Diffusion Board: An End to End Chess Move Prediction Pipeline Leveraging Discrete Diffusion for Long Horizon Prediction

Prerit Choudhary* Akshay Gupta* Ankush Dhawan Rikhil Vagadia

Stanford University

{preritc, agupta23, rvagadia, ankushd}@stanford.edu

June 5, 2025

Abstract

We introduce an end-to-end system that converts smartphone-captured chessboard images into Forsyth-Edwards Notation (FEN) and generates move recommendations tailored to player skill levels. Our system employs YOLOv8 for robust chessboard corner detection and piece identification through pose estimation and bounding box detection. These bounding boxes are then processed by a DINO Vision Transformer to extract visual embeddings, classified by a trained Multi-Layer Perceptron (MLP) into piece types. Classified pieces are mapped onto a standardized chessboard grid, yielding reliable FEN representations. To emulate human-like strategic decisions, we finetune a 270M-parameter transformer on the ChessBench dataset, enriched with skill-specific move annotations derived from 10 million chess games. Unlike traditional search engines, our diffusion-based transformer learns latent representations of playing strength, yielding move predictions that closely match human decision-making patterns at specific skill levels. Additionally, we propose a novel positional encoding strategy that effectively captures and preserves spatial relationships between pieces. Our hybrid YOLOv8-DINO approach achieves 99.83% accuracy in chess piece classification, with YOLOv8 pose estimation providing robust corner detection with approximately 6-7 pixel error on 640×640 images. The discrete diffusion model demonstrates exceptional learning capability, reducing per-token cross-entropy loss from 0.42 to 0.004 across 312-token sequences, indicating near-perfect prediction of human move patterns during training. However, inference reveals position-dependent performance with high confidence in opening positions (entropy 2.17) but declining performance in middle and endgame scenarios (entropy 4.83-6.17), suggesting the need for enhanced model capacity and refined noise scheduling for complex game states.

1 Introduction

1.1 Overview

Automatically converting chess board images into Forsyth-Edwards Notation (FEN) represents a critical challenge at the intersection of computer vision and chess technology. This capability can enable real-time game broadcasting, and enhance online chess education by seamlessly bridging the physical and digital chess worlds. Beyond basic digitization, our project extends this vision by incorporating a novel pipeline that provides humanistic move recommendations tailored to specific skill levels. While traditional engines like Stockfish find optimal moves, they fail to capture human playing patterns at different ELO ratings, making their suggestions impractical for learning.

The fundamental challenge lies in achieving reliable accuracy across diverse real-world conditions. Natural images present multiple simultaneous obstacles: varying lighting conditions create shadows and reflections that obscure piece identification, perspective distortion from arbitrary camera angles complicates board detection, and the enormous variety of chess set designs demands robust generalization. While existing approaches report high per-square accuracy on synthetic datasets, their performance degrades on real-world images, with end-to-end FEN reconstruction accuracy often falling below practical thresholds.

Our project tackles these challenges through a pipeline that combines computer vision for board recognition with reinforcement learning for humanistic analysis. Once we achieve accurate FEN reconstruction, this digital representation serves as input to our transformer-based chess engine trained on 10 million human games, enabling ELO-specific move recommendations that reflect how players at different skill levels actually approach positions.

2 Related Work

The task of converting images of real-world chessboards into structured digital representations such as Forsyth-Edwards Notation (FEN) involves different aspects of computer vision including object detection, pose estimation, and symbolic reasoning. Early classical approaches leveraged rule-based techniques to extract board geometry and detect pieces using shape descriptors and template matching [2]. While effective under controlled lighting and angle conditions, these methods struggle with the diversity of chessboard designs, inconsistent lighting, and camera perspectives encountered in smartphone images.

Recent deep learning approaches have shown significant improvements. Chesscog by Wölflein and Arandjelović employs a multi-stage CNN-based pipeline for square-level classification. Their method performs well on synthetic images, achieving 93.9 percent board-level accuracy using few-shot learning to generalize across unseen piece designs. However, performance deteriorates dramatically on real images, where the compounded effects of poor corner localization, occlusions, and lighting variations result in over 40 misclassified squares per image. These findings highlight the difficulty of supervised classification pipelines when grid alignment is imperfect [6].

Modern single-stage detectors like YOLOv8 [7] offer advantages for chess recognition through unified architectures that simultaneously predict object locations and classes. The anchor-free detection paradigm directly predicts object centers, providing natural robustness to scale variations and partial occlusions common in chess photography. YOLOv8's pose estimation variant extends this capability to keypoint detection, enabling simultaneous board detection and corner localization in a single pass. To improve robustness under such variability, researchers have begun exploring self-supervised visual representations. Vision Transformer-based models like DINO leverage multi-head self-attention to capture long-range dependencies across 16×16 patch embeddings, making them effective in few-shot classification where spatial alignment is uncertain. [3] Unlike CNNs with localized receptive fields, DINO's attention mechanism dynamically weights relationships between all patches simultaneously, preserving semantic structures even when patches contain misaligned chess piece features. Prior applications of DINO in fine-grained recognition and object retrieval suggest its potential for encoding distinctive chess piece features without explicit supervision [1].

While most existing systems stop at visual board reconstruction, some recent work has begun exploring post-visual reasoning aligned with human play. The Maia engine uses supervised fine-tuning on human move data to predict ELO-specific behavior [4], while other methods introduce ChessBench, a dataset of 10 million online games annotated by the chess engine Stockfish, resulting in 15 billion data points. The authors train a series of transformer models up to 270M parameters in size on supervised objectives including action-value prediction, state-value, and behavioral cloning. [5] Collectively, these findings motivate systems that can handle imperfect board alignment, generalize across uncontrolled conditions, and bridge the output of perception modules with human-centered gameplay modeling.

3 Data

3.1 Vision Dataset

For our dataset, we are utilizing the ChessReD2k Dataset, which contains 2,078 chess board images captured by three smartphone models to represent real-world conditions. The dataset provides annotations for chess piece positions using standard algebraic notation (like "a8"), piece types (12 categories covering all pieces in both colors), and corner locations that help determine board orientation. Images come from 20 different chess games and are split into training (1,442 images), validation (330 images), and test sets (306 images), with care taken to distribute images from all three camera types across these sets. These different annotations will allow us to pursue one of our pipeline approaches of detecting the board and the individual squares before detecting the pieces themselves. In addition, because the dataset also contains annotations and bounding boxes for the pieces as well, we gain flexibility in exploring other ways to classify and locate pieces even without full board corner detection. We outline the preprocessing steps in the methods.

3.2 Gameplay Dataset

To train our predictive model's for high level play, we leverage two data sources: ChessBench and LiChess. The ChessBench dataset consists of chess states and their corresponding stockfish depth assessed win percentage, which we leverage to distill the stockfish search policy directly into network parameters. The ChessBench dataset was developed by Google DeepMind for their own explorations on Searchless Chess Engines. To mimic humanistic style, we leverage publicly available LiChess Datasets, which contain data for every game which contains the player's elo, the moves the player's made, and the winner. This data allows us to segment states and the corresponding trajectories for certain ELO buckets which will be used to train our diffusion policy, which aims to predict ELO conditioned trajectories given a state.

4 Methods

Given an image, the goal of our computer vision pipeline was to detect a chessboard and classify all the pieces and their locations within the image.To achieve this, we

conducted an iterative process and explored a variety of strategies described below:

4.1 ResNet Corner Regression

Our baseline approach explored the capability of a large pretrained model to directly regress all the corners of a chess board in an image. Therefore, we explored ResNet as a backbone. For our model architecture, we leverage a pretrained ResNet18 backbone with weights obtained from the 'IMAGENET1K_V1' set. Then, inspired by the ideas discussed in lecture, we essentially changed the fully connected layer to have an output shape of 8 to output the coordinate for the four corners. We froze all layers except the final classification layer due to our small dataset size. Images were resized to 224×224 pixels and normalized. For the actual training process, we used the AdamW optimizer with a learning rate of 0.001 and used Huber Loss as our loss function. We used Huber loss due to its ability to mitigate outlier sensitivity, something which we anticipated for this process. We then trained for 20 epochs and evaluated our model's performance through RMSE on the validation set.

4.2 YOLOv8 Pose Estimation for Corners

Since we planned to use corner detection as a step towards creating a 2D flattened homography of the board, we needed the corner detection to be extremely accurate to yield a perfectly aligned 2D representation. After assessing the performance of the previous technique, we decided to also explore the use of a YOLO model for the dual task of creating a bounding box for the chess board within an image and detecting its corners. Since detecting the corners of a board can be thought of as akin to pose detection, the YOLO-pose architecture served as the clear choice for our approach. We converted annotations to YOLO pose format and used YOLOv8n due to compute constraints. Training was conducted on an 80/20 split with normalized coordinates, and training metrics were tracked across epochs as discussed in the results section. AdamW was used as the optimizer along with a variable learning rate schedule built into the Ultralytics library. We also made use of the data augmentation techniques with YOLO models and used mosaic augmentation and horizontal flipping. An example of a training batch can be seen in Figure 8.

4.3 Homography and Unsupervised DINO Clustering

Using YOLOv8-detected corners, we applied a homography transformation to obtain a standardized top-down board view. Homography allows us to mathematically map points from one plane to another while preserving the structure of the image. We can use this property to map the corners to a 2D board and use that mapping on all pixels to generate a 2D transformed image. Initially, we planned to manually segment the 2D image into 64 squares and classify each square, however, the results from the homography introduced some issues like slanted representations and inconsistent line detection, which necessitated a better way to classify the pieces in the 2D homography. As a result, we decided to explore unsupervised DINO clustering, similar to what was explored in assignment 3. We decided to use Facebooks's DINO Vision Transformer with the pretrained 'dino_vits8' model (smallest model for our compute constraints). This model was used to generate feature representations of the 2D chess board representations at each patch. We split the 2D image into a set of 8 by 8 patches, to mimic a chessboard as close as we could, and then densely sampled each patch with half the stride. Then, we extracted the feature representation of each patch from the DINO model. For piece detection, we then implemented a simple K-means clustering technique to detect any patch into one of 13 possible clusters, where one cluster is for an empty square and the other 12 clusters are for each piece type for their respective color. We hypothesized that the feature representations from DINO would create strong enough clusters to classify each patch accurately into one of the 13 clusters. We also tried different combinations of patch size and stride to gauge the impact on performance. To assess performance, we generated visual cluster mappings for qualitative evaluation.

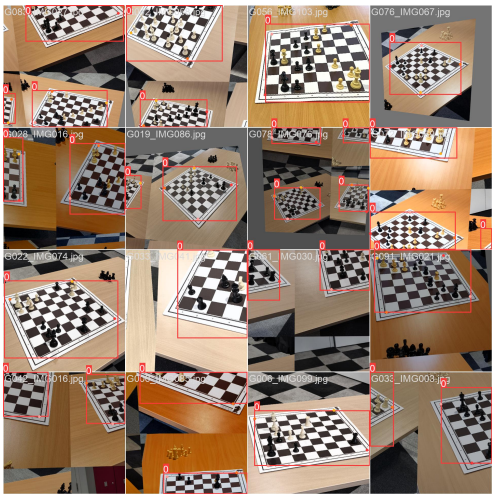

Figure 1: Example of a training batch with mosaic augmentation and horizontal flipping.

4.4 YOLOv8 for Piece Detection and Classification

While the homography pipeline showed promise, it yielded inconsistent results that propagated throughout the pipeline. Because the YOLOv8 model performed well for board corner detection, we tested the ability of the YOLO model to directly detect the bounding boxes for all pieces on a chess board as well as classify them in one

pass. We still utilized the ChessReD2K dataset, however, we had to generate the labels in a different process for the new task of our model. Instead of accessing the board annotations, we access the piece annotations within the data. These annotations are then converted to the proper YOLO labels for a given image. For each image, the number of annotations was stochastic because there were a variable number of pieces on the board. For each piece, a class ID was assigned as a number in the range of 1 to 12 for each type of piece and its respective color on the board. The resultant final format for each piece was `<classid> x_center y_center width height` where the respective bounding box for each piece was also extracted from the annotations. We then trained our YOLO model with the AdamW optimizer and a stochastic learning rate with mosaic and flipping augmentations. One key note is that we used YOLOv8 'detect' instead of pose (which identifies keypoints), as it makes use of anchor-free detection. This allowed us to directly predict the center of the box without needing to predefine anchors and makes the model more flexible for odd shapes like chess pieces. An example training batch for this model can be seen in Figure 1.

4.5 Hybrid YOLOv8 and DINO-MLP Classifier

We explored combining YOLO's detection capabilities with DINO's rich feature representations to overcome limitations of our previous approaches. This hybrid approach addresses two key issues: first, using YOLO to detect piece bounding boxes eliminates our reliance on homography-dependent segmentation, overcoming the alignment issues from our previous DINO clustering approach. Second, we hypothesized that DINO's pretrained feature representations would provide more accurate piece classification than YOLO's learned features alone. The first step in this pipeline is to detect and segment out a piece. Our YOLO model from the previous approach already accomplishes this task, and so we move forward to train a classifier to classify a segmented piece image. We design a DINO based 3 layer MLP classifier, which takes in a 384 dimensional feature vector from the CLS token of the pretrained DINO model. We again use Facebook's self-supervised DINO model with 'dino-vits8'. To each cropped piece, we apply the standard transformations that were discussed in class: Resizing, ToTensor, and Normalize, before passing through the DINO model. For the MLP classification head, we define a three layer model that has hidden layer dimensions of 256 and 128, before finally projecting to 12 for piece logits. Each layer applies ReLU and a dropout of 0.1. The classification head is trained with the AdamW optimizer with weight decay of 0.005 to mitigate overfitting. For training, we used cropped piece images from the labeled bounding boxes as individual data points rather than full images, since YOLO handles the detection step in our pipeline. The classification head was trained using AdamW optimizer with weight decay of 0.005 to mitigate overfitting, minimizing cross-entropy loss between predicted and true piece labels for 10 epochs. The full pipeline is described in the following algorithm:

Algorithm 1 DINO-MLP Chess Piece Detection and Classification Pipeline

Require: *image*: Path to the chess board image
Require: *mlp_weights*: Path to pre-trained MLP classifier weights
Require: *yolo_weights*: Path to pre-trained YOLO model weights
Initialize predictions $\leftarrow$ []
Load classification model MLP
Load object detection model YOLO
Load image I
Get detection results: results $\leftarrow$ YOLO.predict(I)
Extract locations: boxes $\leftarrow$ results.boxes
for each bounding box B in boxes **do**
    Extract coordinates $(x_1, y_1, x_2, y_2) \leftarrow B$
    Crop piece image piece $\leftarrow I[y_1 : y_2, x_1 : x_2]$
    feature $\leftarrow$ DINO(piece) {Extract 384-dimensional feature vector}
    output $\leftarrow$ MLP(feature) {Get class logits}
    predicted_class $\leftarrow \arg\max$(output) {Get predicted class index}
    Append (predicted_class, B) to predictions
end for
return predictions

4.6 Discrete Diffusion for ELO-Conditioned Move Generation

To extend upon our vision pipeline, we implement a discrete diffusion model that learns skill-specific move distributions from human gameplay data, enabling our end-to-end system to generate move recommendations that authentically reflect how players at different ELO ratings approach a given chess position. Modeling this as a discrete diffusion problem allows us to directly model the categorical distribution over the chess move vocabulary.

Our diffusion model leverages an 8 layer pretrained decoder only transformer architecture trained on the ChessBench dataset described previously. The model consists of 8 transformer layers with 256-dimensional embeddings, 8 attention heads, and 1024-dimensional feed-forward networks. The pre-trained model provides crucial chess domain knowledge, including piece interaction patterns, tactical motifs, and positional evaluation heuristics. Rather than training from scratch, we initialize our diffusion model with these learned representations and adapt

them through our novel ELO-conditioning framework.

We employ Low-Rank Adaptation (LoRA) to efficiently adapt the pre-trained architecture for our diffusion task. LoRA decomposes weight updates into low-rank matrices, dramatically reducing the number of trainable parameters while preserving model expressiveness through the expression $h = W_\theta x + \Delta W x = W_\theta x + BAx$ where $B \in R^{d \times r}$ and $A \in R^{r \times k}$ and $rank(r) << min(d, k)$. We apply LoRA with rank r=128 and scaling factor alpha=256 to the query, key, value, and output projection matrices within each attention head. This configuration reduces trainable parameters by approximately 90% while maintaining the model's capacity to learn complex skill-dependent pattern, targeting the attention mechanism where we hypothesize skill-level differences primarily manifest.

Player skill conditioning represents a core innovation in our approach. We model ELO ratings through learnable bucket embeddings that span our target range of 1200-1800 with 100-point granularity, resulting in 7 discrete buckets. To enable smooth skill interpolation rather than discrete jumps, we implement continuous ELO conditioning through linear interpolation between adjacent bucket embeddings. This process is explained further in 7.1. The continuous ELO embedding is broadcast across all sequence positions and added to the position embeddings, ensuring that skill-level information influences every aspect of the model's reasoning process.

4.6.1 Discrete Diffusion process

Our diffusion formulation operates directly on the discrete move vocabulary. The forward diffusion process gradually corrupts clean move sequences by mixing them with uniform noise according to a predefined schedule. For a clean move sequence x_0 and a timestep t, the forward process samples from:

$$q(x_t|x_0) = \alpha_t \cdot \text{OneHot}(x_0) + (1 - \alpha_t) \cdot \frac{1}{|V|}\mathbf{1}_{|V|}$$

where α_t represents the noise schedule, $\text{OneHot}(x_0)$ is the one-hot encoding of the true move token, and $|V| = 2000$ is our move vocabulary size. The term $\frac{1}{|V|}\mathbf{1}_{|V|}$ represents a uniform distribution over all possible moves, where each move has probability $\frac{1}{|V|}$.

We create a categorical mixture distribution that interpolates between the true move (represented as a one-hot vector) and uniform noise over the vocabulary. At $t = 0$, we have $\alpha_0 = 1$, recovering the clean data distribution. As t increases, α_t decreases, gradually shifting probability mass from the true move to uniformly random moves in the vocabulary. The noise schedule follows $\alpha_t = \prod_{i=1}^{t}(1 - \beta_i)$ with linearly increasing β_i values from 0.025 to 0.1 over $T = 40$ diffusion steps. During training, we randomly sample timesteps t Uniform(0, T-1) and apply the corresponding noise level to ground truth move sequences. The model learns to predict the original clean moves given the noisy observations, position context, and target ELO rating.

4.6.2 Model Architecture

The input to the model is represented as follows:

$$x = [\textbf{CLS}]||\text{state\_tokens}||\text{future\_tokens}$$

where state_tokens represent the current chess position encoded as piece-square tokens, and future_tokens represent the (potentially noised) move sequence to be denoised. The exact representations of these tokens is explained in Section 7.2) .

Position embeddings are applied through the inherited positional encoding from the base model, while ELO embeddings are broadcast and added across all positions. Temporal embeddings for the diffusion timestep t are learned through a separate embedding layer initialized to zero, allowing the model to gradually learn timestep-specific behaviors during training. The temporal embedding is critical for the denoising process, as it allows the model to adapt its predictions based on the current noise level. At early timesteps (high noise), the model should focus on broad strategic patterns, while at later timesteps (low noise), it should attend to precise tactical considerations.

4.6.3 Training And Loss

Our training objective focuses specifically on positions where noise was applied, reflecting the insight that the model should learn to correct corrupted tokens rather than simply copying already-clean inputs. The loss function incorporates several masking strategies:

$$L = \frac{1}{B}\sum_{b=1}^{B}\lambda_t \sum_{l=1}^{L}\Big[\text{CE}\big(f_\theta(x_t^{(b)}, s^{(b)}, \text{elo}^{(b)})_l, x_{0,l}^{(b)}\big) \times \text{mask\_changed}_{b,l} \times \text{mask\_valid}_{b,l}\Big]$$

$$\lambda_t = \frac{\beta_t}{1 - \bar{\alpha}_t}.$$

In this formulation, λ_t is given by $\beta_t/(1 - \bar{\alpha}_t)$, which provides timestep-dependent weighting to emphasize harder denoising steps. The term $\text{mask\_changed}_{b,l}$ is an indicator that equals 1 if token l in example b was corrupted by the forward noising process (and 0 otherwise), ensuring we only compute loss on positions that were actually altered. The term $\text{mask\_valid}_{b,l}$ equals 1 if token l in example b is not a padding token (and 0 if it is), which excludes padding from the loss. Finally, $\text{CE}(\cdot, \cdot)$ denotes the cross-entropy loss over the move vocabulary. Weighting by the timestep is a crucial stability component to prevent the model from being dominated by easy denoising tasks and ensures that the model still learns from high-noise scenarios.

4.6.4 Diffusion Inference

During inference, we employ a reverse diffusion process, which we call entropy conditioned denoising, that iteratively denoises a randomly initialized move sequence over $T = 40$ timesteps. Starting from uniform random tokens, the model progressively refines predictions by selecting the highest-confidence tokens (the one's with minimal logit entropy) at each step. We implement an adaptive masking strategy where only the least confident tokens (determined by log-probability percentiles) are updated at each timestep, allowing high-confidence predictions to remain stable. This selective updating mechanism prevents the model from unnecessarily revising correct predictions while focusing computational effort on uncertain regions of the sequence. Additionally, the frozen tokens add to the inherent conditional information the model can use as other tokens that are still to be predicted are now conditioned on the frozen tokens. One key design choice is that we allow tokens to be "unfrozen" so that highly confident incorrect predictions do not derail the entire diffusion process. The process terminates early if no tokens require updating, typically converging within 10-15 iterations for most chess positions.

5 Experiments and Results

5.1 Baseline ResNet Classification

Our first approach, which involved training a classification layer on a baseline ResNet model to detect corners was assessed through validation set RMSE loss which can be seen below:

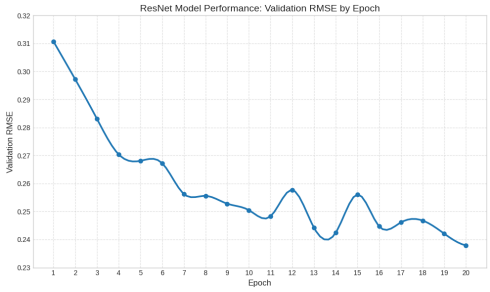

Figure 2: Validation Set RMSE per Epoch for ResNet Corner Classification

Although the model learns through the epochs, the validation loss starts to fluctuate around 0.24. Intuitively, this means that our predicted corner coordinates from this method are off by 0.24 in the normalized coordinate space, and when converted back to the 224 pixel scale, there is approximately 54 pixels of error, which indicates that this method struggles substantially for corner detection. This model probably struggles heavily due to the fact that we are only fine tuning one classification layer on top of pretrained weights. Moreover, the use of a very general model for such a specialized task also factors into the lack of success. This result, therefore necessitates an improved architecture for the task of corner detection, YOLOv8-pose.

5.2 YOLOv8 Pose Estimation

Because of the struggles of the baseline model, we decided to test the use YOLOv8 pose to improve our pipeline's corner detection capabilities. we chose YOLOv8 pose because it was specifically designed for keypoint detection can learn to detect the board and corners simultaneously. The training and validation loss curves for both board detection and corner detection (pose) can be seen in Figure 4. After 60 epochs of training our final training loss for the board bounding box was 0.459 while the final training loss for the corner detection pose for 0.25605. The corresponding validation losses were 0.3501 and 0.0991. The reason that the validation losses are actually lower than the training loss is due to the way that YOLO trains on augmented images, but only tests on the way images are in the dataset (ie. without mosaic or flipping). Additionally, the calculated mAP50-95 value for board detection was 0.993, indicating that 99.3 percent of the board bounding boxes closely matched the ground truth labels across a variety of thresholds, indicating very strong ability to define a board bounding box. Additionally, the validation kobj loss was 0.00354 which indicates very strong corner detection capabilities. Our final validation loss of 0.0991 translates to an approximately 6 or 7 pixel error in corner detection for a 640 by 640 pixel image, based on the way that the Yolo model loss is calculated, which is a substantial improvement over the previous approach.

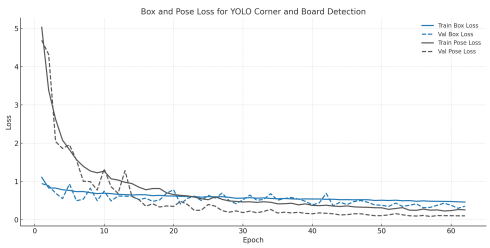

Figure 3: Train and Val Board Detection Loss

To test the model's abilities in a qualitative manner, we applied 2D homography to the image based on the detected corners to see if the corner classification was accurate enough to yield correct 2D image representations. We then evaluating the consistency and accuracy of the generated standardized top-down views. illustrates the transformation's effectiveness in handling diverse camera perspectives and angles.

Because the 2D representations were still slightly misaligned, we couldn't segment the 2D representation into

Figure 4: Homographic transformation used to flatten the board to a top-down view.

an exact grid. We hypothesize that the model occasionally classified the corner of the chessboard sheet rather than the actual chessboard in some cases, which yielded slightly misaligned representations like the one above. To accommodate for this, we decided to use Hough Line transforms from OpenCV to see if we could sidestep the misalignment and still segment the squares accurately. The results from this procedure can be seen in Figure 10. The Hough Transforms were unable to overcome this limitation, sometimes generating multiple lines for one edge and other times missing lines, thus limiting our ability to directly classify pieces from the homography.

5.3 Unsupervised DINO Clustering

To try to classify pieces from the homography in an unsupervised manner, we explored the use of clustering DINO feature embeddings in patches to see if patches could be clustered into their piece category. In preliminary testing, thediscriminative power of the DINO Vision Transformer embeddings was assessed qualitatively through embedding visualization, with applying PCA to 3 dimensions and mapping to RGB values. Figure 9 shows this result for the full board, while Figure 6 below depicts this for a specific patch. The process to obtain the heatmaps mimics Assignment 3.

Figure 6 clearly shows an understanding of what the chessboard edge is (highlighted in green), the center of the piece (highlighted in red) and the square the piece is on (highlighted in dark blue), which initially showed promise for unsupervised clustering as an approach for piece classification. By performing k-means clustering into 13 clusters (12 piece types and empty squares), we can cluster all patches into their effective classification. Figure 7 shows the result of this clustering for each respective image patch.

The clustering is very inconsistent any only seems to work to cluster white pawns effectively. Although we had hoped each cluster would all contain patches from the same pieces, with one for empty squares, unsupervised clustering was not a reliable technique to make use of DINO features.

Figure 5: k-Means clustering of DINO features into 13 groups (12 pieces + empty).

5.4 YOLOv8 Piece Detection

Due to the lack of success in unsupervised clustering with DINO, we explored using YOLOv8 again, but this time on detecting all pieces on the board with a bounding box and simultaneously classifying them. The training and validation loss curves for each task can be seen below.

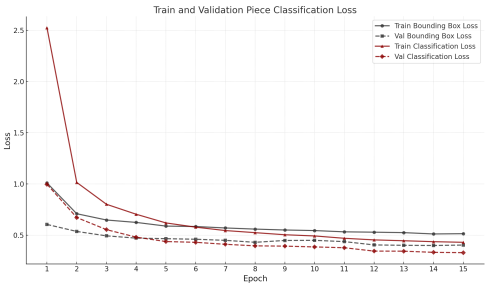

Figure 6: Train and Validation Piece Classification Loss

YOLOv8 showed extensive capability in detecting the pieces with a final validation losses of 0.05146 for detection and 0.4305 for classification. More importantly, the model had extremely high precision and recall of 99.17% and 98.26% respectively, indicating very high discrimination capabilities. The model very rarely misclassified a piece and very rarely misses a given piece across the classification categories. The model's MAP50-95 was 91.988%, which indicates that the model performs well even with highly localized bounding boxes. Of note is that the model's precision started at 53.82% and the recall began at 66.01% emphasizing the model's learning over time.

5.5 Hybrid Approach

Finally, we quantified the accuracy of our classification module for our hybrid YOLO and DINO-MLP Approach to see if we could improve on the sole YOLO model with dense DINO features. We measured these results, using per-class accuracy metrics and found nearly perfect accuracy across all classes, with an overall accuracy of 99.83 percent as seen in Table 1 . While misclassification rates are minimal, we noted that there was a higher misclassification rate for black pieces than white pieces. Our hy-

pothesis is that this issue arises because the white pieces have a tan color, preventing them from blending seamlessly with the white squares on the board, whereas the black pieces blend more naturally. The overall high accuracy, especially the perfect classification performance observed for several piece types, validates our hybrid approach of combining YOLO detections with DINO embeddings and the subsequent MLP classifier. This analysis confirms the high practical reliability of our approach for robust piece classification in real-world chess image scenarios.

Table 1: Per-class Accuracy for Piece Classification

Class ID	Piece Type	Accuracy (%)
0	White Pawn	100.00 (2106/2106)
1	White Knight	100.00 (646/646)
2	White Bishop	99.74 (384/385)
3	White Rook	100.00 (462/462)
4	White Queen	99.55 (219/220)
5	White King	100.00 (416/416)
6	Black Pawn	99.61 (2315/2324)
7	Black Knight	99.85 (659/660)
8	Black Bishop	100.00 (362/362)
9	Black Rook	100.00 (443/443)
10	Black Queen	99.07 (214/216)
11	Black King	99.74 (389/390)
Overall Accuracy		**99.83%**

5.6 Diffusion Results

Due to substantial compute constraints, we were only able to train LoRA adapters on the baseline transformer for 5 million samples extracted from the Lichess dataset. Although we plan to test the model through tournament style self play to assess capabilities, this process could not be completed within the constraints of the class period. Throughout training, the model exhibited stable and consistent convergence patterns. The weighted cross-entropy loss incorporating our timestep-dependent weighting scheme (λ_t) demonstrated effective learning across all diffusion timesteps. Unlike traditional supervised learning where loss simply decreases monotonically, diffusion training showed characteristic oscillations reflecting the varying difficulty of denoising at different noise levels. We therefore evaluate performance using logit entropy as our primary metric. Entropy provides insight into the model's prediction confidence across the 1968-token vocabulary. This metric allows us to assess both training quality and inference prediction, as these differ substantially in diffusion models. Our final loss during training was 1.271, decaying exponentially from a first epoch training loss of 120. Intuitively, our loss function sums the cross entropy loss of each token across the 312 predicted tokens. Because the vocabulary is of size 2000 (1968 actions + 31 states + 1 noise), a fully random prediction would result in a loss of $312 \times (ln(1/2000) \approx 7.6) = 2371.2$. With this understanding, we see that after one epoch, our model had a per token loss of 0.42, with our final epoch per token loss at 0.004. This indicates that the model can denoise artificially noised inputs extremely well, given a state. Since our noising schedule that caps at 0.1, meaning that we also need to assess the model's ability to denoise given a completely random sample during inference. Here we make use of the entropy confidence values. We note an extremely interesting trend: the model's average entropy for a prediction in earlier parts of the game (ie. its confidence) is substantially lower (2.17) compared to when the model approaches the middle (4.83) and end games (6.166), where the model struggles to play reasonable moves. We can see that the model is slightly better than random in the middle, but in the endgame, approaches almost complete randomness, indicating decaying performance. We note that this is likely due to a combination of a distribution mismatch in training data accompanied by the exponentially larger state space in the middle and end game compared to the beginning. Additionally, the contrast between the extremely low training loss and the inference results, suggest that the model may need to have substantially more parameters to capture deeper relationships within the middle and endgame. Moreover, the noise schedule may need to also be refined to enforce substantially more difficult noising tasks for model training. Due to time constraints and the lack of general efficiency, we have still yet to test the capability of the ELO conditional embeddings to alter chess understanding with diffusion, but we plan to test this with simulations against online bots once we have drastically augmented the diffusion model's parameter schema and training time.

6 Conclusion

Our experiments demonstrate the robustness and reliability of each component within our pipeline, demonstrating high accuracy in both homographic alignment and chess piece classification. Moving forward, we will focus on automating the complete integration pipeline to generate accurate Forsyth-Edwards Notation (FEN) strings directly from smartphone-captured chessboard images. We also plan to evaluate the integrated system's performance in diverse real-world conditions, including varied lighting, camera angles, and chess set designs. Future enhancements will involve refining the diffusion-based move recommendation system by incorporating additional human gameplay datasets, thereby improving the model's ability to emulate human strategic decision-making accurately. Ultimately, these developments will enable practical and user-friendly applications, bridging the gap between physical chess play and digital analysis platforms.

7 Appendix

7.1 Continuous ELO Interpolation

For a target ELO rating $\text{elo}_{\text{target}}$, we compute the interpolated embedding as:

$$\text{bucket\_idx} = \frac{\text{elo}_{\text{target}} - \text{elo}_{\text{min}}}{\text{bucket\_size}} \tag{1}$$

$$\text{lo\_idx} = \lfloor \text{bucket\_idx} \rfloor \tag{2}$$

$$\text{hi\_idx} = \lceil \text{bucket\_idx} \rceil \tag{3}$$

$$w_{\text{hi}} = \text{bucket\_idx} - \text{lo\_idx} \tag{4}$$

$$w_{\text{lo}} = 1.0 - w_{\text{hi}} \tag{5}$$

$$\mathbf{e}_{\text{elo}} = w_{\text{lo}}\, \mathbf{E}_{\text{lo}} + w_{\text{hi}}\, \mathbf{E}_{\text{hi}} \tag{6}$$

where $\mathbf{E}_{\text{lo}}$ and $\mathbf{E}_{\text{hi}}$ represent the learnable embeddings for the lower and upper buckets respectively. This interpolation scheme allows our model to generate moves for any ELO rating within the target range, not just the discrete bucket values.

7.2 Diffusion Tokenization

The state token is defined as a length 77 token vector that is obtained by converting all characters within a FEN string into an encoding obtained by indexing the vocabulary space. As a result, each state can be represented as a vector of length 77. The future tokens, for which we use a horizon of 4, where an action and then its corresponding state is 1, are then the combination of each state vector concatenated with the next action token. As a result, the future token is a length 312 vector made up four 78 length action state combinations. In total, with the classification token prepended to the start, the model sees a 390 length vector.

7.3 Figures

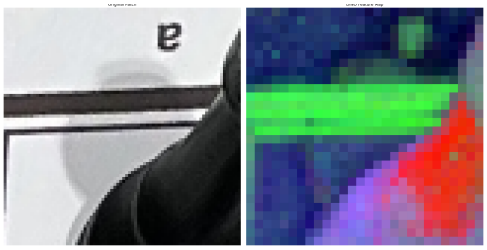

Figure 7: Patch-level feature heatmap using DINO on sample chess images.

Figure 8: Example of a training batch with mosaic augmentation and horizontal flipping.

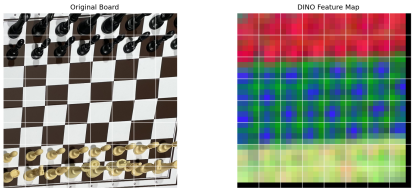

Figure 9: Comparison of DINO-based embeddings across different piece types.

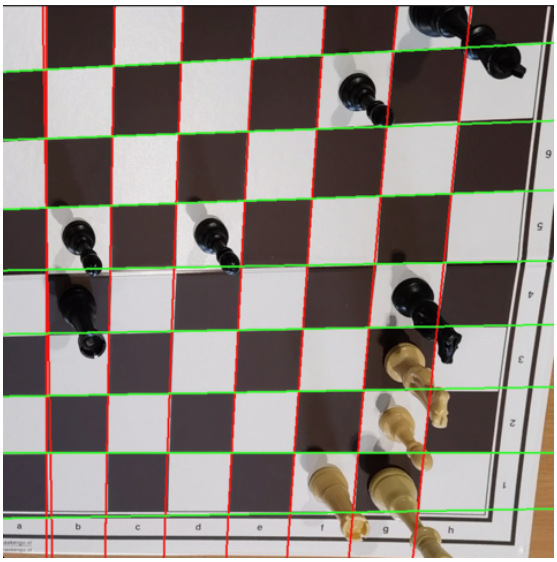

Figure 10: Detected gridlines overlaid on the chessboard using Hough transform to aid in homography computation and square segmentation. Red lines indicate vertical divisions and green lines indicate horizontal divisions.

8 Contributions

Prerit and Akshay implemented and tested the entire image analysis component including data manipulation, training the different architectures, and generating results. Prerit and Akshay also wrote the entire paper. Rikhil and Ankush worked on the reinforcement learning component of the project which included policy gradient, self play and training on soft label embeddings to improve the actual chess engine performance to mimic the humanistic style. As Prerit was enrolled in both classes (CS231N and CS224R), he was the primary contributor on implementing the diffusion model component to experiment with human style, which was inspired from components of both classes.

References

[1] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin. Emerging properties in self-supervised vision transformers. *CoRR*, abs/2104.14294, 2021. 2

[2] M. A. Czyzewski, A. Laskowski, and S. Wasik. Chessboard and chess piece recognition with the support of neural networks, 2020. 2

[3] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. 2

[4] R. McIlroy-Young, S. Sen, J. Kleinberg, and A. Anderson. Aligning superhuman ai with human behavior: Chess as a model system. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20)*, page 11 pages, Virtual Event, CA, USA, 2020. ACM. 2

[5] A. Ruoss, G. Delétang, S. Medapati, J. Grau-Moya, L. K. Wenliang, E. Catt, J. Reid, C. A. Lewis, J. Veness, and T. Genewein. Amortized planning with large-scale transformers: A case study on chess, 2024. 2

[6] G. Wölflein and O. Arandjelović. Determining chess game state from an image. *Journal of Imaging*, 7(6):94, June 2021. 2

[7] M. Yaseen. What is yolov8: An in-depth exploration of the internal features of the next-generation object detector, 2024. 2