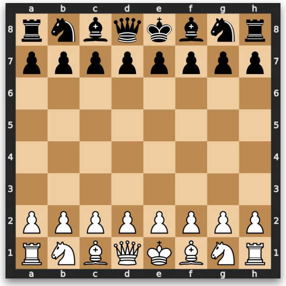
Retrieval-based chess engine

Anirudh Ajith
Gudni Nathan Gunnarsson
David Xu

Motivation

- Chess as milestone in AI
- High branching factor - cannot be solved practically
- Traditional chess engines
- Embedding chess boards as vectors - but which objective?



Can retrieving expert-level chess moves from similar board positions help find good moves?

Related Work

- Chess engines
  - Deep Blue
  - Stockfish
  - AlphaZero
- Chess position embeddings
  - Zobrist hashes
  - Chess2Vec
  - ChessPos
- Retrieval augmented LLMs
  - REALM
  - REPLUG

Methods

- **Dataset**: Lichess database, standard games, with players having elo > 2000, excluding bullet games. This dataset has a total of almost 5B games, which accounts for about 320B unique chess positions.
- **Encoding**: We used the chesspos embedder, which tries to apply similar embeddings for subsequent moves.
- **Pinecone**: We stored ~120M embeddings of chess positions
- **Chesscone**: Queries database, tries to directly play retrieved moves.
- **ChessBERT**: Can a transformer model be trained to use retrieved chess positions to deduce good moves?
- **Evaluation**
  - Winrate vs RANDOM
  - Recall@k of optimal move
  - Average rank of recommended move according to Stockfish engine

Dataset

- High branching factor → dataset needs to be extremely large for good coverage.
- The Lichess dataset has a total of almost 5B games, which accounts for an estimated 320B unique chess positions.
- Filter for only >2000 elo players in hopes of learning only good moves
- Also filter out “bullet” games to learn more well-considered moves.
- Finally we discard duplicate positions.

Pinecone Index

- Pinecone was extremely useful, but presented some challenges.
- Fast upserts. Limitations on queries, and vector storage space.
- Database schema:

```
{
     'id': <FEN>,
     'vector': <ChessPOS embedding>,
     'metadata': {'move': <move>}
}
```

- Using fen as the id for our schema allows us to easily eliminate duplicates via pinecone.

ChessPos Embeddings

Assumption: If positions X and Y are similar, then a good move for X is also good for Y

- Use existing chess autoencoder ChessPos to embed chess positions
    - CNNs serve as backbone architecture for encoder and decoder
- Train with a contrastive triplet loss function as well as reconstruction loss
    - Position P is positive example for position A if P directly follows from A in a chess game
    - Negative examples N are simply randomly drawn positions
- Contrastive loss pushes similar positions together in the latent space
- Reconstruction loss ensures that embeddings still contain info about the position

https://github.com/patrickfrank1/chesspos

Chesscone

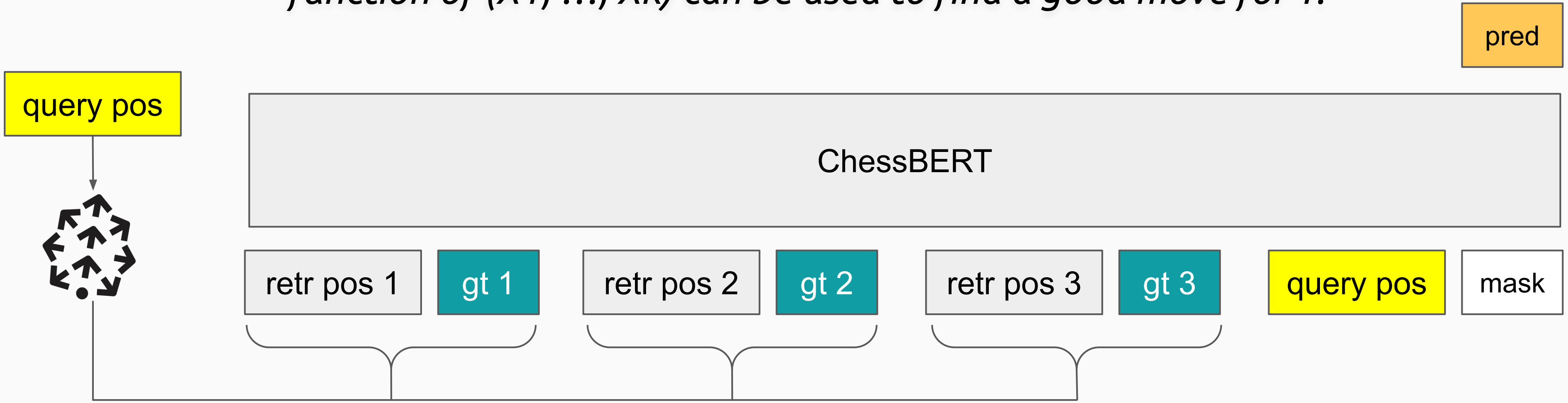
- Perform knn lookup of input position against Pinecone index
- Lookup returns list of positions and corresponding optimal moves
- Iterate through positions in order of highest similarity and returns the first legal move
    - If no legal move found, default to RANDOM policy

Image by ChatGPT4 + DALL·E

ChessBERT

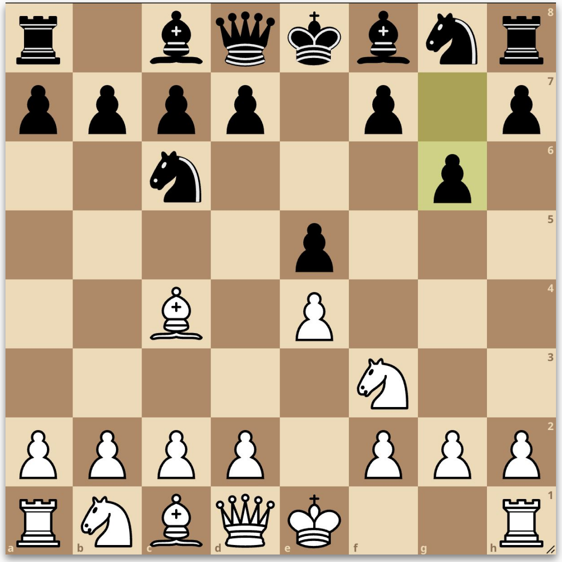
- Chesscone’s naive decision rule is only barely better than RANDOM
- Has difficulty generalizing to positions not seen in the index
- Can a shallow transformer model do better?
    - Attend to k “similar positions + the ground truth moves” to deduce good move for query position

Assumption: If positions X1, X2, …, Xk are similar to Y, then some shallow function of (X1, …, Xk) can be used to find a good move for Y.



ChessBERT: Architecture

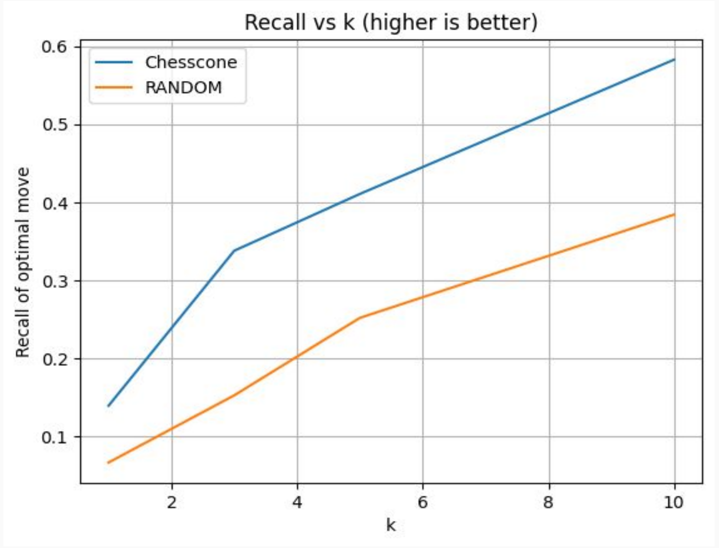
- Goal: Engineer inductive priors for chess positions
    - Represent every game state as a bag of chess pieces
        - Represent every piece as a sum of
            - Piece emb. (eg. Knight, Bishop)
            - File emb. (eg. a, b, c)
            - Row emb. (eg. 1, 2, 3)
            - [Segment emb. (eg. retr pos 1, gt 1, …, query pos)]
    - Represent the target pred embd. using the same scheme
        - Nc3 = “N” + “c” + “3” + (seg emb.)
- Train “actual pred embd” → “target pred embd”
    - We use CLIP loss
    - Train on 300k context-position pairs

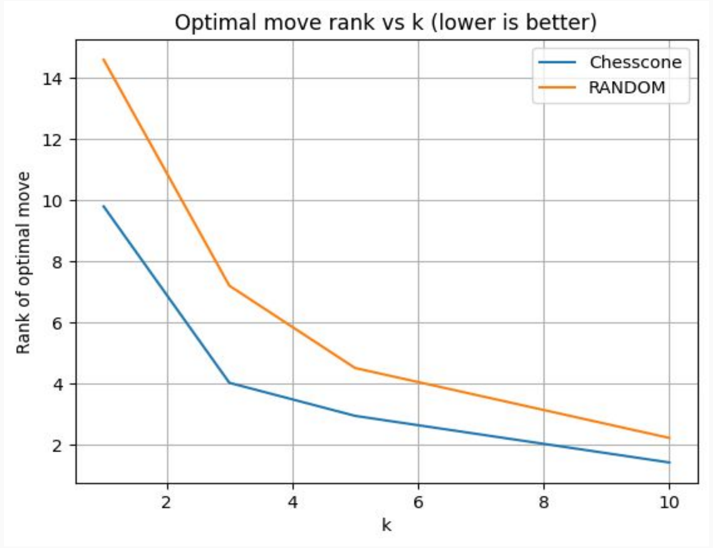


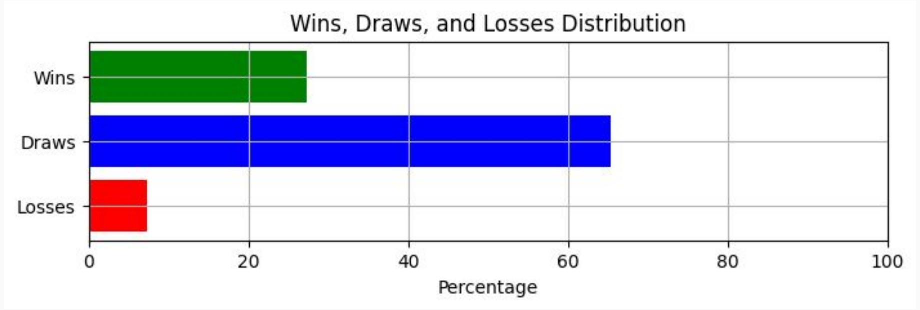
Evaluation Methods

- Original intention was to report steady-state ELO on lichess.org
  - Both engines too weak to get meaningful results
  - Too resource intensive
- Report the recall@k for our engines, treating Stockfish top move as the ground truth
- Simulate games of chess between our engines and RANDOM policy
- Report average rank of our engine’s top recommendation, according to Stockfish

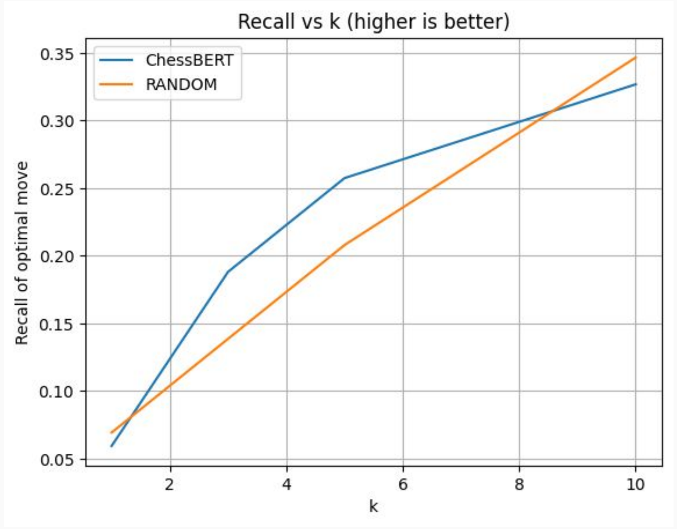
Results: **Chesscone**

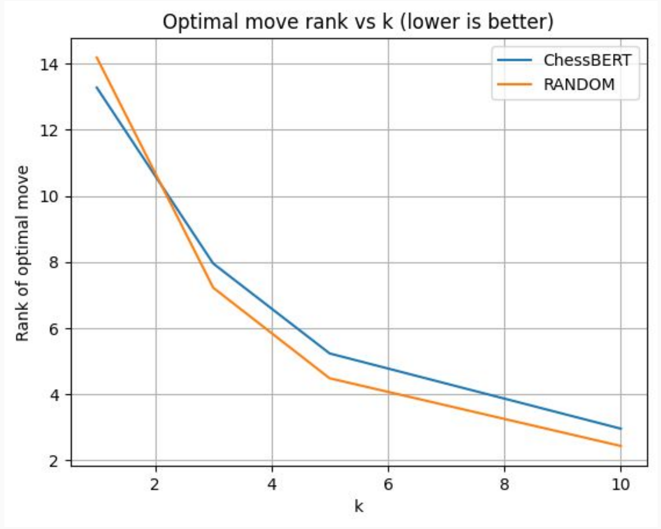


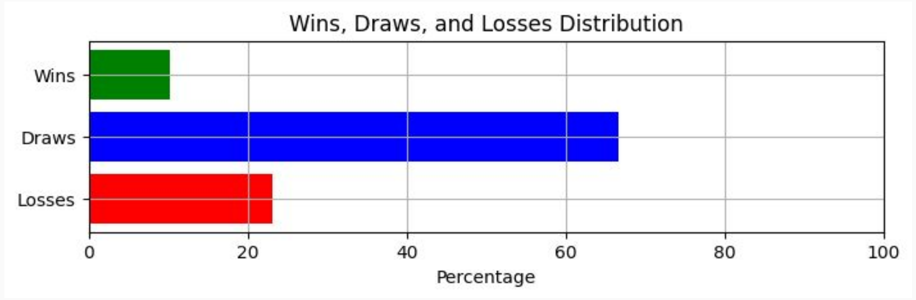




Results: **ChessBERT**







Next Steps

- Bigger Pinecone index - billion-scale
- More context positions
- More training time
- Alternative training objective?
- Hard negative mining
- Modify dataset to increase diversity of positions

COS
597A