online]. Available: `https://facebookresearch.github.io/projectaria_tools/docs/data_utilities`

[6] Meta Reality Labs Research, *Project aria docs: Aria machine perception services*, Accessed: 2025-07-08. [Online]. Available: `https://facebookresearch.github.io/projectaria_tools/docs/ARK/mps`

[7] Meta Reality Labs Research, *Project aria docs: Open datasets*, Accessed: 2025-07-08. [Online]. Available: `https://facebookresearch.github.io/projectaria_tools/docs/open_datasets`

[8] Meta Reality Labs Research, *Project aria docs: Open models*, Accessed: 2025-07-08. [Online]. Available: `https://facebookresearch.github.io/projectaria_tools/docs/open_models`

[9] Meta Reality Labs Research, *Project aria eye tracking pre march 2024 model*, Accessed: 2025-07-08. [Online]. Available: `https://github.com/facebookresearch/projectaria_eyetracking`

[10] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "Wav2vec 2.0: A framework for self-supervised learning of speech representations," *Advances in neural information processing systems*, vol. 33, pp. 12 449–12 460, 2020.

[11] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, "Robust speech recognition via large-scale weak supervision," in *International conference on machine learning*, PMLR, 2023, pp. 28 492–28 518.

[12] Ultralytics, *Ultralytics yolo*, Accessed: 2025-07-08. [Online]. Available: `https://docs.ultralytics.com/`

[13] OpenAI, *Openai clip*, Accessed: 2025-07-08. [Online]. Available: https://openai.com/index/clip/

[14] A. Kirillov et al., "Segment anything," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2023, pp. 4015–4026.

[15] A. Dosovitskiy et al., "An image is worth 16x16 words: Transformers for image recognition at scale," *arXiv preprint arXiv:2010.11929*, 2020.

[16] C. Ryali et al., "Hiera: A hierarchical vision transformer without the bells-and-whistles," in *International conference on machine learning*, PMLR, 2023, pp. 29 441–29 454.

[17] SYSTRAN, *Faster whisper*, Accessed: 2025-07-08. [Online]. Available: https://github.com/SYSTRAN/faster-whisper

[18] Meta Reality Labs Research, *Project aria docs: Image devignetting*, Accessed: 2025-07-08. [Online]. Available: https://facebookresearch.github.io/projectaria_tools/docs/data_utilities/advanced_code_snippets/image_utilities#image-devignetting

[19] *The mobile robot programming toolkit (mrpt) project*, Accessed: 2025-07-08. [Online]. Available: https://github.com/MRPT/mrpt

[20] Meta Reality Labs Research, *Project aria docs: Mps output - eye gaze*, Accessed: 2025-07-08. [Online]. Available: https://facebookresearch.github.io/projectaria_tools/docs/data_formats/mps/mps_eye_gaze

[21] Meta Reality Labs Research, *Project aria docs: In-session eye gaze calibration*, Accessed: 2025-07-08. [Online]. Available: https://facebookresearch.github.io/projectaria_tools/docs/ARK/mps/eye_gaze_calibration

[22] *Coco - common objects in context*, Accessed: 2025-07-08. [Online]. Available: https://cocodataset.org

[23] T. Lindeberg, "Scale invariant feature transform," 2012.

[24] Meta Reality Labs Research, *Project aria gen 2 glasses*, Accessed: 2025-07-08. [Online]. Available: https://www.projectaria.com/glasses/