Object Tracking Comparison Report

YOLOv10n vs YOLOv11n vs YOLOv12n

1. Introduction

This report presents a comprehensive comparison of three lightweight YOLO model variants (YOLOv10n, YOLOv11n, and YOLOv12n) for object tracking tasks. The primary focus of this evaluation is on inference speed, which is a critical factor for applications requiring real-time processing or deployment on resource-constrained devices.

Each model was evaluated using identical input video and processing conditions to ensure a fair comparison. The "n" (nano) variants of these models are specifically designed for speed and efficiency while maintaining acceptable detection accuracy.

2. Methodology

2.1 Experimental Setup

- **Test Video**: A standard video was processed through each model to evaluate tracking performance
- **Hardware Configuration**: All tests were conducted on the same CPU-based system
- **Implementation Framework**: Ultralytics YOLO implementation with PyTorch backend
- **Tracking Method**: Each model used the same tracking algorithm to ensure differences were due to the detection model only
- **Metrics Collection**: Performance metrics were automatically collected during processing and saved for analysis

2.2 Models

All models used in this comparison are "nano" variants of YOLO architectures:

- **YOLOv10n**: The smallest variant of YOLOv10, optimized for speed with reduced parameters
- **YOLOv11n**: The nano version of YOLOv11, building on YOLOv10 with architectural improvements
- **YOLOv12n**: The latest nano YOLO variant, incorporating the newest design principles

These models share similar design philosophies but differ in their specific architectural details, optimization techniques, and parameter counts.

2.3 Evaluation Metrics

- **Frames Per Second (FPS)**: The average number of video frames processed per second

- **Total Processing Time**: The time taken to process the entire video (in seconds)
- **Execution Time**: The total wall clock time for the entire process including I/O operations (in seconds)

3. Results

3.1 Performance Metrics

The following table summarizes the performance metrics for each model:

Model	Average FPS	Processing Time (s)	Execution Time (s)
YOLOv10n	4.59	205.30	234.11
YOLOv11n	5.32	176.95	216.80
YOLOv12n	4.88	193.16	230.41

Speed Comparison:

- **YOLOv11n** demonstrated the highest processing speed at **5.32 FPS**, making it approximately **15.9%** faster than YOLOv10n and **9.0%** faster than YOLOv12n
- **YOLOv12n** achieved **4.88 FPS**, positioning it in the middle of the three models
- **YOLOv10n** was the slowest at **4.59 FPS**

Processing Time:

- **YOLOv11n** completed the video processing in **176.95 seconds**, the fastest among the three models
- **YOLOv12n** required **193.16 seconds**, about 9.2% longer than YOLOv11n
- **YOLOv10n** needed **205.30 seconds**, approximately 16.0% longer than YOLOv11n

Execution Overhead:

- All models showed similar patterns of execution overhead (the difference between total execution time and actual processing time)
- YOLOv10n: 28.81 seconds of overhead (12.3% of execution time)
- YOLOv11n: 39.85 seconds of overhead (18.4% of execution time)

- YOLOv12n: 37.25 seconds of overhead (16.2% of execution time)

4. Discussion

4.1 Performance Analysis

The results reveal interesting patterns in the progression of YOLO models:

1. **YOLOv11n Efficiency**: YOLOv11n demonstrated superior performance in terms of inference speed, suggesting significant optimization improvements over YOLOv10n. These improvements likely stem from architectural refinements, more efficient convolution operations, or better utilization of computational resources.
2. **YOLOv12n Performance**: Interestingly, YOLOv12n did not continue the trend of improved speed, performing slower than YOLOv11n but faster than YOLOv10n. This could indicate that YOLOv12n prioritizes other improvements (possibly detection accuracy, handling complex scenes, or reducing false positives) at a slight cost to raw processing speed.
3. **Execution Overhead**: The differences in overhead between processing time and total execution time suggest variations in model loading times, memory management efficiency, or I/O handling. YOLOv11n showed the highest percentage of overhead, which may indicate more complex initialization steps despite its faster inference.

4.2 Practical Implications

The performance differences observed have several implications for real-world applications:

1. **Real-time Processing**: None of the models achieved true real-time performance (typically considered 30+ FPS) on the tested CPU hardware. For real-time applications, GPU acceleration would be necessary.
2. **Resource-constrained Environments**: In scenarios where processing resources are limited, YOLOv11n would be the preferred choice based purely on speed considerations.
3. **Model Selection Tradeoffs**: The selection between YOLOv11n and YOLOv12n might depend on whether the application prioritizes speed (favoring YOLOv11n) or potentially better detection characteristics (which might favor YOLOv12n, though this would need verification through accuracy testing).

5. Conclusion

Based on our analysis of inference speed:

1. **YOLOv11n** emerged as the fastest model among the three tested variants, making it the best choice for speed-critical applications.

2. The performance regression in **YOLOv12n** relative to YOLOv11n suggests a potential shift in design priorities in the newer model, possibly favoring detection quality over raw speed.
3. Despite being the oldest model in the comparison, **YOLOv10n** still provides reasonable performance that may be sufficient for less demanding applications.

6. Limitations and Future Work

6.1 Limitations

This study focused exclusively on inference speed and did not evaluate:

- Detection accuracy and precision
- Ability to track small or occluded objects
- Performance across different object classes
- Resilience to challenging lighting or weather conditions
- Energy consumption and memory footprint

6.2 Future Work

To build upon this comparison, future work could include:

- Evaluation of detection accuracy metrics (mAP, precision, recall)
- Testing on GPU hardware to evaluate real-time performance capabilities
- Comparing tracking stability across dynamic scenes
- Measuring memory usage during inference
- Analyzing the impact of different confidence thresholds on speed and accuracy

7. References

1. Ultralytics YOLO. (2023). https://github.com/ultralytics/ultralytics
2. Redmon, J., & Farhadi, A. (2018). YOLOv3: An incremental improvement. arXiv preprint arXiv:1804.02767.
3. Bochkovskiy, A., Wang, C. Y., & Liao, H. Y. M. (2020). YOLOv4: Optimal speed and accuracy of object detection. arXiv preprint arXiv:2004.10934.
4. Wang, C. Y., Liao, H. Y. M., Wu, Y. H., Chen, P. Y., Hsieh, J. W., & Yeh, I. H. (2020). CSPNet: A new backbone that can enhance learning capability of CNN. arXiv preprint arXiv:2011.11929.

Appendix: Technical Details

A. Software Environment

- **Python**: 3.8+
- **PyTorch**: 2.0+
- **Ultralytics**: 8.0.0+
- **OpenCV**: 4.5+