Stockfish or Leela Chess Zero? A Comparison Against Endgame Tablebases

Quazi Asif Sadmine[1⋆], Asmaul Husna[1⋆], and Martin Müller[1][0000−0002−5639−5318]

University of Alberta, Edmonton, Canada
{sadmine,asmaul,mmueller}@ualberta.ca

Abstract. The game of chess has long been used as a benchmark for testing human creativity and intelligence. With the advent of powerful chess engines, such as Stockfish and Leela Chess Zero (Lc0), endgame studies have also become a tool for evaluating the capabilities of machine chess engines. In this work, we conduct a detailed study of Stockfish and Lc0, two leading chess engines with distinct methods of play, using chess endgames with varying numbers of remaining pieces. We evaluate the programs' move decision errors when using only the raw policy network as well as when using a small amount of search. We provide insights into the strengths and weaknesses of Stockfish and Lc0 in handling complex endgame positions by exploring common mistakes and identifying interesting behaviours of the engines based on the position of the opponent's last pawn remaining on the board.

Keywords: Computer Chess · Leela Chess Zero · Stockfish · Chess Endgame Tablebases.

1 Introduction

Playing chess requires strategic thinking, planning, and decision-making skills. Endgame studies, which involve analyzing and solving complex chess positions with a limited number of pieces remaining on the board, have traditionally been used to test human creativity and intelligence. Endgame studies have also become a tool for evaluating the capabilities of chess engines [9, 12].

Before the widespread adoption of deep neural networks and the emergence of AlphaZero [18, 19], Stockfish was the leading chess engine. AlphaZero [19] demonstrated superhuman performance in complex board games - chess, shogi, and Go. This neural network-based program has exceptional move selection and state evaluation abilities. Inspired by the success of AlphaZero, Stockfish incorporates a neural network known as NNUE (efficiently updatable neural network) [16] into its traditional chess engine from version 12. However, NNUE is a relatively simple and shallow feedforward neural network, whereas AlphaZero uses a more complex and deep convolutional neural network (CNN).

⋆ equal contribution

Despite their remarkable performance, these programs are not perfect and still make mistakes. To better understand how these modern programs learn to play as well as explore the limits of their playing abilities, we turn to a sample problem that has known exact solutions - chess endgames. While the full game of chess has not been solved, exact solutions for endgames with up to seven pieces have been computed and compiled into endgame tablebases. In this study, we utilize the open-source programs Leela Chess Zero (Lc0) [1], which follows the AlphaZero-style approach, and Stockfish [4] to analyze chess endgames and investigate the gap between strong and perfect play. Additionally, we compare their respective performance in these endgames. Through this analysis, we aim to shed light on their playing abilities and provide insights into their strengths and limitations. We design a comprehensive methodology involving extensive experiments to address the following research questions:

- How well do these two leading chess engines perform compared to perfect play?
- Which is easier to predict for the engines? Wins or draws?
- Which engine's policy networks perform well in evaluation?
- How much do the programs improve after using a small search budget?
- In an interesting board configuration, when only one pawn of the opponent remains, how much do their performances differ from each other?

2 Related Work

The original AlphaZero paper [19] compared the performance of AlphaZero with Stockfish for chess in terms of gameplay. The authors compared win-draw-loss percentage against the baselines in a tournament under the same time settings, and AlphaZero outperformed Stockfish. However, it's worth noting that NNUE had not been introduced into Stockfish at that time.

The work by Haque et al. [9] compares Leela Chess Zero with perfect play from endgame tablebases. The authors also analyze different case studies of Lc0's policy and search, which give more insights into the performance. In our study, we conduct a detailed analysis with more complex endgames and do further experiments where the engines tend to make more mistakes.

Two noteworthy papers in the field of comparing game engines or algorithm performance to perfect play are worth mentioning. Lassabe et al. [12] use genetic programming to solve chess endgames by combining elementary chess patterns defined by domain experts. In Romein and Bal [17], the game of awari was first solved and then used as a basis to measure the performance of two world champion-level engines from the 2000 Computer Olympiad.

3 Background

3.1 Endgame Tablebases

Chess endgames are sub-problems that occur when only a reduced set of game pieces remain on the board, and the full rules of chess still apply. The solutions are publicly available in databases known as endgame tablebases [14]. Each solution in the tablebase includes the outcome of the game assuming perfect play from both players, along with the optimal moves for reaching that outcome and specific metrics such as the number of plies required to achieve the result. Endgame tablebases hosted online differ in storage size and metrics [11, 13].

Tablebase generators are also available and allow for the creation of custom endgame tablebases. Among the available options, Syzygy [13] and Gaviota [5] tablebases are popular and widely used, and they are also free for public access.

3.2 Stockfish

Stockfish is a highly robust open-source chess engine. It takes a position on the chessboard as input and generates a move as output using an alpha-beta pruning search algorithm [7]. To cope with the vast search space of chess, Stockfish employs techniques such as forward pruning and reduction to reduce the search space [10]. The evaluation function of Stockfish determines whether a leaf node is favourable for White or Black by evaluating factors such as the current positions of the pieces, piece activity, game phase, etc.

From version 12, Stockfish uses an efficiently updatable neural network (NNUE) [16] as its evaluation function. This neural network is capable of predicting the output of the evaluation function at a moderate network depth. The architecture of NNUE is shallow, consisting of four layers, and is specifically optimized for speed on the CPU. NNUE has greatly enhanced the performance of Stockfish, making it even more powerful in analyzing chess positions and generating strong moves.

3.3 Leela Chess Zero

Leela Chess Zero (Lc0) is a chess adaptation of the popular Go program Leela Zero. Both open-source programs aim to replicate the success of AlphaZero in their respective games [2]. Similar to AlphaZero [19], Lc0 takes a sequence of consecutive raw board positions as input and utilizes a two-headed network for policy and value estimation. It uses Monte Carlo Tree Search (MCTS) as a search algorithm [6] to find the best move. Over time, the Lc0 developers introduced enhancements that were not present in the original AlphaZero, including additional auxiliary outputs such as the "moves left" head, which predicts the number of plies remaining in the current game [8]. Another auxiliary output called the "WDL head" separately predicts the probabilities of winning, drawing, or losing the game [3].

The raw network policies of Stockfish and Lc0 are very different from each other. It is challenging for humans to comprehend what is happening inside these networks [15]. However, comparing the move decisions of these engines provided solely by the network can help to understand how well they have learned to evaluate endgame positions.

4 Dataset and Evaluation Method

4.1 Dataset Generation and Preprocessing

We follow the dataset generation and preprocessing technique of Kryukov used [11] in [9]. We first place the kings on the board in all possible cases. Then we place other pieces one at a time to generate all positions with three, four, and five pieces. We apply colour swapping, horizontal mirroring, vertical mirroring, and diagonal mirroring for a more efficient generation. Then we remove illegal positions such as pawns on promotion ranks, positions where the player to move can capture the king, etc. We disable castling and en passant captures for all positions for simplicity. We also set the halfmove clock to 0 and the fullmove counter to 1. We obtain all unique legal positions of an endgame and append the *syzygy* endgame tablebase's perfect information for each position. Due to the abundance of five-piece chess endgame positions and limited resources, we sample 1% of all five-piece positions. We consider a total of four three-piece, seven four-piece, and six five-piece tablebases for our experiment. The three-piece tablebases consist of one white queen, rook, knight, or bishop, and both kings. In four-piece tablebases, we include two pieces for each player to maintain a balanced power dynamic between the two sides. In the five-piece tablebases, we position only one pawn for black while white receives any two pieces among the queen, rook, knight, and bishop.

We generate all the positions as Forsyth-Edwards Notation (FEN) [1] strings in our datasets. White pieces are represented by capital letters, and black pieces are represented as lowercase letters in each FEN string. We name our datasets accordingly. For instance, the dataset *KQkp* has a white king, a white queen, a black king, and a black pawn.

We store all positions in a MySQL database and separate the data for each tablebase into two parts: (i) positions that result in a win and (ii) positions that result in a draw. We do this separation to analyze deeply if the engines have any performance variation for winning or drawing positions. We skip losing positions because they are of no use in identifying mistakes made by the engines.

4.2 Engine settings

For our experiment, we consider the latest versions available at the time of this work, which are Stockfish 15.1[2] and Lc0 0.29.0[3]. To fairly compare the engines, we ensure that they have similar strengths. The highest Elo rating available for Stockfish at the time of this work is 2850. Therefore, we use this Elo rating for both engines. Initially, we focus on comparing only the policy. As the backend, we use CPU for Stockfish since it only runs on CPU. For Lc0, we use *cudnn* on a Linux machine equipped with an Nvidia Titan RTX GPU.

[1] https://www.chessprogramming.org/Forsyth-Edwards_Notation
[2] https://stockfishchess.org/blog/2022/stockfish-15-1
[3] https://github.com/LeelaChessZero/lc0/releases/tag/v0.29.0

5 Experimental Results

5.1 Wrong Play Analysis

Table 1 displays the percentage of mistakes made by the raw policies of Stockfish and Lc0 when evaluating move decisions for three, four, and one five-piece endgame tablebases. Mistakes are defined as moves that change the game-theoretic outcome. When there is a winning move available in the endgame tablebase, a mistake is counted if the engine suggests its best move that results in a draw or loss. When there is a drawing move available in the tablebase, a mistake is counted if the engine suggests a move that results in a loss.

Table 1 shows that Stockfish performs better than Lc0 for the three-piece tablebases. However, for the four-piece tablebases, Lc0 shows better results than Stockfish, except for three tablebases: KQkb (win), KPkp (win), and KQkp (win). Figure 1b illustrates that Lc0 consistently outperforms Stockfish in all the tablebases, where perfect play results in a draw. Even though Stockfish shows a lower percentage of mistakes in the winning positions of KQkb, KPkp, and KQkp tablebases in Figure 1a, Lc0 still demonstrates very competitive results in those tablebases as well. In our 1% of the total positions for different five-piece endgame tablebases, Lc0 could only perform better than Stockfish in KRBkp (draw), KBNkp (win), and KBNkp (draw). Calculating from Table 1, we find that Stockfish makes an overall 1.47% and 1.67% of errors in winning and drawing positions, respectively, whereas Lc0 makes 1.32% and 1.07% of errors.

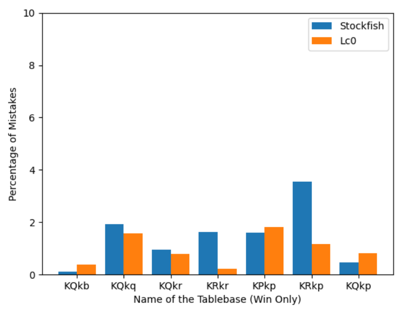
Based on this result, we conduct a study on the Average Centipawn Loss (ACPL) to gain further insights. A Centipawn[4] is 1/100 of a pawn used to evaluate a chess position. ACPL identifies how much 'value' a player loses while playing a wrong move. An ACPL close to zero indicates a very strong move. We choose to calculate the ACPL for the 4-piece tablebases of our datasets, as the results on the three-piece tablebases are almost perfect, and we have limited data on the five-piece tablebases.

To calculate the ACPL, we divide the mistaken positions of each 4-piece tablebase into two parts: (a) Positions where Stockfish plays the correct move but Lc0 plays the wrong move, and (b) Positions where Lc0 plays the correct move but Stockfish plays the wrong move. We choose this division to assess the relative strength of the incorrect move compared to the correct one. In (a), we obtain an ACPL of -408.28 with a standard deviation of 1739.31. In (b), we obtain an ACPL of 281.71 with a standard deviation of 1476.47. These values portray that both engines recognize their moves as very weak. However, when Lc0 plays the wrong move, the ACPL is further from zero compared to the other case. This suggests that Lc0 evaluates its mistaken position more accurately than Stockfish in this study.

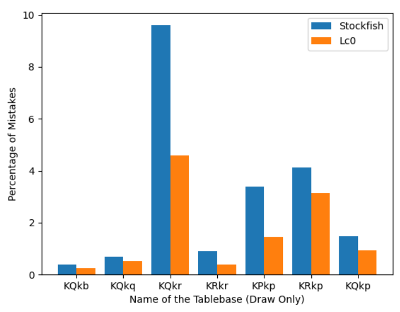
[4] https://chess.fandom.com/wiki/Centipawn

Table 1: Total number of mistakes by the policy net of Stockfish and Lc0.

#Pieces	**EGTB**	**W/D**	**Total Positions**	**Stockfish(Policy)**	**Lc0(Policy)**
3	KQk	W	18081	19 **(0.1%)**	173 (0.96%)
		D	2896	0	0
	KBk	D	52234	0	0
	KNk	D	53806	0	0
	KRk	W	21959	0 **(0%)**	23 (0.1%)
		D	2796	0	0
4	KQkb	W	701738	787 **(0.11%)**	2638 (0.38%)
		D	220956	843 (0.38%)	538 **(0.24%)**
	KQkq	W	934428	18038 (1.93%)	14682 **(1.57%)**
		D	1293823	8874 (0.7%)	6714 **(0.52%)**
	KQkr	W	890800	8512 (0.96%)	7057 **(0.8%)**
		D	49184	4745 (9.6%)	2261 **(4.6%)**
	KRkr	W	784918	12759 (1.63%)	1839 **(0.23%)**
		D	1892778	16313 (0.9%)	7311 **(0.4%)**
	KPkp	W	321303	5130 **(1.6%)**	5857 (1.82%)
		D	248509	8411 (3.38%)	3580 **(1.44%)**
	KRkp	W	1110806	39490 (3.56%)	13028 **(1.17%)**
		D	398282	16417 (4.12%)	12508 **(3.14%)**
	KQkp	W	945359	4359 **(0.46%)**	7762 (0.82%)
		D	155352	2284 (1.47%)	1461 **(0.94%)**
5	KQBkp	W	1050708	4215 **(0.4%)**	12011 (1.14%)
		D	252429	215 **(0.08%)**	1560 (0.62%)
	KQNkp	W	1148101	7136 **(0.62%)**	11923 (1.04%)
		D	265648	513 **(0.19%)**	2085 (0.78%)
	KQRkp	W	931942	2388 **(0.26%)**	10634 (1.14%)
		D	30495	165 **(0.54%)**	1049 (3.44%)
	KRBkp	W	1274054	11684 **(0.92%)**	16498 (1.29%)
		D	339055	4566 (1.35%)	3924 **(1.16%)**
	KRNkp	W	1081372	16086 **(1.49%)**	19198 (1.78%)
		D	282217	5000 **(1.77%)**	6178 (2.19%)
	KBNkp	W	1208553	52401 (4.34%)	40607 **(3.36%)**
		D	410661	30795 (7.5%)	14596 **(3.55%)**



(a) In winning positions

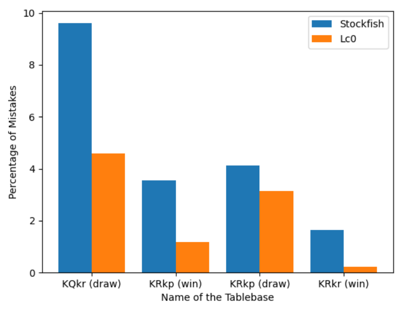


(b) In drawing positions

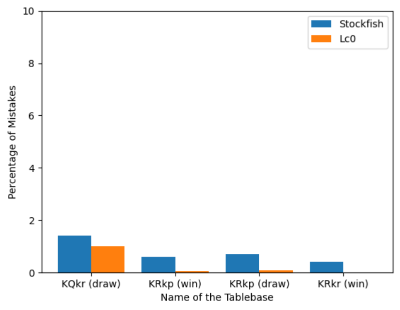
Fig. 1: Performance comparison between the raw policies of Stockfish and Lc0 in the four-piece tablebases

5.2 Improvement Analysis After Incorporating Search

Based on Table 1, we decide to investigate the four-piece tablebases KQkr (draw), KRkp (win), KRkp (draw), and KRkr (win). This selection is motivated by the statistically significant performance difference between the two engines. We search up to 400 nodes with these engines to examine the changes.



(a) Mistakes without search



(b) Mistakes with search

Fig. 2: Performance of Stockfish and Lc0 with and without search (400 Nodes)

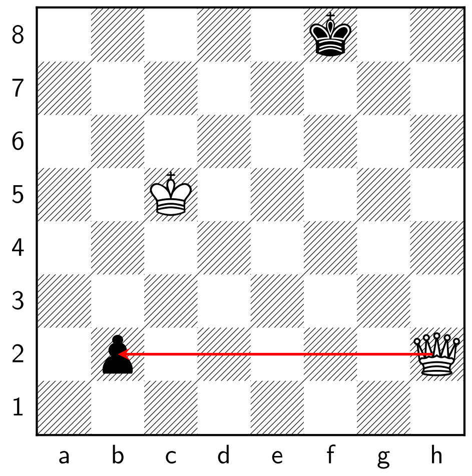
Figure 2 reflects the improvement in the performance of both engines after incorporating search. In the four tablebases used here, Lc0 still performs better than Stockfish even after applying a search of up to 400 nodes. However, incorporating search leads to a significant reduction of mistakes for both engines.

5.3 Engine Behaviour Analysis in Positions with a Single Pawn for the Weaker Side

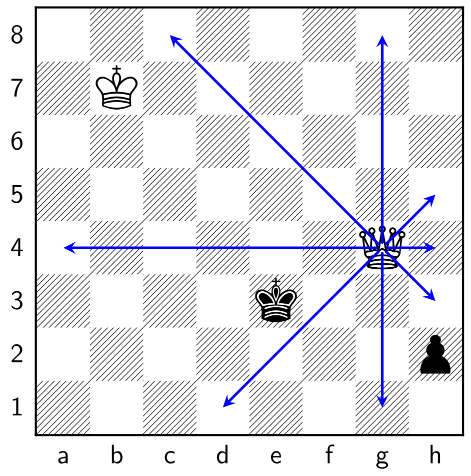
We investigate the behaviour of Stockfish and Lc0 using tablebases where there is only one black pawn and one or more stronger white pieces. We compare the behaviours of the engines based on the fact whether the black pawn is attacked or safe. Figure 3 shows examples of two boards; in one, the pawn is under attack, and in the other, the pawn is on a safe square. We consider only cases where the result is a win in perfect play. We choose to analyze this behaviour because, in these endgames, human chess players usually do not miss the chance to capture this pawn to make the opponent helpless with only the king. We decide to observe how many mistakes occur in this kind of scenario.

Table 2 displays the total number of correct moves, the percentage of attacked pawns in those positions, the total number of mistakes, and the percentage of attacked pawns in those positions for the raw networks of Stockfish and Lc0, respectively. The results show that Lc0 has a lower percentage of mistakes when the black pawn is under attack.

Interestingly, for the tablebases KQBkp, KQNkp and KQRkp, the percentages of mistakes are very high for both engines. In positions where the white can capture the black pawn, such a high rate of mistakes is not expected at all. Specifically, the performance of Stockfish for KQRkp (87.06%) is a cause for concern. Here, Stockfish makes almost 90% of its mistakes when the black pawn is under attack, whereas this percentage is 57.5% for attacked pawns in cases with no mistakes.



(a) Black pawn is attacked



(b) Black pawn is safe

Fig. 3: Example of black attacked and safe pawns (white to play)

Table 2: Percentages of Mistakes for Attacked and Safe Pawns (Win Only).

	Stockfish				**Lc0**			
	No Mistake		**Mistake**		**No Mistake**		**Mistake**	
EGTB	**Total Positions**	**% of Attacked Pawns**	**Total Positions**	**% of Attacked Pawns**	**Total Positions**	**% of Attacked Pawns**	**Total Positions**	**% of Attacked Pawns**
KRkp	1071316	29.2	39490	9.95	1097774	28.82	13032	**3.4**
KQkp	941000	40.5	4359	35.21	937608	40.6	7751	**25.21**
KQBkp	1046493	52.6	4215	48.6	1038698	52.7	12010	**40.84**
KQNkp	1140965	48.5	7136	40.72	1136179	48.5	11922	**37.61**
KQRkp	929554	57.5	2388	87.06	921309	57.7	10633	**50.89**
KRBkp	1262371	41.3	11683	24.4	1257557	41.5	16497	**19.06**
KRNkp	1065286	36.5	16086	16.31	1062174	36.6	19198	**13.27**
KBNkp	1156153	29.07	52401	30.7	1167947	29.5	40607	**18.06**

6 Conclusion and Future Work

Through this work, we aim to contribute to the fascinating world of chess engines by uncovering new insights. The important findings of this work are - (1) The Stockfish policy is strictly better than or equal to the Lc0 policy in 3-piece endgames for predicting a perfect move, (2) The Lc0 policy produces fewer mistakes than the Stockfish policy in most four-piece endgames, (3) Lc0 identifies a weak position better than Stockfish in four-piece endgames, (4) With search, both engines improve their performance by a significant margin, and the difference in their performances becomes narrower, (5) Predicting wins is easier for Stockfish, whereas predicting draws is easier for Lc0, (6) Lc0 makes fewer mistakes than Stockfish when the opponent's last pawn is under attack.

As a future research direction, these experiments can be extended to other endgame tablebases. Increasing the number of pieces and sampling more positions will lead to a deeper understanding of these engines. Finding more interesting patterns and behaviours in the positions misplayed by the two engines would also be a significant extension of this study.

References

1. Lc0 authors: lc0. **https://lczero.org**, accessed: 2023–03–04
2. Lc0 authors: What is lc0? **lczero.org/dev/wiki/what-is-lc0/**, accessed: 2023–03–04
3. Lc0 authors: Win-draw-loss evaluation. **lczero.org/blog/2020/04/wdl-head/**, accessed: 2023–03–24
4. Stockfish authors: Stockfish. **https://stockfishchess.org**, accessed: 2023–03–04
5. Ballicora, M.: Gaviota. **sites.google.com/site/gaviotachessengine/Home/endgame-tablebases-1**, accessed: 2023–02–27

6. Coulom, R.: Efficient selectivity and backup operators in monte-carlo tree search. In: International conference on computers and games. pp. 72–83. Springer (2006)
7. Edwards, D.J., Hart, T.: The alpha-beta heuristic. Tech. Rep. AIM-30, MIT (1961)
8. Forst´en, H.: Purpose of the moves left head. **https://github.com/LeelaChessZero/lc0/pull/961#issuecomment-587112109**, accessed: 2023–03–24
9. Haque, R., Wei, T.H., Müller, M.: On the road to perfection? evaluating leela chess zero against endgame tablebases. In: Advances in Computer Games: 17th International Conference, ACG 2021, Virtual Event, November 23–25, 2021, Revised Selected Papers. pp. 142–152. Springer (2022)
10. Heinz, E.A.: Extended futility pruning. ICGA Journal **21**(2), 75–83 (1998)
11. Kryukov, K.: Number of unique legal positions in chess endgames. **http://kirillkryukov.com/chess/nulp/**, accessed: 2023–02–26 (2014)
12. Lassabe, N., Sanchez, S., Luga, H., Duthen, Y.: Genetically programmed strategies for chess endgame. In: Proceedings of the 8th annual conference on genetic and evolutionary computation. pp. 831–838 (2006)
13. de Man, R., Guo, B.: Syzygy endgame tablebases. **syzygy-tables.info/**, accessed: 2023–03–06
14. de Man, R.: Syzygy. **https://github.com/syzygy1/tb**, accessed: 2023–03–08
15. McGrath, T., Kapishnikov, A., Tomašev, N., Pearce, A., Wattenberg, M., Hassabis, D., Kim, B., Paquet, U., Kramnik, V.: Acquisition of chess knowledge in alphazero. Proceedings of the National Academy of Sciences **119**(47), e2206625119 (2022)
16. Nasu, Y.: Efficiently updatable neural-network-based evaluation functions for computer shogi. The 28th World Computer Shogi Championship Appeal Document **185** (2018)
17. Romein, J.W., Bal, H.E.: Awari is solved. ICGA Journal **25**(3), 162–165 (2002)
18. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. nature **529**(7587), 484–489 (2016)
19. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al.: A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. Science **362**(6419), 1140–1144 (2018)