To use a YOLO (You Only Look Once) exported model in C, you'd typically follow this approach:

**✅ 1. Understand the YOLO Export Format**

YOLO models are often exported in formats like:

* ONNX (.onnx)
* Darknet (.cfg + .weights)
* TensorRT engine (.engine)
* OpenVINO IR (.xml + .bin)

The format determines how you use it in C.

**✅ 2. Choose a Compatible Inference Framework**

YOLO models are not run "directly" in C—you need an inference engine that:

* Can be used in C
* Supports the format your model is in

| **Format** | **C-Compatible Framework** | **Notes** |
| --- | --- | --- |
| .onnx | ONNX Runtime C API | Lightweight, widely used |
| .cfg + .weights | [Darknet C library](https://github.com/AlexeyAB/darknet) | Original YOLO format |
| .engine | NVIDIA TensorRT C++ API | For high-performance inference on NVIDIA GPUs |
| .xml + .bin | OpenVINO C API | Intel's optimized inference engine |

**✅ 3. Example: Using Darknet in C**

If you're using YOLOv4 or similar via Darknet, here's a sample:

**a. Clone Darknet**

bash

コピーする編集する

git clone https://github.com/AlexeyAB/darknet

cd darknet

make

**b. C Code to Use YOLO**

1. #include "darknet.h"

2.

3. int main() {

4. char \*cfg = "yolov4.cfg";

5. char \*weights = "yolov4.weights";

6. char \*data = "coco.data";

7. char \*image\_file = "dog.jpg";

8.

9. network \*net = load\_network(cfg, weights, 0);

10. set\_batch\_network(net, 1);

11. srand(2222222);

12.

13. image im = load\_image\_color(image\_file, 0, 0);

14. float \*X = im.data;

15.

16. network\_predict(net, X);

17.

18. int nboxes = 0;

19. detection \*dets = get\_network\_boxes(net, im.w, im.h, 0.5, 0.5, 0, 1, &nboxes);

20.

21. // Post-process and draw boxes...

22.

23. free\_detections(dets, nboxes);

24. free\_image(im);

25. return 0;

26. }

27.

You'll need to compile this with the darknet static/shared library and include paths.

**✅ 4. Alternative: ONNX Runtime in C**

If you exported the model as ONNX:

* Use ONNX Runtime C API
* Load the model and run inference with OrtSession

Great! Here's a complete **ONNX Runtime C API example** to load and run inference on a YOLO model exported as .onnx:

**✅ Prerequisites**

**1. Install ONNX Runtime C API**

Download prebuilt ONNX Runtime C library from:

* <https://github.com/microsoft/onnxruntime/releases>

Extract it and add the include and lib paths to your build environment.

**✅ Sample C Code**

**main.c**

1. #include <stdio.h>

2. #include <stdlib.h>

3. #include <onnxruntime\_c\_api.h>

4.

5. int main() {

6. // Initialize environment

7. const OrtApi \*api = OrtGetApiBase()->GetApi(ORT\_API\_VERSION);

8. OrtEnv \*env;

9. api->CreateEnv(ORT\_LOGGING\_LEVEL\_WARNING, "yolo", &env);

10.

11. // Create session options

12. OrtSessionOptions \*session\_options;

13. api->CreateSessionOptions(&session\_options);

14. api->SetIntraOpNumThreads(session\_options, 1);

15.

16. // Load the YOLO ONNX model

17. OrtSession \*session;

18. const char \*model\_path = "yolov5.onnx"; // Replace with your model path

19. api->CreateSession(env, model\_path, session\_options, &session);

20.

21. // Print input node names and shapes

22. size\_t num\_input\_nodes;

23. OrtAllocator \*allocator;

24. api->GetAllocatorWithDefaultOptions(&allocator);

25. api->SessionGetInputCount(session, &num\_input\_nodes);

26.

27. for (size\_t i = 0; i < num\_input\_nodes; i++) {

28. char \*input\_name;

29. api->SessionGetInputName(session, i, allocator, &input\_name);

30. printf("Input %zu : name = %s\n", i, input\_name);

31. api->AllocatorFree(allocator, input\_name);

32. }

33.

34. // Cleanup

35. api->ReleaseSession(session);

36. api->ReleaseSessionOptions(session\_options);

37. api->ReleaseEnv(env);

38. return 0;

39. }

40.

**✅ Build It**

Assuming ONNX Runtime is in ./onnxruntime-linux-x64/, build with:

bash

gcc -o yolo\_infer main.c -I./onnxruntime-linux-x64/include \

-L./onnxruntime-linux-x64/lib -lonnxruntime

**🧠 Notes:**

* You’ll need to convert your input image (e.g. from OpenCV) into a float array and shape it as 1x3x640x640 or whatever your model expects.
* After SessionRun(), you'll read back the output tensor and post-process detections (usually NMS, box scaling, etc.).
* ONNX YOLO models (like YOLOv5/8) are often exported from PyTorch.