Architecting Real-Time Object Tracking and Scoring Systems with MediaPipe

Section 1: The MediaPipe Ecosystem for Computer Vision

MediaPipe, an open-source framework from Google, provides a robust and versatile platform for building complex, cross-platform machine learning (ML) pipelines. It is engineered not merely as a repository of pre-trained models but as a comprehensive ecosystem designed to address the entire lifecycle of a computer vision application—from prototyping and customization to efficient, real-time deployment on a wide array of hardware.[1] Understanding this foundational architecture is paramount for developers aiming to build sophisticated applications like real-time object trackers.

1.1 A Foundational Overview of MediaPipe's Architecture

The core of MediaPipe is a dataflow programming framework. Perception pipelines are constructed as directed acyclic graphs (DAGs), where modular components, known as **Calculators**, process data packets.[2] These packets flow between calculators via

Streams. This modular design is the key to MediaPipe's flexibility, allowing developers to assemble, modify, and reuse components to create highly customized pipelines.[2]

A significant advantage of this architecture is its inherent cross-platform nature. MediaPipe is engineered to build a solution once and deploy it across numerous targets, including mobile (Android, iOS), web browsers (JavaScript), desktop and server environments (Python, C++), and even resource-constrained embedded devices like Raspberry Pi.[1] This capability is central to the concept of "edge computing," where complex ML inference is performed on-device rather than in the cloud. This approach dramatically reduces latency, lowers data transmission costs, and enhances user privacy, making it ideal for real-time interactive applications.[4]

The value proposition of MediaPipe extends beyond the quality of its individual models to its holistic solution for the ML deployment lifecycle. While many tutorials conclude after training a model in a development environment, MediaPipe provides the necessary tools to overcome the significant engineering challenge of making that model performant and reliable in a real-world application on diverse hardware platforms.[2]

1.2 The MediaPipe Solutions Suite: A Developer's Toolkit

To streamline development, Google has packaged the framework's power into the MediaPipe Solutions suite, a collection of libraries, models, and tools.

- **MediaPipe Tasks:** These are high-level, cross-platform APIs that provide a simplified, object-oriented interface for deploying pre-built solutions. The Tasks API abstracts the underlying complexity of the graph and calculator system, allowing developers to integrate powerful ML capabilities with just a few lines of code. For any new development, it is strongly recommended to use the modern mediapipe.tasks API, as Google has phased out support for many "Legacy Solutions" as of March 1, 2023, in favor of this more accessible and maintainable approach.
- **MediaPipe Models:** Each solution is accompanied by pre-trained, ready-to-run TensorFlow Lite (.tflite) models. These models are optimized for on-device performance and are often trained on large, canonical datasets like COCO (Common Objects in Context) to recognize a broad range of objects.[1]
- **MediaPipe Model Maker:** This is a critical tool for creating custom object detectors. It utilizes a technique called transfer learning to retrain existing models on a user's custom dataset. This dramatically simplifies the process of creating a specialized model, making it accessible even to developers without deep expertise in ML model engineering.[1] This tool will be explored in detail in Section 3.
- **MediaPipe Studio:** A browser-based visualization tool that allows developers to evaluate, benchmark, and interact with MediaPipe solutions without writing any code. It is an excellent resource for understanding a model's capabilities and limitations before integration.[1]

1.3 Performance Analysis and Optimization

For applications that demand high performance, MediaPipe provides tools for analysis and pathways for optimization.

- **The MediaPipe Visualizer:** This tool is essential for advanced debugging and performance tuning. It renders a pipeline's graph topology and provides a timeline view of its execution. Developers can use the visualizer to diagnose performance bottlenecks, identify unexpected real-time delays caused by specific calculators, and observe memory accumulation from packet buffering. This allows for targeted optimization of the nodes along the pipeline's critical path that determine end-to-end latency.[2]
- **Optimization with OpenVINO:** For deployment on Intel hardware (CPU, Integrated GPU, VPU), Intel's OpenVINO™ toolkit offers a powerful optimization path. OpenVINO can enhance model performance by converting and optimizing them for specific Intel architectures.[8] The OpenVINO Model Server (OVMS) can be used to serve MediaPipe graphs, and official demos exist for tasks like Image Classification and Holistic Tracking, demonstrating this integration.[10] While this represents an advanced optimization route, it can yield significant performance gains for production systems deployed on compatible hardware. A community-driven
  MediapipeOpenvino package on PyPI also exists as another potential avenue for exploration.[12]

Section 2: A Comparative Analysis of MediaPipe's Object Tracking Solutions

To build an effective object tracker, one must first select the appropriate tool from MediaPipe's offerings. The framework provides several distinct solutions for object tracking, each with its own underlying technology, capabilities, and ideal use cases. The choice between them is a critical architectural decision that will shape the entire application.

2.1 In-Depth Analysis of the ObjectDetector Task

The ObjectDetector task is the primary ML-based solution for identifying the presence and location of objects within an image or video frame.[13]

- **Core Functionality:** It takes an image as input and outputs a list of detected objects, each with a class label and a confidence score. It is designed for per-frame detection.[13]
- **Pre-trained Models:** The task comes with several pre-trained models, such as EfficientDet-Lite and SSD MobileNetV2, which are trained on the COCO dataset and can recognize 80 common object categories.[4] These models offer a trade-off between inference speed and detection accuracy.
- **Configuration and Output:** The ObjectDetector is highly configurable through its options. Key parameters include running_mode (to handle single images, videos, or live streams), score_threshold (to filter out low-confidence detections), max_results (to limit the number of objects returned), and category_allowlist (to focus on specific objects).[4] The output is a DetectionResult object containing a list of Detections, where each detection provides a bounding_box and a list of categories with a category_name and a confidence score.[4]

2.2 The Mechanics of BoxTracking

In contrast to the ML-based ObjectDetector, the BoxTracking solution relies on classic computer vision techniques.[17]

- **Core Technology:** It does not perform detection itself. Instead, it requires an initial set of bounding boxes to track. Its pipeline consists of three main components:
  1. A MotionAnalysis calculator extracts and tracks low-level image features (like corners) across the frame.
  2. A FlowPackager calculator efficiently compresses this motion data.
  3. A BoxTracker calculator uses this motion data to predict the new positions of the initial boxes, without needing to re-process the full-resolution RGB frames.[17]

- **The "Detect-then-Track" Paradigm:** The primary and most powerful use of BoxTracking is to pair it with an ML-based detector like ObjectDetector. This architectural pattern is central to building efficient, real-time tracking systems in MediaPipe. The computationally expensive detector runs only intermittently (e.g., every 5-10 frames) to establish or correct object locations. In the frames in between, the lightweight BoxTracking algorithm takes over, providing smooth and temporally consistent tracking.[17] This approach offers several profound advantages:
    - **Performance:** It allows the use of larger, more accurate detection models while still achieving real-time frame rates, even on mobile devices.
    - **Temporal Consistency:** It significantly reduces the visual "jitter" that can occur when running a detector on every frame.
    - **Instance Persistence:** It inherently maintains a consistent identity for each tracked object across frames.[17]

2.3 Advanced 3D Perception with Objectron

The Objectron solution moves beyond 2D boxes to provide real-time 3D object detection, estimating an object's pose, orientation, and physical size.[19]

- **Core Functionality:** Objectron is designed for specific categories of everyday objects for which it has been trained, such as shoes, chairs, cups, and cameras.[22] It uses models trained on the specialized
  Objectron dataset, which includes object-centric videos captured with AR metadata to provide ground-truth 3D information.[19]
- **Internal Architecture:** Objectron itself is a powerful example of the "detect-then-track" paradigm. It contains an internal detection subgraph that runs periodically and a tracking subgraph that uses the very same BoxTracker technology to track the 2D projection of the 3D bounding box between ML inferences.[19]
- **Output Structure:** The output is significantly richer than that of the 2D detector, providing 3D landmarks in camera coordinates, 2D landmarks for visualization, and the object's rotation, translation, and scale matrices.[19]

2.4 Comparison of MediaPipe Tracking Solutions

The following table summarizes the key characteristics of each solution to guide the selection process. For a real-time tracker with a score keeper, the most common and effective approach is to combine the ObjectDetector with a tracking algorithm, embodying the "detect-then-track" pattern. Objectron is a specialized tool for when 3D data is essential to the application's logic.

Feature	MediaPipe ObjectDetector	MediaPipe BoxTracking	MediaPipe Objectron
Dimensionality	2D	2D	3D
Core Technology	Machine Learning (TF Lite)	Classic Computer Vision (Feature Tracking)	Hybrid: ML Detection + CV Tracking
Primary Use Case	Detecting object presence and class in a single frame.	Efficiently tracking known 2D regions of interest across frames.	Detecting 3D pose, orientation, and size of specific object categories.
Input	Image/Video Frame	Image/Video Frame + Initial Bounding Boxes	Image/Video Frame
Output Data	Bounding Box, Category Name, Confidence Score	Tracked Bounding Box Positions	3D Bounding Box Landmarks, 2D Landmarks, Rotation, Translation, Scale
Performance	Computationally moderate to heavy (depends on model).	Computationally very lightweight.	Heavy for detection, lightweight for tracking.
Customization	Highly customizable with MediaPipe Model Maker for any object class.	Not applicable (algorithm-based).	Not easily customizable; requires a complex 3D dataset with AR metadata.
When to Use	For simple detection tasks or as the "detector" in a detect-then-track pipeline.	As the "tracker" in a detect-then-track pipeline to improve performance and temporal stability.	When 3D information (orientation, real-world size) is required for the application logic.

Section 3: Custom Object Recognition with MediaPipe Model Maker

While MediaPipe's pre-trained object detectors are powerful, they are limited to the classes in their training data (e.g., the 80 categories in the COCO dataset).[4] To build a tracker for custom objects—be it a specific type of ball for a game, a unique product, or any other specialized item—it is necessary to train a custom model. MediaPipe Model Maker is a high-level library designed to make this process accessible and efficient.[1]

Model Maker powerfully abstracts the complexities of deep learning, shifting the developer's primary focus from intricate model engineering to the more manageable task of data engineering. The most critical step for success becomes the collection and accurate annotation of a high-quality dataset; the tool handles the complex training and conversion process.[26]

3.1 The Principles of Transfer Learning for Rapid Model Development

Model Maker employs a technique called **transfer learning.**[7] Instead of training a new neural network from scratch, which would require vast amounts of data and computational resources, transfer learning starts with a sophisticated, pre-trained model (like MobileNetV2). It then retrains only the final layers of this model using the new, custom dataset.

This approach has two key advantages:

- **Reduced Data Requirement:** Because the model has already learned general visual features (edges, textures, shapes) from a large dataset, it needs far less data to learn the new classes. A general guideline is to have approximately 100 annotated images per class.[7]
- **Faster Training:** Retraining only a small portion of the network is significantly

faster than training the entire model from the ground up.[7]

The main limitation is that the retrained model is specialized. It can only recognize the new classes it was trained on and cannot perform a fundamentally different task (e.g., an object detector cannot be retrained to become an image classifier).[7]

3.2 Step-by-Step Guide to Preparing a Custom Dataset

The quality of the training data is the single most important factor in the performance of the final model. Model Maker for object detection supports two standard annotation formats.[27]

- **PASCAL VOC Format:** This format consists of a folder of images and a parallel folder of XML files. Each XML file corresponds to an image and contains metadata, including the filename, image dimensions, and one or more <object> tags. Each <object> tag defines a single object instance with its class <name> and its bounding box coordinates (<bndbox>).
- **COCO Format:** This format uses a data folder for all images and a single labels.json file to hold all annotations. This JSON file contains sections for images, annotations (which link bounding boxes to images), and categories (the list of all class names).

For robust model evaluation, the dataset should be split into at least two subsets: a train set for training the model and a validation set to assess its performance on unseen data and prevent overfitting.[26] Tools like

Label Studio can be invaluable for the practical task of drawing bounding boxes and exporting annotations in these standard formats.[29]

3.3 The Training and Evaluation Pipeline (Python/Colab)

Training a custom model with Model Maker can be accomplished with a straightforward Python script, often executed in a Google Colab notebook to leverage free GPU resources.

1. **Setup:** Install the necessary packages.

!pip install --upgrade pip
!pip install mediapipe-model-maker
Then, import the required modules.python
from mediapipe_model_maker import object_detector
```

25

2. **Load the Dataset:** Use the Dataset class to load the annotated data from your prepared folders.
   Python
   ```
   train_data = object_detector.Dataset.from_pascal_voc_folder('path/to/train_data/')
   validation_data =
   object_detector.Dataset.from_pascal_voc_folder('path/to/validation_data/')
   ```

   26

3. **Select Model and Configure Training:** Choose a supported model architecture and define the training hyperparameters.
   Python
   ```
   # Define the model architecture
   spec = object_detector.SupportedModels.MOBILENET_V2

   # Define hyperparameters like batch size, learning rate, and epochs
   hparams = object_detector.HParams(export_dir='exported_model',
                                     batch_size=8,
                                     learning_rate=0.3,
                                     epochs=50)

   # Combine into training options
   options = object_detector.ObjectDetectorOptions(supported_model=spec,
                                                   hparams=hparams)
   ```

   25

4. **Run Training:** Create and train the model with a single command.
   Python
   ```
   model = object_detector.ObjectDetector.create(train_data=train_data,
                                                 validation_data=validation_data,
                                                 options=options)
   ```

   26
   ```

5. **Evaluate the Model:** After training, use the validation set to measure the model's performance. This is a crucial step to ensure the model generalizes well.
   Python
   ```python
   loss, coco_metrics = model.evaluate(validation_data)
   print(f"Validation loss: {loss}")
   print(f"Validation coco metrics: {coco_metrics}")
   ```
   26
6. **Export the Model:** Finally, export the trained model to the .tflite format. This file is now ready to be used with the MediaPipe ObjectDetector task in your application.
   Python
   ```python
   model.export_model('my_custom_model.tflite')
   ```
   26

For advanced use cases, Model Maker also supports **model quantization**, a process that reduces the model's file size and can speed up inference by converting its weights to lower-precision numbers (e.g., 16-bit floats or 8-bit integers). This is particularly useful for deployment on resource-constrained devices.[25]

Section 4: Building a Real-Time Object Tracker with a Score Keeper in Python

This section integrates the concepts from the previous sections into a practical, step-by-step guide for building a complete application: a real-time object tracker with a functional score keeper, using a live webcam feed. The core of this application lies in managing state on top of a tracking pipeline. The tracker provides persistent identity, and the scoring logic acts as a state machine that increments a score based on an object's interaction with predefined zones.

4.1 Part I: Foundational Setup and Real-Time Detection

First, establish the basic application structure for capturing video and running the

MediaPipe ObjectDetector.

1. **Environment Setup:** Ensure opencv-python and mediapipe are installed.
   Bash
   ```bash
   pip install opencv-python mediapipe
   ```
   42
2. **Core Video Loop and Detection:** The following script sets up the webcam, initializes the ObjectDetector with a model file (either pre-trained or a custom one from Section 3), and runs detection on each frame.
   Python
   ```python
   import cv2
   import mediapipe as mp
   import numpy as np

   # Import MediaPipe Tasks
   from mediapipe.tasks import python
   from mediapipe.tasks.python import vision

   # --- INITIALIZATION ---
   # Create ObjectDetectorOptions
   model_path = 'efficientdet.tflite' # Or 'my_custom_model.tflite'
   base_options = python.BaseOptions(model_asset_path=model_path)
   options = vision.ObjectDetectorOptions(
       base_options=base_options,
       running_mode=vision.RunningMode.VIDEO,
       score_threshold=0.5 # Set a confidence threshold
   )

   # Create the object detector
   detector = vision.ObjectDetector.create_from_options(options)

   # Initialize webcam
   cap = cv2.VideoCapture(0)
   if not cap.isOpened():
       print("Cannot open camera")
       exit()

   # --- MAIN LOOP ---
   frame_index = 0
   while True:
   ```

```
    # Capture frame-by-frame
    success, frame = cap.read()
    if not success:
        print("Can't receive frame. Exiting...")
        break

    # Convert the frame from BGR to RGB
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # Create a MediaPipe Image object
    mp_image = mp.Image(image_format=mp.ImageFormat.SRGB,
data=rgb_frame)

    # Get frame timestamp for video mode
    frame_timestamp_ms = int(cap.get(cv2.CAP_PROP_POS_MSEC))

    # Detect objects
    detection_result = detector.detect_for_video(mp_image, frame_timestamp_ms)

    # --- VISUALIZATION ---
    if detection_result:
        for detection in detection_result.detections:
            bbox = detection.bounding_box
            start_point = (bbox.origin_x, bbox.origin_y)
            end_point = (bbox.origin_x + bbox.width, bbox.origin_y + bbox.height)

            # Draw the bounding box
            cv2.rectangle(frame, start_point, end_point, (0, 255, 0), 2)

            # Draw the label and score
            category = detection.categories
            category_name = category.category_name
            probability = round(category.score, 2)
            result_text = f"{category_name} ({probability})"
            text_location = (start_point, start_point - 10)
            cv2.putText(frame, result_text, text_location, cv2.FONT_HERSHEY_PLAIN, 1,
(0, 255, 0), 2)

    # Display the resulting frame
    cv2.imshow('Object Tracker', frame)

    # Exit on 'q' key press
    if cv2.waitKey(1) == ord('q'):
        break

# --- CLEANUP ---
cap.release()
cv2.destroyAllWindows()
```

13

4.2 Part II: Implementing Persistent Tracking and Unique Object IDs

The script above detects objects but cannot track them individually. To build a score keeper, we must solve the **identity problem**: knowing that an object in one frame is the same object in the next.[30] This requires a tracking algorithm to assign and maintain a unique ID for each object.

Below is an implementation of a simple but effective **Centroid Tracking** algorithm. This approach associates detections across frames based on the Euclidean distance between their centroids.[31]

Python

```python
from collections import OrderedDict
from scipy.spatial import distance as dist
import numpy as np

class CentroidTracker:
    def __init__(self, maxDisappeared=50):
        self.nextObjectID = 0
        self.objects = OrderedDict()
        self.disappeared = OrderedDict()
        self.maxDisappeared = maxDisappeared
```

```
def register(self, centroid):
    self.objects = centroid
    self.disappeared = 0
    self.nextObjectID += 1

def deregister(self, objectID):
    del self.objects
    del self.disappeared

def update(self, rects):
    if len(rects) == 0:
        for objectID in list(self.disappeared.keys()):
            self.disappeared += 1
            if self.disappeared > self.maxDisappeared:
                self.deregister(objectID)
        return self.objects

    inputCentroids = np.zeros((len(rects), 2), dtype="int")
    for (i, (startX, startY, endX, endY)) in enumerate(rects):
        cX = int((startX + endX) / 2.0)
        cY = int((startY + endY) / 2.0)
        inputCentroids[i] = (cX, cY)

    if len(self.objects) == 0:
        for i in range(len(inputCentroids)):
            self.register(inputCentroids[i])
    else:
        objectIDs = list(self.objects.keys())
        objectCentroids = list(self.objects.values())
        D = dist.cdist(np.array(objectCentroids), inputCentroids)
        rows = D.min(axis=1).argsort()
        cols = D.argmin(axis=1)[rows]

        usedRows = set()
        usedCols = set()

        for (row, col) in zip(rows, cols):
            if row in usedRows or col in usedCols:
                continue
            objectID = objectIDs[row]
            self.objects = inputCentroids[col]
            self.disappeared = 0
            usedRows.add(row)
            usedCols.add(col)

        unusedRows = set(range(0, D.shape)).difference(usedRows)
        unusedCols = set(range(0, D.shape)).difference(usedCols)

        if D.shape >= D.shape:
            for row in unusedRows:
                objectID = objectIDs[row]
                self.disappeared += 1
                if self.disappeared > self.maxDisappeared:
                    self.deregister(objectID)
        else:
            for col in unusedCols:
                self.register(inputCentroids[col])
    return self.objects
```

31

For more complex scenarios involving significant occlusion or camera movement, integrating a dedicated library like **Norfair** is recommended. Norfair is a lightweight, customizable tracker that can work with any detector and offers advanced features like Re-Identification (ReID) and 3D tracking.[32]

4.3 Part III: Developing the Score-Keeping Logic

With persistent tracking established, the final step is to implement the scoring logic. This involves defining an interactive zone and managing the state of each tracked object relative to that zone.

1. **Integrate the Tracker:** In the main script, instantiate the CentroidTracker and in each frame, pass the list of detected bounding boxes (rects) to tracker.update().
2. **Define a Scoring Zone:** Define a region of interest (ROI) on the frame. A simple

horizontal line can serve as a counter.

```
Python
# Inside the main loop, before drawing
height, width, _ = frame.shape
scoring_line_y = int(height * 0.75)
cv2.line(frame, (0, scoring_line_y), (width, scoring_line_y), (255, 0, 0), 2)
```

3. **Create the Stateful Scoring Function:** This logic is analogous to how exercise counters work in pose estimation apps, where a repetition is counted on a state change (e.g., from "up" to "down").[34] Here, a score is added when an object crosses the line. We need to track the previous position of each object to detect a crossing event.

```
Python
# Before the main loop
score = 0
tracked_objects_positions = {} # To store previous positions {id: y_coord}

# Inside the main loop, after tracker.update()
objects = tracker.update(rects) # rects is the list of bounding boxes from detector

for (objectID, centroid) in objects.items():
  # Check if this is a new object or an existing one
  if objectID not in tracked_objects_positions:
    tracked_objects_positions = centroid # Store initial Y
  else:
    prev_y = tracked_objects_positions
    current_y = centroid

    # Check for crossing the line (from above to below)
    if prev_y < scoring_line_y and current_y >= scoring_line_y:
      score += 1
      # Optional: Add visual feedback for scoring event
      cv2.line(frame, (0, scoring_line_y), (width, scoring_line_y), (0, 255, 0), 4)

    # Update the position for the next frame
    tracked_objects_positions = current_y

  # Draw the object ID and centroid
  text = f"ID {objectID}"
  cv2.putText(frame, text, (centroid - 10, centroid - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
  cv2.circle(frame, (centroid, centroid), 4, (0, 255, 0), -1)

# Display the score
cv2.putText(frame, f"Score: {score}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0,
255), 2)
```

This stateful approach—checking the transition from a previous state (prev_y < scoring_line_y) to a current state (current_y >= scoring_line_y)—is the key to correctly implementing a score keeper and preventing the score from incrementing on every frame an object is below the line.

Section 5: Conclusion and Future Directions

This report has detailed a comprehensive approach to building a real-time object tracking and scoring system using Google's MediaPipe framework. The analysis has demonstrated that MediaPipe is not just a collection of models but a complete ecosystem for developing and deploying high-performance computer vision pipelines.

5.1 Summary of Key Architectural Patterns

Several critical architectural patterns have emerged as essential for success:

- **The MediaPipe Ecosystem:** Leveraging the full suite of tools—Tasks for easy deployment, Models as a starting point, and Model Maker for crucial customization—is the most effective way to build applications.
- **The "Detect-then-Track" Paradigm:** For efficient and temporally stable real-time tracking, combining a periodic, heavy ML-based detector with a lightweight, per-frame classical tracker is the optimal strategy. This pattern is fundamental to MediaPipe's own advanced solutions like BoxTracking and Objectron.
- **Persistent ID for State Management:** Simple detection is insufficient for interactive applications like a score keeper. A tracking algorithm that provides persistent object identity is a prerequisite for managing state.

- **Scoring as a State Machine:** The core logic of a score keeper is a state machine that triggers actions (like incrementing a score) based on state transitions (e.g., an object moving from "outside" a zone to "inside").

5.2 Recommendations for Further Exploration

The system developed in this report serves as a strong foundation that can be extended in several exciting directions:

- **Web Deployment:** The entire application can be ported to run in a web browser using **MediaPipe.js**. This allows for creating highly accessible applications without requiring any installation from the end-user. Official JavaScript examples and CodePen demos provide a clear starting point for this transition.[15]
- **Leveraging 3D Data:** For more sophisticated scoring logic, **MediaPipe Objectron** could be used. This would allow the system to score based not just on 2D position but on an object's 3D orientation, real-world size, or pose. For example, a game could award points only if a specific object is thrown with the correct orientation.[19]
- **Multi-Modal Applications:** The object tracker can be combined with other MediaPipe tasks to create richer interactions. For instance, using **Pose Landmarker** or **Hand Landmarker**, the system could detect when a user physically interacts with a tracked object (e.g., "picking up" an item).[37] The **MediaPipe Holistic** solution, which combines pose, face, and hand tracking, is perfectly suited for such complex, full-body interactive experiences.[6]
- **Advanced Performance Optimization:** For production deployments on compatible Intel hardware, using the **OpenVINO™ toolkit** to optimize and serve the ML model can provide a significant boost in inference speed and efficiency.[8]
- **Robustness to Occlusion:** In scenarios where objects frequently overlap or are occluded, the simple centroid tracker may struggle. Exploring the advanced features of a dedicated library like **Norfair**, specifically its support for Re-Identification (ReID) using appearance embeddings, can help re-acquire object identities after they have been lost from view.[32]

Works cited

1. MediaPipe Solutions guide | Google AI Edge - Gemini API, accessed July 7, 2025, https://ai.google.dev/edge/mediapipe/solutions/guide
2. MediaPipe: A Guide Google's Open Source Framework - viso.ai, accessed July 7,

2025, https://viso.ai/computer-vision/mediapipe/
3. What is MediaPipe? A Guide for Beginners - Roboflow Blog, accessed July 7, 2025, https://blog.roboflow.com/what-is-mediapipe/
4. Computer Vision with MediaPipe - GSAPP SMORGASBORD, accessed July 7, 2025, https://smorgasbord.cdp.arch.columbia.edu/modules/32-ai-vision-imagery/321-cv-mediapipe/
5. AISoltani/MediaPipe_Pose_Face_Hand_Detection: MediaPipe Solutions provides a suite of libraries and tools for you to quickly apply artificial intelligence (AI) - GitHub, accessed July 7, 2025, https://github.com/AISoltani/MediaPipe_Pose_Face_Hand_Detection
6. MediaPipe Holistic — Simultaneous Face, Hand and Pose Prediction, on Device, accessed July 7, 2025, https://research.google/blog/mediapipe-holistic-simultaneous-face-hand-and-pose-prediction-on-device/
7. MediaPipe Model Maker | Google AI Edge - Gemini API, accessed July 7, 2025, https://ai.google.dev/edge/mediapipe/solutions/model_maker
8. ToolRepoName-Functionality-Platforms-Reference.csv
9. Person and Face Detection using Intel OpenVINO toolkit - GeeksforGeeks, accessed July 7, 2025, https://www.geeksforgeeks.org/machine-learning/person-and-face-detection-using-intel-openvino-toolkit/
10. model_server/demos/mediapipe/holistic_tracking/README.md at main - GitHub, accessed July 7, 2025, https://github.com/openvinotoolkit/model_server/blob/main/demos/mediapipe/holistic_tracking/README.md
11. MediaPipe Image Classification Demo - OpenVINO™ documentation, accessed July 7, 2025, https://docs.openvino.ai/2025/model-server/ovms_docs_demo_mediapipe_image_classification.html
12. MediapipeOpenvino - PyPI, accessed July 7, 2025, https://pypi.org/project/MediapipeOpenvino/
13. Object detection guide for Python | Google AI Edge - Gemini API, accessed July 7, 2025, https://ai.google.dev/edge/mediapipe/solutions/vision/object_detector/python
14. Object detection task guide | Google AI Edge - Gemini API, accessed July 7, 2025, https://ai.google.dev/edge/mediapipe/solutions/vision/object_detector
15. Object detection guide for Web | Google AI Edge - Gemini API, accessed July 7, 2025, https://ai.google.dev/edge/mediapipe/solutions/vision/object_detector/web_js
16. object_detector.ipynb - Colab, accessed July 7, 2025, https://colab.research.google.com/github/googlesamples/mediapipe/blob/main/examples/object_detection/python/object_detector.ipynb
17. mediapipe/docs/solutions/box_tracking.md at master · google-ai ..., accessed July 7, 2025,

https://github.com/google/mediapipe/blob/master/docs/solutions/box_tracking.md
18. layout: forward target: https://developers.google.com/mediapipe ..., accessed July 7, 2025, https://mediapipe.readthedocs.io/en/latest/solutions/box_tracking.html
19. mediapipe/docs/solutions/objectron.md at master - GitHub, accessed July 7, 2025, https://github.com/google/mediapipe/blob/master/docs/solutions/objectron.md
20. Real-Time 3D Object Detection on Mobile Devices with MediaPipe - Fritz ai, accessed July 7, 2025, https://fritz.ai/real-time-3d-object-detection-on-mobile-devices/
21. layout: forward target: https://developers.google.com/mediapipe/solutions/guide#legacy title: Objectron (3D Object Detection) parent: MediaPipe Legacy Solutions nav_order: 12 — MediaPipe v0.7.5 documentation - MediaPipe Docs, accessed July 7, 2025, https://mediapipe.readthedocs.io/en/latest/solutions/objectron.html
22. mediapipe/mediapipe/python/solutions/objectron.py at master - GitHub, accessed July 7, 2025, https://github.com/google/mediapipe/blob/master/mediapipe/python/solutions/objectron.py
23. 3D Object Detection (3D Bounding Boxes) in Python with MediaPipe ..., accessed July 7, 2025, https://stackabuse.com/3d-object-detection-3d-bounding-boxes-in-python-with-mediapipe-objectron/
24. google-research-datasets/Objectron: Objectron is a dataset of short, object-centric video clips. In addition, the videos also contain AR session metadata including camera poses, sparse point-clouds and planes. In each video, the camera moves around and above the object and captures it from different views. - GitHub, accessed July 7, 2025, https://github.com/google-research-datasets/Objectron
25. Object detection model customization guide | Google AI Edge - Gemini API, accessed July 7, 2025, https://ai.google.dev/edge/mediapipe/solutions/customization/object_detector
26. Train a custom object detection model with MediaPipe Model Maker - Colab - Google, accessed July 7, 2025, https://colab.research.google.com/github/googlesamples/mediapipe/blob/main/tutorials/object_detection/Object_Detection_for_3_dogs.ipynb
27. How to Create Custom ML Models with Google MediaPipe Model Maker - DigiKey, accessed July 7, 2025, https://www.digikey.com/en/maker/tutorials/2024/how-to-create-custom-ml-models-with-google-mediapipe-model-maker
28. mediapipe-samples/examples/customization/object_detector.ipynb at main - GitHub, accessed July 7, 2025, https://github.com/googlesamples/mediapipe/blob/main/examples/customization/object_detector.ipynb

29. Training an object detection model - ML on Raspberry Pi with MediaPipe Series - YouTube, accessed July 7, 2025, https://www.youtube.com/watch?v=X9554zNNtEY
30. Object Tracking using OpenCV (C++/Python) - LearnOpenCV, accessed July 7, 2025, https://learnopencv.com/object-tracking-using-opencv-cpp-python/
31. Simple object tracking with OpenCV - PyImageSearch, accessed July 7, 2025, https://pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/
32. tryolabs/norfair: Lightweight Python library for adding real-time multi-object tracking to any detector. - GitHub, accessed July 7, 2025, https://github.com/tryolabs/norfair
33. Announcing Norfair 2.0: an open-source real-time multi-object ..., accessed July 7, 2025, https://tryolabs.com/blog/2022/09/20/announcing-norfair-2.0-open-source-real-time-multi-object-tracking-library
34. Sayedalihassaan/GYM-pose-estimation-using-mediapipe - GitHub, accessed July 7, 2025, https://github.com/Sayedalihassaan/GYM-pose-estimation-using-mediapipe
35. Create a custom object detection web app with MediaPipe - Google Codelabs, accessed July 7, 2025, https://codelabs.developers.google.com/mp-object-detection-web
36. MediaPipe Objectron · Models - Dataloop, accessed July 7, 2025, https://dataloop.ai/library/model/mediapipe_objectron/
37. Pose landmark detection guide | Google AI Edge | Google AI for ..., accessed July 7, 2025, https://ai.google.dev/edge/mediapipe/solutions/vision/pose_landmarker
38. Hand landmarks detection guide | Google AI Edge - Gemini API, accessed July 7, 2025, https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker
39. Holistic landmarks detection task guide | Google AI Edge - Gemini API, accessed July 7, 2025, https://ai.google.dev/edge/mediapipe/solutions/vision/holistic_landmarker
40. MediaPipe Holistic · Models - Dataloop, accessed July 7, 2025, https://dataloop.ai/library/model/mediapipe_face-holistic/
41. Getting Started - Norfair, accessed July 7, 2025, https://tryolabs.github.io/norfair/2.2/getting_started/
42. Python MediaPipe: real-time hand tracking and landmarks ..., accessed July 7, 2025, https://techtutorialsx.com/2021/04/20/python-real-time-hand-tracking/
43. Implementation of Human Pose Estimation Using MediaPipe | by CodeTrade India | Medium, accessed July 7, 2025, https://medium.com/@codetrade/implementation-of-human-pose-estimation-using-mediapipe-23a57968356b
44. Face and Hand Landmarks Detection using Python - Mediapipe, OpenCV - GeeksforGeeks, accessed July 7, 2025, https://www.geeksforgeeks.org/machine-learning/face-and-hand-landmarks-detection-using-python-mediapipe-opencv/
45. Object Tracking and Detection using OpenCV in python - GitHub, accessed July 7, 2025, https://github.com/ayush-thakur02/object-tracking-opencv

46. Practical-CV/Simple-object-tracking-with-OpenCV - GitHub, accessed July 7, 2025, https://github.com/Practical-CV/Simple-object-tracking-with-OpenCV