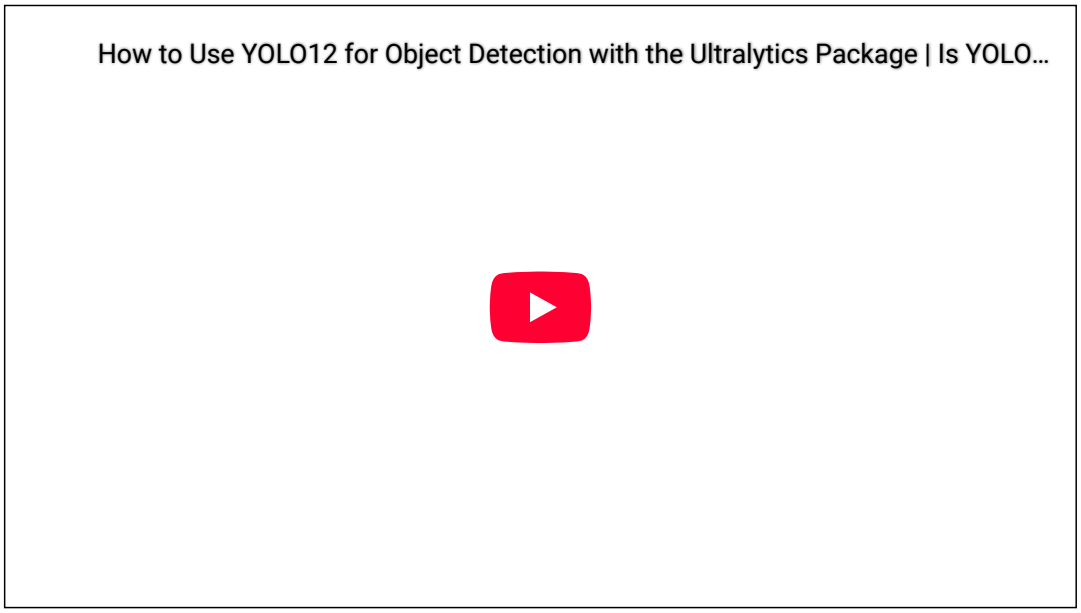
YOLO12: Attention-Centric Object Detection

Overview

YOLO12 introduces an attention-centric architecture that departs from the traditional CNN-based approaches used in previous YOLO models, yet retains the real-time inference speed essential for many applications. This model achieves state-of-the-art object detection accuracy through novel methodological innovations in attention mechanisms and overall network architecture, while maintaining real-time performance.



Watch: How to Use YOLO12 for Object Detection with the Ultralytics Package | Is YOLO12 Fast or Slow? 🚀

Key Features

- **Area Attention Mechanism**: A new self-attention approach that processes large receptive fields efficiently. It divides feature maps into *l* equal-sized regions (defaulting to 4), either horizontally or vertically, avoiding complex operations and maintaining a large effective receptive field. This significantly reduces computational cost compared to standard self-attention.
- **Residual Efficient Layer Aggregation Networks (R-ELAN)**: An improved feature aggregation module based on ELAN, designed to address optimization challenges, especially in larger-scale attention-centric models. R-ELAN introduces:
  - Block-level residual connections with scaling (similar to layer scaling).
  - A redesigned feature aggregation method creating a bottleneck-like structure.
- **Optimized Attention Architecture**: YOLO12 streamlines the standard attention mechanism for greater efficiency and compatibility with the YOLO framework. This includes:
  - Using FlashAttention to minimize memory access overhead.
  - Removing positional encoding for a cleaner and faster model.
  - Adjusting the MLP ratio (from the typical 4 to 1.2 or 2) to better balance computation between attention a layers.
  - Reducing the depth of stacked blocks for improved optimization.

- Leveraging convolution operations (where appropriate) for their computational efficiency.
- Adding a 7x7 separable convolution (the "position perceiver") to the attention mechanism to implicitly encode positional information.

- **Comprehensive Task Support**: YOLO12 supports a range of core computer vision tasks: object detection, instance segmentation, image classification, pose estimation, and oriented object detection (OBB).
- **Enhanced Efficiency**: Achieves higher accuracy with fewer parameters compared to many prior models, demonstrating an improved balance between speed and accuracy.
- **Flexible Deployment**: Designed for deployment across diverse platforms, from edge devices to cloud infrastructure.

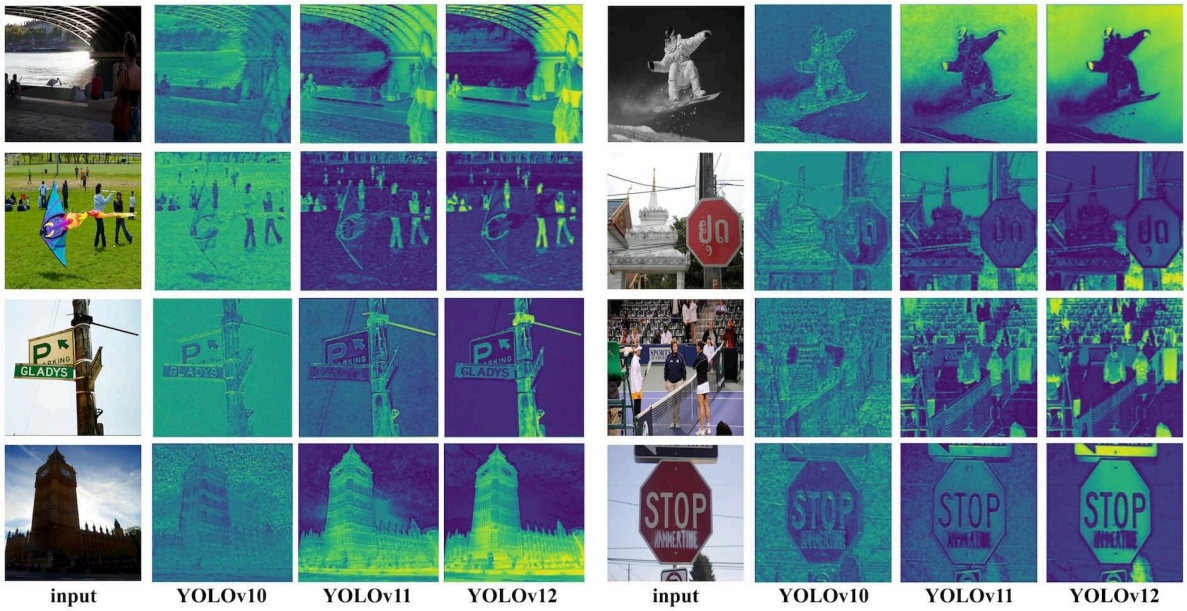


Figure 5. **Comparison of heat maps between YOLOv10 [53], YOLOv11 [28], and the proposed YOLOv12.** Compared to the advanced YOLOv10 and YOLOv11, YOLOv12 demonstrates a clearer perception of objects in the image. All the results are obtained using the X scale models. *Zoom in to compare the details.*

Supported Tasks and Modes

YOLO12 supports a variety of computer vision tasks. The table below shows task support and the operational modes (Inference, Validation, Training, and Export) enabled for each:

Model Type	Task	Inference	Validation	Training	Export
YOLO12	Detection	✅	✅	✅	✅
YOLO12-seg	Segmentation	✅	✅	✅	✅
YOLO12-pose	Pose	✅	✅	✅	✅
YOLO12-cls	Classification	✅	✅	✅	✅
YOLO12-obb	OBB	✅	✅	✅	✅

Performance Metrics

YOLO12 demonstrates significant accuracy improvements across all model scales, with some trade-offs in speed compared to the *fastest* prior YOLO models. Below are quantitative results for object detection on the COCO validation dataset:

Detection Performance (COCO val2017)

Performance

Detection (COCO)

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed T4 TensorRT (ms)	params (M)	FLOPs (B)	Comparison (mAP/Speed)
YOLO12n	640	40.6	-	1.64	2.6	6.5	+2.1%/-9% (vs. YOLOv10n)
YOLO12s	640	48.0	-	2.61	9.3	21.4	+0.1%/+42% (vs. RT-DETRv2)
YOLO12m	640	52.5	-	4.86	20.2	67.5	+1.0%/-3% (vs. YOLO11m)
YOLO12l	640	53.7	-	6.77	26.4	88.9	+0.4%/-8% (vs. YOLO11l)
YOLO12x	640	55.2	-	11.79	59.1	199.0	+0.6%/-4% (vs. YOLO11x)

- Inference speed measured on an NVIDIA T4 GPU with TensorRT FP16 precision.
- Comparisons show the relative improvement in mAP and the percentage change in speed (positive indicates faster; negative indicates slower). Comparisons are made against published results for YOLOv10, YOLO11, and RT-DETR where available.

Usage Examples

This section provides examples for training and inference with YOLO12. For more comprehensive documentation on these and other modes (including Validation and Export), consult the dedicated Predict and Train pages.

The examples below focus on YOLO12 Detect models (for object detection). For other supported tasks (segmentation, classification, oriented object detection, and pose estimation), refer to the respective task-specific documentation: Segment, Classify, OBB, and Pose.

Example

Python

Pretrained `*.pt` models (using PyTorch) and configuration `*.yaml` files can be passed to the `YOLO()` class to create a model instance in Python:

```python
from ultralytics import YOLO

# Load a COCO-pretrained YOLO12n model
model = YOLO("yolo12n.pt")

# Train the model on the COCO8 example dataset for 100 epochs
results = model.train(data="coco8.yaml", epochs=100, imgsz=640)

# Run inference with the YOLO12n model on the 'bus.jpg' image
results = model("path/to/bus.jpg")
```

CLI

Command Line Interface (CLI) commands are also available:

```bash
# Load a COCO-pretrained YOLO12n model and train on the COCO8 example dataset for 100 epochs
yolo train model=yolo12n.pt data=coco8.yaml epochs=100 imgsz=640

# Load a COCO-pretrained YOLO12n model and run inference on the 'bus.jpg' image
yolo predict model=yolo12n.pt source=path/to/bus.jpg
```

Key Improvements

1. **Enhanced Feature Extraction**:
   - **Area Attention**: Efficiently handles large receptive fields, reducing computational cost.
   - **Optimized Balance**: Improved balance between attention and feed-forward network computations.
   - **R-ELAN**: Enhances feature aggregation using the R-ELAN architecture.
2. **Optimization Innovations**:
   - **Residual Connections**: Introduces residual connections with scaling to stabilize training, especially in larger models.
   - **Refined Feature Integration**: Implements an improved method for feature integration within R-ELAN.
   - **FlashAttention**: Incorporates FlashAttention to reduce memory access overhead.
3. **Architectural Efficiency**:
   - **Reduced Parameters**: Achieves a lower parameter count while maintaining or improving accuracy compared to many previous models.
   - **Streamlined Attention**: Uses a simplified attention implementation, avoiding positional encoding.
   - **Optimized MLP Ratios**: Adjusts MLP ratios to more effectively allocate computational resources.

Requirements

The Ultralytics YOLO12 implementation, by default, *does not require* FlashAttention. However, FlashAttention can be optionally compiled and used with YOLO12. To compile FlashAttention, one of the following NVIDIA GPUs is needed:

- Turing GPUs (e.g., T4, Quadro RTX series)
- Ampere GPUs (e.g., RTX30 series, A30/40/100)
- Ada Lovelace GPUs (e.g., RTX40 series)
- Hopper GPUs (e.g., H100/H200)

Citations and Acknowledgements

If you use YOLO12 in your research, please cite the original work by University at Buffalo and the University of Chinese Academy of Sciences:

BibTeX

```
@article{tian2025yolov12,
  title={YOLOv12: Attention-Centric Real-Time Object Detectors},
  author={Tian, Yunjie and Ye, Qixiang and Doermann, David},
  journal={arXiv preprint arXiv:2502.12524},
  year={2025}
}

@software{yolo12,
  author = {Tian, Yunjie and Ye, Qixiang and Doermann, David},
  title = {YOLOv12: Attention-Centric Real-Time Object Detectors},
  year = {2025},
  url = {https://github.com/sunsmarterjie/yolov12},
  license = {AGPL-3.0}
}
```

FAQ

How does YOLO12 achieve real-time object detection while maintaining high accuracy?

YOLO12 incorporates several key innovations to balance speed and accuracy. The Area Attention mechanism efficiently processes large receptive fields, reducing computational cost compared to standard self-attention. The Residual Efficient Layer Aggregation Networks (R-ELAN) improve feature aggregation, addressing optimization challenges in larger attention-centric models. Optimized Attention Architecture, including the use of FlashAttention and removal of positional encoding, further enhances efficiency. These features allow YOLO12 to achieve state-of-the-art accuracy while maintaining the real-time inference speed crucial for many applications.

What computer vision tasks does YOLO12 support?

YOLO12 is a versatile model that supports a wide range of core computer vision tasks. It excels in object detection, instance segmentation, image classification, pose estimation, and oriented object detection (OBB) (see details). This comprehensive task support makes YOLO12 a powerful tool for diverse applications, from robotics and autonomous driving to medical imaging and industrial inspection. Each of these tasks can be performed in Inference, Validation, Training, and Export modes.

How does YOLO12 compare to other YOLO models and competitors like RT-DETR?

YOLO12 demonstrates significant accuracy improvements across all model scales compared to prior YOLO models like YOLOv10 and YOLO11, with some trade-offs in speed compared to the *fastest* prior models. For example, YOLO12n achieves a +2.1% mAP improvement over YOLOv10n and +1.2% over YOLO11n on the COCO val2017 dataset. Compared to models like RT-DETR, YOLO12s offers a +1.5% mAP improvement and a substantial +42% speed increase. These metrics highlight YOLO12's strong balance between accuracy and efficiency. See the performance metrics section for detailed comparisons.

What are the hardware requirements for running YOLO12, especially for using FlashAttention?

By default, the Ultralytics YOLO12 implementation does *not* require FlashAttention. However, FlashAttention can be optionally compiled and used with YOLO12 to minimize memory access overhead. To compile FlashAttention, one of the following NVIDIA GPUs is needed: Turing GPUs (e.g., T4, Quadro RTX series), Ampere GPUs (e.g., RTX30 series, A30/40/100), Ada Lovelace GPUs (e.g., RTX40 series), or Hopper GPUs (e.g., H100/H200). This flexibility allows users to leverage FlashAttention's benefits when hardware resources permit.

Where can I find usage examples and more detailed documentation for YOLO12?

This page provides basic usage examples for training and inference. For comprehensive documentation on these and other modes, including Validation and Export, consult the dedicated Predict and Train pages. For task-specific information (segmentation, classification, oriented object detection, and pose estimation), refer to the respective documentation: Segment, Classify, OBB, and Pose. These resources provide in-depth guidance for effectively utilizing YOLO12 in various scenarios.