# Video Transcript: YOLO11 PyTorch to ONNX Conversion

## Introduction (0:00 - 0:30)

"Hi everyone! Today I'm demonstrating YOLO11 model conversion from PyTorch to ONNX format. I'll run inference with both models and compare the results using IoU metrics. I'll also show how I used Cursor AI to accelerate development. Let's get started!"

---

## Environment Setup (0:30 - 1:15)

"First, I'm setting up a fresh Python environment. Here are the required packages:

```
pip install ultralytics torch torchvision onnx onnxruntime opencv-python numpy pillow
```

I used Cursor AI to identify these dependencies. I simply asked: 'What packages do I need for YOLO11 and ONNX conversion?' and it provided this complete list. The main ones are ultralytics for YOLO, torch for PyTorch, and onnx/onnxruntime for conversion and inference."

---

## Code Structure (1:15 - 2:00)

"The script has three helper functions:

- **calculate_iou**: Compares bounding boxes between models
- **print_results**: Displays detection details
- **main**: Orchestrates the four-step pipeline

I asked Cursor to 'organize this into modular functions with documentation' and it suggested this clean structure. This makes debugging much easier."

---

## Helper Functions (2:00 - 3:00)

"The IoU function calculates overlap between two bounding boxes - it's key for comparing our models:

```
python
```

```
def calculate_iou(box1, box2):
    # Calculate intersection area
    inter_x_min = max(x1_min, x2_min)
    # ... find overlapping rectangle
    # Return intersection / union
```

I prompted Cursor: 'implement IoU for bounding boxes' and it generated this. An IoU of 1.0 means perfect overlap, 0.0 means no overlap.

The print_results function extracts and displays detection info - class names, confidence scores, and bounding box coordinates. This makes our console output readable."

---

## Step 1: PyTorch Inference (3:00 - 4:00)

"Now the main pipeline. Step 1 loads and runs the PyTorch model:

```
python

model_pt = YOLO("yolo11n.pt")
results_pt = model_pt(["image-2.png"])
```

I used Cursor's autocomplete here - when I typed 'model_pt = YOLO(' it suggested the correct syntax. The model runs inference and we save results to 'result_pytorch.jpg'. Then we print detailed detection info with class names and confidence scores."

---

## Step 2: ONNX Export (4:00 - 5:00)

"Step 2 converts the model to ONNX format:

```
python

model_pt.export(format="onnx")
```

That's it! One line. Ultralytics handles all the complexity. Then we validate:

```
python

onnx_model_check = onnx.load("yolo11n.onnx")
onnx.checker.check_model(onnx_model_check)
```

This ensures the exported model is valid and properly formatted. When I asked Cursor 'how to validate an ONNX model', it showed me this checker approach."

---

## Step 3: ONNX Inference (5:00 - 5:45)

"Step 3 runs inference with the ONNX model:

```python
model_onnx = YOLO("yolo11n.onnx")
results_onnx = model_onnx("image-2.png")
```

Same YOLO interface, different backend. We save results to 'result_onnx.jpg' for side-by-side comparison. The beauty of ultralytics is it handles both PyTorch and ONNX with the same API."

---

## Step 4: IoU Comparison (5:45 - 7:00)

"Step 4 compares predictions from both models:

```python
# Extract bounding boxes from both results
for box in results_pt[0].boxes:
    pt_boxes.append(box.xyxy[0].tolist())
```

Then we calculate IoU for each detection pair:

```python
iou = calculate_iou(pt_boxes[i], onnx_boxes[i])
print(f"Detection {i+1} IoU: {iou:.4f}")
```

We categorize the match quality:

- Above 0.95: Excellent match
- 0.8-0.95: Good match
- Below 0.8: Poor match - needs investigation

Finally, we compute average IoU across all detections. If it's above 0.95, the models produce nearly identical results, confirming our conversion was successful."

---

## AI Tool Usage Demonstration (7:00 - 8:00)

"Let me highlight how I used Cursor AI throughout:

1. **Dependency Discovery**: Asked 'What packages needed for YOLO11?' - got complete list
2. **Code Structure**: Prompted 'organize into modular functions' - got clean architecture
3. **Implementation Help**: For IoU calculation, validation steps, error handling
4. **Autocomplete**: Cursor suggested correct syntax as I typed
5. **Documentation**: Asked it to add docstrings and comments

The key is asking specific questions and then reviewing the suggestions. AI accelerates coding, but you still need to understand and validate the output."

---

## Results and Conclusion (8:00 - 8:45)

"When we run this script, we get:

- Two annotated images showing detections
- Console output with all detection details
- IoU comparison showing model agreement
- Confirmation that conversion preserved accuracy

The output files are:

- result_pytorch.jpg
- result_onnx.jpg
- yolo11n.onnx (the converted model)

This demonstrates a complete pipeline from PyTorch inference through ONNX conversion to validation. Using AI tools like Cursor cut development time significantly while maintaining code quality.

Thanks for watching! The complete code is available in the description."