YOLOv8 `model.track` Benchmark
(macOS M4 Pro, Parallels, UTM)

Emre Saraç

September 16, 2025

Abstract

This short report benchmarks the YOLOv8 `model.track` (ByteTrack) pipeline on a single test video across different execution environments: macOS (CPU, MPS, CoreML) and virtualized Linux (Parallels, UTM). The results highlight the performance differences in end-to-end (pipeline) frames per second (FPS) and inference-only FPS.

Hardware and Software

Host (macOS): Apple M4 Pro; **CPU**: 14 cores, **GPU**: 20 cores, **Unified Memory**: 24 GB.
Model: YOLOv8n (detection) + ByteTrack (tracking).
Video: 13-second clip (341 frames) containing multiple pedestrians.
Parameters: `imgsz=640`, `conf=0.25`, `iou=0.5`, `max_det=100`, `tracker=bytetrack.yaml`.
Visualization: `--show` disabled (when enabled, FPS decreases due to rendering).
Measurement: FPS computed with exponential moving average (EMA, $\alpha = 0.9$) and total elapsed frames.

Method

The benchmarking script streams the input video into YOLOv8 tracking and measures both:

- **Pipeline FPS**: End-to-end frame processing (decode, inference, tracking, postprocess).
- **Inference FPS**: Inference-only speed reported by Ultralytics.

The experiment was repeated under:

- **UTM** (ARM Ubuntu, 7 vCPUs),
- **Parallels** (ARM Ubuntu, 2/4/7 vCPUs),
- **macOS native** (CPU, MPS, CoreML).

Results

Observations.

- **CoreML** delivers the best end-to-end FPS (≈ 39.5).
- **MPS** achieves extremely high inference speed ($\approx$ 162 FPS), but total pipeline FPS is limited by CPU-GPU synchronization and tracking overhead.
- **macOS CPU** significantly outperforms virtualized environments.

Table 1: YOLOv8 `model.track` results on the same 341-frame video.

Environment (Config)	**Avg FPS (pipeline)**	**EMA pipeline**	**EMA infer**
UTM (7 cores)	10.2	12.6	13.5
Parallels (2 cores)	11.9	15.0	15.9
Parallels (4 cores)	12.8	16.6	17.8
Parallels (7 cores)	12.6	16.0	17.1
macOS CPU	30.3	52.5	70.1
macOS MPS	25.6	50.0	**162.3**
macOS CoreML	**39.5**	**81.7**	138.3

- Increasing vCPUs in Parallels from 2 to 7 yields marginal gains, suggesting bottlenecks are not purely parallelizable.

Conclusion

For Apple Silicon (M4 Pro), CoreML provides the best balance of speed and integration for YOLOv8 tracking workloads. Native execution (CPU/MPS/CoreML) is far superior to virtualized (UTM/Parallels) performance. Developers targeting real-time tracking on macOS should prioritize CoreML deployment.