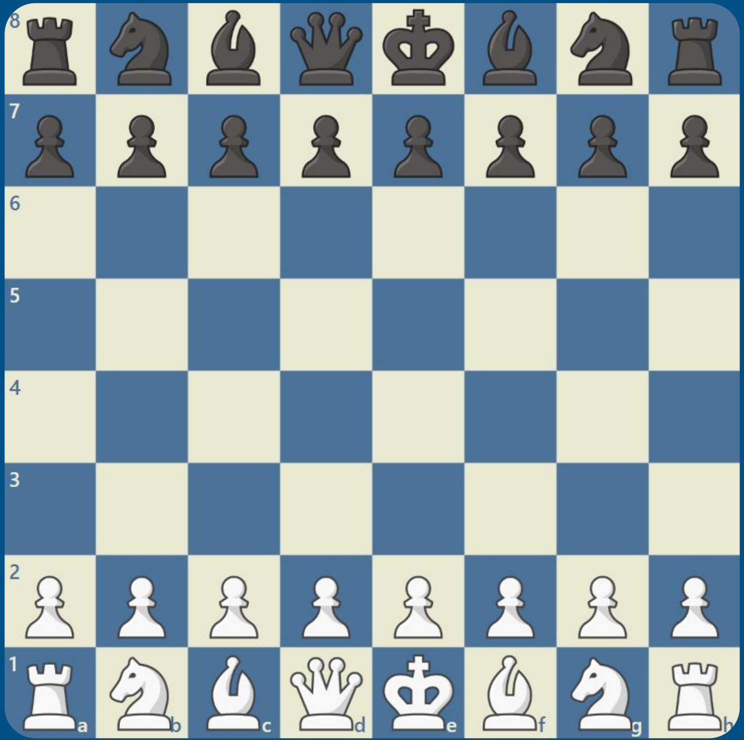
Chess Analysis Tool

Farees Siddiqui
100-780-513



1. Defining the Problem
2. Chess Vision Parser
3. Transformer Chess Chatbot
4. Chess Engine
5. Chess Engine Fine Tuning
6. Results

For this project I decided to make a Chess analysis tool.

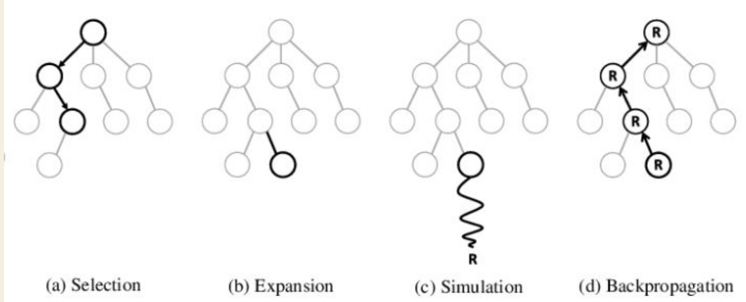
I have created an analysis tool for the game of chess.

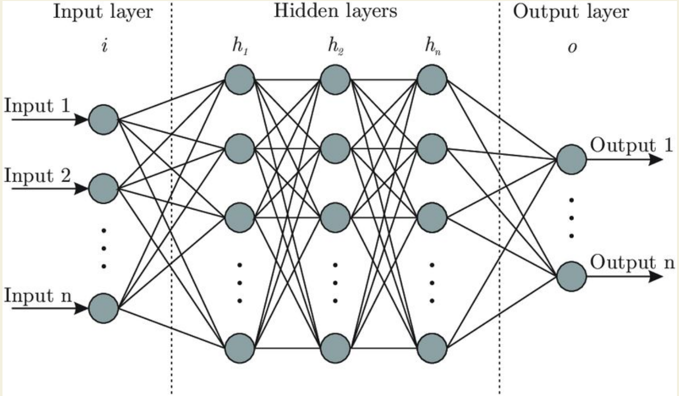
The motivation behind this project, lies in its complexity, chess is often known to be a difficult game for humans, and is even more difficult (computationally) for computers. My tool has the following features:

- Computer Vision board parser
    - Used so that the over the board (in person) games can be parsed and analyzed easily without the need for a human to enter the position into a computer manually

- Transformers based Chatbot
    - Used so that the user can ask questions about the current position, as well as ask for assistance, such as the next moves in a certain opening

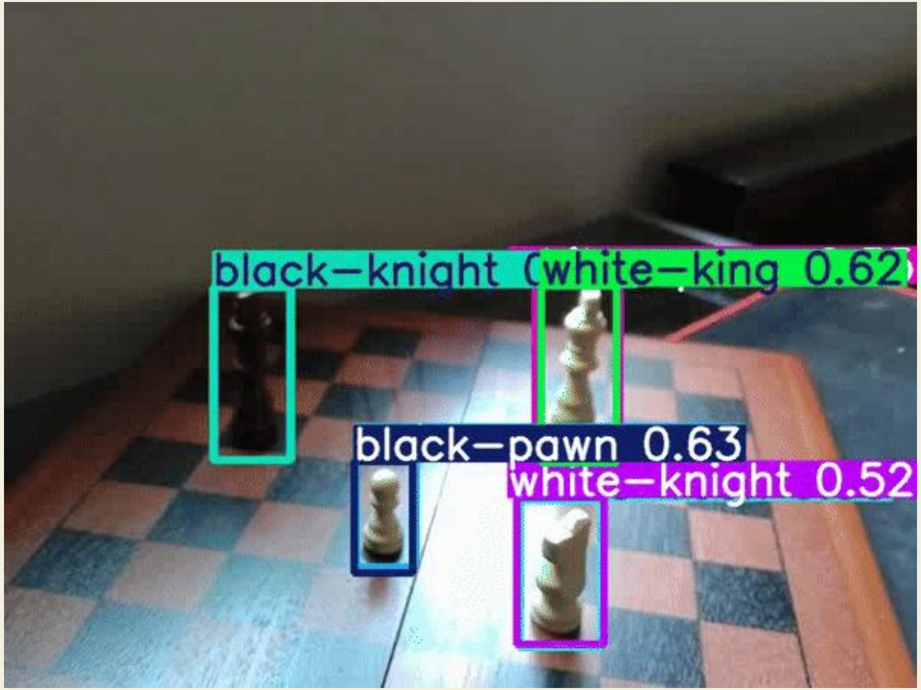
- Neural Network based Chess Engine
    - Better Generalization than MCTS based approaches
    - Faster Predictions than MCTS based approaches
    - Fine Tuned using self play and Proximal Policy Optimization (PPO)





Chess Vision Parser

- Yolo V5 (You Only Look Once)
- Fine Tuning based on chess pieces dataset
- Data annotation
- Object Detection
- Usage



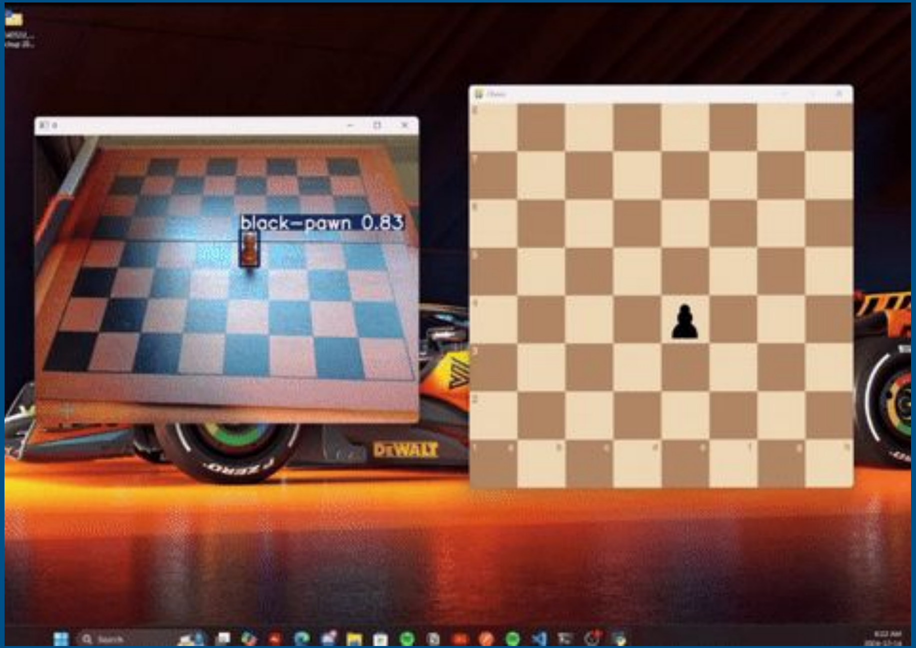
How Does it Work

- Pre-trained YoloV5 model
- Fine tuned with chess dataset
  - Images of over the board games (top img)
  - Contains annotations of piece labels and positions (bottom img)
- Calculates the FEN string for the position
  - FEN is used to represent a chess position textually
- FEN string is used throughout the tool.

```
6 0.85009765625 0.782798833819242 0.07177734375 0.13192419825072887
6 0.73291015625 0.08163265306122448 0.05322265625 0.10787172011661808
5 0.70556640625 0.413265306122449 0.0634765625 0.16909620991253643
2 0.77197265625 0.2988338192419825 0.078125 0.1924198250728863
3 0.81640625 0.543002915451895 0.06982421875 0.13119533527696792
```

Chess Vision Parser Results

- Non Optimal Lighting
- Glare on board and pieces
- Inaccurate detection of square
- Longer Training
- Larger Dataset
- Better Camera Angle
  - Purely overhead would be ideal



Transformer Chess Chatbot

- GPT-2 Model that was fine tuned for chess queries called chessgpt-chat-v1
- For this project, I used the pretrained weights that were available online
- Powered by the transformers architecture
- Training Data:
    - Scraped chess blogs, books, and forums
      Filtered and structured into a conversational format
- Answer Questions about chess strategies, openings, and rules

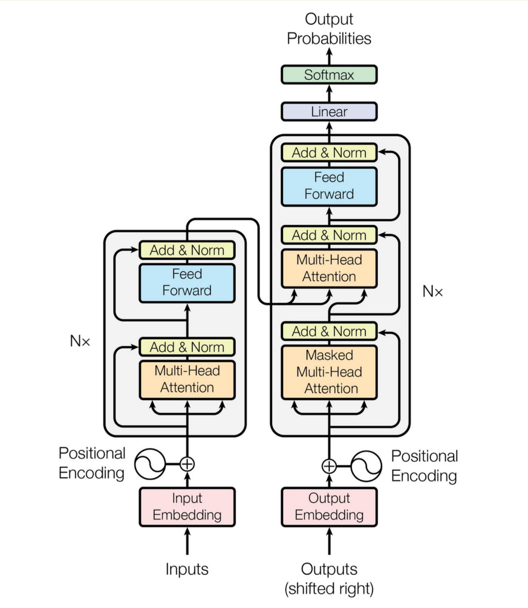


Figure 1: The Transformer - model architecture.

```
prompt = "what is the sicialian defense?"
```

```
Human 1 A: The Sicilian is a defense to 1. e4. It is played by black against 1. e4.

The Sicilian is not considered a "system" opening per se, but the Kan (4... a6) and Taimanov
```

Dataset & Engine Overview

Dataset Generation

- Extracted 100’000 positions from the Lichess Database
- Data Augmented using stockfish for position evaluations
  - FEN String (Board position)
  - Human Move
  - Stockfish move
  - Stockfish Evaluation of position

Engine Architecture

- Input: Board state encoded as a 12x8x8 tensor
  - 12 chess pieces, 8 rows, 8 columns
- CNN layers to extract spatial features
- Fully Connected Linear layers to predict next moves


```
{
  "fen": "rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1",
  "human_move": "e2e4",
  "stockfish_best_move": "e2e4",
  "stockfish_eval": 40
},
```


```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1             [-1, 64, 8, 8]           6,976
              ReLU-2             [-1, 64, 8, 8]               0
            Conv2d-3            [-1, 128, 8, 8]          73,856
              ReLU-4            [-1, 128, 8, 8]               0
            Conv2d-5            [-1, 256, 8, 8]         295,168
              ReLU-6            [-1, 256, 8, 8]               0
           Flatten-7                [-1, 16384]               0
            Linear-8                 [-1, 2048]      33,556,480
              ReLU-9                 [-1, 2048]               0
           Linear-10                 [-1, 1024]       2,098,176
             ReLU-11                 [-1, 1024]               0
           Linear-12                   [-1, 64]          65,600
           Linear-13                  [-1, 512]       8,421,888
             ReLU-14                  [-1, 512]               0
           Linear-15                   [-1, 64]          32,832
================================================================
Total params: 44,550,976
Trainable params: 44,550,976
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.62
Params size (MB): 169.95
Estimated Total Size (MB): 170.57
----------------------------------------------------------------
```

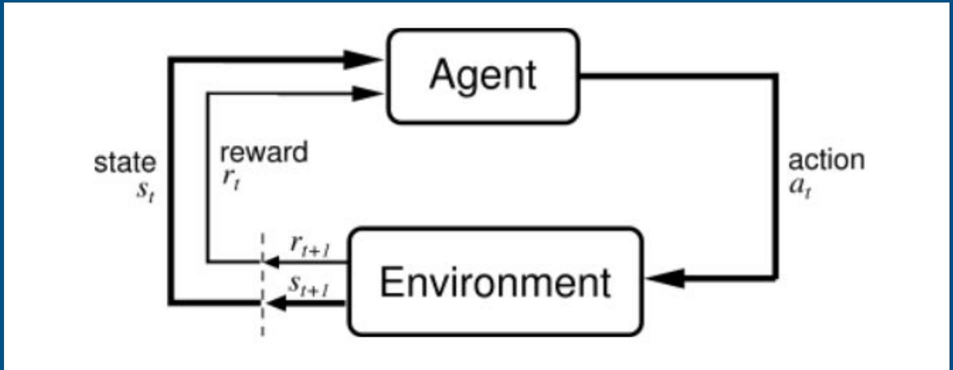
Reinforcement Learning & PPO?

RL Overview

- **Agent**: Learns a policy to decide actions based on the current state
- **Environment**: Provides feedback by returning a new state and a reward after the agent performs an action
- **State Space**: represents the state space of the environment
- **Action Space**: represents all possible actions that an agent can perform in a given state

PPO Overview

- Policy Optimization method that helps to stabilize the learned policy
- Optimizes a policy using small updates
  - Avoids drastic changes
- Clipped loss function is used to prevent the new policy from diverging too far from the old one



Algorithm 1 PPO-Clip

Input: initial policy parameters θ_0, initial value function parameters ϕ_0

for $k = 0, 1, 2, ...$ **do**

Collect set of trajectories $\mathcal{D}_k = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.

Compute rewards-to-go $\hat{R}_t$.

Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}.

Update the policy by maximizing the PPO-Clip objective:

$$\theta_{k+1} = \arg\max_{\theta} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} A^{\pi_{\theta_k}}(s_t, a_t),\ \ g(\epsilon, A^{\pi_{\theta_k}}(s_t, a_t)) \right),$$

typically via stochastic gradient ascent with Adam.

Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k| T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2,$$

typically via some gradient descent algorithm.

end for

RL Fine Tuning for NN Engine

Objective

- Use Proximal Policy Optimization (PPO) to train a reinforcement learning agent to play chess
- Improve the neural network based model by having it play against stockfish and learn from it

Custom Chess Environment

- Encode the chessboard as a 12x8x8 tensor
  - To conform to NN input
- 64x64 action space for all possible moves
  - Possible moves != valid moves
  - Possible moves are any combination of moving from a square to another
  - Use stockfish as an opponent to learn from