Chess fortresses, a causal test for state of the art Symbolic[Neuro] architectures

Hedinn Steingrimsson
Electrical and Computer Engineering
Rice University
Houston, TX 77005
hedinn.steingrimsson@rice.edu

***Abstract*—The AI community's growing interest in causality is motivating the need for benchmarks that test the limits of neural network based probabilistic reasoning and state of the art search algorithms. In this paper we present such a benchmark, chess fortresses. To make the benchmarking task as challenging as possible, we compile a test set of positions in which the defender has a unique best move for entering a fortress defense formation. The defense formation can be invariant to adding a certain type of chess piece to the attacking side thereby defying the laws of probabilistic reasoning. The new dataset enables efficient testing of move prediction since every experiment yields a conclusive outcome, which is a significant improvement over traditional methods. We carry out extensive, large scale tests of the convolutional neural networks of Leela Chess Zero [1], an open source chess engine built on the same principles as AlphaZero (AZ), which has passed AZ in chess playing ability according to many rating lists. Our results show that a novel, experimental network, which was intended for efficient endgame play, performed best despite being one of the smallest networks.**

***Index Terms*—Benchmark, causation, intervention, Deep Learning, Convolutional Neural Network, Halting-Problem, chess fortresses, Monte Carlo Tree Search, AlphaZero, Leela Chess Zero, logical reasoning, human cognition, material invariance**

I. Introduction

Despite the success of deep learning based techniques such as AlphaZero (AZ), the AI community is interested in developing methods that can handle tasks that are of causal nature. For example, Judea Pearl, recently argued that "all the impressive achievements of deep learning amount to just curve fitting" [2]–[4]. Yoshua Bengio and Bernhard Schölkopf have also recently emphasized the importance of causal relations [5]. In this paper we pursue this line of research by examining the performance of symbolic[neuro] architectures (notation from [6], an architecture where a neural network runs as a subroutine within symbolic Monte Carlo Tree Search (MCT)), on an original, new benchmark which is of causal nature. The benchmark task is invariant to modifications, that do not change the key causal elements of the task. An example is presented in Figure 1, where the fortress is invariant to adding black white-squared bishops. An arbitrary number of white-squared black bishops can be added to the position without changing the fortress evaluation. The reason is that this type of a piece (which can only move on white squares) does not contribute to battling the key g5 square (which is on a black square), which is sufficiently protected by the white pieces. These characteristics beat the laws of probabilistic reasoning, where extra material typically means increased chances of winning the game. This feature makes the new benchmark especially challenging for neural based architectures.

Our goal in this paper is to provide a new benchmark of logical nature - aimed at being challenging or a hard class - which modern architectures can be measured against. This new dataset is provided in the Supplement [1]. Exploring hard classes has proven to be fruitful ground for fundamental architectural improvements in the past as shown by Olga Russakovsky's influential work on ImageNet's hard classes [7] and more recently ObjectNet [8].

Following a long tradition in AI and more recently AZ [9], [10], we use chess as our experimental domain. For our simulations, we use Leela chess zero (lc0), an open source reverse engineered version of AZ [11], which is based on published information about AZ [12].

In our dataset we focus on fortresses. Fortresses are considered one of the most challenging concepts in chess, also for the latest chess engines [13]. Fortresses are a task of causal intervention. In a fortress scenario the defending side (There can for example be material disadvantage.), builds an impregnable barrier that the attacking side can not successfully breach. Whether a fortress is impregnable or breakable involves the full depth and complexity of chess which makes fortresses ideally suited for testing the limits of modern architectures. We explore what kinds of neural network modifications make partial progress on the fortress task.

AZ is a general purpose architecture that has been applied to a wide range of tasks [14], [15] such as modeling the COVID-19 virus protein structure with AlphaFold [16].

The core AZ architecture [17], with the convolutional neural network having only a policy and a value head, is kept intact in subsequent DeepMind papers [17]. Architectural improvements which are motivated by testing a wide variety of different AZ

[1] https://drive.google.com/drive/folders/1du7P4vgi7X7kL9BTCZ6A2VKENGvzLY-B

978-1-6654-3886-5/21/$31.00 ©2021 IEEE

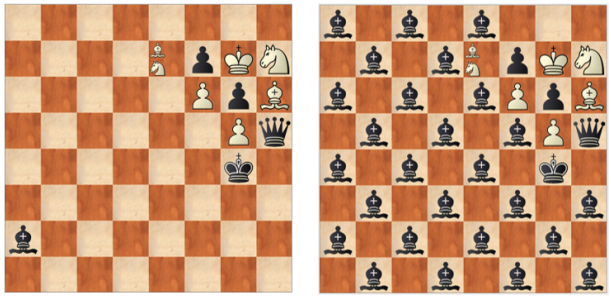

Fig. 1: A fortress, which is invariant to adding extra black white-squared bishops, defies the laws of probabilistic reasoning. There can be either a white knight or bishop on the e7 square. The extra pieces do not contribute to battling the key g5 square.

inspired neural work architectures, could potentially have a wide ranging impact.

In the work presented here, we tested all 66 of the main and experimental lc0 [1] convolutional neural networks which have been developed since the lc0 project started in 2018.

We created a novel fortress chess positions dataset, which has causality and logical intervention characteristics, see Figure 1. It emphasizes the importance of making progress. Each position in the dataset has a *provably unique best move*, which for a defending side leads to a fortress scenario or maintains it. We treat this move as the optimal strategy and measure the success rate of the different networks' preferred choices or policy in terms of matching this ground truth chess move.

Our benchmark approach is different from the traditional approach for testing the performance of deep neural network architectures in chess, which is to let different architectures play full chess matches against each other. This approach is also used in self play. The architecture that wins more matches is considered superior.

One drawback of this approach is that a very high percentage of chess games played between expert players ends in a draw. This means that the large majority of the experiments are inconclusive. When AZ played Stockfish 8 in their first match, the draw rate was 83.9% (out of 1000 games), [10]. The overall score in this match was 57.45% (574.5 points out of possible 1000). A more recent result with a Leela (lc0) network (384x30-t60-3010) playing 100 games against the current version, Stockfish 11, ended with lc0 winning 52.5-47.5 [18], with 71% of the games ending in a draw. (See discussion [19].) Leela is not inferior to AZ in terms of chess strength, see [20] where a modern lc0 achieved a higher score against Stockfish 8, than AZ did. Generally top level chess games, both engine and human, show the same trend towards a draw as is also observed in the last World Championship match with all the 12 games ending in a draw. It can thus be argued that playing full chess games is inefficient in terms of achieving conclusive experimental outcomes.

To avoid this outcome, we focused on comparing the networks' most preferred move choices and position evaluation with ground truth data. By comparing the networks' performance on our carefully designed new dataset, we always get a conclusive result in terms of whether the network finds the correct move or not. Thus, no computational resources are wasted on inconclusive experiments.

In addition to comparing the policy head of the networks with our unique ground truth moves, we also analyze the network's value head, which expresses how feasible the neural network thinks a chess position is after a given move by the policy head (see Figure 2 for the architecture of the network). Given optimal play from both sides, a fortress position will end in a draw. We are interested in exploring how far the evaluation is from this ground truth. Our experimental results indicate that a modified deep learning (DL) network can enhance performance for chess fortresses, while saving computational costs due to a smaller network size.

Our contributions are:

- a new test dataset for entering as a defending side into a fortress chess position class,
- an experimental evaluation of the impact of different modifications to the lc0 neural network using our new dataset and observe that a network with an additional moves left head consistently outperforms the others.
- an efficient approach for evaluating state of the art chess architectures on hard chess position classes.

II. Related work

Previously, it was believed that more data enabled training larger architectures with more expressive power, which enabled them to handle a more diverse set of tasks [21]. The self-play approach mastered by AZ was perhaps the pinnacle of this approach, creating a limitless quantity of data. Currently there is a growing consensus that more data is not enough and that the focus should shift towards fundamental architectural improvements with a special focus on logical reasoning and intervention tasks [2]–[6], [22]–[24].

A. Fortresses

A chess fortress is a local feature F in a given starting position such that (a) the game cannot be won by the attacker without destroying F, and (b) all sequences of play that break F expressly harm the attacker so that the attacker cannot win. In a fortress, invasion roads or critical squares are sufficiently defended, making it impossible to break the barrier with pawn breakthroughs or material sacrifices that eliminate key

defending pieces. In other words, even if the superior side has a material or positional advantage, best play from both sides will still result in a draw (see Figure 1). Fortresses are generally considered to rank among the most challenging phenomena in chess. Nikolaos Ntirlis called fortresses in Appendix 2 Advanced Engine Management [13] "one of the big problems in computer chess analysis."

Fortresses display a hardness feature associated generally with the Halting Problem. If a fortress can be broken, then the fact of breaking it is verifiable, otherwise we may not know whether its appearance of holding is because the algorithm has not searched long enough.

Prior work on fortresses explored the different ways in which the attacking side could attempt to break a potential fortress position. All of these approaches involved a mini-max-based search [25] such as Stockfish and were designed to determine whether a given chess position is an impregnable fortress.

A human chess player will approach fortresses as a logical reasoning and intervention task: first define critical squares and the optimal offensive and defensive formations and only then start a search process to determine whether the critical squares can be penetrated.

This human behavior inspired Eiko Bleicher's Freezer program [26], [27], which focuses on constraining the search space when searching for moves that might break the potential fortress. A human operator defines the results of boundary conditions (pawn moves and pawn or piece exchanges) and the possible defensive piece movements that seem to keep the potential fortress intact. The chess position needs to be close to a tablebases position [28]. Tablebases are chess positions which have been analytically solved using retrograde analysis. Given this constrained search space, Freezer is able to determine whether a chess position is a fortress or not.

In [29] all moves are searched to the same depth and the chess engine position evaluations at each depth are recorded. The focus is on finding moves which evaluation rises with increasing depth. A temporary concession might open up entry points for the attacking pieces. The most recent attempt with the Stockfish variant Crystal [30] penalizes moves that lead to cyclic, repetitive variations.

In the work presented here, unlike prior approaches, we focus on finding the unique correct moves for the defending side for entering a fortress. The idea is that this makes the task harder, since the defending side does not have the luxury of making any mistakes.

B. Extended AlphaZero neural network architecture

Figure 2 shows the extended AlphaZero neural network architecture as presented in [31]. It was developed for the purpose of making lc0's endgame play more effective [31], finishing off games where there is a winning position instead of maneuvering without making progress or so called "trolling" [32], [33]. It was trained with tablebases data, which includes how many moves are left to a checkmate given best play by both sides.

The vanilla AlphaZero architecture does not have the moves left head or "horizon" and has a value head as a direct output. The advantage of splitting the value head into win, draw, loss (W, D, L) is that it provides more information about the position evaluation. For example a value head evaluation of zero could mean that W=0, D=100, L=0 or W=50, D=0, L=50. In the former case the position is a dead draw, while in the latter case the position is highly complicated and dynamic, with both sides having a chance, but a draw is an unlikely result. We refer to [9], [10], [34] for further details on the AZ architecture.

C. Monte Carlo Tree Search

With MCTS different moves are searched to a different depths. For its search component, the AZ architecture and lc0 use the Polynomial Upper Confidence Trees (PUCT) [35] variant of the MCTS. PUCT is a best-first search. This is in contrast to mini-max search, which Stockfish uses, where all plausible moves tend to be searched to a similar depth. The original PUCT formula will assign a weight to each move, and select the move with the highest weight:

$$a_t = PUCT(s,a) = \underset{a}{\operatorname{argmax}} \big(Q(s,a) + U(s,a)\big) \quad (1)$$

In its most basic form, the PUCT formula has two components: Q, the mean action value or average game results across current simulations that took action a in state s, and U a regulating term which makes sure that moves, other than the moves that are thought to be best, also eventually get explored.

U is calculated in its most basic form as:

$$U(s,a) = c_{puct} P(s,a) \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)}$$

where c_{puct} is an adjustable constant, $P(s,a)$ is the prior probability given by the policy output of the neural network, and $N(s,a)$ is the visit count for action a at state s. We sum over all potential actions (b) in the numerator, and the visit count for the action under consideration (a) in the denominator. Therefore the less we have explored this action, the greater U will be. The U component of the PUCT formula thus encourages exploring moves suggested by the policy $P(s,a)$.

A recent development [36], adds another term to the PUCT formula renaming it as $S = Q + U +$ *moves_left_effect*

The *moves_left_effect* comes from an experimental version of lc0 which adds a moves left head to the vanilla AZ architecture. It estimates how many moves are left of the chess game (until game ends).

The *moves_left_effect* is currently defined as $(-1) * (a + b|Q| + cQ^2)(dM)$ where M is the number of moves that the network estimates are left of the game and a, b, c and d are pre-defined constants. With the additional moves left head, the neural network's role as a universal function approximator is changed from $f : position \mapsto (value, bestmoves)$ to $f : position \mapsto (win, draw, loose, bestmoves, \mapsto estimated\_number\_of\_moves\_left)$.

III. Experimental Setup

We ran version, 24.1 of lc0's Monte Carlo Tree Search (MCTS), which is comparable to AZ. For the MCTS, we

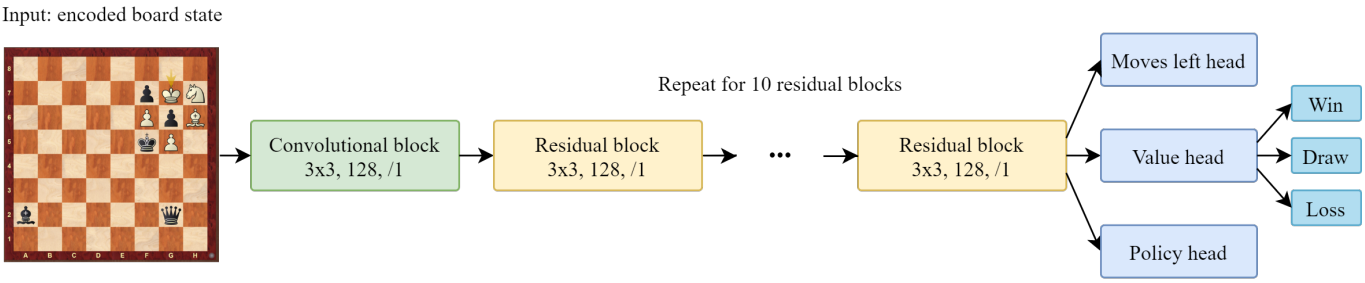

Fig. 2: The AlphaZero neural network architecture with an additional moves left head and a split value head

chose the lc0 default value which is $c_{puct} = 2.15$ [37] (see Supplement [2] for a complete list of the lc0 parameters).

a) Fortresses dataset: To create the fortresses dataset, we searched the largest available chess composition database [38]. We also contacted known chess composition authors and got databases from their personal collection, which focused on the theme "a positional draw". We also got references to books which we searched and composition authors that focused on this theme. We contacted professional chess trainers and got fortress databases from their personal collection, which focus on fortresses that occurred in practical chess matches. We searched some of the largest collections of chess matches such as the Chessbase Mega 2021 [39] and Corr 2020 [40] databases. We used the Chessbase [39] chess database program to filter the resulting databases. This resulted in a database with 150 potential fortress games, each of which can potentially have many fortress positions. We filtered out positions that were in the tablebases [28] domain, since the these positions might have been included in the set of chess positions that the networks were trained on. Finally, we went through each game and position manually in search of chess positions, where there is a unique move for the defending side to enter the fortress. In order to make a network's task of finding the provably unique best move as hard as possible, we selected positions where the defending side must enter a fortress position from afar. In other words, the defending side is not yet in the fortress position, which means that an impregnable barrier has not yet been built, but there is provably a unique way to enter it. We filtered out positions where it was possible to arrive at the fortress scenario or find the correct move by the method of exclusion, meaning that all other moves lead to immediate material loss. We selected only positions which had a unique provably best move because we wanted to have a clear way to measure performance. This rigorous filtering resulted in 18 test positions in our dataset.

Our objective was to gather all the relevant lc0 networks that have been trained since the lc0 project started in the year 2018. We selected all of the strongest networks and all of the experimental versions that tried out different modifications to the vanilla AZ architecture. The best networks included the ones on the lc0 best networks list [41], the final networks from the main lc0 training runs, and the most recent networks from the current runs [42]. We gathered networks from lc0 enthusiasts personal repositories [43]–[45]. We also got networks via the lc0 Discord channel [46]. The network which added a moves left head to the vanilla AZ architecture was found on Discord. It is also available on [31]. We collected 66 networks.

We chose the v0.24.1, of the lc0 MCTS engine [47] CUDA version, which resembles AZ (There have been subsequent improvements which move away from AZ). We chose the default engine parameters. The main variable, apart from the networks, was the number of nodes searched by the lc0 engine. We chose that metric rather than search time as was done in the original AZ paper [9] in order to make our experiments hardware independent. For a list of the 66 networks and the exact lc0 engine parameters see the supplementary material.

Since we kept the MCTS fixed, we subsequently refer to move choices as the network's move choices, although more precisely a move is made by the network and the MCTS.

In the experiments, the networks were faced with each of the 18 fortress positions from our new dataset. As mentioned, each position has a unique provably best move. We registered the five most preferred moves of each of the networks at the different levels of nodes searched and compared it with the ground truth chess move. We also registered the chess position evaluation associated with each of the preferred five moves.

IV. Finding the best architecture

All the selected networks were used together with a MCTS search as discussed in Section II to find the best move for entering a fortress position for each of the positions in our dataset. We varied the number of nodes searched for each position to investigate how the performance of the networks depends on the search effort.

The supplement [3] shows the results of these experiments for all 66 networks. The network 128x10-ccrl-moves-le consistently outperformed the other networks independent of the search effort. This network has an extra moves left head (see Figure 2 for the architecture of the network), tracking how many moves are left in the game, and is also one of the smallest networks tested. It is only 1/27th of the size of the 384x30-t60-3070 network, which is ranked as 8th best. We qualitatively observed,

[2]https://drive.google.com/drive/folders/1du7P4vgi7X7kL9BTCZ6A2VKENGvzLY-B [3]https://drive.google.com/drive/folders/1du7P4vgi7X7kL9BTCZ6A2VKENGvzLY-B

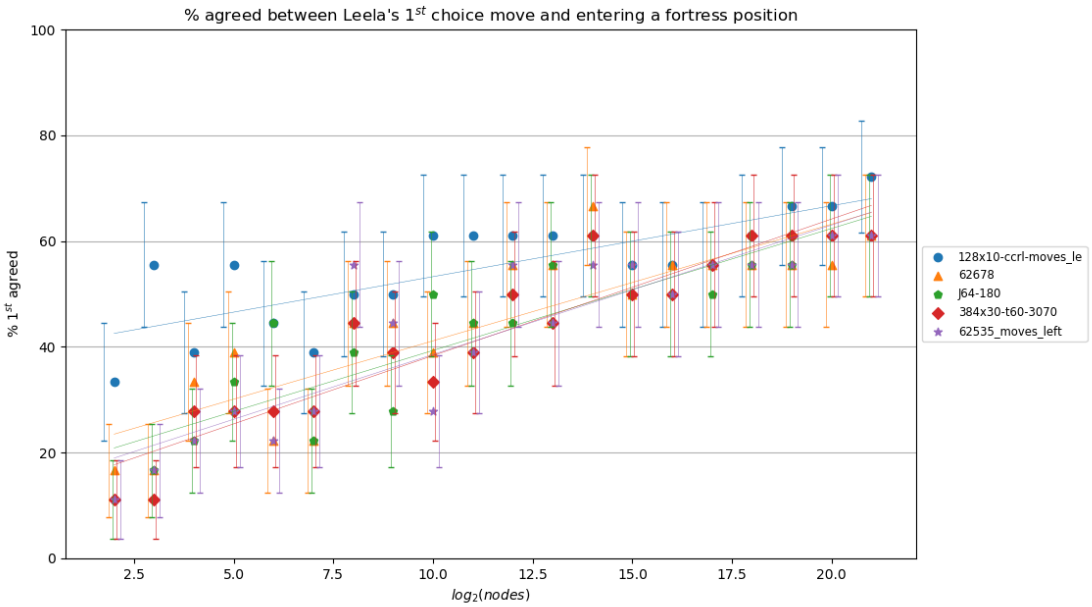

Fig. 3: Entering a fortress with a larger number of maximum nodes searched. The five best networks displayed based on the average of the match between their most preferred move choice and the ground truth moves.

that searching more nodes may or may not lead to an improved performance, depending on the chosen network.

For improved visual clarity of the graphical representation, we selected the best networks based on the performance of their most preferred move choice. We ran the search with this subset of the networks to a higher node count. The results of those experiments are shown in Figure 3. We observe that all five best networks improve when given more time. We also observe that the best performance is achieved by 128x10-ccrl-moves-le, that correctly identified the best move in 72.2% of the positions when given enough time to search 2^{21} nodes per move. This network has the additional moves left head and consistently performs best over the range of nodes searched.

We were interested in discovering which networks would be able to partially master the fortress positions and in how stable the performance would be when we varied the number of nodes searched. Thus, we considered not only the most preferred move of the networks, but the top five preferred moves.

In Figure 4 we see that not only is the most preferred move of the 128x10-ccrl-moves-le network better than the most preferred move of the other networks, its other less favored choices are also at least as good as the ones of the other networks. In terms of accumulated performance, it shows the best performance for all the up to five most preferred moves that we tested (results shown up to the accumulated best four).

In a fortress scenario, the correct winning percentage is 50% or a draw, assuming best play for both sides. MCTS will be slightly off this evaluation, favoring the side which has fewer possibilities for making a mistake. In our case this means that the evaluation would be slightly lower than 50%. We see, however, in Figure 5 that all of the top 5 networks are overly pessimistic about their defensive chances. Thus, there is considerable room for improvement when it comes to showing understanding of the fortress concept with an evaluation that is close to the correct 50%. Surprisingly, we observe that more search depth leads to an even worse evaluation (negative slope of the best fit line), except for the best network, 128x10-ccrl-moves-le.

a) Main empirical results: The moves left network, 128x10-ccrl-moves-le, outperforms the other architectures consistently in all the metrics measured and in every experiment despite it being one of the smallest of the networks tested.

It has a better first move choice and generally makes better move choices as seen by the fact that it maintains its lead when more than one move guess is allowed, for the accumulated percentage for the, 2, 3, 4 and 5 move choices (The results for the 2^{nd}, 3^{rd} and 5^{th} are not shown).

It is the only network whose value head has a positive best fit slope indicating that it benefits from searching more nodes – moving closer to the correct 50% winning percent evaluation. This is in contrast to the vanilla AZ architecture networks, which have a negative best fit line slope indicating that searching more nodes has a detrimental effect on their evaluation of fortress positions.

The value head of all the networks is off by a large margin from the correct evaluation of a draw indicating that despite the policy head of the moves left network performing well, its value head still lags behind in terms of evaluating the fortress scenario correctly. Nevertheless, the moves left network's value head is still closer to the correct value than the other networks,

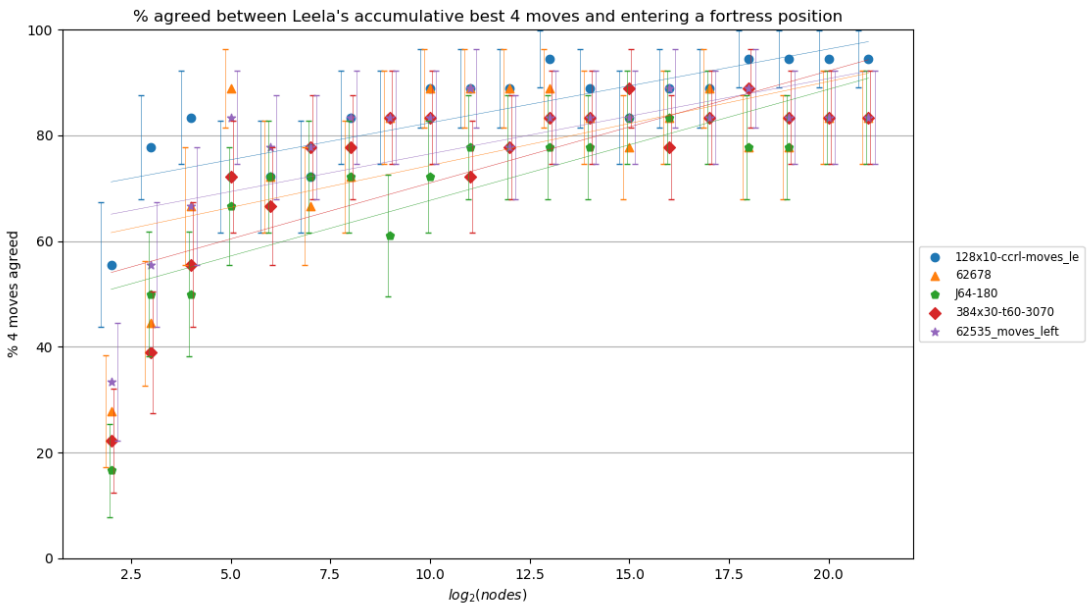

Fig. 4: The five best networks displayed based on the average of the match between their accumulated four best move choices and the provably unique best move. We see that the highest ranked network, the moves left network, outperforms the other networks over a wide range of nodes searched.

especially when more nodes are searched, as seen in Figure 5.

A qualitative follow up experiment – the hard classes for the moves left head network

In order to understand why the moves left head is outperforming other neural networks on the fortress task, we carried out further qualitative tests. We entered the chess positions from the fortress dataset into a graphical user interface and had an expert chess player interpret the moves preference and evaluation of the moves left network in an attempt to understand its approach. We were especially interested in the chess positions in the dataset, that it did not master.

This qualitative analysis indicates that while learning to win games where there is a winning position quickly, the network also has learned to prolong defeat once faced with what it perceives as an inferior position. A hard class for it seems to be positions where two or more moves lead to a prolonged game or a plausible fortress, but only one of them leads to a genuine fortress. Also, the moves left architecture can underestimate transformative moves, such as sacrificial moves breaking the fortress. This is due to the search not giving enough attention to these sacrificial moves. Prior approaches such as Bratko's approach [29] with mini-max engines tried to find transformative moves and give them special attention in the form of a deeper search. The reason for this problem is that sacrificial moves get a low position evaluation score (Q value) at lower search depth, while the other moves that maintain the status quo have a higher Q value. MCTS searches the moves with a higher Q value deeper than others (see $PUCT$ formula 1), and therefore does not pay enough attention to sacrificial moves. This problem is magnified if there are many possible neutral moves as in most/all our fortress positions. Thus, the architecture struggled with finding sacrificial moves that penetrated the defensive formation, but once such a move had been played or this move had been searched deeper, the evaluation started climbing, and it did realize its strength. Detailed qualitative analyses of the moves left hard classes are presented in the supplementary material.

V. Discussion, Conclusions and Future Work

Neuro and symbolic[neuro] architectures have achieved impressive results in a wide range of domains. As described in [2], [5], tasks of logical nature are perhaps their main Achilles heel. Following that reasoning, our paper presents a new benchmark which is of logical nature and explores which extension of the vanilla AZ architecture empirically results in progress being made on the challenging intervention task.

As expected, our new dataset for entering, as the defensive side, into a fortress scenario proved to be challenging. Surprisingly, a new experimental network, 128x10-ccrl-moves-le, designed for efficient endgame play which adds a moves left head to the neural network, tracking how many moves are left in the game, consistently outperformed the other networks despite being one of the smallest networks tested, only 1/27th of the size of the largest ones. The new architecture was designed for the purpose of finishing off superior endgame positions.

A DeepMind paper [48] highlights a challenge of self play emphasizing the importance of exploring the agent state space

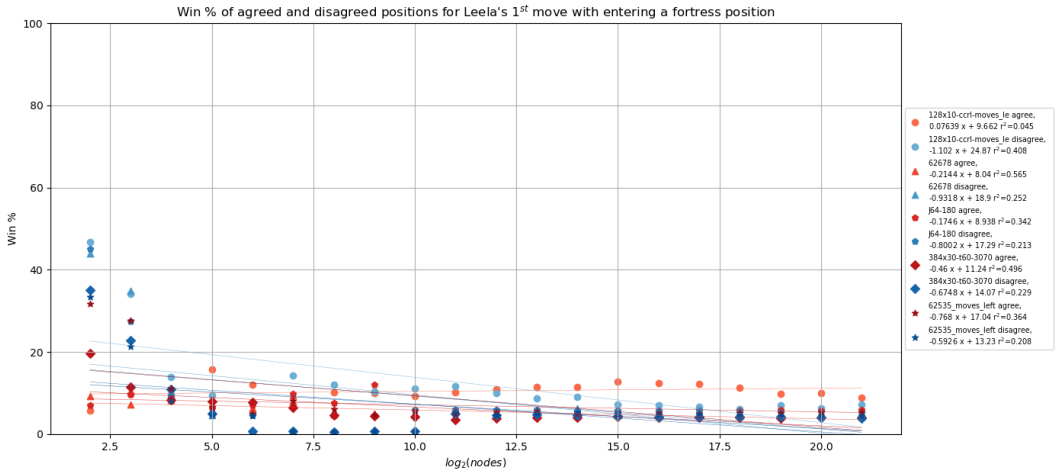

Fig. 5: Winning probability for the networks' most preferred move. Red indicates that the move was the correct one, blue the value for an incorrect move guess. The correct move choices of the 128x10-ccrl-moves-le network are the only ones which evaluation does not decrease with more nodes searched as observed by the slope of the best fit line, although the evaluation is still far below 50%.

and having diverse agents with a heterogeneous skill set rather than focusing on comparing the performance of the self-play generated agents. If none of the agents masters a part of the agent state space or a chess position class, then this position class will be a blind spot. Focusing on hard classes, such as chess fortresses, can help highlight this problem.

A limitation of our work is that despite searching the largest possible chess composition collections [38], the largest databases with chess matches [39] and [40] and contacting well known composition authors and professional chess trainers and getting access to their personal collections, due to our rigorous filtering, our dataset size was only 18 chess positions. It is possible to extrapolate and augment the dataset in various ways, see Figure 1 e.g. where extra pieces are added to the attacking side. The resulting positions will, however, not be completely independent, since they will share the same logical theme. It is of interest to explore ways to gather a larger dataset. Fortresses are generally considered one of the very hardest chess phenomena, which according to [13], even chess professionals working in tandem with the latest engines struggle with. To our knowledge this is the first time that a dataset has been compiled which consists of positions where a defending side has to find unique best moves for entering a fortress.

Another limitation is that we show only one architecture where the moves left extension performs best. We leave it for follow up work to carry out an ablation study (removing the moves left head from the best performer) and adding a new moves left head to other architectures (e.g., the 2-5th best).

When creating the fortress dataset, we excluded chess positions, where modern chess engines could find the unique correct move by the method of exclusion since all the other moves loose chess material immediately. It is possible that the moves left network is also finding the right moves by the method of elimination from its point of view, which is different. Although material does not change, the network's metric to consider how long the game will last, might lead it to preferring moves that prolong the game.

It is possible that ideas described in [49] for proving program termination could be explored including moving away from the search for a single ranking function and toward a search for a set of ranking functions. Using ideas from Bratko [29] and specialized Stockfish implementations such as Crystal [30] a new type of a ranking function could be added to the MCTS algorithm to give more attention to transformative moves. These approaches place emphasis on moves that rise in evaluation with a deeper search, and as is the case in Crystal, penalizing, when having a favorable position, chess variations that lead to repetitive cycles of moves where progress is not made. Another possibility is that the MCTS treats the move choice as either status quo or a transformative move and splits the nodes search between these two types of a choices.

We also may explore whether the new moves left head architecture can play a similar role as the human operator in the Freezer program [26], [27], restricting the search space after which a mini-max engine can finish off the job to decide whether a position is a fortress or not.

In conclusion, our evaluation of a new dataset comprised of hard classes of chess positions indicates that a moves left head can improve the performance of the AZ architecture in chess. Since AZ is a general purpose architecture, this extension to the architecture could have positive impact on a range of similar domains. For example, tasks associated with the Halting Problem may benefit in a way similar to chess fortresses. The tasks can have logical characteristics and invariant properties such as not being sensitive to the addition of a certain type of extra material or chess pieces, which defies the laws of

probabilistic reasoning.

REFERENCES

[1] L. C. Z. open source community, "Leela chess zero," 2020, [Online; accessed 04-June-2020]. [Online]. Available: https://lczero.org/

[2] J. Pearl, ""radical empiricism and machine learning research," causal analysis in theory and practice (blog)," 2020.

[3] ——, ""the limitations of opaque learning machines," possible minds: twenty-five ways of looking at ai," 2019.

[4] J. Pearl and D. Mackenzie, ""ai can't reason why,"," 2018.

[5] B. Schölkopf, F. Locatello, S. Bauer, N. R. Ke, N. Kalchbrenner, A. Goyal, and Y. Bengio, "Towards causal representation learning," 2021.

[6] H. Kautz, "The third ai summer, aaai 2020 robert s. engelmore memorial award lecture," Youtube, 2020. [Online]. Available: https://www.youtube.com/watch?v=_cQITY0SPiw

[7] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[8] A. Barbu, D. Mayo, J. Alverio, W. Luo, C. Wang, D. Gutfreund, J. Tenenbaum, and B. Katz, "Objectnet: A large-scale bias-controlled dataset for pushing the limits of object recognition models," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. dÁlché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 9453–9463.

[9] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *CoRR*, vol. abs/1712.01815, 2017.

[10] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018. [Online]. Available: https://science.sciencemag.org/content/362/6419/1140

[11] D. Tanksley and D. Wunsch, "Reproducibility via crowdsourced reverse engineering: A neural network case study with deepmind's alpha zero," *CoRR*, vol. abs/1909.03032, 2019. [Online]. Available: http://arxiv.org/abs/1909.03032

[12] crem, "Alphazero paper, and lc0 v0.19.1," dec 2018. [Online]. Available: https://lczero.org/blog/2018/12/alphazero-paper-and-lc0-v0191

[13] J. Aagaard and N. Ntirlis, ""grandmaster preparation - thinking inside the box, appendix 2 advanced engine management by nikolaos ntirlis"," 2018. [Online]. Available: http://www.qualitychess.co.uk/products/2/154/grandmaster_preparation_-_thinking_inside_the_box_by_jacob_aagaard/

[14] M. Sadler and N. Regan, *Game Changer: AlphaZero's Groundbreaking Chess Strategies and the Promise of AI*. New in Chess, 2019. [Online]. Available: https://www.newinchess.com/en_US/game-changer

[15] L. Fridman, "David silver: Alphago, alphazero, and deep reinforcement learning — ai podcast #86 with lex fridman," Youtube, apr 2020. [Online]. Available: https://www.youtube.com/watch?v=uPUEq8d73JI&t=2s

[16] A. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. Žídek, A. Nelson, A. Bridgland, H. Penedones, S. Petersen, K. Simonyan, S. Crossan, P. Kohli, D. Jones, D. Silver, K. Kavukcuoglu, and D. Hassabis, "Improved protein structure prediction using potentials from deep learning," *Nature*, vol. 577, pp. 1–5, 01 2020.

[17] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. P. Lillicrap, and D. Silver, "Mastering atari, go, chess and shogi by planning with a learned model," *ArXiv*, vol. abs/1911.08265, 2019.

[18] Chessdom, "Top chess engine championship: Stockfish 20200407dc vs. lczero v0.24-sv-t60-3010," 2020. [Online]. Available: https://www.tcec-chess.com/archive.html?season=17&div=sf&game=1

[19] glbchess64, "Tcec s17 super final report," apr 2020. [Online]. Available: https://lczero.org/blog/2020/04/tcec-s17-super-final-report/

[20] W. Canary-Reed, "Alphazero match will be replicated in computer chess champs," 2020. [Online]. Available: https://www.chess.com/news/view/alphazero-match-will-be-replicated-in-computer-chess-champs

[21] Y. LeCun and G. Marcus, "Debate: "does ai need more innate machinery?"," Youtube, oct 2017. [Online]. Available: https://www.youtube.com/watch?v=vdWPQ6iAkT4&t=227s

[22] Y. Bengio, "From system 1 deep learning to system 2 deep learning," dec 2019. [Online]. Available: https://slideslive.com/38922304

[23] Y. Bengio and G. Marcus, "Debate : The best way forward for ai," MONTREAL.AI, dec 2019. [Online]. Available: https://montrealartificialintelligence.com/aidebate/

[24] B. Liu, ""weak ai" is likely to never become "strong ai", so what is its greatest value for us?" 2021.

[25] Chessprogramming contributors, "Minimax — chessprogramming wiki," 2020, [Online; accessed 4-June-2020]. [Online]. Available: https://www.chessprogramming.org/index.php?title=Minimax&oldid=13281

[26] E. Bleicher, "Freezerchess," 2020, [Online; accessed 04-June-2020]. [Online]. Available: http://www.freezerchess.com/

[27] ——, "Building chess endgame databases for positions with many pieces using a-priori information," 2004. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.217.3356

[28] Chessprogramming contributors, "Endgame tablebases — chessprogramming wiki," 2020, [Online; accessed 4-June-2020]. [Online]. Available: https://www.chessprogramming.org/index.php?title=Endgame_Tablebases&oldid=18896

[29] M. Guid and I. Bratko, "Detecting fortresses in chess," *Elektrotehniški vestnik (English Edition)*, vol. 79, pp. 35–40, 01 2012.

[30] J. Ellis, "Crystal chess engine," 2020, [Online; accessed 4-June-2020]. [Online]. Available: https://github.com/jhellis3/Stockfish/tree/crystal

[31] H. Forstén, "Moves left head pull request - github," mar 2020. [Online]. Available: https://github.com/LeelaChessZero/lc0/pull/961

[32] crem, "Tablebase support and leela weirdness in endgame," oct 2018. [Online]. Available: https://lczero.org/blog/2018/08/tablebase-support-and-leela-weirdness/

[33] Ipmanchess, "Lc0: why play for a mate if i can get a extra 50moves?!" 2020. [Online]. Available: https://github.com/LeelaChessZero/lc0/issues/688

[34] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. P. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 2016.

[35] D. Auger, A. Couëtoux, and O. Teytaud, "Continuous upper confidence trees with polynomial exploration – consistency," in *Machine Learning and Knowledge Discovery in Databases*, H. Blockeel, K. Kersting, S. Nijssen, and F. Železný, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 194–209.

[36] LCZero, "Score," 2020. [Online]. Available: https://github.com/LeelaChessZero/lc0/blob/master/src/mcts/search.cc#L1244

[37] borg, "A layman's guide to configuring lc0," apr 2020. [Online]. Available: https://lczero.org/blog/2020/04/a-laymans-guide-to-configuring-lc0/

[38] H. van der Heijden, ""endgame study database hhdbvi, the world's largest collection of endgame studies"," 2020. [Online]. Available: https://www.hhdbvi.nl/

[39] M. Wuellenweber and M. Feist, ""chessbase database program"," 2021. [Online]. Available: https://shop.chessbase.com/en/products/chessbase_16_premium_package

[40] ——, ""chessbase correspondance games database"," 2021. [Online]. Available: https://shop.chessbase.com/en/products/corr_2020_upgrade

[41] LCZero, "Selecting network to use," apr 2020. [Online]. Available: https://lczero.org/play/bestnets/

[42] ——, "Leela chess zero networks," apr 2020. [Online]. Available: http://training.lczero.org/networks/?show_all=1

[43] J. H. Thomas, "Leela training," mar 2020. [Online]. Available: https://github.com/jhorthos/lczero-training/wiki/Leela-Training

[44] S. V., "Sv nets," 2020. [Online]. Available: https://www.comp.nus.edu.sg/~sergio-v/

[45] dkappe, "Dkappe leela chess zero home page," jul 2019. [Online]. Available: https://github.com/dkappe/leela-chess-weights/wiki

[46] LCZero, "Leela chess zero discord server," 2020. [Online]. Available: https://discordapp.com/invite/pKujYxD

[47] ——, "Leela chess zero download page," 2020. [Online]. Available: https://lczero.org/play/download

[48] D. Balduzzi, M. Garnelo, Y. Bachrach, W. Czarnecki, J. Pérolat, M. Jaderberg, and T. Graepel, "Open-ended learning in symmetric zero-sum games," in *ICML*, 2019.

[49] G. W. Hamilton, "Distilling programs to prove termination," *CoRR*, vol. abs/2008.02936, 2020. [Online]. Available: https://arxiv.org/abs/2008.02936