YZV302E-Deep Learning Term Project: Real-World Chess Position Recognition Using Deep Learning

Ömer Faruk San
150220307
Artificial Intelligence and Data Engineering
Istanbul Technical University
san22@itu.edu.tr

Mustafa Kerem Bulut
150220303
Artificial Intelligence and Data Engineering
Istanbul Technical University
bulutm22@itu.edu.tr

***Abstract*—We build a full pipeline that reads a real chess position from one photo of a chessboard. Our system detects chess pieces, finds the four board corners, fixes perspective by warping, detects board orientation using OCR on the board letters, and converts the final result into a FEN string [1]. Then we send the FEN to Stockfish [8] to get the best move and an evaluation score. We train and compare YOLOv11 [3] (nano and small) and RT-DETR-L [4] for piece detection, and we also test different training settings such as augmentation, image size, epoch count, and dataset size. We report detection metrics (mAP, confusion matrices), timing, ablation results, and final end-to-end FEN accuracy.**

***Index Terms*—chessboard understanding, object detection, YOLO, RT-DETR, keypoints, OCR, perspective warping, FEN, Stockfish**

I. Introduction

Understanding a chess position from a real photo is not easy. The photo can have strong perspective distortion, shadows, reflections, and background objects. Also, small errors can break the final board representation. Our goal is to take one picture from a phone camera and output a correct chess position.

We solve this with this pipeline: (1) detect all pieces with a detector model, (2) detect the four board corners and warp the image into a square board, (3) detect the board orientation using OCR of the letters around the board, (4) place each detected piece into one of the 64 squares and build a FEN string, (5) run a chess engine to get the best move and evaluation.

Contributions.

- We trained and compared multiple detectors (YOLOv11n, YOLOv11s, RT-DETR-L) for 12 chess piece classes.
- We implemented a corner detection module using keypoint detection and a two-pass refinement step.
- We estimated board orientation with OCR + RANSAC [6] line fitting and convert the final board into FEN.
- We provided detailed analysis such as metrics, plots and ablation studies.

II. Related Work

Our work is close to vision-based chessboard understanding systems, where the main goal is to detect pieces and recover a correct board state from a single image.

For piece detection, one common solution is to use fast object detectors such as YOLO [3]. Recently, real-time transformer detectors like RT-DETR also became a strong option and can achieve high accuracy with competitive speed [4]. In this project, we compare these two detector families for 12 chess piece classes.

For board geometry, a practical baseline is to regress the four board corners with a CNN backbone such as ResNet [5]. Another approach is to predict the corners as keypoints using a keypoint detector (e.g., a pose-style model). We test both regression and keypoint-based corner detection in our pipeline.

To get the correct square naming, we also need board orientation. A simple way is to read the coordinate letters on the board edges with OCR. We use Tesseract, which includes LSTM-based recognition [7], and we filter noisy OCR outputs with RANSAC [6]. Finally, we represent the position using FEN [1] and send it to Stockfish for analysis [8].

III. Dataset

A. Data Sources

For the piece detection we used a public chess piece dataset from Roboflow[2] with **9729 images** and **12 classes** (white/black pawn, knight, bishop, rook, queen, king). We also collect our own data and add **200 images** taken with our mobile phones. We label them with the same 12 classes and bounding boxes.

B. Data Collection and Labeling

Our additional images include different angles, distances, and lighting conditions. We labeled bounding boxes for pieces using the same label format as the public dataset. We keep a consistent label mapping across all experiments.

In addition to piece detection labels, we also created a separate dataset for board corner detection. For this part, we manually labeled **1000 images** with the four chessboard

TABLE I: Dataset split

Split	Images	Source	Notes
Train	6971	Roboflow + ours	used for training
Val	2000	Roboflow + ours	used for early stop / selection
Test	996	Roboflow + ours	final evaluation

corners in a fixed order (a1, a8, h1, h8). This dataset contains **800 images** selected from the Roboflow dataset and **200 images** from our own photos (the same images we used in piece detection). This gaved us ground truth corner keypoints for training and evaluating our corner detector.

C. Train/Validation/Test Split

We split the dataset as 70% for training, 20% for validation, and 10% for testing. We reported the final split sizes in Table I.

D. Exploratory Data Analysis (EDA)

We visualize class distribution on the test set to check balance. Fig. 1 shows the test distribution.

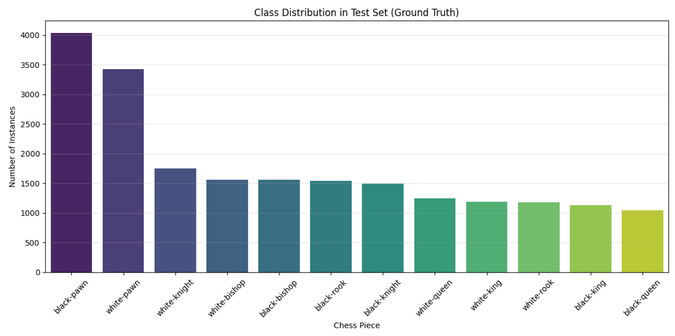

Fig. 1: Test set class distribution

IV. Method

Fig. 2 shows our full pipeline.

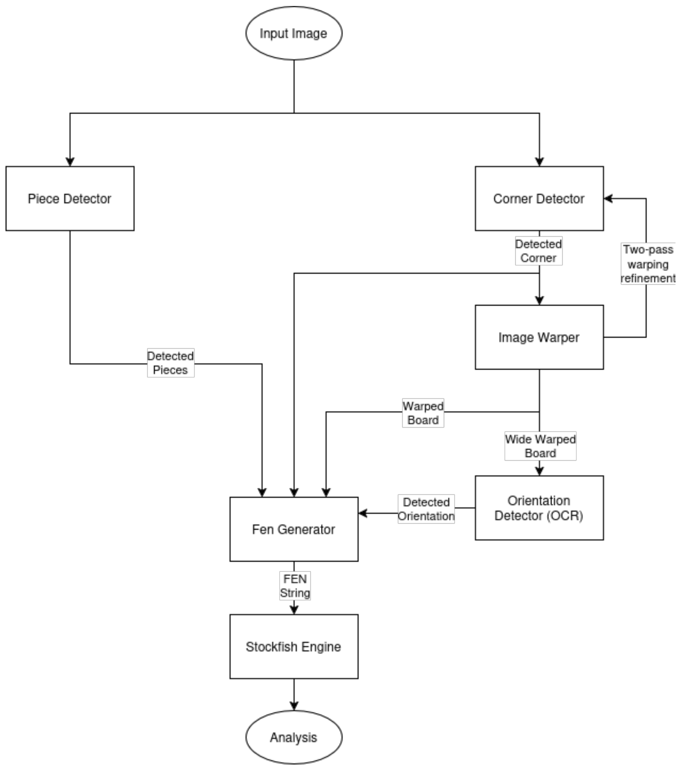

Fig. 2: Pipeline overview.

A. Piece Detection

We trained detection models to predict bounding boxes and class labels for 12 chess piece classes. We use:

- YOLOv11-nano as a baseline,
- YOLOv11-small as a stronger YOLO model,
- RT-DETR-L as a transformer-based detector.

For each run we report standard detection metrics: Precision, Recall, mAP50, and mAP50-95. We also analyze the results with plots such as loss curves, confusion matrices, per-class confidence, overall confidence distribution, and qualitative predictions on a real-world image.

B. Board Corner Detection and Warping

We detect the four outer corners of the chessboard to remove perspective distortion and to build a stable 8×8 grid. We tested two approaches for corner prediction: (i) a regression model based on ResNet-18, and (ii) a keypoint-based model using YOLO-Pose.

a) ResNet-18 corner regression: We use a ResNet-18 backbone and replace the final layer to output 8 values, representing (x, y) coordinates for 4 corner points. The coordinates are normalized by the image width/height and the model is trained with an MSE regression loss.

The output corner order is important. We first tried the semantic chess ordering $(a1, a8, h1, h8)$ to see if the model can learn the board orientation from the edge letters/numbers. In practice this was not stable. We then used geometric order (top-left, top-right, bottom-left, bottom-right), which avoids orientation ambiguity and works reliably.

b) YOLO-Pose: We also train a YOLO-Pose model to predict the same four corners as keypoints. This gives valid predictions, but in our setup the ResNet-18 regressor performs better.

Given the predicted corners, we compute a perspective transform (homography) and warp the board into a square top-down view. To improve stability, we use a two-pass refinement: we slightly expand the first-pass corners outward, warp again, predict corners again on the wide-warped board, map them back to the original image, and average the two corner estimates.

C. Board Orientation Detection

To build the correct chess position, we need to know the board orientation, i.e., which side belongs to White and which side belongs to Black. In our system, we determine this by reading the coordinate letters printed on the board edges.

We use an OCR-based approach with Tesseract [7], which uses an LSTM-based text recognition model. We first crop the four board edges (top, bottom, left, and right) from the input image. Then we run the OCR model separately on each cropped edge and collect the detected characters and their locations.

Not every OCR output is reliable, so we select the edge that has the highest number of letter detections (this edge is the most informative and usually corresponds to the side where the letters are clearly visible). To remove wrong detections (outliers), we apply a RANSAC-based filtering step on the detected character positions. This helps us keep only the detections that lie on a consistent line/structure along the board edge.

After filtering, we analyze the remaining detected letters to decide the direction of the sequence. We check pairwise combinations of detections and compare their order along the edge with the expected alphabetic order. If the letters follow the normal direction ($a \to h$), we label that edge as the White side. If the detected order is reversed ($h \to a$), we label it as the Black side. This orientation label is then used to assign squares correctly when converting piece detections into a full chessboard state.

D. Mapping Pieces to Squares and FEN Generation

After warping the board into a top-down square view, we split the warped image into an 8×8 grid, where each cell corresponds to one chess square. The next step is to assign each detected piece to the correct square.

A simple approach is to take the center of the detected bounding box and map it to a grid cell. However, this can fail in practice because pieces are tall and the bounding box may cover multiple squares even if the piece is standing on only one square. To make the assignment more robust, we focus on the bottom part of the piece, since this region is closest to where the piece actually touches the board.

For each piece bounding box detected in the original image, we crop a small rectangle from the lower portion of the box. Then we apply the same perspective transformation (homography) that was used for board warping, and transform this cropped bottom-region into the *warped image* coordinate system. In the warped image, we compute the Intersection-over-Union (IoU) between the transformed region and each of the 64 square regions. The piece is assigned to the square that gives the maximum IoU.

After assigning all pieces, we fill a 64-cell board representation (rank by rank) using the predicted piece classes. Finally, we convert this board array into a standard FEN string, using the usual FEN rules (piece letters and run-length encoding for empty squares).

E. Chess Engine Integration

We pass the final board position as a FEN string to Stockfish. For each analysis, we output the best move (SAN and UCI), the evaluation score (centipawns or mate-in-n), and the principal variation if needed.

We run this analysis twice: once assuming White to move and once assuming Black to move, so we can report results for both sides.

V. Experiments

A. Experiments on Piece Detection

1) Training Setup: We train all detectors using the Ultralytics framework. We keep input size 640 for most experiments, unless stated otherwise. We report epoch counts, augmentation settings, and dataset size for each run.

2) Models and Runs: Table II lists the main experiments on piece detection models.

TABLE II: Main training runs.

Run	Model	Epochs	Notes
Baseline	YOLOv11n	50	public+ours
Main-1	YOLOv11s	50	default aug
Main-2	YOLOv11s	50	+mosaic/+rotation
Main-3	RT-DETR-L	30	baseline epochs
Main-4	RT-DETR-L	50	longer training
Abl-1	RT-DETR-L	50	half dataset
Abl-2	RT-DETR-L	50	half dataset + img=320

3) Evaluation Metrics: **Detection:** mAP50, mAP50-95 per-class AP, confusion matrix, and confidence distributions. **End-to-end:** FEN accuracy, defined as the fraction of correct squares among 64 squares:

$$\text{FEN Accuracy} = \frac{\text{Correct Squares}}{64} \times 100\%. \qquad (1)$$

B. Experiments on Corner Detection

1) Setup: We evaluate two corner detection approaches: (i) a ResNet-18 regression model that outputs 4 corner points (8 values), and (ii) a YOLO-Pose model that predicts 4 corner keypoints. For ResNet-18, we compare two target orderings: semantic ordering $(a1, a8, h1, h8)$ and geometric ordering (top-left, top-right, bottom-left, bottom-right). We also test the effect of our two-pass warping refinement.

2) Metric: We report **Average Euclidean Error (AEE)**, computed as the mean Euclidean distance between predicted and ground-truth corner points, averaged over the four corners. We use the normalized version of this error so it is comparable across images.

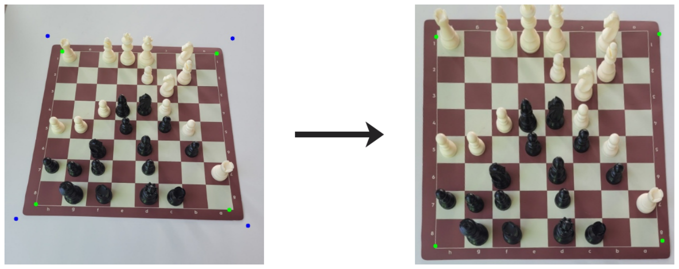

Fig. 3: Two-pass warping refinement for board corner detection.

Left: first-pass corner predictions (green). We expand these corners outward (blue) to create a wider crop and compute a "wide-warp" of the board. Right: the wide-warped board image used for the second pass, where corners are detected again (green) under reduced perspective distortion. Second-pass corners are mapped back to the original image and averaged with the first-pass corners to obtain the final corner estimates.

C. Experiments on OCR

We experimented with a few simple OCR preprocessing steps to make edge-letter detection more reliable.

First, we convert the cropped edge regions to grayscale before running Tesseract. In cases where OCR fails on the grayscale image, we apply a second attempt with an inverted black/white version (threshold + invert). This helps when the board letters have low contrast or the background is brighter than the text.

We also tested resizing the cropped edge images. We observed that upscaling improves character readability for the OCR model, so we use a $4\times$ upscale for the cropped edge regions.

With grayscale + optional inversion and $4\times$ upscaling, OCR performance was good enough for robust board orientation detection in our pipeline.

VI. Results and Analysis

A. Piece Detection Results

We summarize our main detection results in Table III. We report Precision (P), Recall (R), mAP@0.5 (mAP50), and mAP@0.5:0.95 (mAP50-95). We also include model size (parameters) and compute (GFLOPs) to show the cost-performance trade-off.

Our main observation. YOLOv11s with default configurations gives the best overall accuracy with mAP50=0.891 and mAP50-95=0.664. RT-DETR-L (30 epochs) is also strong (mAP50-95=0.655) but it is much heavier (103.5 GFLOPs vs 21.3 GFLOPs). YOLOv11n is lightweight and already performed well as a baseline.

B. Detailed Plots for Our Two Main Models

To avoid too many figures, we show detailed plots only for our two main models: YOLOv11s (50 epochs) and RT-DETR-L (50 epochs). For each of them, we include (1) training curves, (2) confusion matrix, (3) confidence analysis, and (4) one real-world example.

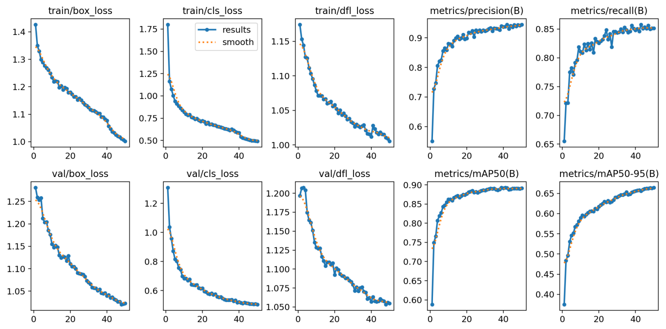

Fig. 4: YOLOv11s (50 epochs): training/validation curves

Training and validation losses go down steadily, which shows stable learning. Precision, recall, and mAP increase fast in the first epochs and then improve slowly. The curves do not show a big gap between train and val so overfitting is not strong.

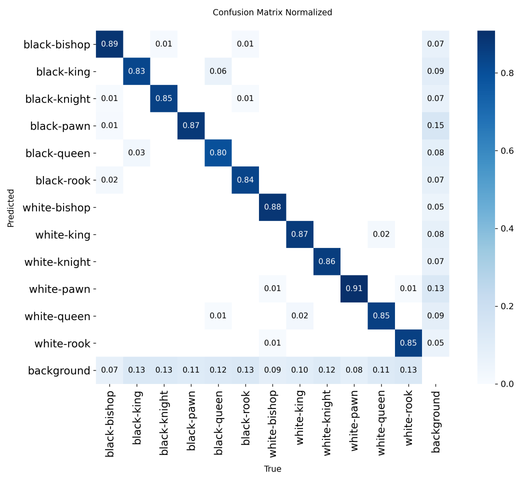

Fig. 5: YOLOv11s (50 epochs): normalized confusion matrix.

Most classes are correctly predicted (strong diagonal values, around 0.80–0.91). The main confusion is between king and queen of the same color. We also see that some pieces are predicted as background(missed detections), roughly in the 0.07–0.13 range for several classes. False positives on background exist as well, and they are more visible for pawns compared to other pieces.

TABLE III: Piece detection results on the test set. YOLOv11n is the baseline. Other rows are our main models and ablations.

Run / Model	Params (M)	GFLOPs	P	R	mAP50	mAP50-95
YOLOv11n (baseline, 50 ep)	2.58	6.3	0.925	0.829	0.879	0.635
YOLOv11s (50 ep, default)	9.42	21.3	**0.944**	**0.852**	**0.891**	**0.664**
YOLOv11s (50 ep, augmented)	9.42	21.3	0.939	0.844	0.887	0.638
RT-DETR-L (30 ep)	32.01	103.5	0.942	**0.860**	0.879	0.655
RT-DETR-L (50 ep)	32.01	103.5	**0.947**	0.848	0.873	0.639
RT-DETR-L (half data, 47 ep early stop)	32.01	103.5	0.918	0.827	0.851	0.603
RT-DETR-L (half data, img=320, 50 ep)	32.01	103.5	0.632	0.626	0.542	0.354

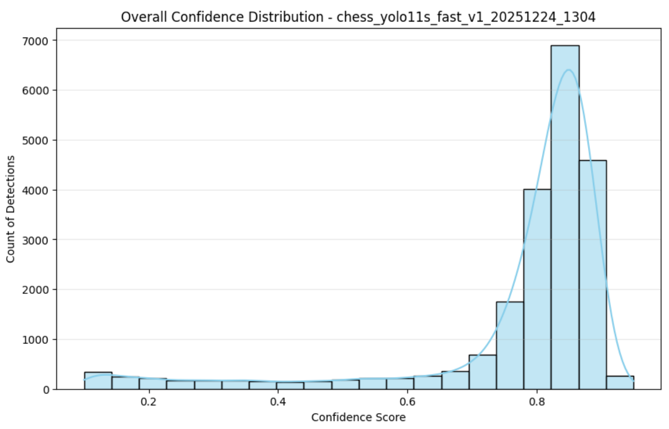

Fig. 6: YOLOv11s (50 epochs): overall confidence distribution

Most predictions are in the high confidence range (around 0.8–0.9). Only a small number of detections have low confidence, which usually means harder images.

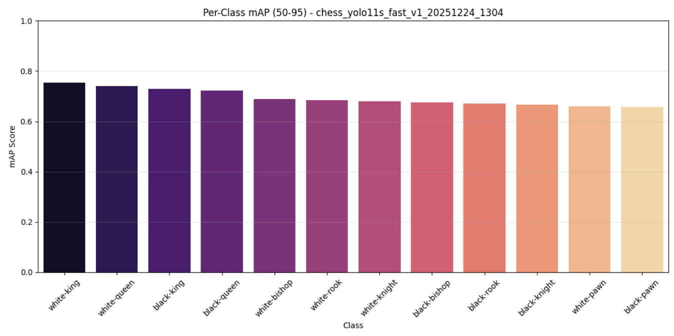

Fig. 7: YOLOv11s (50 epochs): per-class confidence

Confidence scores are similar across most classes. Kings and queens tend to have higher confidence, while pawns are slightly lower because we think they are smaller and look more similar to each other.

1) YOLOv11s (50 epochs):

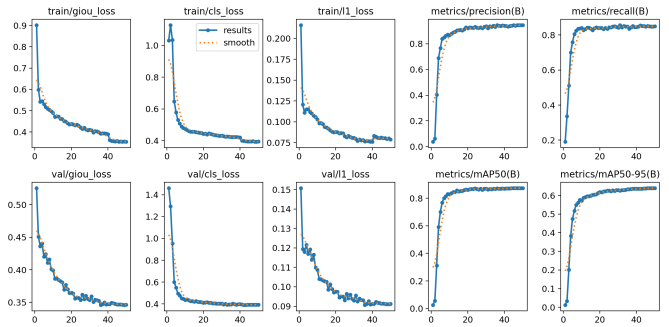

Fig. 8: RT-DETR-L (50 epochs): training/validation curves

Training and validation losses decrease over epochs, so the model learns steadily. Precision and recall increase quickly at the beginning and then become stable. mAP also improves fast in early epochs and then saturates near the end.

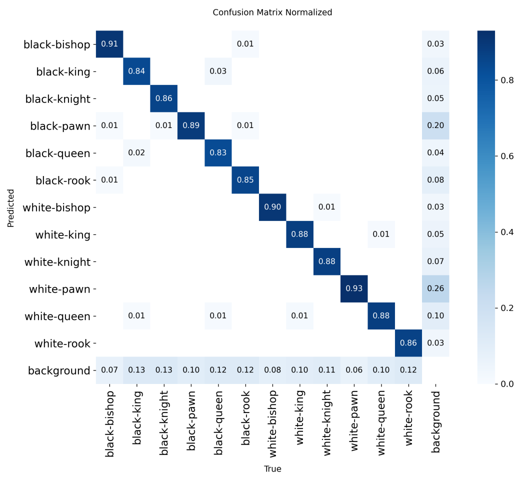

Fig. 9: RT-DETR-L (50 epochs): normalized confusion matrix

We see in the confusion matrix that it has a strong diagonal (most classes are correct, around 0.83–0.93). Similar to YOLOv11s, the main confusion is between king and queen of the same color. We also see missed detections as background for some classes (roughly around 0.06-0.13). Compared to YOLOv11s, RT-DETR-L looks slightly cleaner on some diagonal classes (bishops and pawns), but overall the confusion patterns are very similar.

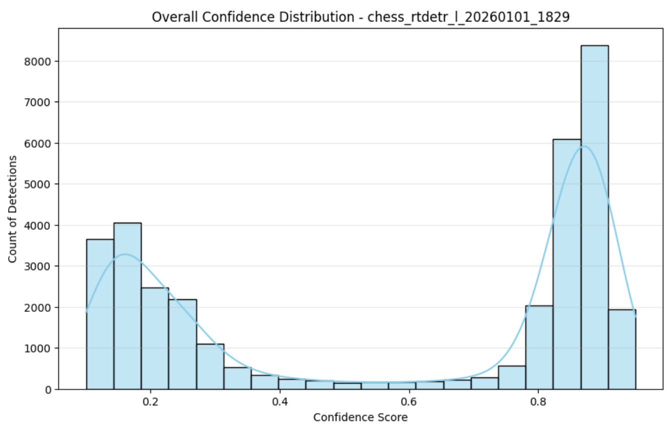

Fig. 10: RT-DETR-L (50 epochs): overall confidence distribution

Many detections have high confidence (around 0.8-0.9). Compared to YOLOv11s, RT-DETR-L also shows more low-confidence detections (around 0.1-0.3), which may come from harder images or weaker predictions.

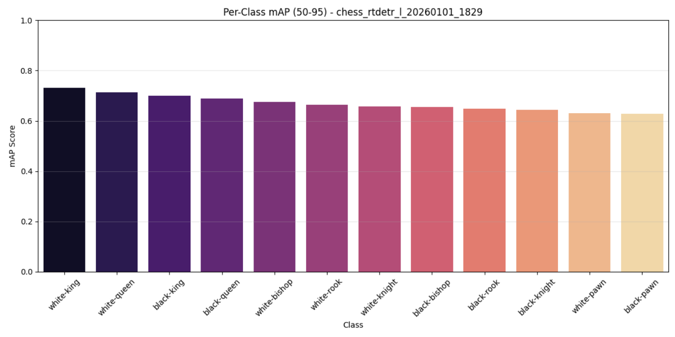

Fig. 11: RT-DETR-L (50 epochs): per-class confidence.

Per-class confidence is mostly close to YOLOv11s. Kings and queens are usually higher, while pawns are lower since they are smaller and harder to separate.

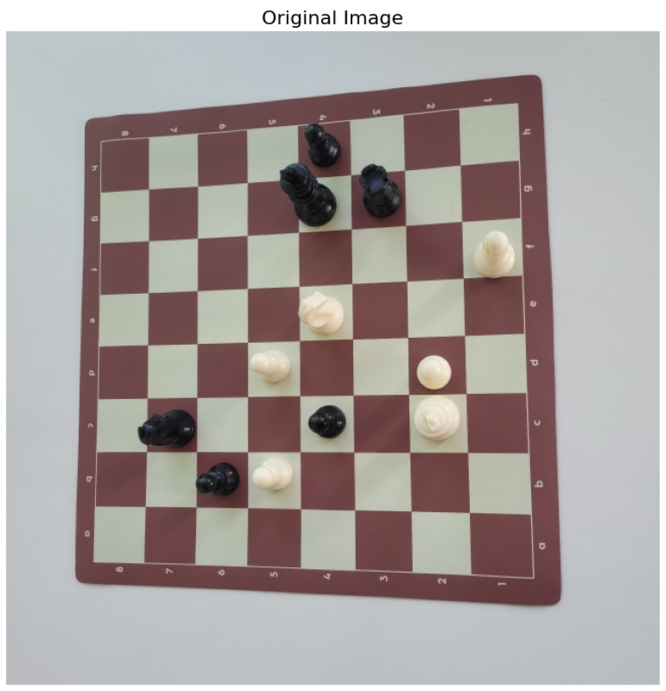

Fig. 12: Original Image.

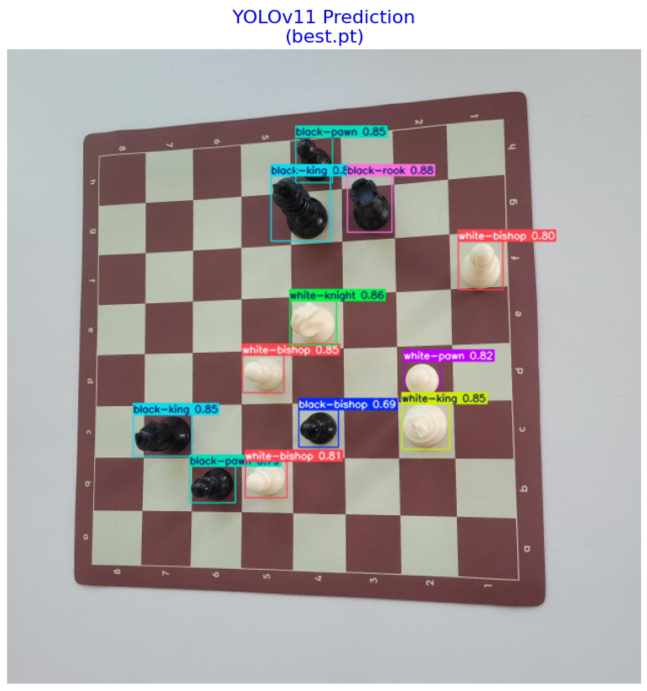

Fig. 13: YOLOv11s (50 epochs): one real-world example with predicted boxes, labels, and confidence scores

The model detects most pieces correctly and gives high confidence scores. Some labels can still be confused when pieces look similar or when the view angle is hard.

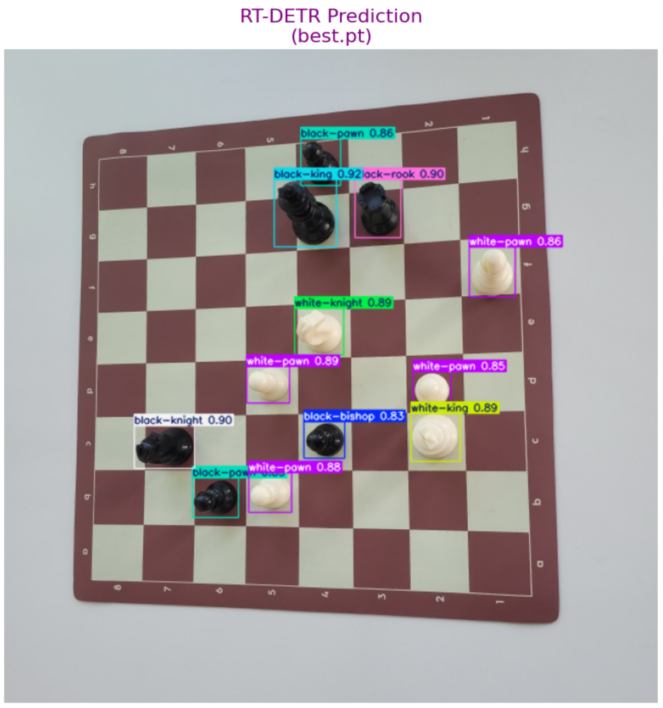

Fig. 14: RT-DETR-L (50 epochs): one real-world example with predicted boxes, labels, and confidence scores

RT-DETR-L also detects the pieces clearly and usually outputs high confidence scores. Compared to YOLOv11s, the predictions look similar on this image, but confidences are slightly higher on some pieces.

2) *RT-DETR-L (50 epochs):*

C. *Ablation Studies*

We ran ablations to understand what changes the performance.

For Ablation 1: YOLOv11s with more augmentations. We train YOLOv11s for 50 epochs with a fixed training setup (AdamW, lr0=0.001, cosine LR, imgsz=640, seed=42). Then we train again with extra augmentations (HSV jitter, rotation, translate, scale, perspective and mosaic). The augmented version slightly decreases performance (mAP50-95 drops from 0.664 to 0.638). This suggests that the augmentation strength may be too high for this dataset, or it may make some piece details harder to learn.

For Ablation 2: RT-DETR-L epochs (30 vs 50). We keep the same config and only change epochs. 30 epochs gives better results than 50 epochs (mAP50-95: 0.655 vs 0.639). We think that a simple reason is that longer training may start to overfit, or it may not benefit more after saturation.

For Ablation 3: RT-DETR-L dataset size (full vs half). When we use half of the dataset, performance drops (mAP50-95: 0.639 $\rightarrow$ 0.603). This shows the model depends on data size and needs more samples to generalize well.

For Ablation 4: RT-DETR-L image size (640 vs 320) with half data. Reducing image size to 320 caused a large accuracy drop (mAP50-95: 0.603 $\rightarrow$ 0.354). This is expected for us because small chess pieces lose important details at lower resolution.

Model size note. YOLOv11s has 9.42M parameters and 21.3 GFLOPs, while RT-DETR-L has 32.01M parameters and 103.5 GFLOPs. So RT-DETR-L is much heavier, and in our case it does not give a clear accuracy gain over YOLOv11s.

D. *Corner Detection Results*

Table IV compares corner detection models. Overall, ResNet-18 with geometric ordering (TL, TR, BL, BR) gives the best accuracy. The semantic ordering $(a1, a8, h1, h8)$ performs poorly, likely because the same physical corner can correspond to different semantic labels depending on board orientation.

TABLE IV: Corner detection results (Average Euclidean Error, normalized; lower is better).

Model	Average Euclidean Error
YOLO-Pose (4 keypoints)	0.035
ResNet-18 (ordering: $a1, a8, h1, h8$)	0.500
ResNet-18 (ordering: TL, TR, BL, BR)	**0.017**

1) *Two-pass warping refinement:*

We also tested a two-pass refinement: we predict corners, warp with wide corners, predict corners again on the warped board, map them back, and average the two corner sets. Table V shows the effect. This refinement improves YOLO-Pose, but slightly hurts ResNet-18, so we keep single-pass corners for ResNet-18.

TABLE V: Effect of two-pass refinement on corner error (Average Euclidean Error, normalized; lower is better).

Model	Original Corners	Averaged Corners
YOLO-Pose	0.03533	**0.03124**
ResNet-18	**0.01740**	0.01914

E. *Orientation Detection Results*

We evaluate board orientation detection based on OCR of the edge letters. We apply the OCR pipeline described in the experiments. With this setup, we correctly predict the White side in **94%** of the test images. This accuracy is sufficient for our end-to-end pipeline, since orientation directly affects the square naming and the final FEN construction.

F. *FEN Accuracy*

We evaluate the end-to-end reconstruction quality using **FEN Accuracy**, defined as the fraction of correctly predicted squares among 64 squares:

$$\text{FEN Accuracy} = \frac{\text{Correct Squares}}{64} \times 100\%. \tag{2}$$

On our test set, the **average FEN Accuracy** is **88.98%**. We also report the **90th percentile FEN Accuracy** as **98.44%**, which shows that many images are reconstructed almost perfectly, while the remaining cases include harder examples that reduce the mean.

VII. Conclusion and Future Work

We presented a full end-to-end system that reads a real chess position from a single photo and produces a FEN string and a best-move suggestion from Stockfish. Our pipeline includes (1) piece detection, (2) board corner detection and perspective warping, (3) board orientation detection using OCR + RANSAC, and (4) mapping pieces to the 8×8 grid to generate FEN.

For piece detection, we trained and compared YOLOv11n, YOLOv11s, and RT-DETR-L on 12 chess piece classes using a Roboflow dataset (9729 images) plus 200 images collected and labeled by us. Among all runs, YOLOv11s with default settings achieved the best overall performance (mAP50=0.891, mAP50-95=0.664) with a much lower compute cost than RT-DETR-L. RT-DETR-L was competitive in accuracy but it was significantly heavier in GFLOPs and parameters. Our ablation studies showed that stronger augmentations slightly reduced YOLOv11s performance, and RT-DETR-L performance dropped with fewer training images and with smaller input resolution (img=320), which is expected because pieces lose details at low resolution.

For corner detection, we created an extra labeled corner dataset of 1000 images (800 from Roboflow + 200 from our own photos) with ordered corners. We tested ResNet-18 regression and YOLO-Pose keypoint detection. ResNet-18 with geometric ordering (TL, TR, BL, BR) achieved the lowest corner error, while the semantic ordering $(a1, a8, h1, h8)$ failed to generalize well. We also tested a two-pass corner refinement with wide-warping, which improved YOLO-Pose but did not improve ResNet-18 in our setup.

For orientation detection, we used OCR on the board edge letters with simple preprocessing and RANSAC filtering. This allowed us to estimate whether the letters follow $a \rightarrow h$ or $h \rightarrow a$, and we achieved 94% orientation accuracy on the test set. Finally, we evaluated the full pipeline using FEN Accuracy (correct squares out of 64). We obtained an average FEN Accuracy of 88.98% and a 90th percentile of 98.44%, which shows that many images are reconstructed almost perfectly, while the remaining failures come from hard cases (missed/incorrect piece detections, corner errors, or OCR failures).

Limitations. Our system can still fail when (i) pieces are covered by other pieces or look very small, (ii) the board is rotated a lot or part of it is outside the photo, and (iii) the board letters cannot be seen, so OCR is not reliable. Also, we did not run experiments with many different random seeds due to limited time, so small differences between similar models should be taken with care.

Future work. We plan to (1) collect more real images under harder lighting and viewpoints, (2) improve orientation detection when letters are not visible (e.g., using a learned orientation classifier), (3) add chess-rule consistency checks to fix small errors and reject illegal boards, and (4) run stronger statistical validation (multiple seeds and optimizer comparisons) to better support the reported trends.

References

[1] Forsyth–Edwards Notation (FEN), Chessprogramming Wiki.

[2] Roboflow Universe, "C1 Dataset (visualizan2/c1-zhnfm), Dataset v1," available online: https://universe.roboflow.com/visualizan2/c1-zhnfm/dataset/1, accessed: Jan. 2026.

[3] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," in *CVPR*, 2016.

[4] Y. Zhao, W. Lv, S. Xu, J. Wei, G. Wang, Q. Dang, Y. Liu, and J. Chen, "DETRs Beat YOLOs on Real-time Object Detection," arXiv:2304.08069, 2023.

[5] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," in *CVPR*, 2016.

[6] H. Cantzler, "Random Sample Consensus (RANSAC)," University of Edinburgh, CVonline local copy.

[7] R. Smith, "A Simple and Efficient LSTM Architecture for Optical Character Recognition," in *ICDAR*, 2017.

[8] Stockfish Chess Engine, official documentation / repo.