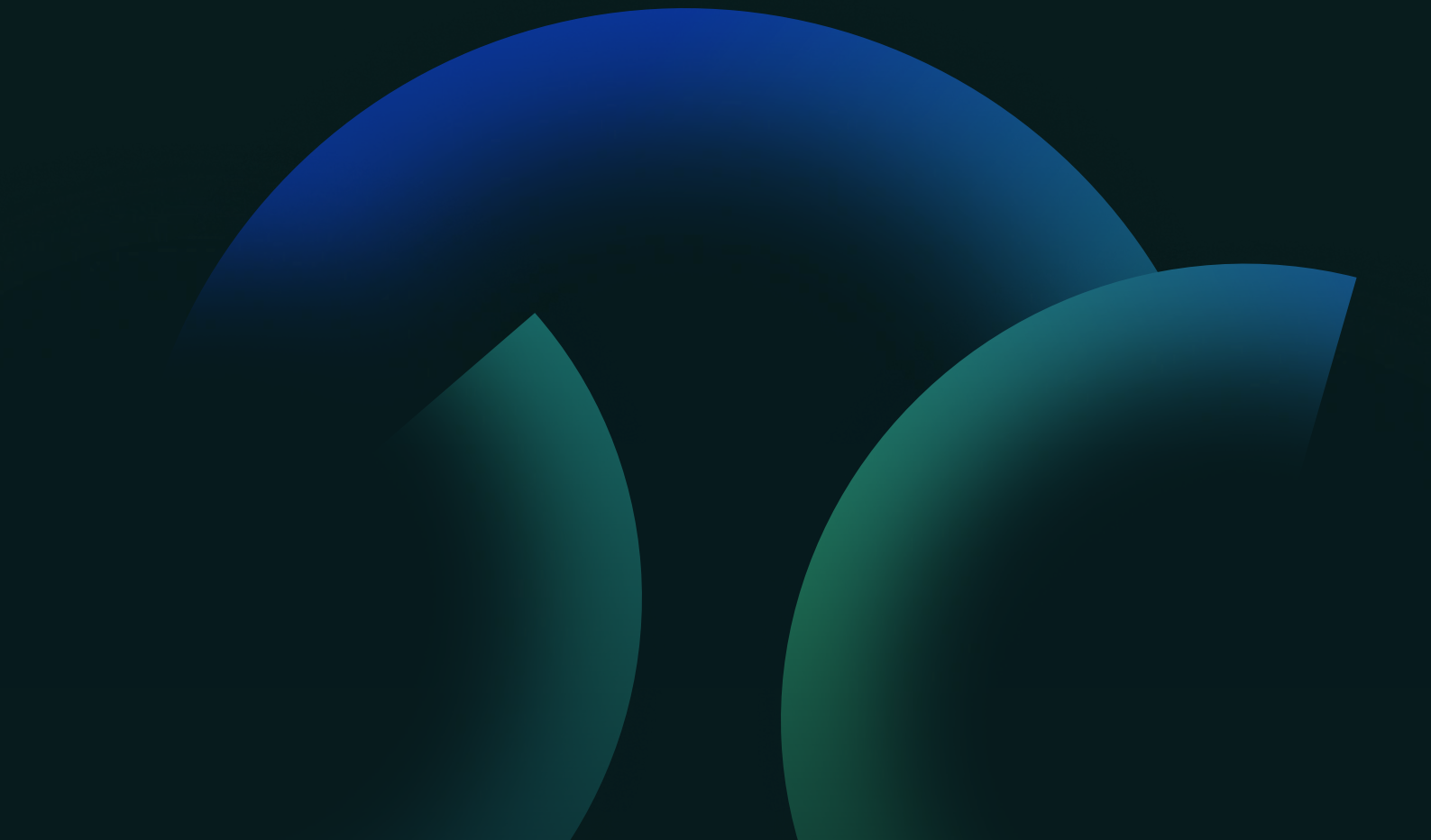
LIGHTLY

Computer Vision Architecture Decision Tree

How to pick the right model



Computer Vision Architecture Decision Tree

This short guide shows you how to go from use-case to model with a simple path:

- Choose your task
- Apply your constraint
- Select from strong defaults

It covers the common workloads, object detection, instance/semantic/panoptic segmentation, classification (including zero-shot), and keypoint detection, and offers options for accuracy targets, real-time needs, and edge limits. The Year and Backbone columns help you judge recency and fit for your stack.

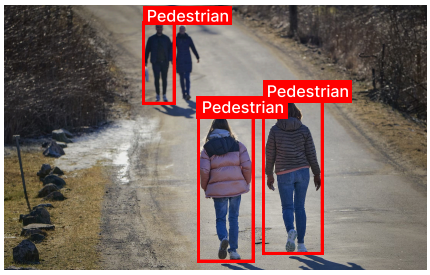
> Picking a vision model should be a decision, not a debate.

Once you have chosen a model, use the checklist further below to verify that the model fits all your needs.

Task	Constraint	Model
Object Detection	High Accuracy	Co-DETR DINOv3 + Plain-DETR
	Real Time	RT-DETRv2 RF-DETR
	Real Time on Edge Devices	YOLOv5 YOLO11
Instance Segmentation	High Accuracy	Co-DETR
	Real Time	DINOv3 + EoMT
	Real Time on Edge Devices	YOLOv5-seg
Semantic Segmentation	High Accuracy	DINOv3 + Mask2Former
	Real Time	DINOv3 + EoMT
	Real Time on Edge Devices	MobileNetV4 + U-Net
Panoptic Segmentation	High Accuracy	DINOv3 + Mask2Former
	Real Time	DINOv3 + EoMT

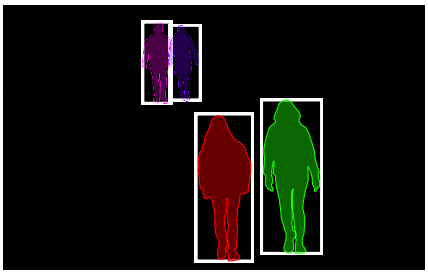
Task	Constraint	Model
Classification	High Accuracy	PE-Core SigLIP 2
	Real Time	PE-Core (B/L) SigLIP 2 (S/B/L)
	Real Time on Edge Devices	MobileNetV4 ConvNeXt V2
Zero-Shot Classification	High Accuracy	PE-Core SigLIP 2
	Real Time	PE-Core (B/L) SigLIP 2 (S/B/L)
	Real Time on Edge Devices	MobileCLIP2
Keypoint Detection	High Accuracy	ViTPose MogaNet
	Real Time	ViTPose MogaNet
	Real Time on Edge Devices	RTMPose YOLO11-pose

Image classification



Object detection

Semantic segmentation



Instance segmentation

Panoptic segmentation

Keypoint detection

Choosing the Right Vision Model - Checklist

Use the checklist to confirm a candidate model is fit for your problem and environment. It covers all important aspects of a model, from accuracy to deployment and maintenance. We provide detailed descriptions for each task and tips on what to look out for further below.

Model Property	Description	Check
Task	Supports the required task (detection, segmentation, classification, ...)	
Accuracy	Reaches required accuracy	
Size	Fits into memory of target hardware	
Latency	Runs fast enough	
Open Source	Code and weights are open source	
License	Code and weights license allows commercial use	
Framework	Choice of machine learning framework (PyTorch, TensorFlow, Keras, JAX)	
Deployment	...	
Python Package	Is available as Python package	
API	Is available as API endpoint	
Real-time/Edge	Can be exported to ONNX/TensorRT	
Container	Is available as Docker image	
Cloud integration	Has cloud integration (AWS SageMaker, Vertex AI, Azure Machine Learning, Nvidia TAO)	
Maintained	Is actively maintained	
Documentation	Documentation is available and maintained	

Task

Verify that the model can solve your task. If you have to solve an object detection problem, make sure the model supports object detections. There are many different tasks and every model supports a different subset of them. Common tasks are object detection, instance segmentation, semantic segmentation, classification, and keypoint detection.

> If your task involves videos you might also want to check whether the model has special video support. This could be creating predictions based on multiple frames at the same time or object tracking over time.

Model Accuracy, Size, and Latency

Once you know a model supports your task, you have to verify whether it is good enough for your use-case. Good enough depends on how you intend to use the model.

- If the model has to run in real-time on an edge device you have to look up whether the inference latency is low enough.
- If memory is a concern you have to check the size and memory requirements of the model.
- If you are happy with the latency and memory you have to check whether the model achieves the required accuracy for your use-case.
- If not, you can usually trade higher accuracy for higher latency and larger model size.

Open Source

Check whether the model code and model weights are open source and available for download. Open source model code gives you the ability to verify the correctness of the code and the flexibility to add any modifications if required to do so.

License

The license is an important component when choosing a model architecture. Open-source licenses like MIT and Apache 2.0 provide the most flexibility and permit the model to be used for commercial purposes. Dual licenses like AGPL 3.0 combined with a commercial license are also popular.

Special attention must be paid to verify whether the model code and model weights have different licenses. Oftentimes, the model code is under a permissive license (MIT/Apache) whereas model weights are under a more restrictive license. If the model weights license is

too restrictive for your use-case, you will still be allowed to use the model code but will have to train any models from scratch. It is also recommended to check on which datasets any available model weights were trained as not all datasets might fulfill the license requirements for your use-case.

Framework

Verify in which machine learning framework the model is implemented. This is commonly one of PyTorch, TensorFlow, Keras, or JAX. We recommend choosing the framework based on how familiar the engineers in your team are with the framework. If there is no preference we recommend to go with PyTorch as it is the most widely adopted framework.

Deployment Options

Check available deployment options depending on your use-case:

- Python package: A Python package provides you with the most flexibility and is great for development, custom model training, and deployment in the cloud.
- API: Some models are also available through an API. If this is enough for your use-case then check if the model is available on any of the popular model providers. Examples are Google Cloud Vision API, Amazon Rekognition (AWS), Microsoft Azure Computer Vision, and OpenAI.
- Real-time model: If you need a real-time edge model, check that it can be exported to ONNX or TensorRT. If your edge devices have special hardware, make sure the model works on that hardware. If you require export to ONNX check whether the model supports the ONNX opset version required by your hardware.
- Container: Check if the model is available as a Docker image if you require containerized workflows.
- Cloud integrations: Check if your cloud provider supports the model if you intend to deploy it in the cloud. This could be through AWS SageMaker, Vertex AI, Azure Machine Learning, or Nvidia TAO.

Maintainance

Avoid models that are not actively maintained. Check the date of the last GitHub commit to verify that the model still receives updates. Check the number of open GitHub issues and whether maintainers answer questions. Finally, check the number of contributors, the more contributors a library has, the higher the chance it will keep being maintained.

Documentation

Documentation is another critical aspect when choosing a model architecture. Make sure that documentation is available and up-to-date or you risk spending a lot of time trying to understand and reverse engineer the model code if something goes wrong.

Conclusion

This guide has shown you how to navigate the decision process for computer vision architectures The goal is not only to match a model to a task but to make choices that keep your solutions future-proof and practical.

> Models continue to evolve rapidly, and what works best today may shift within months. By grounding your decision in clear requirements and applying a systematic verification step, you ensure that your architecture supports both current needs and long-term scalability.