Analysis of ByteTrack and YOLOX on Various Input Sizes

Sara Larson and Ethan Sims

Selected Application

- YOLOX
  - Repository: https://github.com/Megvii-BaseDetection/YOLOX/blob/main/README.md
  - Downloaded YOLOX-s weights
    - Best for real-time performance
- ByteTrack
  - Reminder: Multi-Object Tracking Algorithm, uses high and low scoring detection confidences
  - Repository: https://github.com/mikel-brostrom/boxmot
- MOT20 Dataset
  - Full dataset download: https://motchallenge.net/data/MOT20/
  - Utilized training data (4 videos): MOT20-01, MOT20-02, MOT20-03, MOT20-05

MOT20 Training Set

Sample	Name	FPS	Resolution	Length	Tracks	Boxes
	MOT20-05	25	1654x1080	3315 (02:13)	1211	751330
	MOT20-03	25	1173x880	2405 (01:36)	735	356728
	MOT20-02	25	1920x1080	2782 (01:51)	296	202215
	MOT20-01	25	1920x1080	429 (00:17)	90	26647
	Total			8931 frm. (357 s.)	2332	1336920

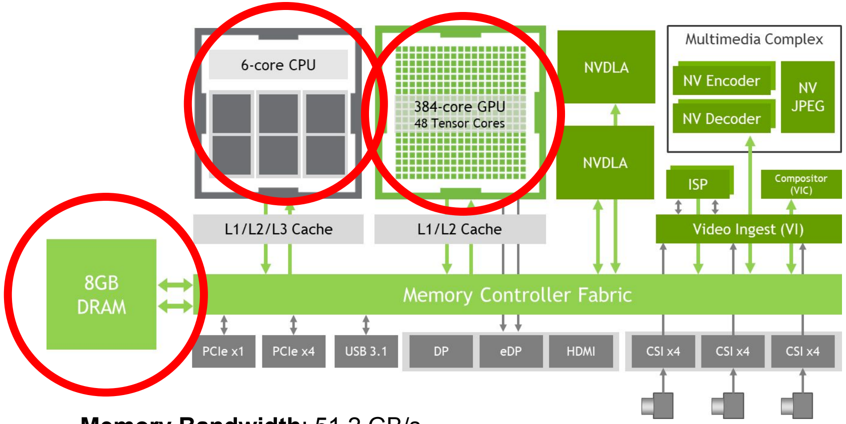
Parallelization

- Frames must be processed in order
  - Cannot process two frames at once
- Detection must happen before tracking
  - Detect all frames (YOLOX)
  - Then track all detections (ByteTrack)
- Pixels in each frame can be processed in parallel
  - Detection
- Each detection box can be processed in parallel
  - Tracking

Targeted Accelerator

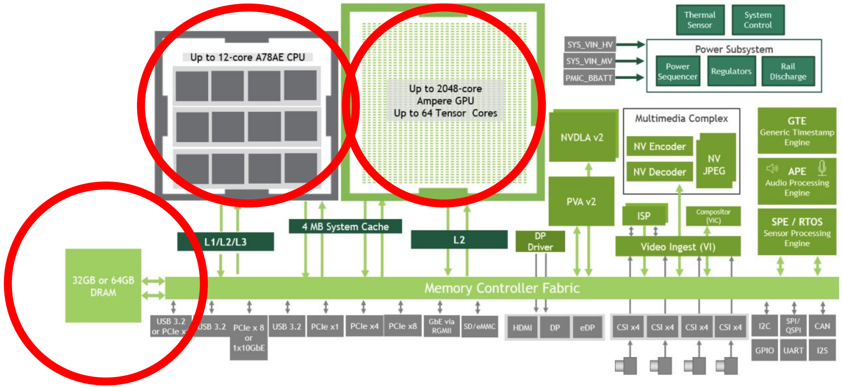
- NVIDIA Xavier NX
  - Symmetric CPU: 6 Camel Cores
  - GPU: 384 CUDA Cores, 48 Tensor Cores Volta Generation
- NVIDIA Orin AGX
  - Symmetric CPU: 12 Cortex-A78AE
  - GPU: 2048 CUDA Cores, 64 Tensor Cores Ampere Generation
- Preliminary Notes:
  - Xavier: **Fewer CUDA cores**, lower power, **less expensive** than Orin AGX
  - Orin AGX: **More CUDA cores**, higher power, **more expensive** than Xavier NX
  - (***foreshadowing…***)

Xavier NX Functional Block Diagram



Memory Bandwidth: 51.2 GB/s

Figure 9: Jetson AGX Orin Series Functional Block Diagram



Memory Bandwidth: 204.8 GB/s

Acceleration Framework

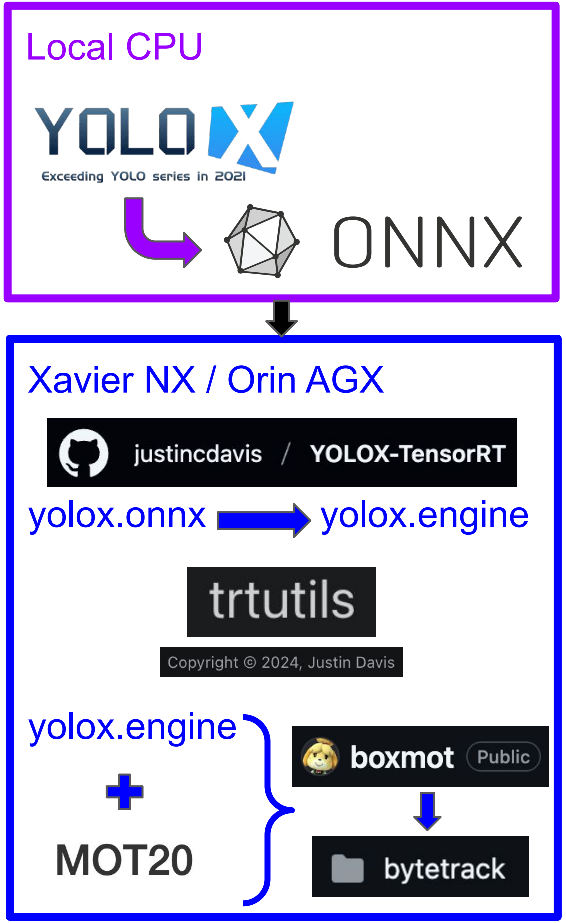
- TensorRT / ONNX
  - Exported PyTorch model as ONNX
  - TensorRT used ONNX model to construct YOLOX model for NVIDIA devices
- CUDA
  - Both Xavier NX and Orin AGX: pip install cuda-python==11.*
  - Creating YOLOX model:
    - # initialize YOLOX model with CUDA
      yolo = YOLOX("yolox.engine", preprocessor="cuda")

    - # preprocess the image
      tensor, ratio, padding = yolo.preprocess(img, method="cuda", no_copy=True)

  - **TL;DR: Image processing done on GPU with CUDA, Object detection done on GPU with CUDA**

Overall Pipeline

- On local CPU:
  - Clone YOLOX
  - Export YOLOX model to ONNX format for TensorRT usage
  - scp yolox.onnx to Xavier NX
- SSH'ed to Xavier NX:
  - Clone YOLOX-TensorRT
  - Install trtutils (for using TensorRT to run YOLOX model)
  - Export yolox.onnx to TensorRT engine: yolox.engine
  - YOLOX inference (detection) on MOT20 videos using TensorRT engine with CUDA-accelerated preprocessing and tracking
  - Create Python v3.9 virtual environment
  - Install boxmot (tracking package)
  - Create ByteTrack object and feed in YOLOX outputted detections
- Orin AGX:
  - Nearly same process, needed a different yolox.engine (specific to device)



Challenges with Frameworks

- torch and torchvision
  - Jetson devices use custom firmware and drivers
  - Pre-built PyTorch and TorchVision versions (e.g., from `pip`) often don’t match Jetson’s CUDA version
  - We did PyTorch → ONNX → TensorRT for running on GPU
  - TensorRT 2x-10x faster than PyTorch on Jetson hardware
    - In our case, for running trained neural network model (YOLOX)

Running YOLOX model:

TensorRT

PyTorch

- NVRTC (NVIDIA Runtime Compilation)
  - CUDA library for compiling CUDA C++ code at runtime, opposed to build-time (runtime better for flexibility, performance)
  - Issues with version dependencies
    - RuntimeError: Failed to dlopen libnvrtc.so.12
  - CUDA-python v12 → **cuda-python==11.***

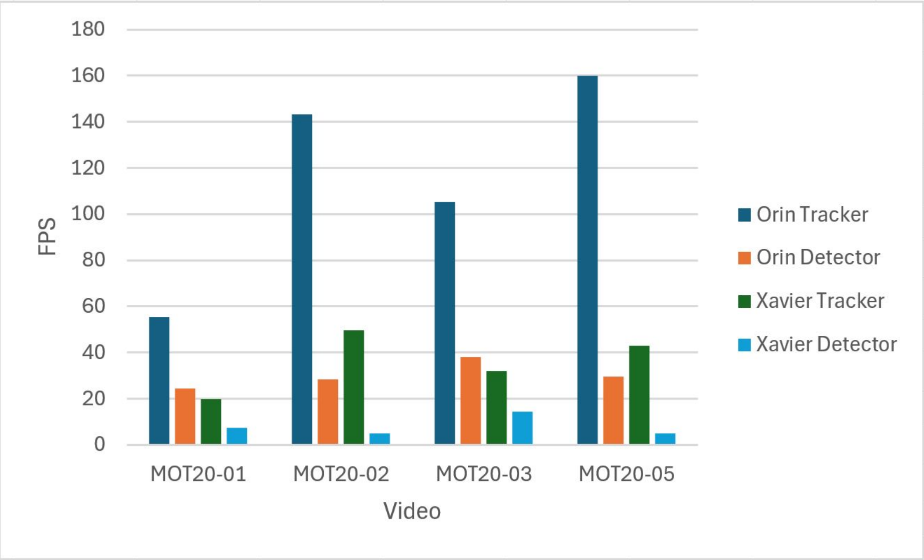
Performance Metrics

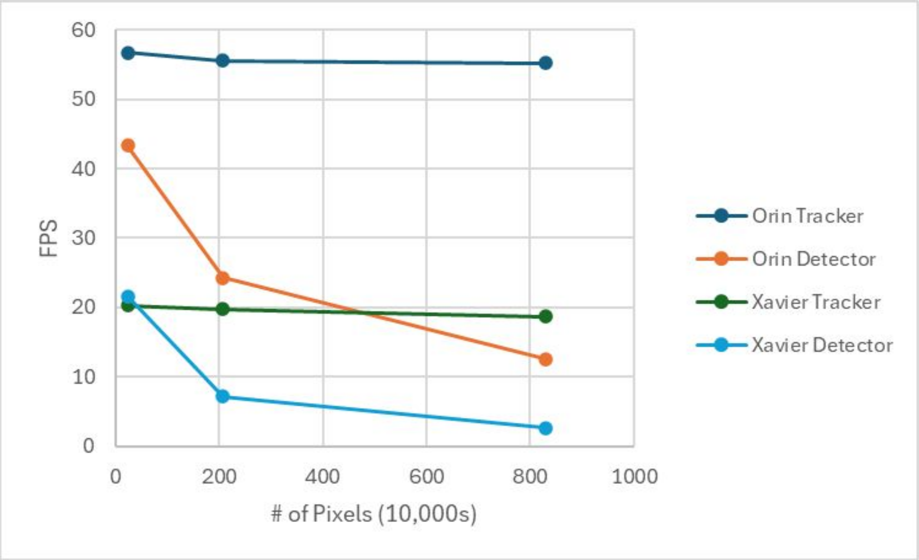
- Time/FPS
- Memory Usage
- Utilization of CPU and GPU
- CPU temperatures

Across different resolutions and videos

Measured separately for detector and tracker

Framerate

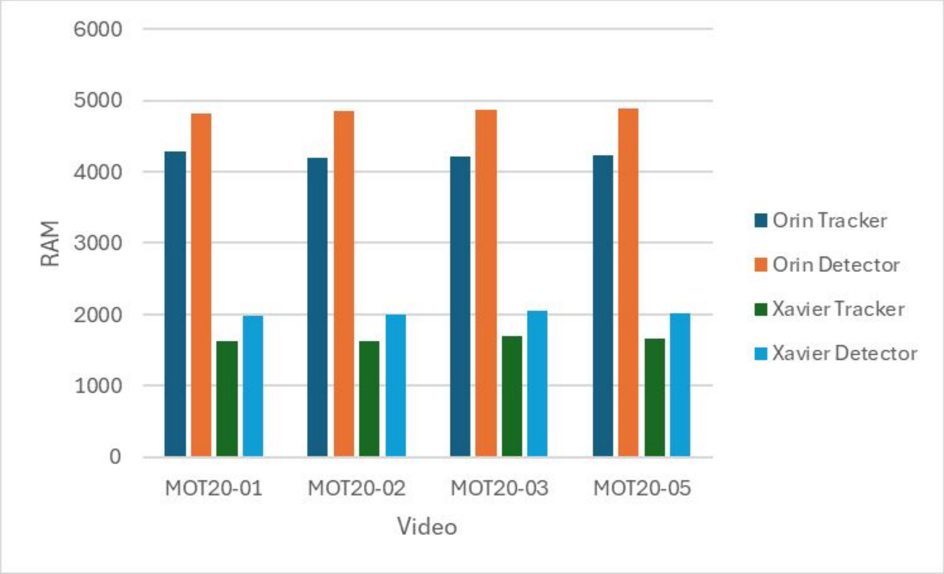




- Tracker is much faster
- Tracker is robust to resolution
- Tracker and detector heavily affected by complexity of scene

Memory Usage

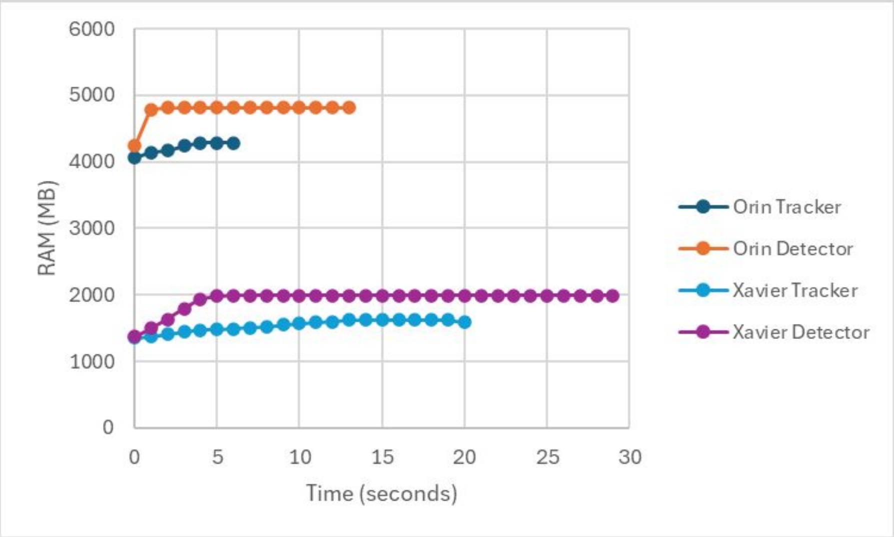
- Orin uses more memory
  - It has more memory to use
- Detector uses (slightly) more memory
- No notable differences for different videos/resolutions



*Note: This is max RAM, but it is very comparable to average RAM

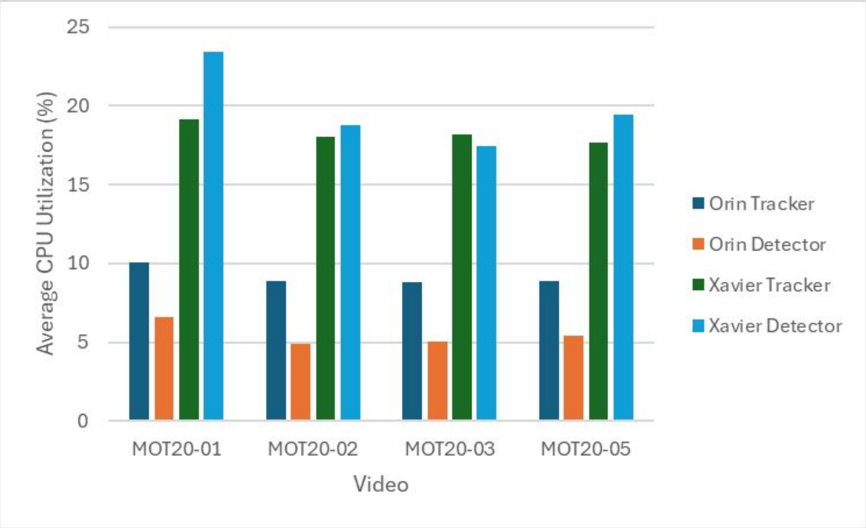
Memory Over Time

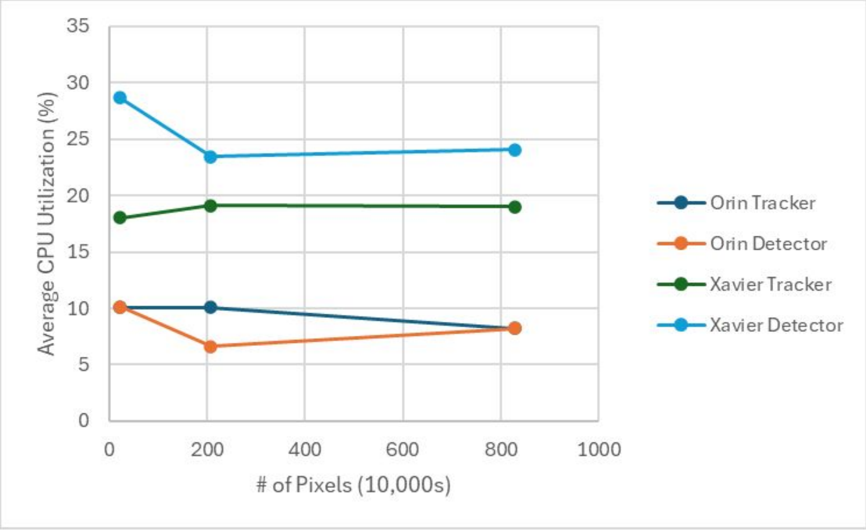
- Memory stays very steady after initial set up



CPU Usage

- Does not vary much based on input
- Much higher on Xavier (because Xavier's CPU capacity is lower)

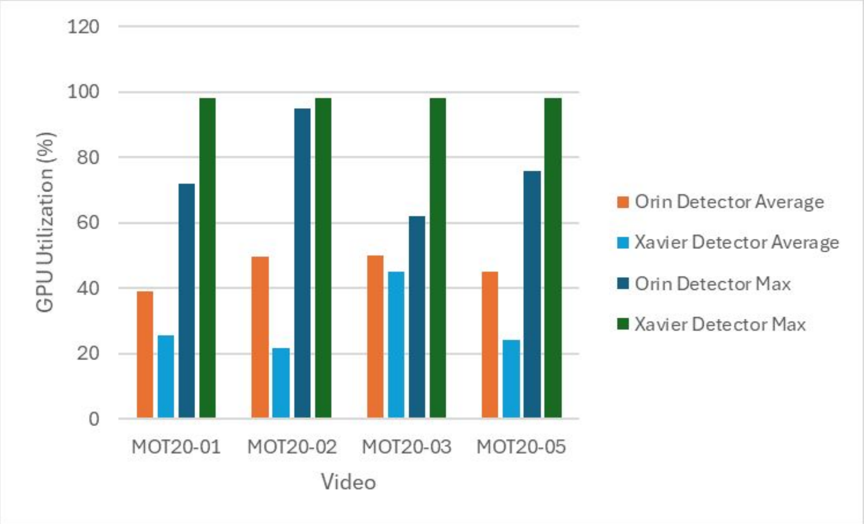


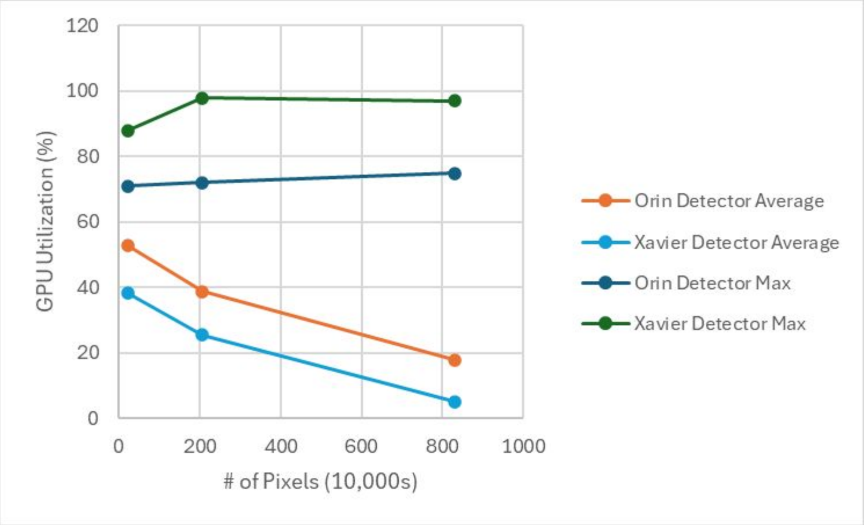


GPU Usage

- More pixels decreases average utilization
- Lower average usage on Xavier (with a weaker GPU)

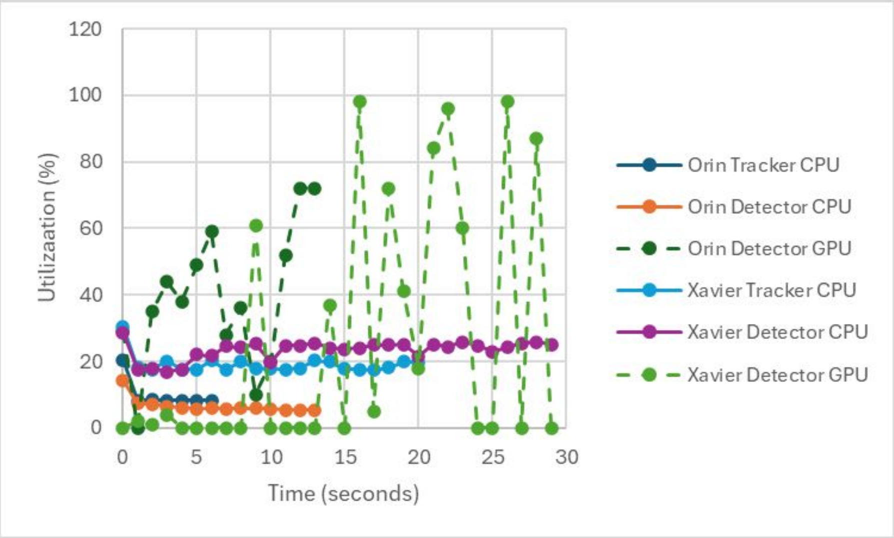
Both of these findings seem counter-intuitive, could be because CPU actions take up more time





Processor Utilization Over Time

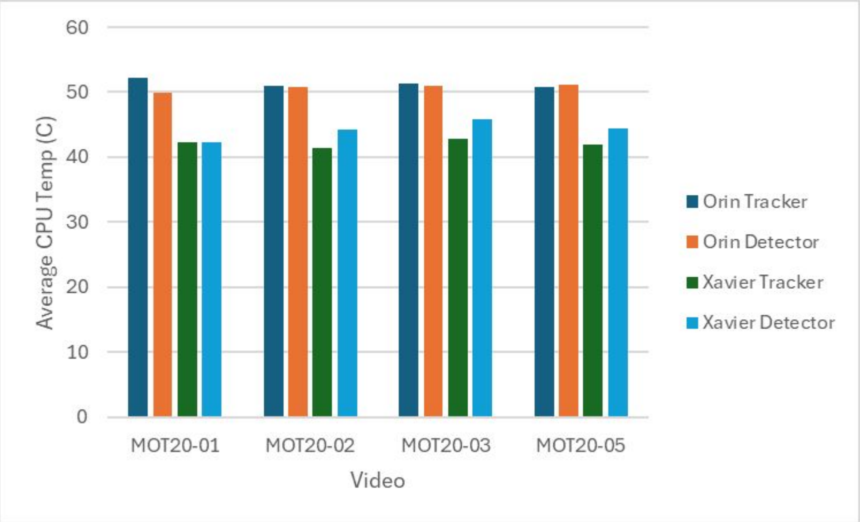
- GPUs very sporadic
- CPUs most utilized at start of tracking or detecting
    - Setup tasks

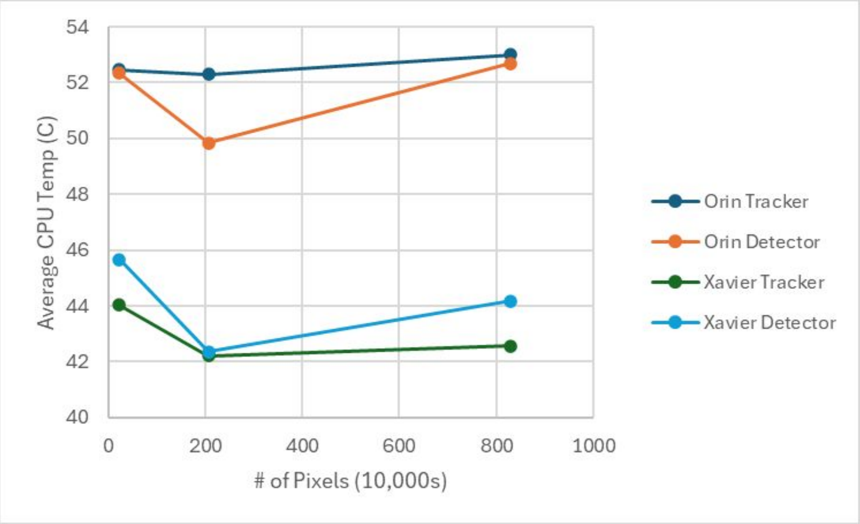


Temperature

- Orin runs hotter
  - More powerful CPU
- Seems to be an optimal resolution around 2,000,000 pixels (1080p)

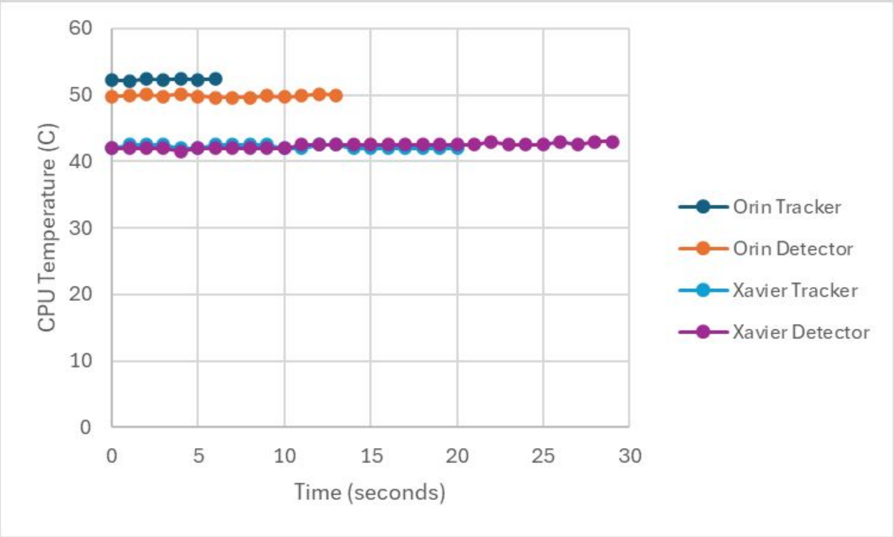
*Note: ~2°C difference between average and max





Temperature Over Time

- Very steady over time
- No significant findings here



Future Directions and Takeaways

- CPU parts may be a bottleneck in detector
  - GPU has a lot of idle time
  - Future work could aim to improve parallelization if possible
- Detector takes up the most time (almost 2x tracker time)
  - Future work could look into faster detectors
  - Initial detection could use slower, more accurate detector. Then subsequent detections could use faster, less accurate detectors
- Future work could investigate domain-specific hardware
  - Special hardware could be developed/used for detections, tracker could still use CPU