Final Report: Transformer Chess Bot for Move Prediction and Strategy Evaluation

Kyle Tranfaglia, Dustin O'Brien

Summary

This report details the completion of our LLM-powered chess bot project, which aimed to develop a probabilistic chess AI that demonstrates human-like gameplay while maintaining competitive performance. Building on our midway progress, we further refined our model architecture, training data, and search algorithms. While we achieved some success in producing a functional chess AI with probabilistic decision-making, we encountered significant challenges in training the model to play at the initially targeted competitive level. Our final system achieved an estimated Elo rating of approximately 400, falling short of our initial goal of 2200, yet still demonstrating valuable insights about applying language model techniques to chess play and the importance of training data quality in developing game-playing AI systems.

Project Overview and Approach

Our project approached chess AI development from a probabilistic perspective rather than the traditional deterministic approaches used by engines like Stockfish. The core hypothesis was that by training a transformer-based model on human chess games and integrating it with Monte Carlo Tree Search (MCTS), we could create a system that plays more human-like chess while maintaining reasonable competitive strength.

This approach was chosen for several reasons:

- Chess, like natural language, involves nuanced pattern recognition that transformer architectures excel at identifying
- Probabilistic decision-making better reflects human gameplay compared to traditional minimax algorithms
- Recent research has demonstrated the viability of using language model techniques for chess position evaluation
- The approach allows for adjustable "personality" in gameplay by varying search parameters

Our implementation comprised three main components:

1. A transformer-based neural network for position evaluation
2. A Monte Carlo Tree Search system for move selection
3. A user interface that displays position evaluations and allows for configuration

We explored the solution space across multiple dimensions:

Model Architecture Variations:

- Initial implementation of a standard neural network for basic position evaluation

- Enhanced transformer architecture with self-attention mechanisms
- Introduction of residual connections to attempt to mitigate vanishing gradient problems
- An improved reordered layer attention pipeline
- Higher dropout attention model

Training Data Exploration:

- Initial training on a subset of the Lichess dataset of 6.25 million games
- Transition to exclusively professional GM games to improve quality
- Final hybrid approach combining high-quality GM games with selected mid-Elo games
- Limiting the opening repertoire to reduce variability in the early game
- Began inclusion of "landslide" games, where tactical mistakes were severely punished

Preprocessing Techniques:

- Vector encoding of chess positions based on established research
- Position averaging to handle contradictory evaluations
- Filtering out short GM draws (less than 30 moves) to reduce noise
- Balanced representation of game outcomes (wins, losses, draws)
- Validation to remove illegal moves and positions

Search Algorithm Adjustments:

- Integration of neural network evaluations with MCTS
- Experimentation with various exploration-exploitation parameters
- Fixed iteration count to ensure consistent performance and time complexity

Evaluation Methodology

We evaluated our model's performance using multiple complementary methods:

Move Prediction Accuracy:

- Measuring the model's ability to predict expert moves using the test dataset
- Comparing the model's top three move suggestions against actual moves played

Human-likeness Assessment:

- Qualitative analysis of the model's playing style and strategic tendencies
- Evaluation of the model's handling of different game phases (opening, middlegame, endgame)

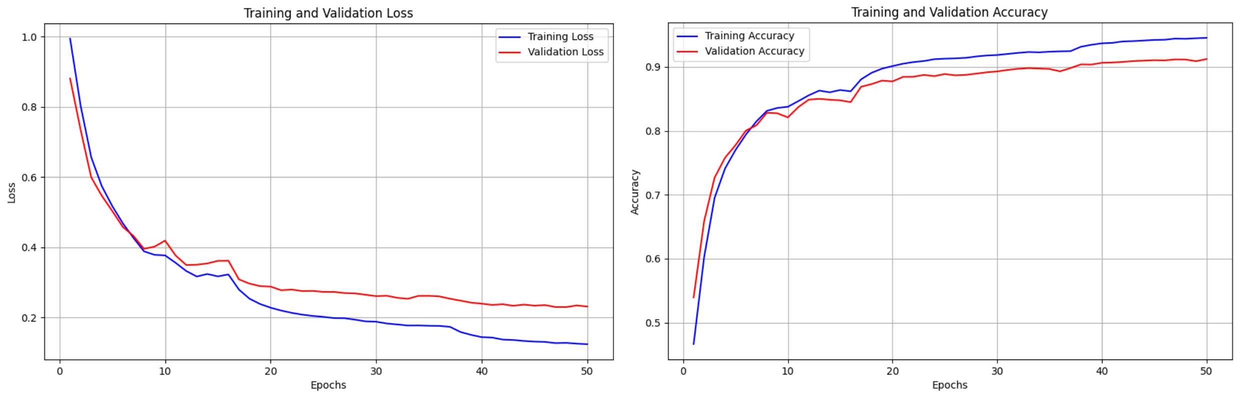
Live Performance Testing:

- Direct gameplay against human players and itself
- Estimating Elo rating through match outcomes against benchmarked opponents

Data Quality Validation:

- Using dataTester.py to assess training data for bias, completeness, and consistency
- Testing for NaN values, missing attributes, and game phase representation

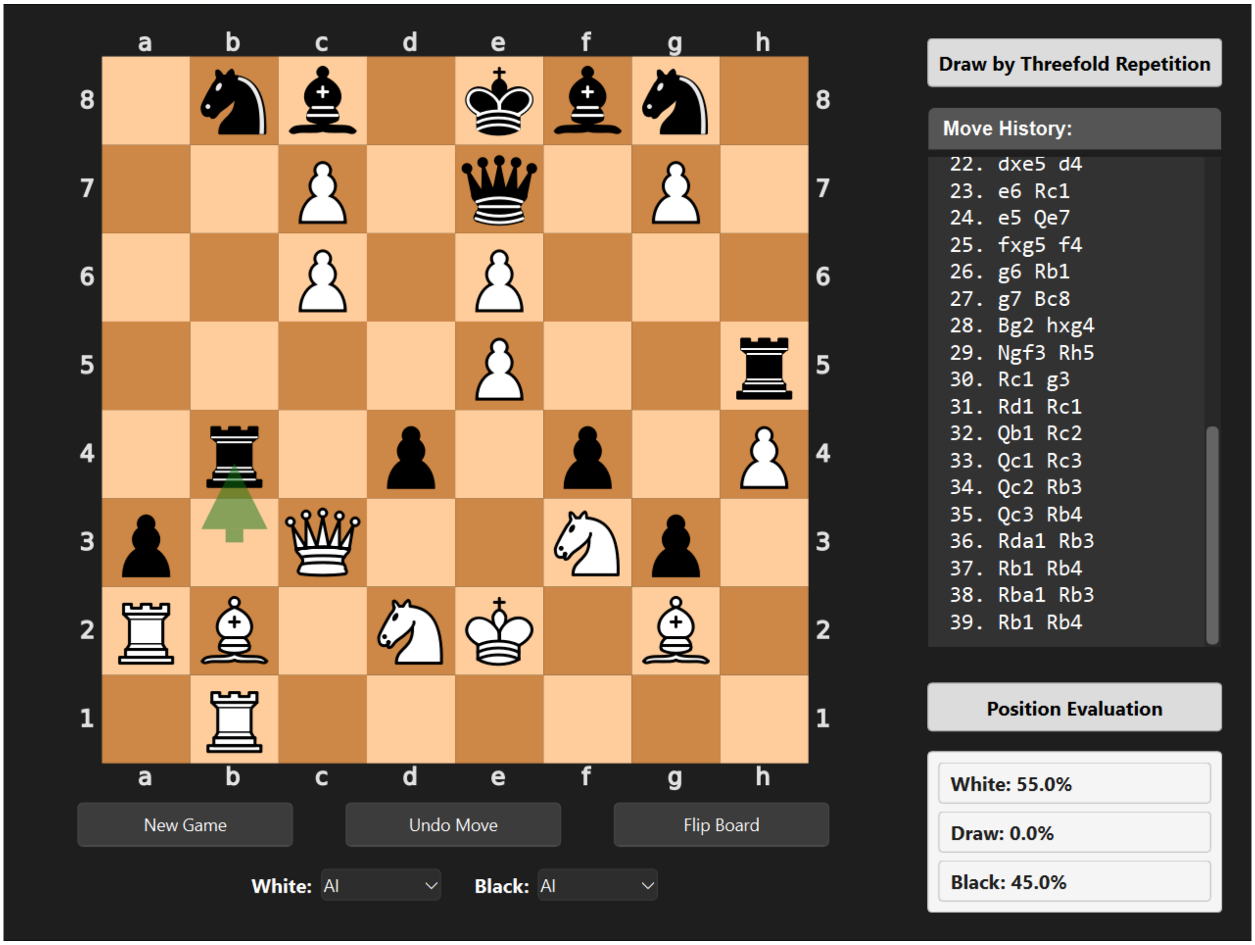
Results and Analysis



What Worked:

1. Architecture Implementation:
   - ○ The transformer-based architecture with attention mechanisms was successfully implemented
   - ○ Position averaging techniques effectively reduced contradictions in the training data
   - ○ Integration of the neural network evaluator with MCTS created a functional system that quickly plays moves with the model weights
2. GUI Development:
   - ○ Fully functional chess interface with move validation, history display, evaluation visualization, and some quality of life features
   - ○ Real-time display of model evaluations and probability distributions
3. Data Processing Pipeline:
   - ○ Efficient conversion of algebraic notation to tensor-based board representations
   - ○ Robust filtering system to improve data quality and balance
4. Pattern Recognition**:**

- The model demonstrated some ability to recognize basic chess patterns
- Showed improvement in logical play after dataset refinement
- Played human-like moves at a low level in comparison to popular low-performance bots, although this is difficult to distinguish for beginner play



Draw by Threefold Repetition

```
22. dxe5 d4
23. e6 Rc1
24. e5 Qe7
25. fxg5 f4
26. g6 Rb1
27. g7 Bc8
28. Bg2 hxg4
29. Ngf3 Rh5
30. Rc1 g3
31. Rd1 Rc1
32. Qb1 Rc2
33. Qc1 Rc3
34. Qc2 Rb3
35. Qc3 Rb4
36. Rda1 Rb3
37. Rb1 Rb4
38. Rba1 Rb3
39. Rb1 Rb4
```

White: 55.0%

Draw: 0.0%

Black: 45.0%

White: AI Black: AI

What Did Not Work:

1. Performance Below Target:
   - Final Elo rating of approximately 400, substantially below our target of 2200
   - The model struggled to consistently recognize tactical opportunities and threats
2. Training Limitations:
   - Hardware limitations prevented large data usage, such that a suboptimal amount of chess games were used to prevent hitting memory thresholds
3. Opening Knowledge:
   - Even after dataset refinement, the model showed weaknesses in opening theory

  - Failed to consistently apply standard opening principles

4. Tactical Awareness:
   - The model frequently missed opportunities to capture undefended pieces
   - Struggled to recognize immediate threats and blunders

Insights and Analysis

- The most critical insight from our project was the fundamental importance of training data quality and composition for chess AI development. Our initial hypothesis that a transformer architecture trained on human games would develop strong chess understanding proved only partially correct.

- The training data composition presented a critical dilemma: professional GM games contained high-quality chess but lacked examples of punished mistakes, while lower-Elo games contained instructive mistakes but also introduced noise from suboptimal play patterns. Our final hybrid approach improved results but still faced limitations in learning tactical patterns.

- The balance between dataset size and quality proved challenging. While we had access to millions of games, the sheer volume of data introduced contradictory evaluations for identical positions. Our position averaging techniques helped mitigate this issue, but couldn't fully resolve the underlying tension between data quantity and consistency.

- Memory and computational constraints limited our ability to fully explore very large model architectures. The resource requirements for training larger models with more extensive datasets exceeded our available hardware capabilities, forcing compromises in model complexity and training duration.

- The most successful aspect of our approach was the integration of the transformer model with MCTS, which created a functional system capable of playing complete chess games with reasonable move selection. The probabilistic nature of the system did produce more varied and human-like play patterns compared to traditional engines, partially validating our core hypothesis.

- The order of layer functions is crucial for good models. All early models had significantly reduced performance caused by late layer normalization over starting layer normalization.

Conclusion and Future Work

Our project demonstrated both the potential and limitations of applying transformer-based models to chess play. While we successfully created a functional chess AI using a probabilistic approach, the system fell short of our performance targets, achieving an estimated Elo of 400 rather than the targeted 2200.

The project yielded valuable insights about the challenges of training neural networks for chess, particularly the critical importance of training data quality and the difficulty of balancing dataset size with consistency. The successful implementation of our architecture and integration with MCTS provides a foundation for future improvements.

Moving forward, several promising directions exist for enhancing the system:

1. Reinforcement Learning: Implementing a self-play reinforcement learning approach similar to AlphaZero could overcome the limitations of supervised learning on human games.

2. Hybrid Evaluation: Combining neural network evaluations with traditional chess heuristics might improve tactical awareness while maintaining human-like strategic understanding.

3. Targeted Training: Creating specialized datasets focused on specific chess skills (tactics, endgames, etc.), concentrating on games that punish bad moves and reward strong moves, and training separate models might produce better overall performance.

4. Enhanced Hardware Resources: Access to more powerful computing resources would enable training of larger models on more extensive datasets with longer training durations, ultimately increasing the ability to capture simple and nuanced patterns.

5. Distributed Computing: Splitting computation and, more specifically, data across multiple computers, can allow for greater effective memory. Our current approach, although not finished, is to split large Lichess data into multiple data “shards.” We currently have 150 shards of data, roughly 700 - 800 MB each. We believe it to be possible using MPI or Slurm to have multiple clustered computers select a small set of shards to train and using Pytorch DDP implementations to average together gradients, allowing for effective training on large data.

6. Tree Integration: Unknown till late in research, Alpha Zero, one of the main inspirations for this research, operates on a specialized loss function for MCST integration, where the model directly outputs policy and value for the tree rather than outputting win probabilities. Integrating such functions may be beneficial to performance; however,

current understanding is limited.

7. Data Quantization: Quantization is a common technique to compress large data into smaller datasets. This is particularly convenient given the relation to our memory bottleneck. Although insufficient to train on the scale we intended, it remains a possibility for allowing smaller clusters of computers to use large chess datasets. Our models currently use the default 32-bit floating point single precision numbers, given the limited range of our data, usually 1 or 0, with only some exceptions, our tensor datasets can likely use significantly smaller approximations, such as 8-bit for position distributions.

8. Fine-tuning Approach: Starting with a pre-trained model on general chess data and then fine-tuning on high-quality games might produce better results than our current approach.

Despite falling short of our performance target, the project successfully demonstrated the feasibility of creating a chess AI using transformer-based models and probabilistic decision-making. The human-like qualities observed in the system's play patterns suggest that further refinement of this approach could yield a chess AI that better reflects human cognitive processes while maintaining competitive strength.

Contributions

Kyle's Contributions

- Creation of Chess GUI and all UI elements in the main application
- Construction of a larger attention-based model
  - Implemented a transformer-like architecture
  - Included a MCTS for move choice
  - Introduced position averaging
  - Included self-attention blocks, residual connections, feed-forward networks, and layer normalization
- Dataset creation
  - Initial Lichess datasets (small and large)
  - GM datasets (varying-size GM-only games)
  - Balanced GM dataset (balanced wins, losses, and draws)
  - 10 varying datasets with different minimum Elo ranges and minimum player Elos, with balancing
- Pre-loaded model weights integration testing with the GUI application
- Fine-tuning MCTS and the main application to more optimally select and display moves
- Data reading and conversion to tensors
- Model and pre-loaded weight testing on the GUI application

Dustin’s Contributions

- Focused on Neural Network implementation
- Created PyTorch model designs and researched methods to improve models
    - Found and recommended MCTS for the tree algorithm
    - Implemented Xavier Initialization, GeLU, Adam optimization, and an extensive list of common model techniques used in cutting-edge theory
- We formulated strategies to create and improve Chess architecture capabilities based on industry-standard research
- Wrote and ran code handling converting Lichess PGN data to Tensor-based datasets for Model usage
- Built and trained multiple models, including
    - Initial large MLP
    - Initial Attention architecture
    - Modified and improved Attention Architecture
    - High dropout attention architecture