

Gesture Recognition for Enabling Control of Electrical Devices – Proposed solution

Reviewed by Nick Gallo, Ph.D

DAVID T. TUNG

04/04/2025



Objective

Introduction of an ML Pipeline for Device Detection via Pointing Gestures

This presentation is divided into five sections:

Overview, Stage 1, Stage 2, Stage 3, and End-to-End Inference Flow



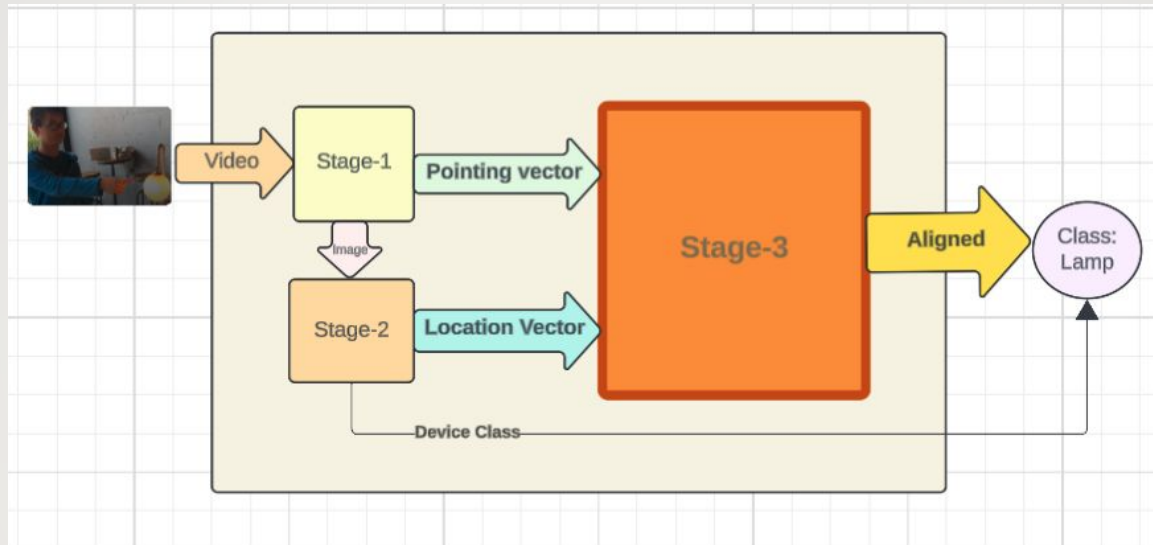
Image illustrates a system capable of identifying the target device based on the user's pointing gesture

Comparable real-world technology



Image demonstrates how pointing gestures enable intuitive, hands-free interaction with in-car devices. (Source: BMW Aftab 2020)

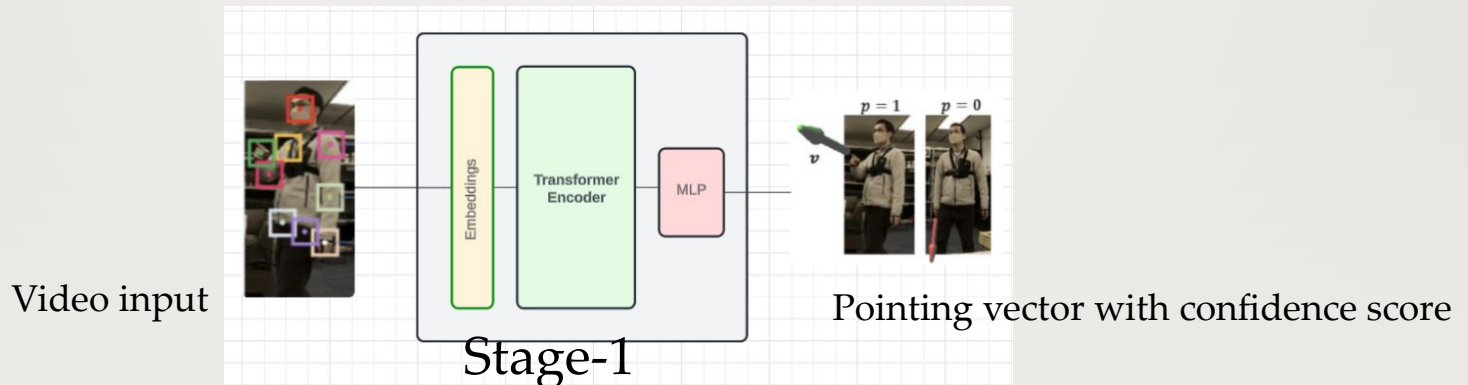
Overview: Pointing-to-Device End-to-End Flow



- Stage 1: Estimate pointing direction from video.
- Stage 2: Detect and localize devices in the scene.
- Stage 3: Determine alignment between pointing and device. (main contribution)
 - If aligned, return the corresponding class from Stage 2.

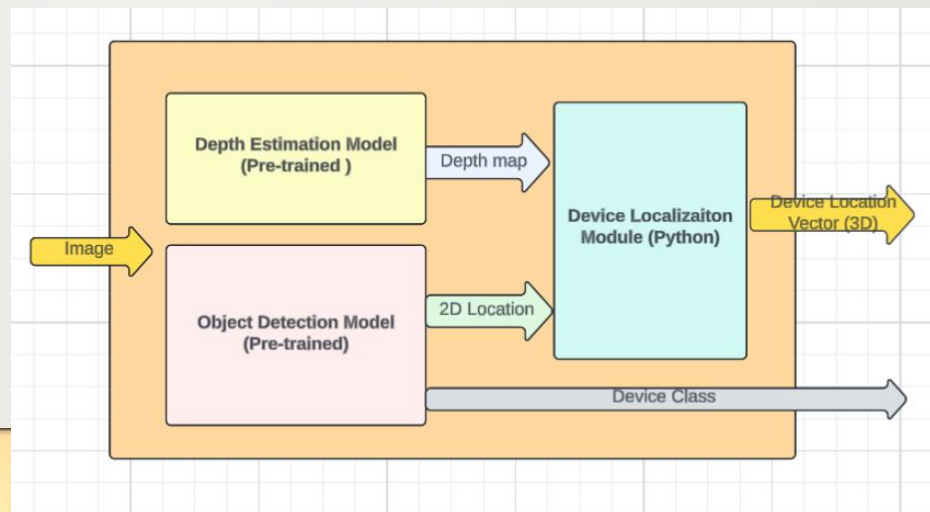
Stage-1: Estimate pointing direction

- Uses the 2023 DeePoint model. (without any modifications or fine-tuning)
- A transformer-based model estimates the user's pointing direction by tracking body, hand, and finger poses.
- It takes vidoes input and estimates a 3D pointing vector along with a confidence score.



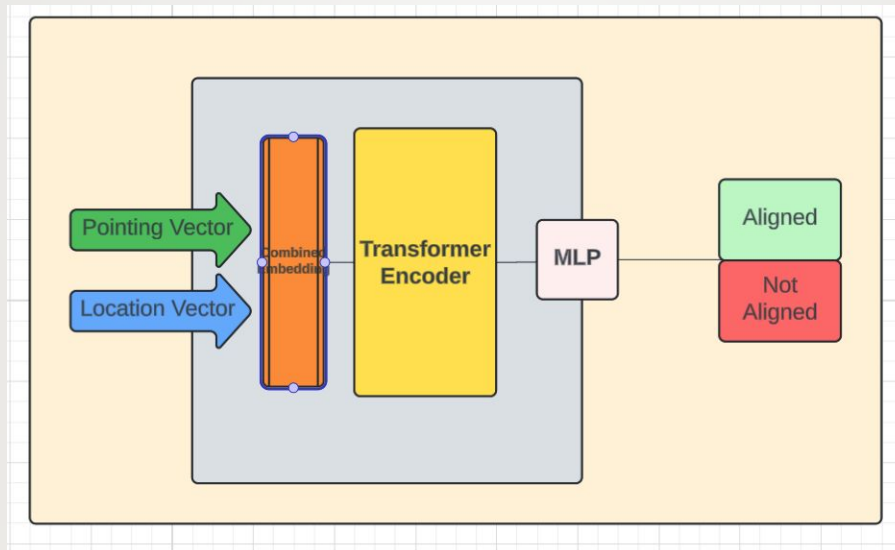
Stage-2: Device Detection & Localization

- Goal: Identify and localize the device user is pointing at.
- Inputs: A single RGB image.
- Models Used: Pre-trained object detection and monocular depth estimation.
- Output: 1. A 3D vector representing the spatial location of the detected device. 2. Device class.



Stage-3 Vectors Alignment

- Classify the alignment between pointing vector and device location vectors.
- The pointing vector and location vector are the inputs to a transformer which determines whether these vectors are aligned.

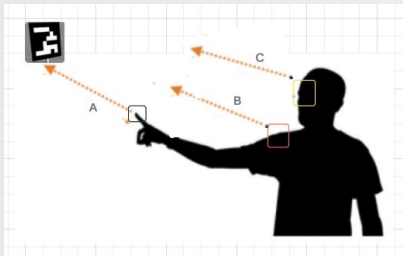


Stage-3 Train Data

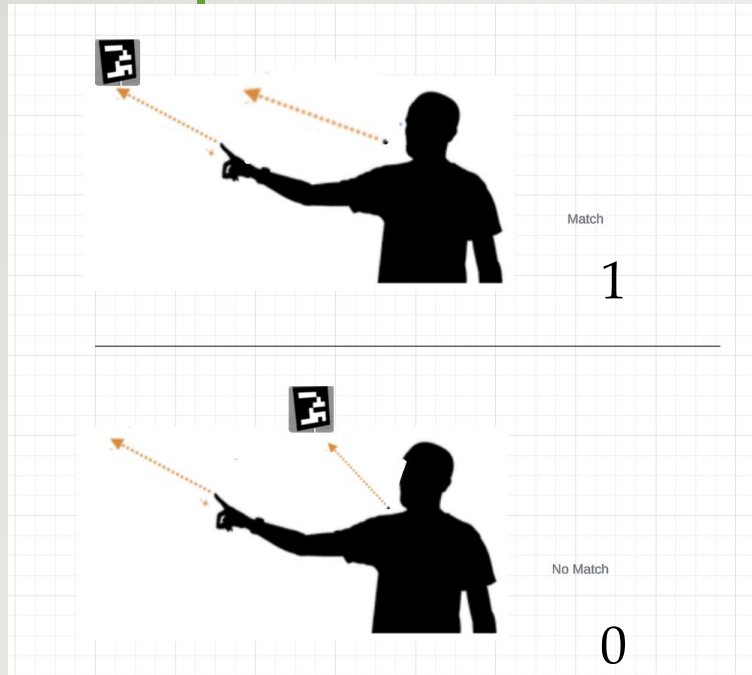
- Stage-3 took subset of DeePoint dataset for training, it is about 90k rows of data.
- The dataset is split into 80% training, 10% validation, and 10% testing.
- Output labels indicating whether the vectors correspond to a valid pointing gesture toward an object (aligned).

Ground Truth Vectors

- Three vectors need to be computed during training:
 - (A) 3D pointing direction vector, calculated as a vector from fingertip keypoint to marker.
 - (B) 3D object direction vector, calculated as vector from center (close to the shoulder) keypoint to marker.
 - (C) 3D gaze direction, calculated as vector from eyes keypoint to marker. [C is used for comparison purposes.]



Training Labels



- Stage-3 labels data as either 1 (aligned) or 0 (not aligned).
- Samples with valid pointing to the marker are labelled 1, else 0.

Stage-3 Training Data

Data Column	Data Column <u>Informaiton</u>
date	Datetime
site	Place, e.g. office, living room
frame	Image frame, e.g. 00000001.jpg
<u>pointing direction</u>	Pointing gesture direction vector
<u>device direction</u>	Device location direction vector
<u>gaze direction</u>	Gaze direction vector (for comparison)
<u>is aligned</u>	Are vectors aligned or not

Stage-3 Results: Transformer

Model	Type	Accuracy	Precision	Recall	F1	Loss
Transformer	Train (80%)	0.8012	0.78	0.82446	0.8017	4.2798
	Validation (10%)	0.7068	0.674	0.7554	0.7125	1.3059
	Test (10%)	0.725	0.688	0.79	0.738	1.294

- Data: 90k rows
- Epoch: 400

Stage-3 Results: MLP

Model	Type	Accuracy	Precision	Recall	F1	Loss
MLP	Train (80%)	0.78066	0.76204	0.8043	0.7826	4.30642
	Validation (10%)	0.64618	0.62445	0.69436	0.65755	4.38983
	Test (10%)	0.682	0.649	0.7554	0.69866	4.395

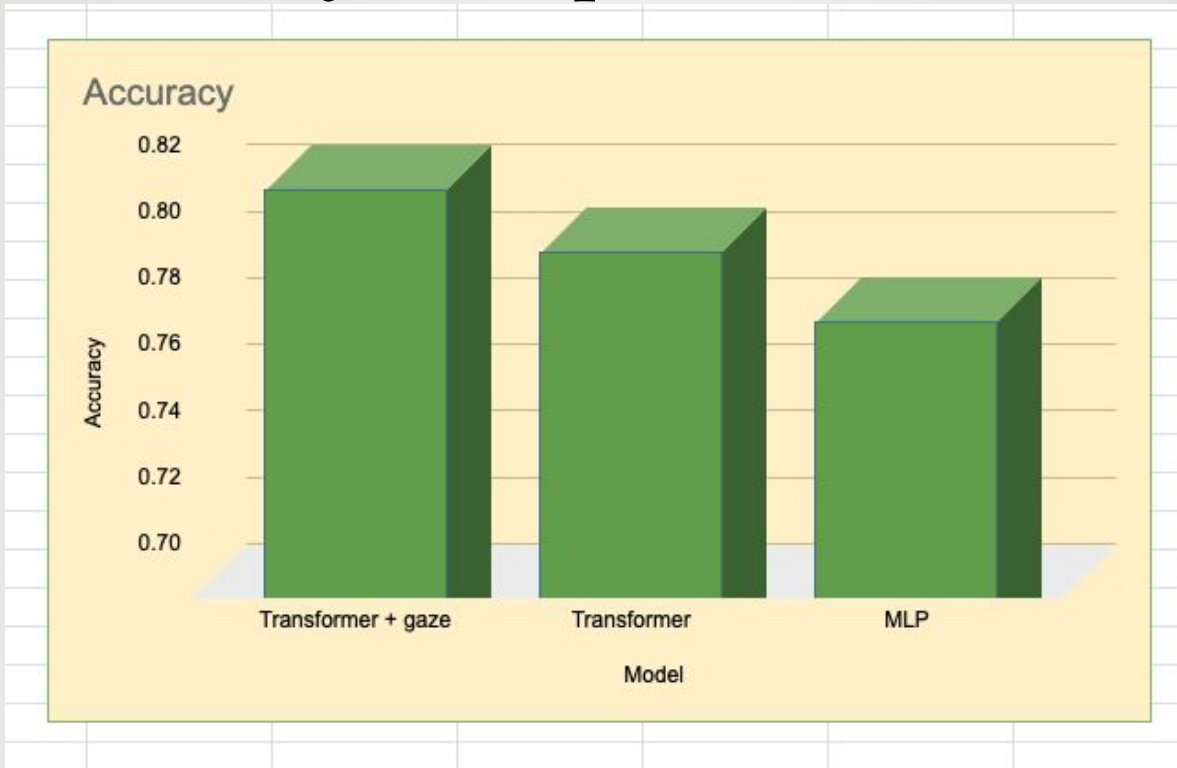
- Data: 90k rows
- Epoch: 400

Stage-3 Results: Transformer + Gaze

Model	Type	Accuracy	Precision	Recall	F1	Loss
Transformer + Gaze	Train (80%)	0.8102	0.81209	0.79795	0.80496	4.281
	Validation (10%)	0.70313	0.71223	0.7087	0.71024	1.6754
	Test (10%)	0.72175	0.70238	0.70917	0.71919	0.49797

- Data: 90k rows
- Epoch: 400

Accuracy Comparison



Takeaway: The Transformer model with gaze direction as an additional input achieved the highest accuracy.

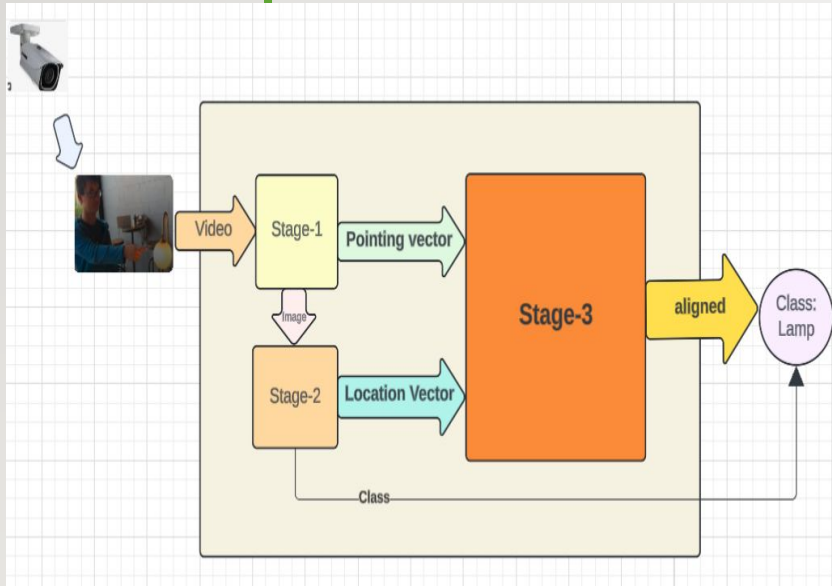
Summary for Stage-3

- Stage-3 training accuracy: 0.8012 and Loss: 1.2387 and result is reproducible.
- Directly learning from pointing vector and location vector is a valid approach that is close to multi-modal fusion research.
- Alternatives design for stage-3 include simpler geometric methods (ray intersection) or MLP based architecture or hybrid approaches.

End-to-End Inference

Integrate all three stages as one flow

End-to-end Inference



This diagram illustrates the pipeline connecting the three stages of the inference flow.

- Stage 1: Estimates the pointing direction.
- Stage 2: Detect and localize devices in the scene.
- Stage 3: Determine alignment between pointing and device.
 - If aligned, return the class from Stage 2.

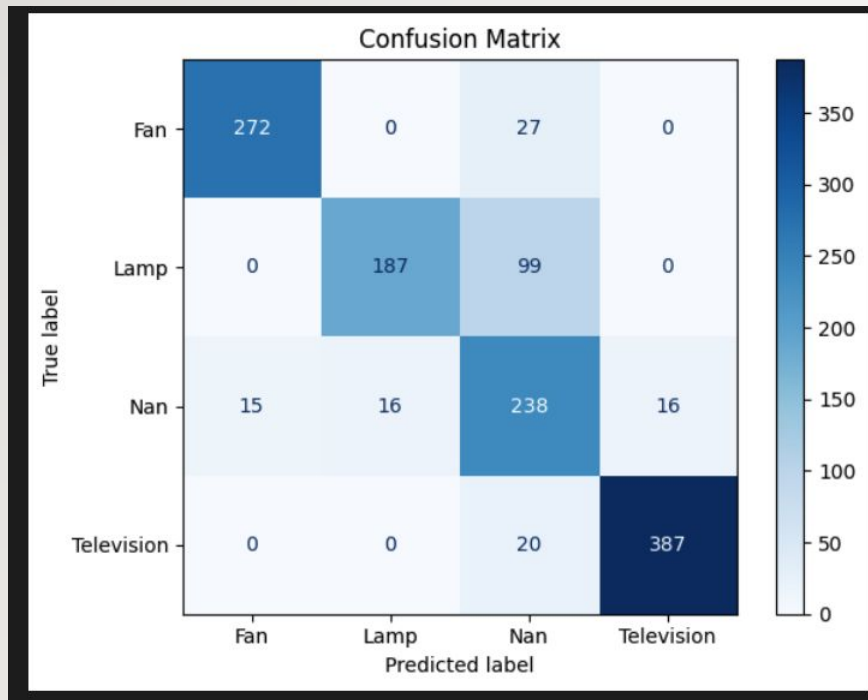
End-to-end Data Samples



- This is an example of video input for end-to-end system.
- AI tool, **superwork.ai** is used to generated videos for evaluation.



End-to-End results



Classification Report:

	precision	recall	f1-score	support
Fan	0.947735	0.909699	0.928328	299.000000
Lamp	0.921182	0.653846	0.764826	286.000000
Nan	0.619792	0.835088	0.711510	285.000000
Television	0.960298	0.950860	0.955556	407.000000
accuracy	0.848865	0.848865	0.848865	0.848865
macro avg	0.862252	0.837373	0.840055	1277.000000
weighted avg	0.872602	0.848865	0.851998	1277.000000

Overall Accuracy: 0.8489

The results indicate that the proposed classification system achieved an accuracy of 0.8489, demonstrating promising performance.

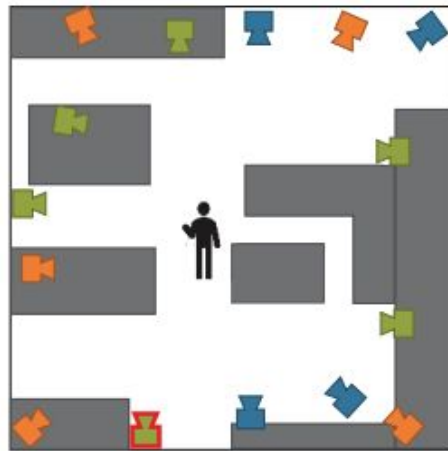
Edge Cases Are Not Covered

- Partial device occlusion
- Multiple device classes in one sample (multi-labels)
- Multiple persons with a pointing gesture
- Left hand is not covered

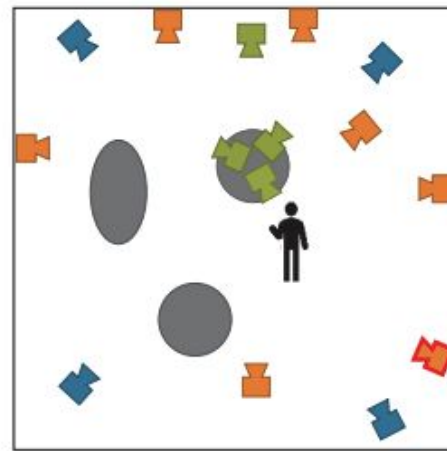


Backup

Stage-1 Training Data



(i) Living Room



(ii) Office

Figure 3. Camera layout of DP Dataset. We mount cameras at fixed viewpoints in the room to capture the pointing gestures from a variety of directions at once. The orange cameras are installed on the ceiling, the blue ones are put on the floor, and the green ones are installed on the mid-level. The gray objects depict tables, sofas, and obstacles.

Block Diagram of Pointing Objects Classification – Level 1

1. Collect data

1. Configure AWS cloud storage infrastructure (S3, EFS) to store the collected dataset.
2. Obtain the required video and image datasets for model training.

Tools: AWS, S3, EFS

2. Preprocess data

1. Process and clean the DeePoint datasets for model training.
2. Implement the dataset class to support efficient data loading during training.
3. Split the data into training, testing, and validation subsets. (80-10-10).

Tools: Python, Spark, S3, EC2, EFS

3. Perform pointing direction estimation. (Stage-1)

1. Prepare and configure DeePoint model for pointing direction estimation on AWS cloud
2. Test model estimation with video samples.

Tools: Python, PyTorch, AWS EC2

4. Perform device detection & localization. (Stage-2)

1. Prepare and configure pre-trained object detection and depth estimation models.
2. Develop python codes to compute 3D device location vector.

Tools: Python, PyTorch, EC2

5. Train model to classify pointing-device alignment (Stage-3)

1. Develop and train transformer model with the dataset from step 1 and 2
2. Evaluate the model with the prepared dataset

Tools: Python, PyTorch, SageMaker, EC2, EFS

6. Evaluate how gaze direction influences Stage-3 model accuracy.

1. Modify the model training logic to account for gaze direction and train the model.
2. Compare the performance with baseline.

Tools: Python, SageMaker, PyTorch, EC2

7. Assess the impact of model choice on Stage-3 outcomes

1. Replace Transformer model with MLP model and compare the performance differences.

Tools: Python, EC2, PyTorch.

8. Build a single pipeline connecting all stages.

1. Setup infrastructure required for the end-to-end experiments
2. Create videos demonstrating the device being pointed at for experiment evaluation purposes.

Tools: Python, Pytorch, EC2, SageMaker

9. Evaluate the performance of the End-to-End pipeline

1. Run the evaluation pipeline with the test videos dataset.
2. Document performance (precision and recall)

Tools: Python, Pytorch, CF matrix

Reference architecture

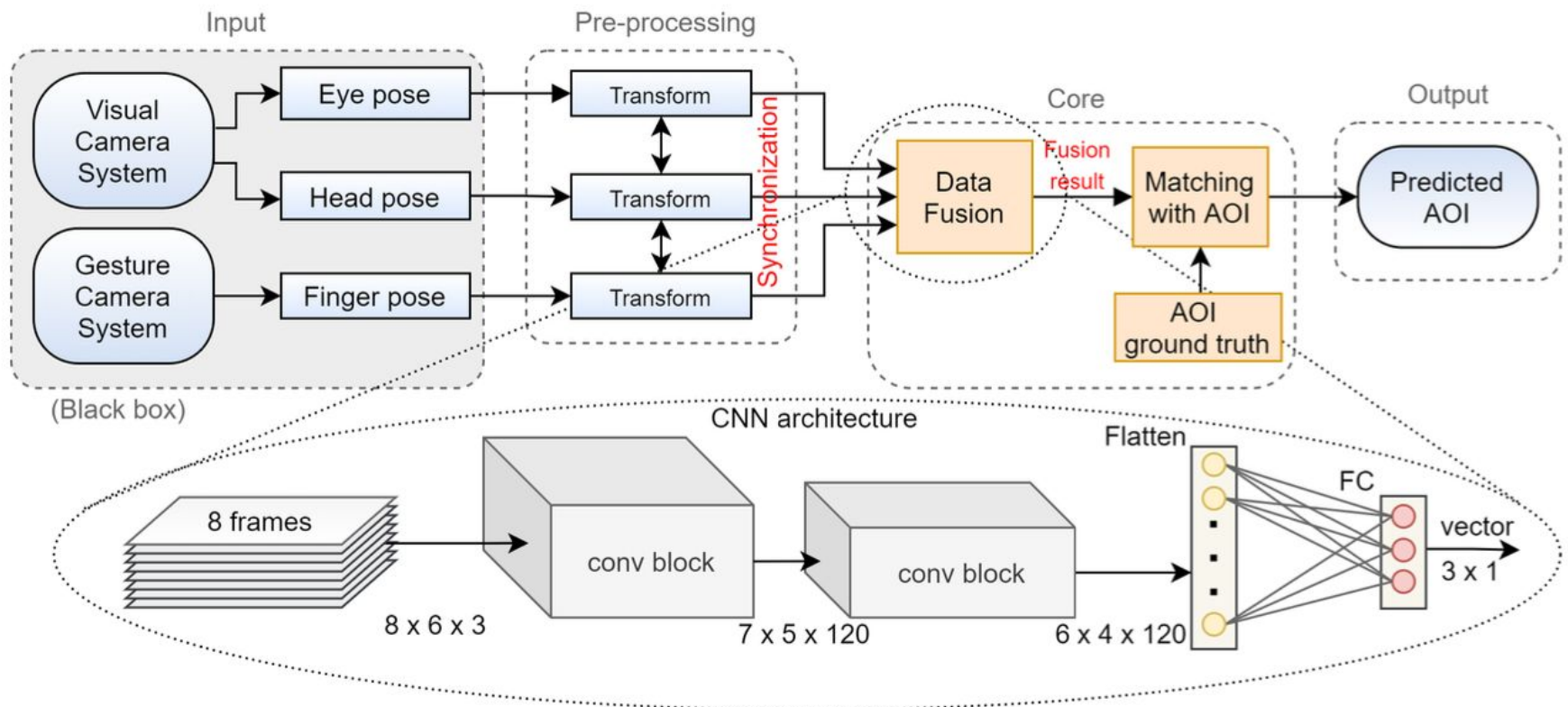



Figure 6: A multimodal late fusion architecture



Research questions

RQ1: Can a two-stage classification system be developed to identify the electrical devices a person points at, enabling touchless device control and improving accessibility?

RQ2: Does tracking gaze direction enhance the accuracy of the second stage in the two-stage classification system?

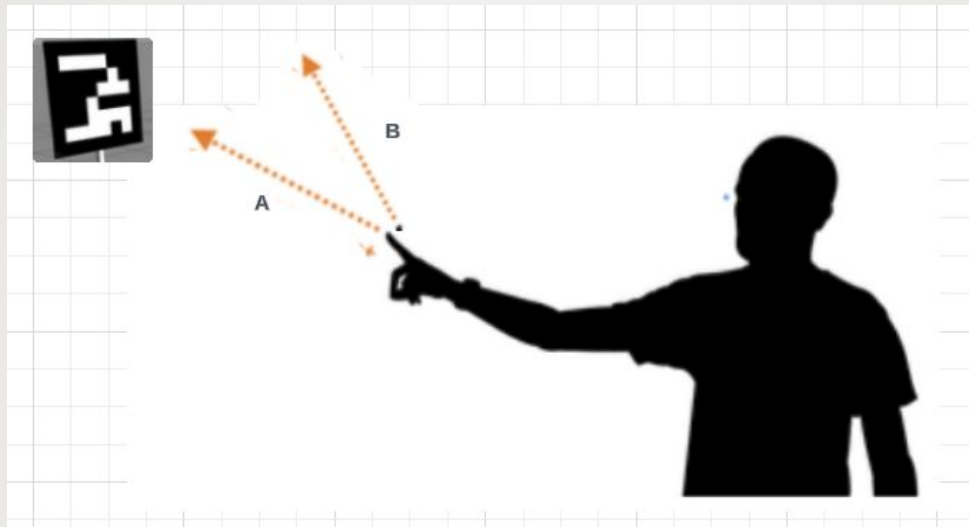
RQ3: Which ML model works best for the second stage of the two-stage classification system for identification of electrical devices, as pointed by an individual?

H1: The proposed two-stage classification system can reach 70% accuracy in identifying the electrical devices an individual points at.

H2: Tracking gaze direction can improve the accuracy by approximately 5% in the second stages of a two-stage classification system.

H3: In the second stage, the proposed device classification system incorporating the Transformer is expected to outperform the model using the MLP architecture.

Stage-1 Loss Function



Takeaway:

- Stage-1 uses cosine similarity between the target and predict pointing direction vector as the loss function.

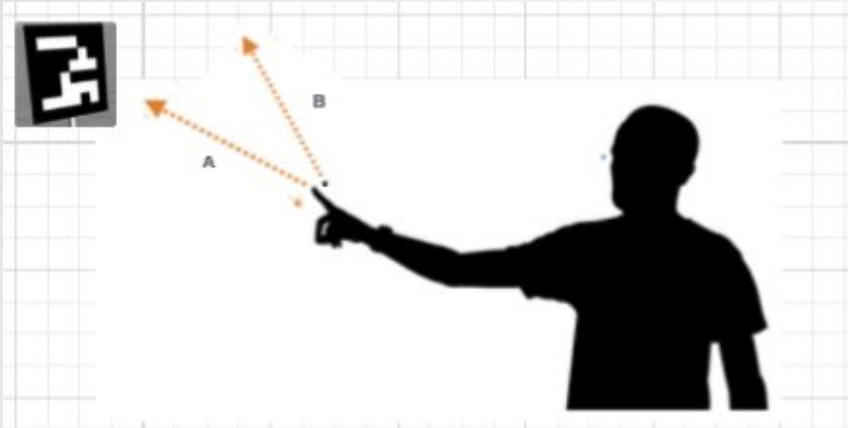
Stage-3 Sample Data (cont)

Keypoint : `[[0.22081189, 0.21458275, ...,
0.19111519, 0.11919417, ...,
0.19223424, ..., 0.08478543], ...](17 keypoints)`

Marker : `[[[-0.3831681 -1.08948806 3.72966626]
[-0.32619304 -1.10173681 3.78169319]
[-0.33496759 -1.02888857 3.8138551]
[-0.39108514 -1.01628521 3.75419499]] (4 corners)`

- The values above are samples from Stage-1 training data, showing keypoint coordinates and marker positions.
- Pointing direction vectors are computed from the fingertip keypoints to marker locations.

Stage-1 Loss Function



Takeaway: Stage-1 uses the cosine similarity between the ground truth pointing direction (toward the marker) and the predicted pointing direction as the loss function.

Device Location Vector



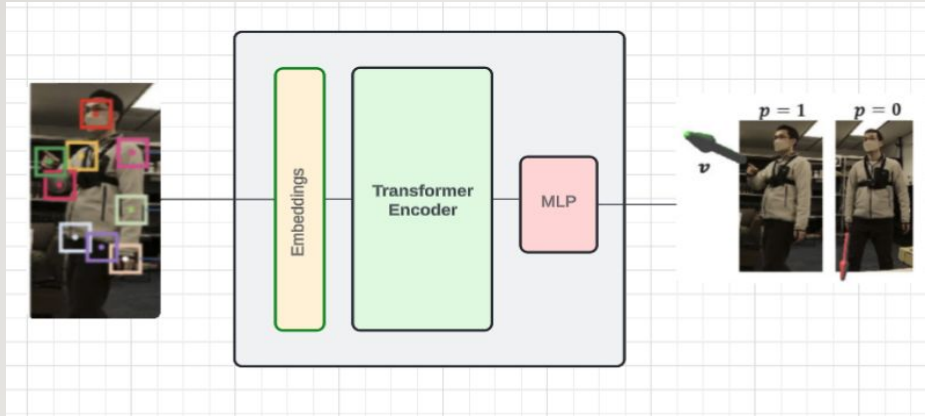
Takeaway:

The device location vector (red line) is computed by connecting the center of the person to the center of the detected device, and then reprojected into 3D space.

Pointing Estimation

- Input: Video stream.
- Model: a transformer-based model estimates the user's pointing direction by tracking body, hand, and finger poses.
- Loss Function: Cosine similarity between the predicted pointing direction and ground truth (toward the marker) is used to train the model.
- Output: A 3D pointing direction vector along with a confidence score.

Architecture



- **Input:** Video stream shows a person performing a pointing gesture.
 - Joints are extracted (e.g., hand, shoulder, etc.) using a pose estimation algorithm.
 - Joint is encoded, capturing spatial features.
- The encoded output is passed through an MLP to predict:
- p:** A binary alignment label
- $p = 1 \rightarrow$ pointing and device are aligned
 - $p = 0 \rightarrow$ not aligned
 - \mathbf{v} : The 3D pointing direction vector

Stage-2: Device detection & Localization

- **YOLO-World** performs 2D object detection on the input image.
- **MiDaS (DPT)** provides depth estimation from the same image.
- The 2D detections are projected into 3D space using the depth map and camera intrinsics.
- This approach is lightweight and modular — ideal when only RGB input is available and avoids to train a full 3D model from scratch.

Stage-3 Model

- The model uses a ***pointing direction vector*** and a ***device location vector*** as input for training and outputs whether they are aligned or not.
- Inputs: (x)
 - 3D pointing direction vector.
 - 3D device location vector.
- Model Architecture: Transformer-based
- Training data
 - Subset of DeePoint dataset. (720k records)
 - Data split: 80% training, 10% validation, and 10% testing.
- Output: 1 for aligned, 0 for not aligned.
-