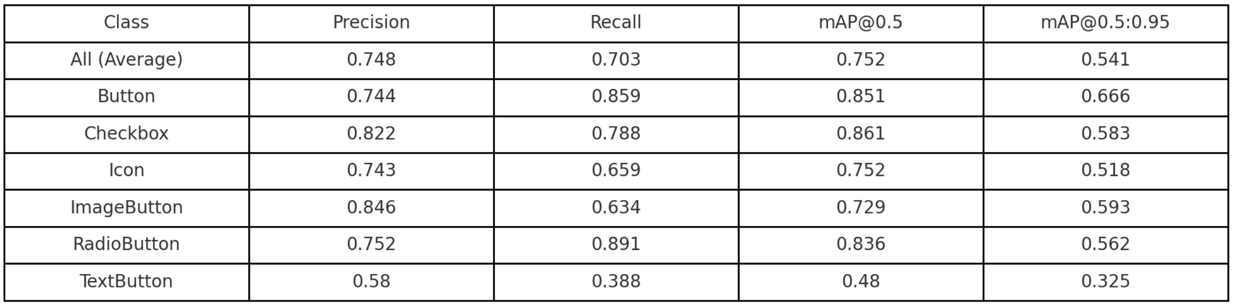
After the training was completed, I evaluated the model using the test set, which includes images the model had never seen before. This helped measure how well the model could generalize and detect real UI elements outside of the training data.

To do this, I used the val() function from the Ultralytics library and passed the test image directory directly. This returned standard object detection metrics like Precision, Recall, and mAP (mean Average Precision), which I used to assess the model's accuracy.

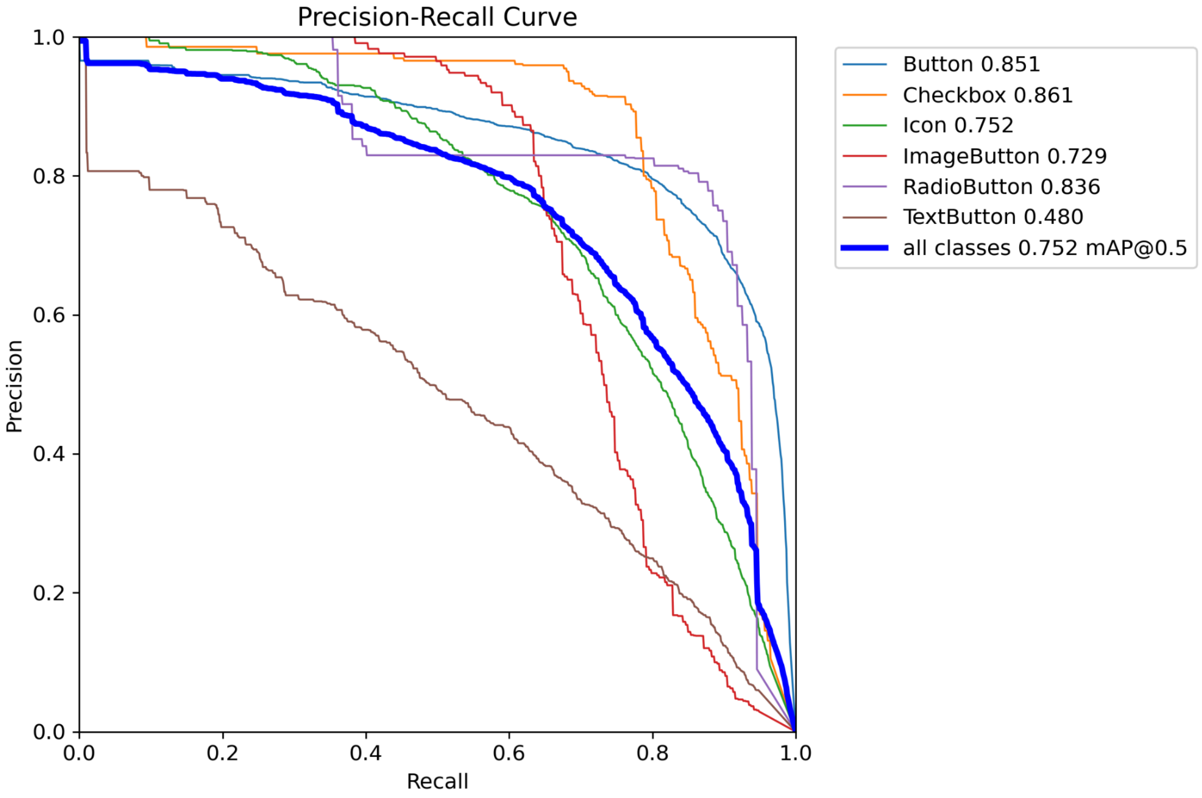
The table below shows the results for each class as well as the average scores:



Class	Precision	Recall	mAP@0.5	mAP@0.5:0.95
All (Average)	0.748	0.703	0.752	0.541
Button	0.744	0.859	0.851	0.666
Checkbox	0.822	0.788	0.861	0.583
Icon	0.743	0.659	0.752	0.518
ImageButton	0.846	0.634	0.729	0.593
RadioButton	0.752	0.891	0.836	0.562
TextButton	0.58	0.388	0.48	0.325

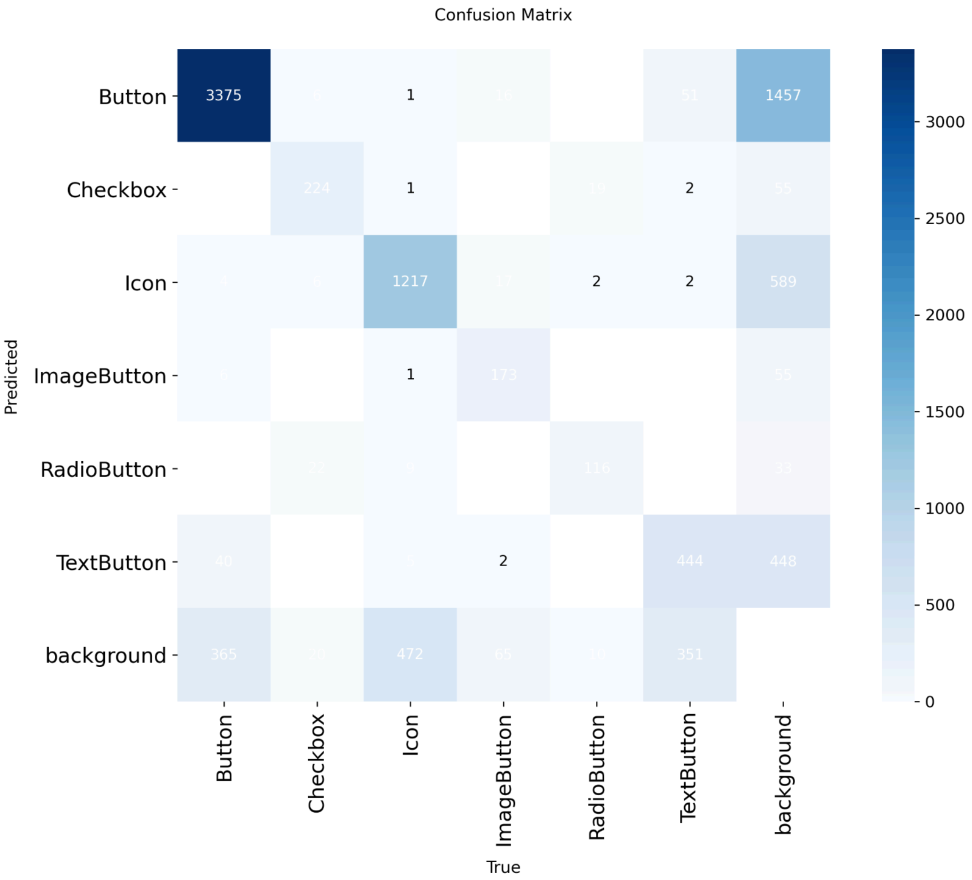
As seen above, most of the UI classes had strong scores. Checkbox, RadioButton, and Button had the best results, with precision and recall values above 0.74. On the other hand, the lowest performing class was TextButton, which only achieved 0.48 mAP@0.5 and 0.325 mAP@0.5:0.95. This may be due to class imbalance or visual similarity with other elements like Button.

The following chart shows the Precision-Recall curves for each class. These curves indicate the model’s ability to balance between precision and recall for different detection thresholds:



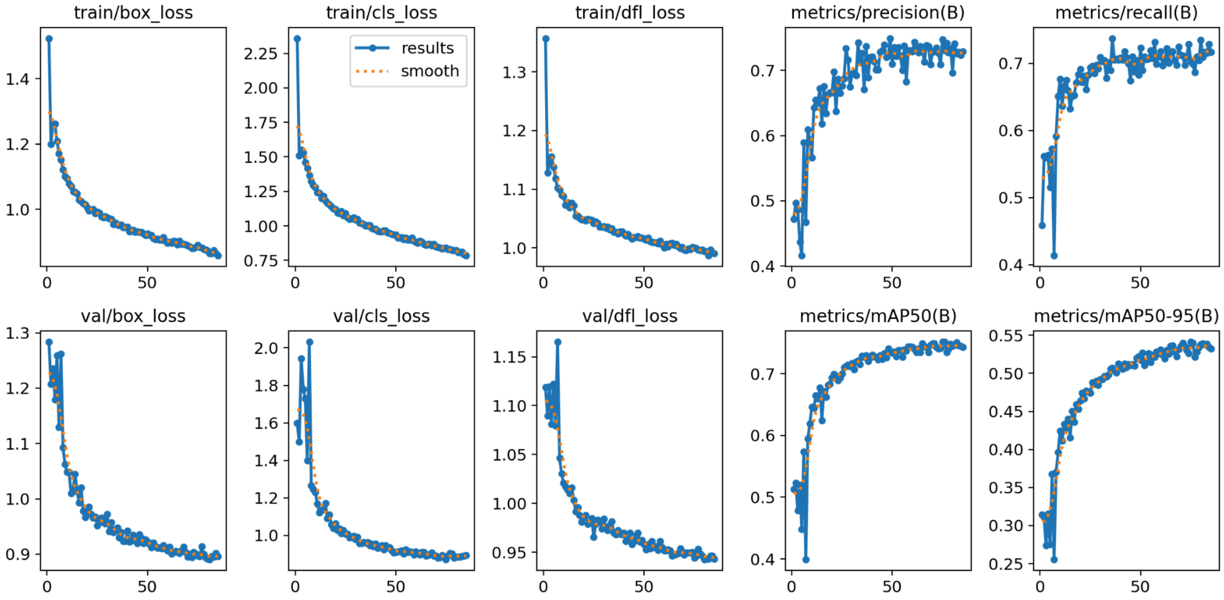
As shown in the graph, Checkbox and RadioButton have smooth and high PR curves, which means the model confidently and consistently detects these components. However, TextButton has a relatively poor curve, confirming what we saw in the numerical results.

Below is the confusion matrix, which shows how often the model confused one class with another:



We can see that most predictions are aligned with their true labels (the diagonal). However, some common mistakes happened between Button and TextButton, which are visually similar in some screenshots. There was also some confusion with background elements.

Lastly, the graph below summarizes the training process. We can observe how the loss decreased and the performance metrics (Precision, Recall, mAP) improved during training:



From the graph, we can say that the model steadily learned over time. The early stopping was triggered at epoch 85, and the best model was saved at epoch 75. The mAP@0.5 and mAP@0.5:0.95 values reached their peaks and stabilized, showing that the model had converged.

Overall, the evaluation shows that the YOLOv11 model learned to detect most UI elements accurately. While some classes need improvement (like TextButton), the general performance is strong enough to be used in UI analysis or automated testing systems.