Amortized Planning with Large-Scale Transformers: A Case Study on Chess

Report by Takuma Osaka, Pun Chaixanien, Koji Kimura, Luke Leh
github.com/Pun2341/transformers_chess

1. Introduction

Our project aimed to reproduce the searchless chess model presented in Google DeepMind's paper, "Amortized Planning with Large-Scale Transformers: A Case Study on Chess" by Ruoss et al. (2024). The paper investigates the capacity of decoder-only transformers in approximating planning algorithms, specifically Stockfish 16's (a state of the art chess engine) chess evaluation, without performing explicit search. The authors employ large-scale transformers trained via supervised learning on ChessBench, a dataset of 10 million chess games annotated by Stockfish 16 with state and action-values. Their key finding demonstrates that sufficiently scaled transformers can generalize effectively to unseen positions and achieve Elo ratings comparable to traditional search-based chess engines.

2. Chosen Result

We aimed to reproduce the paper's decoder-only transformer chess engine focusing on the reproduction of the action-value prediction task described in Figure 1 of the original paper. This involves predicting the win percentages for all legal moves given a chess board state. This task was chosen because action-value predictions yielded the strongest performance in the original study, directly correlating with the model's Elo ratings and puzzle-solving abilities. Reproducing this result allowed us to evaluate how closely a resource-constrained implementation of the transformer model could match the reported performance in the paper.

3. Methodology

Our re-implementation of the Searchless Chess engine followed the four main components outlined in the original paper: preprocessing, model architecture, training, and evaluation. We worked in parallel to complete each component. Due to limited compute (2 hrs of GPU credits per day), we focused our re-implementation on action-value prediction, i.e. using board state and move to predict win percentage, since it performed best in the paper and used simple heuristics to help guide the engine's best move.

In the preprocessing stage, we recreated the FEN string tokenizer and value binning methods used by the authors. Additionally, we built a lightweight BagReader to read from the ChessBench Datasets' .bag files, focusing on read functionality since we did not require data writing or modification. Our model architecture is a decoder-only Transformer built using PyTorch, modeled after the original JAX-based implementation. We chose to use PyTorch for ease of use and consistency with our coursework.

For training, we integrated a simplified training pipeline (no sharding or parallel training) with Google Colab to maximize GPU usage and used Google Drive to load datasets and manage model checkpoints across sessions. For evaluation, we used the puzzle dataset included in ChessBench to

estimate Elo ratings and conducted head-to-head matches between our bots to assess model performance and scaling.

4. Results & Analysis

Our implementation of the transformer-based chess model yielded performance below that reported in the original paper, yet it demonstrated promising results within the scope of our computational and data constraints. As shown in Figure 1, our model achieved an average puzzle accuracy score above 0.6 for puzzles rated under 800, with the accuracy gradually decreasing as the puzzle difficulty increased. This general trend is expected and reflects the transformer model's reliance on common game patterns, and highlights the limitations of generalization when faced with high-complexity positions that deviate from the training distribution.

In contrast, Figure 2 (from the original paper) compares the puzzle accuracy of their 270M model with several well-established chess engines, including Stockfish, Leela Chess Zero, and AlphaZero. Their model exhibits competitive performance across a wide range of puzzle ratings. The discrepancy between our model and the paper's results can largely be attributed to two factors: model size and data scale. Whereas the original study used a 270M parameter transformer trained on approximately 1.1 TB of game data, our model consisted of only 270K parameters and was trained on approximately 1 GB of data. These differences inherently limit the model's capacity and generalization ability. Nonetheless, our model achieved an estimated playing strength of around 600 Elo, in contrast to the paper's 2800 Elo estimate for their largest model.

Despite these limitations, our model still achieved an estimated Elo of approximately 600, and when enhanced with domain-specific heuristics (our own addition), including evaluations for material balance, center control, threats, and blunders, it reached an estimated puzzle-solving rating of 1400 Elo as shown in Figure 3. The improvement is particularly evident in low- to mid-rated puzzles, where rule-based heuristics can complement learned priors effectively.

Notably, our model performed strongly in early-game scenarios, benefiting from high overlap with common openings seen in the training data. However, as the number of moves increased and board positions diverged from typical patterns, the model struggled, likely due to data sparsity and limited representational capacity. Addressing this with larger models or more data was infeasible, so we instead integrated heuristic scoring with the model's predicted move probabilities, forming a hybrid decision-making engine.

We further evaluated our model by playing matches against Lichess AI bots of increasing difficulty. It consistently outperformed Lichess Level 2, which has an estimated Elo rating of 1000, further validating the efficacy of our hybrid approach despite resource constraints.

In summary, although we couldn't match the full performance of the 270M-parameter transformer described in the original paper, our results underscore the feasibility of building competent chess models with modest resources. Moreover, the integration of our heuristics with transformer-based predictions presents a promising direction for improving low-resource chess engines.

5. Reflections

The project highlighted the importance of computational resources, dataset quality, and strategic preprocessing steps in supervised transformer training. The effective tokenization and binning

strategies that we employed were crucial in helping the model learn best. The introduction of specific learning heuristics provided a meaningful boost to learning efficiency, illustrating how strategic adjustments can mitigate computational constraints. Future work should explore scaling model size, refining training protocols, and experimenting with deeper transformer architectures. Furthermore, advanced loss functions such as HL-Gauss, which provides continuous gradient signals beneficial for the inherently continuous nature of chess action-values, show promise in improving model generalization and overall chess-playing strength.

Our experience confirmed the feasibility of transformer-based supervised learning as a viable approach to approximating sophisticated planning algorithms, aligning with the broader discourse of employing large-scale models for algorithmic approximation tasks.

6. References

Ruoss, A., Delétang, G., Medapati, S., Grau-Moya, J., Wenliang, L. K., Catt, E., Reid, J., Lewis, C. A., Veness, J., & Genewein, T.

(2024). Amortized Planning with Large-Scale Transformers: A Case Study on Chess. arXiv. https://doi.org/10.48550/arXiv.2402.04494Google Scholar+8arXiv+8Hugging Face+8

7. Appendix

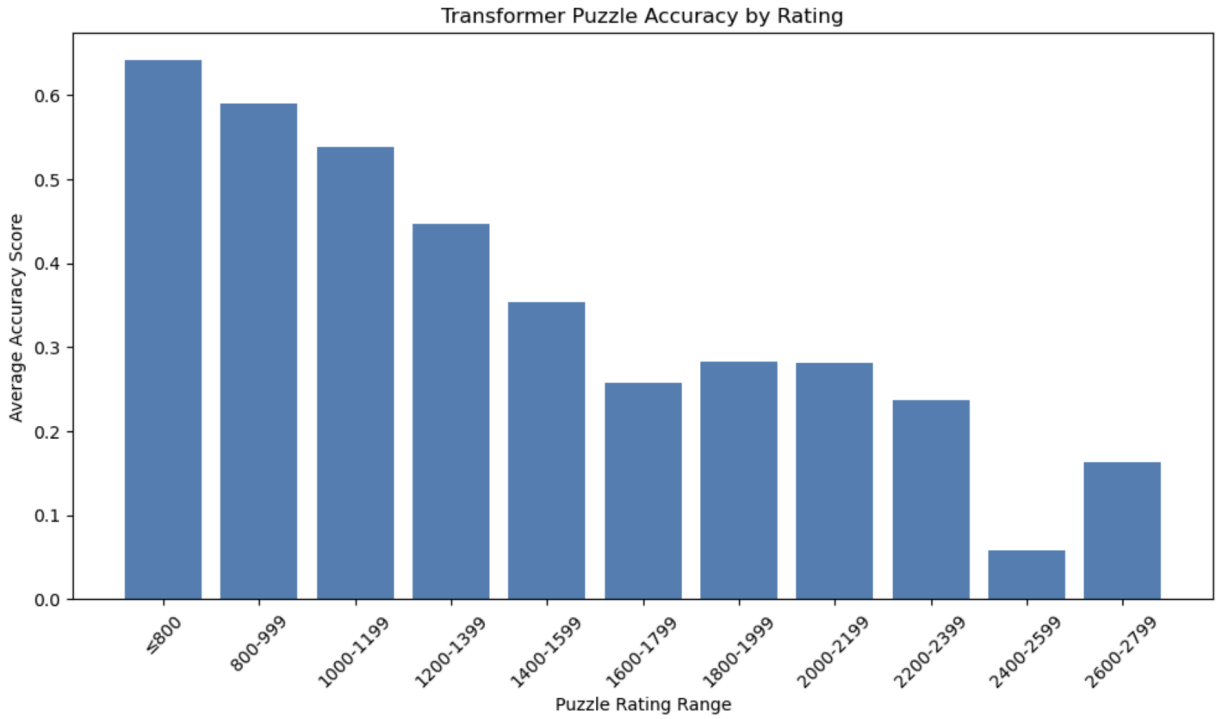

Figure 1: Our transformer engine's average puzzle accuracy score by puzzle rating

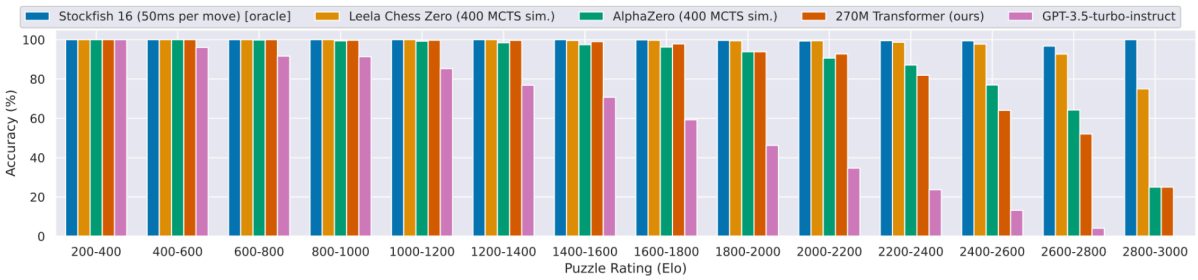

Figure 2: The paper's puzzle accuracy by rating comparison between their 270M transformer model with other well known chess engines

```
Puzzle Evaluation Summary
------------------------------
Total puzzles evaluated: 979
Weighted mean score: 0.353
Estimated ELO (weighted mean-based): 1402

Score distribution by rating bin:
  ≤800: avg score = 0.642, puzzles = 74
  800–999: avg score = 0.590, puzzles = 136
  1000–1199: avg score = 0.539, puzzles = 154
  1200–1399: avg score = 0.447, puzzles = 111
  1400–1599: avg score = 0.354, puzzles = 114
  1600–1799: avg score = 0.259, puzzles = 98
  1800–1999: avg score = 0.282, puzzles = 100
  2000–2199: avg score = 0.282, puzzles = 93
  2200–2399: avg score = 0.237, puzzles = 55
  2400–2599: avg score = 0.058, puzzles = 33
  2600–2799: avg score = 0.163, puzzles = 11
```

Figure 3: Summary statistics of our transformer model tested on the puzzle dataset for accuracy by rating