Object Tracking Assessment Documentation

1. Objective

The purpose of this assessment is to implement an end-to-end computer vision pipeline using **Segment Anything Model 2 (SAM2)** for:

1. **Object Detection** — identifying and segmenting an object in the first image.
2. **Bounding Box Extraction** — deriving the object's location from segmentation masks.
3. **Object Tracking** — tracking the detected object across subsequent frames/images.
4. **Visualization** — displaying results for validation and reporting.

This notebook serves as a demonstration of applying advanced segmentation and tracking in a reproducible and optimized workflow.

---

2. Pipeline Steps & Justification

Step 1: Environment Setup & Imports

Purpose:

- Load essential libraries: torch, numpy, PIL, matplotlib, glob, shutil, gc for data handling, visualization, and memory management.
- Import video_predictor from SAM2 for object tracking.

Why:

- Organizing imports first ensures reproducibility and makes dependency checks easier.
- Explicit device selection (CPU or CUDA) allows optimization based on available resources.

Step 2: Data Preparation

Process:

- Load dataset containing **paired images** and **ground truth segmentation masks**.
- Select **Image 1** (source) and **Image 2** (target) along with its mask.

Why:

- The first image + mask provides accurate initialization for object location.
- The second image tests tracking performance.

Step 3: Bounding Box Extraction

Process:

- Convert the segmentation mask to a NumPy array.
- Extract **non-zero pixels** representing the object.
- Compute **min/max coordinates** → (x_min, y_min, x_max, y_max).

Why:

- Bounding boxes simplify initialization for the tracker by giving it precise object coordinates.
- Works as a compact representation compared to full pixel masks.

Step 4: Tracking Initialization

Process:

- Create a temporary directory for frame storage.
- Store Image 1 and Image 2 as sequential video frames (00000.jpg, 00001.jpg).
- Initialize **SAM2 video predictor state**.
- Add the bounding box as the initial object position.

Why:

- Video predictor requires sequential frame inputs.
- Using a temp directory makes the process modular and easily reusable.

Step 5: Object Tracking

Process:

- Propagate the tracking model across frames.
- Store results as binary masks for each frame.

Why:

- Tracking verifies the model's temporal consistency and ability to handle object motion between frames.

Step 6: Visualization

Process:

- Draw bounding boxes on source images.
- Overlay tracking masks on target images with transparency.

Why:

- Provides qualitative validation of model predictions.
- Overlay visualization highlights the tracked object without obscuring context.

Step 7: Testing Suite

Process:

- Test on multiple object categories.
- Validate bounding box extraction, tracking accuracy, and visualization.
- Calculate **success rates** across test cases.

Why:

- Ensures pipeline robustness across varied objects.
- Helps in identifying category-specific weaknesses.

Step 8: Performance Benchmarking

Process:

- Measure bounding box extraction time.

- Measure tracking time.
- Classify performance (EXCELLENT, GOOD, SLOW).

Why:

- Provides quantitative evidence of pipeline efficiency.
- Important for production-readiness evaluation.

---

3. Optimization Techniques Used

Optimization Area	Technique	Benefit
Memory Management	Explicit torch.cuda.empty_cache() & gc.collect() after heavy operations	Prevents GPU memory fragmentation, avoids OOM errors
Data Handling	Temporary directory with only required frames	Reduces I/O overhead, prevents unnecessary memory load
Model Resetting	video_predictor.reset_state() before new object	Ensures no state leakage between runs
Bounding Box Calculation	NumPy min/max over mask pixels	Computationally fast compared to contour methods
GPU Utilization	Conditional CUDA usage (torch.cuda.is_available())	Adapts to hardware for faster computation
Batch Clearing	Deleting intermediate tensors (del out_mask_logits)	Frees VRAM immediately during multi-frame tracking
Testing Pipeline	Automated multi-object tests	Ensures correctness without manual verification

---

4. Final Output & Deliverables

Upon completion, the notebook provides:

- **Bounding Box Coordinates** — [x_min, x_max, y_min, y_max]
- **Tracked Object Masks** for each frame
- **Visualization Images** — with overlays for human validation
- **Performance Metrics** — time per operation, success rate
- **Testing Report** — for multiple objects

---

5. Key Takeaways

- SAM2 provides high-accuracy segmentation and temporal tracking.
- Efficient **mask-to-bbox** conversion is critical for fast initialization.
- GPU memory management is essential for large video datasets.
- The pipeline is modular and can be extended for multi-object tracking or real-time applications.