Mastering Chess with a Transformer Model

Daniel Monroe[®], Philip A. Chalmers[®]

Abstract—Transformer models have demonstrated impressive capabilities when trained at scale, excelling at difficult cognitive tasks requiring complex reasoning and rational decision-making. In this paper, we explore the application of transformers to chess, focusing on the critical role of the position representation within the attention mechanism. We show that transformers endowed with a sufficiently expressive position representation can match existing chess-playing models at a fraction of the computational cost. Our architecture, which we call the Chessformer, significantly outperforms AlphaZero in both playing strength and puzzle solving ability with 8x less computation and matches prior grandmaster-level transformer-based agents in those metrics with 30x less computation. Our models also display an understanding of chess dissimilar and orthogonal to that of top traditional engines, detecting high-level positional features like trapped pieces and fortresses that those engines struggle with. This work demonstrates that domain-specific enhancements can in large part replace the need for model scale, while also highlighting that deep learning can make strides even in areas dominated by search-based methods.

Index Terms—Chess, AI, Transformer, Machine Learning

I. INTRODUCTION

HESS has long been regarded as a proving ground for artificial intelligence (AI). The importance of strategic planning and rational decision-making in strong chess play make the game an ideal domain in which to test systems aiming to master these human-like cognitive capabilities.

Traditional chess engines employ a specialized tree-search algorithm paired with a handcrafted evaluation function. Most modern chess engines follow this scheme but use an efficiently updated neural network (NNUE) as their evaluation function.

The AlphaZero engine [1] introduced a different recipe based on Monte Carlo Tree Search (MCTS) and deep neural networks. This approach was recreated by the open-source Leela Chess Zero (Lc0), which builds on the same principles with several subsequent enhancements. These engines use large networks that predict not only a position evaluation but also a policy vector, a distribution over subsequent moves which is used to guide the search process.

AlphaZero used a convolution-based residual network, which was the state of the art at the time. It was based on AlphaGo, which had prior achieved superhuman performance in Go [2]. However, as the authors of the AlphaZero paper note, convolution-based models may be poorly suited for

(Daniel Monroe and Philip A. Chalmers are co-first authors.) (Corresponding author: Daniel Monroe)

Daniel Monroe is with the Department of Mathematics, University of California at San Diego, La Jolla, CA 92093, USA (e-mail: lc0@danielmonroe.net).

Philip A. Chalmers is with Williams College, Williamstown, MA 01267, USA. (e-mail: pac4@williams.edu).

chess. Long-range interactions feature much more prominently in chess than in Go, and convolutions are poorly equipped to deal with these because of their small receptive fields. Transformer models, on the other hand, are based on a global self-attention operation and resolve the issue of small receptive fields. They can also exhibit impressive abilities when trained at scale, powering large language models such as OpenAI's GPT-4 [3].

We show that the effectiveness of transformers in chess depends heavily on the choice of position representation in the attention mechanism. Among three representations of varying expressivity, we select for our final architecture the scheme of Shaw et al. [4], training models endowed with this technique and other enhancements at scale. We call the resulting architecture the Chessformer.

Our main contributions are:

- We present a simple yet performant architecture for transformers in chess and apply this architecture at scale.
- We ablate the position representation in the attention mechanism of our models to demonstrate the criticality of a sufficiently expressive representation.
- We provide a detailed comparison of our models against prior work to demonstrate the superiority of our methods.
- We analyze the attention maps and playstyle of our largest model, CF-240M.
- We open-source our training code.¹

This paper is organized as follows. Section II goes over the vanilla self-attention formulation and compares three position representations of varying expressivity. Section III describes our training setup. Section IV compares our models to prior work and reports results for ablating the position representation. Section V analyzes the attention maps of our largest model, CF-240M, while Section VI describes its playstyle. Section VII reviews related work, and Section VIII provides concluding remarks.

II. BACKGROUND

A. Self-Attention

Given a sequence of tokens $(\mathbf{x}_1, \dots \mathbf{x}_n)$ where $\mathbf{x}_i \in \mathbb{R}^d$, self-attention returns a sequence $(\mathbf{z}_1, \dots \mathbf{z}_n)$ where $\mathbf{z}_i \in \mathbb{R}^d$. Using projection matrices W^Q , W^V , and W^K in $\mathbb{R}^{d \times d}$, a logit e_{ij} for each pair of tokens (i,j) is computed through scaled dot-product attention:

$$e_{ij} = \frac{(\mathbf{x}_i W^Q)(\mathbf{x}_j W^K)^T}{\sqrt{d}} \tag{1}$$

Attention weights are obtained via softmax:

¹Available on Github at https://github.com/Ergodice/lczero-training.

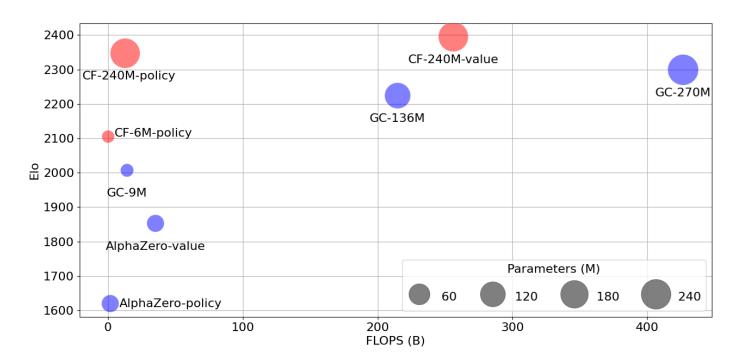


Fig. 1. Elo strength by floating point operations per evaluation (FLOPS) of agents constructed from our CF-6M and CF-240M models (red) against prior art (blue). Our evaluation methodology is described in Section IV.

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{n} \exp(e_{ik})}$$
 (2)

Finally, the output is computed as a weighted sum of value information:

$$\mathbf{z}_i = \sum_{j=1}^n \alpha_{ij}(\mathbf{x}_j W^V) \tag{3}$$

This process is repeated for each head in the self-attention layer, and the outputs are concatenated before being passed through a final linear projection.

B. Position Representations

Because self-attention is permutation-invariant, positional information must be introduced to the model through some kind of position representation. Many position representation techniques, like rotary position embeddings [5], are designed to decay the attention weights with the Euclidean distance between tokens.

While Euclidean distance is a useful inductive bias for language and vision applications, it fails to capture the topology of the chessboard. A square with a bishop, for example, might need to attend to squares on the same diagonal rather than squares that are nearby. To demonstrate the need for more expressivity in the position representation, we compare three options: absolution position embeddings, relative biases, and the scheme of Shaw et al. [4]. Changes to the vanilla self-attention formulation described in Section II-A are marked in blue.

The simplest form of position representation, the absolute position embedding, adds a possibly learnable bias to

each token prior to the attention layer. Calling these biases $(\mathbf{c}_1, \dots \mathbf{c}_n)$, the absolute position embedding adds these biases to the \mathbf{x}_i prior to the attention calculation:

$$\mathbf{x}_i \mapsto \mathbf{x}_i + \mathbf{c}_i$$
 (4)

Unlike absolute position embeddings, relative position representations model the positions of tokens relative to each other. One simple variant introduces relative biases d_{ij} which are added to the attention logits:

$$e_{ij} = \frac{(\mathbf{x}_i W^Q)(\mathbf{x}_j W^K)^T}{\sqrt{d}} + d_{ij}$$
 (5)

For one-dimensional inputs, these bias terms are shared among pairs of tokens (i,j) and (k,l) for which i-j=k-l. We adopt the two-dimensional analog of this technique, where two pairs of tokens share a relative bias if their horizontal and vertical displacements are the same.

A more general position representation introduced by Shaw et al. [4], which we adopt in our final architecture, models the positional relationship between tokens \mathbf{x}_i and \mathbf{x}_j by introducing learnable vectors a_{ij}^Q , a_{ij}^K , and a_{ij}^V in \mathbb{R}^d . The calculation for attention logits is altered to:

$$e_{ij} = \frac{(\mathbf{x}_i W^Q + a_{ij}^Q)(\mathbf{x}_j W^K + a_{ij}^K)^T}{\sqrt{d}}$$
(6)

The output, meanwhile, is updated to propagate signal from a_{ij}^V in addition to the output of the value projection:

$$\mathbf{z}_i = \sum_{j=1}^n \alpha_{ij} (\mathbf{x}_j W^V + a_{ij}^V) \tag{7}$$

While this technique is computationally expensive at large token counts, our models have a context length of 64, making the additional cost negligible.

III. EXPERIMENTAL SETUP

Our training data consists of self-play games generated with the AlphaZero process [1]. However, unlike that work, which worked in the reinforcement learning setting, we create a static dataset of self-play games from an older reinforcement learning run. That run used a transformer model of roughly 100 million parameters at 600 nodes per move which was almost identical in architecture to our main runs, barring its use of a slightly weaker position representation than that of our final architecture. We therefore work in the supervised setting without the need for online data generation, allowing for much faster training. Initial experiments showed no quality degradation on a static dataset.

All models we train employ a standard encoder-only backbone with a context of 64 tokens corresponding to the squares on the chessboard, with one of the position representations described in Section II. The training targets consist of the main policy and value targets in addition to a mix of auxiliary targets, and we use the Nadam optimzer with $\beta_1=0.9$, $\beta=0.98$, $\epsilon=10^{-7}$, and gradient clipping 10. The final model checkpoints used for evaluation are generated with stochastic weight averaging [6]. A more detailed description of the architecture and losses is provided in the Appendix.

Our largest model, CF-240M, has 15 encoder layers with an embedding depth of 1024, a head count of 32, and a feedforward depth of 4096, for a total of 243 million parameters. It was trained for 3.7 million steps with a batch size of 4096 on 8 A100 GPUs using data parallelism. The learning rate was initialized at $1*10^{-3}$ and manually reduced to $3*10^{-4}$ at 3.2 million steps and $1*10^{-4}$ at 3.6 million steps. The dataset consisted of 500 million games generated from mid-2023 to mid-2024, with each game containing roughly 200 positions.

We also train a smaller model, CF-6M, with 8 encoder layers, an embedding depth of 256, a head count of 8, and a feedforward depth of 256, for a total of 6 million parameters. The same configuration is used for ablations of the position representation. CF-6M and its ablations were each trained on a single A100 GPU with a batch size of 2048. The learning rate was initialized at $5*10^{-4}$ and reduced to $1.58*10^{-4}$ and $5*10^{-5}$ at 1.6 and 1.8 million steps, respectively. The dataset consisted of 53 million games generated in April 2024. None of our training runs exhibited overfitting, which is likely due to the size of the datasets relative to the parameter counts of the models.

IV. RESULTS

We compare the playing strength and puzzle-solving ability of agents constructed from our models to prior work and ablate the position representation in the attention mechanism to demonstrate the criticality of accurately modeling positional information.

We include in our analysis two types of agents: those that function based on policy information by picking the move which is ranked highest in the policy vector predicted by the model, and those that function based on value information, emulating a search of depth 1 by evaluating the model for each legal move and selecting the move that maximizes the position evaluation. The policy strategy requires a single model evaluation, while the value maximization strategy requires an evaluation for each legal move. To estimate the floating point operations per evaluation (FLOPS) used by an agent of the value maximization type, we multiply the model FLOPS by 20, which is a rough estimate of the average number of legal moves available in a position.

We construct agents from our models with both strategies, denoting an agent by its model name followed by the strategy it uses (e.g., CF-240M-policy and CF-240M-value). Policy and value agents are constructed from the AlphaZero model in the same way. Our analysis also includes models from Ruoss et al. [7] which use the value maximization approach. We use the final checkpoints of their main runs having parameter counts of 9 million, 136 million, and 270 million, referring to these as GC-9M, GC-136M, and GC-270M, respectively.

A. Playing Strength

To estimate the playing strength of our agents, we play 1,000 games with them against the GC-270M agent of Ruoss et al. [7], tying the Elo rating of GC-270M to the value reported in that paper. We use the same opening book and testing configuration as that work to achieve a similar comparison to theirs. Elo values for the GC and AlphaZero agents are taken from that paper. The results are shown in Table II and Fig. 1.

B. Puzzles

We also evaluate the puzzle-solving ability of our agents on the puzzle database curated by Ruoss et al. [7]. Our evaluation follows the same methodology as that paper, with an agent deemed to solve a puzzle only if it correctly picks each move in the sequence of the solution. The results are shown in Table II. Fig. 2 graphs the accuracy of our two CF-240M agents on this test set against the GC-136M and GC-270M agents of Ruoss et al. [7].

C. Ablations

We ablate the position representation of the CF-6M model, comparing the final test metrics of the three position representations described in Section II on a held-out test set of around 1 million games sharded from the training dataset. The reported statistics are for the main policy and value targets. As shown in Table I, our chosen technique, that of Shaw et al. [4], substantially outperforms both the relative bias and absolute position embedding representations. At this parameter count, a doubling in model size yields an increase in policy accuracy of around 1.5%. The Shaw encoding outperforms the learned absolute position embedding by 1.83% policy accuracy, suggesting that a good choice of position representation can in large part replace the need for model scale.

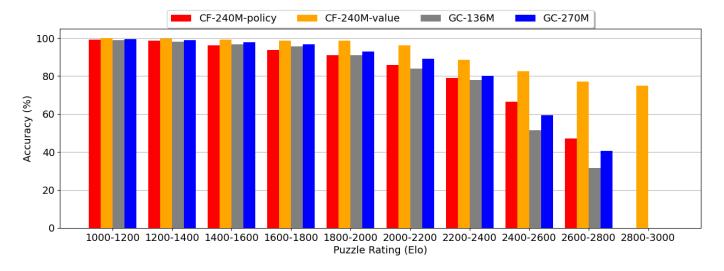


Fig. 2. Puzzle-solving ability on puzzles rated 1000-3000 of our CF-240M-policy and CF-240M-value agents against the GC-136M and GC-270M agents of Ruoss et al. [7].

TABLE I
RESULTS FOR ABLATING POSITION REPRESENTATION

	Loss		Accuracy (%)	
Representation	Policy	Value	Policy	Value
Absolute	0.3460	0.5607	57.44	89.11
Relative bias	0.3321	0.5586	58.23	89.26
Shaw et al. [4]	0.313	0.5549	59.27	89.53

D. Analysis

The main results can be found in Table II. Our agents consistently outperform prior work in both puzzle-solving ability and Elo performance at a fraction of the computational cost. Our CF-6M-policy agent outperforms the AlphaZero-policy agent in both metrics at 8x fewer FLOPS. At 30x fewer FLOPS, our CF-240M-policy agent matches the puzzle performance and exceeds the Elo performance of the grandmaster-level agent GC-270M of Ruoss et al. [7].

TABLE II
COMPARISON OF PLAYING STRENGTH AND PUZZLE SOLVING ABILITY

Agent	Elo	Puzzles (%)	FLOPS
CF-6M-policy (ours)	2105 (±28)	65.3	214M
CF-240M-policy (ours)	2347 (±10)	93.5	12.8B
CF-240M-value (ours)	2385 (±10)	97.6	256B
GC-9M	2007 (±15)	85.5	14.2B
GC-136M	2224 (±14)	92.1	215B
GC-270M	2299 (±14)	93.5	427B
AlphaZero-policy	1620 (±22)	61.0	1.77B
AlphaZero-value	1853 (±16)	82.1	35.3B

V. ATTENTION MAPS

One advantage of transformers is the interpretability of their attention maps. We generate visualizations² of the attention maps of our CF-240M model, which we plot in Fig. 3 and Fig. 4. We select a querying square and color squares based on their attention weights for the querying square. Small weights are colored purple and large weights are colored yellow.

The majority of attention heads in our models represent the movement of a particular piece, attending to squares which are a bishop's, knight's, rook's, or king's move away, depending on the specialization of the head. We give four examples from CF-240M in Fig. 3. From left to right then top to bottom, the heads shown are layer 5 head 24, layer 1 head 28, layer 1 head 14, and layer 15 head 22. The attention maps are fairly static across positions, always retaining their specialization even if the piece whose movement type they specialize in is no longer on the board.

Interspersed with these piece movement heads are several other heads with recognizable patterns. We give four examples in Fig. 4, describing them from left to right then top to bottom. Head 3 in layer 1 always attends to the opponent's queen. Head 29 in layer 2 attends to squares which are of the same color on the checkerboard as the querying square. Head 14 in layer 15 attends to squares occupied by the opponent's pieces with roughly equal weights. Finally, head 2 in layer 11 attends to pieces that can move to the querying square. This final type of head appears across model scales, though always in the second half of the encoder layers, suggesting that they rely on information about piece movement accumulated in early layers.

The variety of these miscellaneous heads suggests that the model develops a broad view of chess principles. The same-checkerboard-color head, for example, may provide information about the maneuverability of bishops. The head attending to the opponent's pieces may be estimating the

 $^{^2\}mathrm{Code}$ for generating attention visualizations is available at https://github.com/Ergodice/lc0-attention-visualizer.

material difference between the players or gleaning the game stage, i.e., opening, middlegame, or endgame, from the types of pieces on the board.

Unlike in vision and language applications, proximity of tokens seems to have little effect on attention weights, confirming the importance of a position representation which can adapt to the topology of the chessboard. Another notable feature of the attention maps is that tokens rarely attend to themselves. This is in contrast to language models, where it is quite common [8].

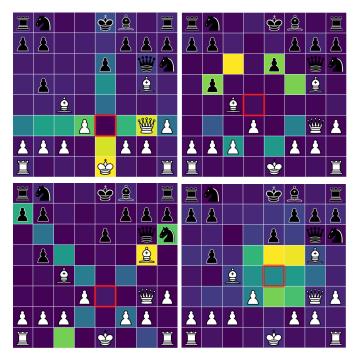


Fig. 3. Attention maps of heads corresponding to the movement of a particular piece. The square highlighted in red is the one producing the query.

VI. PLAYSTYLE

Our largest model, CF-240M, appears to have an understanding of chess which is more akin to humans than that of conventional engines. In particular, it is capable of detecting difficult positional motifs that top chess engines struggle with. One position demonstrating this humanlike game understanding arose between Lc0 and Stockfish in Game 189 of the 23rd Computer Chess Championship Rapid Tournament,³ where the former used a transformer with 190 million parameters trained by the authors. Lc0 managed to construct a fortress from a lost position, leading the game into a draw. A position from this game is shown as Position 1 in Fig. 5. CF-240M assigns the only drawing move, Kg7, a policy of 92.8%, and evaluates the position as drawn (2.2%W, 76.3%D, 21.5%L). Stockfish assigned the same position a +7 evaluation after searching 11 billion nodes, and its evaluation stayed above +3 for over 100 moves. Stockfish evaluations not directly interpretable as win probabilities, but a +3 evaluation corresponds roughly to an over 99% probability of winning.

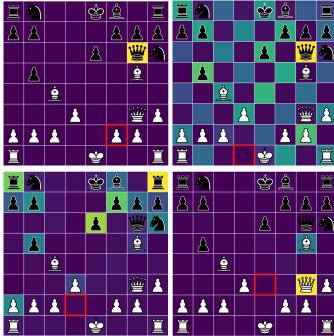


Fig. 4. Attention maps of several additional heads with easily interpretable patterns. The square highlighted in red is the one producing the query.

Another position demonstrating this humanlike positional understanding arose between Lc0 and Stockfish in Game 163 of the 21st Computer Chess Championship Rapid Tournament,⁴ where Lc0 used a transformer of around 100 million parameters trained by the authors. Shown as Position 2 in Fig. 5 is a critical moment in the game when Lc0 played b4, trapping the white rook on b3. CF-240M assigns this move the highest policy at 37% and evaluates the position as slightly winning (40.3%W, 54.1%D, 5.6%L). Lc0 ended up winning the game, with Stockfish's evaluation only going negative several moves later despite searching hundreds of millions of positions.

If the pawn on b2 is removed and this trapping idea is no longer feasible, the evaluation becomes much more drawish (17.1%W, 77.7%D, 5.2%L), despite black gaining a material advantage. Additionally, the model loses interest in b4, with the policy for that move decreasing to 4.76% and Rc7 becoming the favored move.

The positional understanding of our CF-240M model seems to extend to long-term planning as well. As shown in Position 3 in Fig. 5, CF-240M picks the right move at each of white's turns in a famous king walk played by GM Nigel Short as white against GM Jan Timman. The line continues 31. Kh2 Rc8 32. Kg3 Rce8 33. Kf4 Bc8 34. Kg5.

VII. RELATED WORK

Independent work in the same vein as this paper by Ruoss et al. [7] trains transformers at scale for chess, demonstrating that a transformer-based approximation of a strong oracle can achieve grandmaster-level strength without search. Unlike this

³See https://www.chess.com/computer-chess-championship#event=ccc23-rapid-finals&game=189.

⁴See https://www.chess.com/computer-chess-championship#event=ccc21-rapid-semifinals&game=163.



Position 1: fortress detection. CF-240M finds the only drawing move, Kg7, and evaluates the position as drawn.



Position 2: trapped piece detection. If white has a pawn on b2, CF-240M opts to trap the white rook on b3 by pushing b4. However, if the pawn on b2 is not present, it abandons this idea.



Position 3: long-term planning. CF-240M finds a famous king walk that was played out by Nigel Short.

Fig. 5. Positions in which our models exhibit a humanlike understanding of the game, detecting positional ideas that elude top minimax-based engines.

paper, they focus on distilling a search-based algorithm into a transformer model rather than optimizing for playing strength.

Czech et al. [9] design a lightweight transformer block for chess and show that the performance of neural chess models can be improved with carefully chosen input representations and value targets. Similar work in Go [10], [11] adds auxiliary training targets and enhanced input representations to accelerate learning in the AlphaZero process.

Other work interprets the inner workings of neural chess models through ablations, probing of internal states, and analysis of attention maps. McGrath et al. [12] analyze the acquisition of chess knowledge by AlphaZero during training,

while Jenner et al. [13] examine the role of attention maps and lookahead in an older iteration of transformer models trained by the authors.

VIII. CONCLUSION

Though the vanilla transformer is an effective generalist architecture, as we have shown, there can be substantial benefit to exploring domain-specific enhancements and inductive biases like effective position representations. This holds true in resource-limited scenarios but especially in search-based applications like computer chess where speed and accuracy are interchangeable. This idea is exemplified by work such as AlphaFold [14], which predicts the structure of proteins by modeling them with a graph where edges correspond to relationships between residues.

The fungibility of quantity and quality has featured prominently in recent work aiming to improve the reasoning capabilities of language models by encouraging them to delineate their thought process before giving a final answer. For example, chain-of-thought prompting [15] prompts a language model with example reasoning paths, increasing accuracy at the expense of additional computational cost. OpenAI's ol model [16] takes this idea further and is trained to perform a large reasoning step before giving its final answer. In the domain of automated theorem-proving, AlphaProof [17] solves olympiad-level math problems by searching through potential proofs with a model specialized for theorem proving.

We hope our work, which demonstrates the potential usefulness of domain-specific enhancements and shows that in some cases deep learning can succeed where search fails, can provide lessons for this new search-based reasoning paradigm.

APPENDIX A MODEL ARCHITECTURE

A. Input Encoding

The input to our network is a sequence of 64 tokens: one token for each square on the board read from left to right then bottom to top. The board is flipped with the side to move. Each input token has length 112 and consists of a concatenation of:

- 8 one-hot vectors of length 12 describing the piece at that square for the current position and past 7 positions.
- En passant and castling information.
- The number of positions since the last capture, pawn move, or castle, divided by 100.
- Whether each of the current and past 7 positions is a repetition.

To generate token embeddings, we apply a linear projection to the input tokens, then add and multiply by learned offset vectors which are separate across tokens and depth. This gives the model absolute positional information and was found in initial experiments to incrementally improve model quality.

B. Body

The body of our models consists of a stack of encoder layers with Post-LN normalization and the initialization/gain scheme DeepNorm [18]. We use a fixed context length of 64. We

use Mish [19] activations in the feedforward sublayer, and all projections in the output heads are followed by this activation unless otherwise stated.

Following [20], we omit biases in the QKV projections and omit centering and biases in the encoder normalization layers, finding this to increase training throughput by around 10% without degrading model quality. However, we retain biases in the feedforward and post-attention projections since removing them degraded quality in initial experiments.

C. Output Heads

Similar to AlphaZero, our models have heads for policy and value predictions. However, we add several auxiliary policy and value targets to increase convergence speed, following research in Go [10], [11].

Policy Heads We introduce a new policy head based on a modified self-attention mechanism. Moves are encoded by the starting square and destination square of the piece moved. Given the sequence of tokens outputted by the body, we generate policy embeddings by applying a dense layer of depth equal to the model's embedding size. From this we generate, via linear projection, a set of query vectors corresponding to the starting square and a set of key vectors corresponding to the destination square, both with depth equal to the depth of the encoder body.

Logits for moves are calculated via scaled dot product as described in Eq. (1). The result is a 64x64 matrix representing all possible traversals from one square on the chessboard to another. These traversals are sufficient to represent all moves that can occur within the rules of chess, with the exception of promotions. When a pawn advances to the last rank of the board, it must be promoted to a knight, bishop, rook, or queen. To represent these special moves, we apply a linear projection to the key vectors representing the promotion rank, generating an additive bias for each possible promotion piece. This bias is then applied to the logits representing all possible traversals between the penultimate rank and the promotion rank to generate additional logits for each possible promotion.

To generate the final policy vector, we apply a softmax over all logits, masking illegal moves to increase training stability. We did not see performance gain from using any of the position representations described in Section II, likely because the embedding size is much larger than the token count.

Value Heads To model value information, we apply a linear projection of depth $d_{\rm value}$ to the output of the body, where $d_{\rm value}=32$ by default. We then flatten the result and apply another projection of size 128, which we call the value embedding. Each of our models has three value heads, each of which generates its own value embedding separately from the body output. Each of these value heads uses this embedding to predict one or more training targets.

The "result" value head predicts the result of the game from among win, draw, and loss with cross-entropy loss. The "q" and "short-term" value heads each predict three targets: a reward trained with L2 loss, a categorical distribution over rewards trained with cross-entropy loss, and the error between

the predicted and true reward. Respectively, these latter two heads use the reward estimate produced during self-play and an exponential moving average of future rewards with expected depth 6. The gradient of the reward prediction is detached when calculating the error loss to prevent the error head from affecting the reward prediction.

APPENDIX B TRAIN LOSSES

As noted above, our models train on several auxiliary targets, which were found to accelerate model convergence and improve final performance in Go [10], [11]. Here we describe the losses for the main and auxiliary targets. The final loss function is a weighted sum of these terms.

Policy Targets

 Vanilla policy head: predicts the policy target produced by self-play.

$$-c_{\text{pol}} \sum_{m \in \text{moves}} \pi(m) \log(\hat{\pi}(m))$$

where π is the policy target, $\hat{\pi}$ is the model's prediction of π , and $c_{\text{pol}} = 1$.

 Soft policy head: predicts a high-temperature version of the policy target.

$$-c_{\text{softpol}} \sum_{m \in \text{moves}} \pi_{\text{soft}}(m) \log(\hat{\pi}_{\text{soft}}(m)),$$

where $\pi_{\rm soft}$ is taken from π by setting the temperature to 4 and $\hat{\pi}_{\rm soft}$ is the model's prediction of $\pi_{\rm soft}$. We set $c_{\rm softpol}=8$, though this term is still small compared to the main policy loss because the loss is muted by the high temperature.

Value Targets

• Game result: predicts the outcome of the game.

$$c_{\text{value-wdl}} \sum_{r \in \{\text{win, loss, draw}\}} v(r) \log(\hat{v}(r))$$

where v is the game result, \hat{v} is the model's prediction of v, and $c_{\text{value-wdl}} = 1$.

• L2 value: predicts a scalar reward produced by self-play.

$$c_{\text{value-l2}}(q - \hat{q})^2$$

where q is the true reward predicted by \hat{q} and $c_{\text{value-I2}}=1$.

 Categorical value: predicts a categorical distribution over reward estimates.

$$c_{\text{value-cat}} \sum_{x \in \text{score buckets}} z(x) \log(\hat{z}(x))$$

where z is a one-hot vector representing the bucket into which the recorded reward q falls into. We set $c_{\text{value-cat}} = 0.1$ for CF-240M and excluded this target for all other training runs.

• Value error: predicts the error of the L2 value prediction.

$$c_{\text{value-error}}(\hat{e} - (q - \hat{q})^2)^2$$

where $c_{\text{value-error}} = 1$ and \hat{e} is the model's prediction of the squared difference between the predicted and true reward.

APPENDIX C OTHER TECHNIQUES TRIED

In addition to the techniques reported here, we tried several architectural modifications that did not bear fruit. For example, replacing the feed-forward layer with a Gated Linear Unit [21] slightly degraded quality at constant model FLOPS. Using a sparsely gated [22] or soft [23] mixture of experts in place of the feed-forward layer also did not improve performance.

ACKNOWLEDGMENT

This paper relied on extensive contributions from the Lc0 community. We are grateful to Twitch user KittenKaboodle for generously providing the hardware on which experiments were run, and to Github user jkormu for implementing the attention visualization. We are also thankful to David Elliott, Hunter Monroe, Denys Ivanov, and Evan Engler for proofreading and providing insight. Any errors that remain are our own.

Training code and templates for Fig. 1, Fig. 2, and Fig. 5 were generated with help from Github Copilot, and much of the text was checked for spelling and writing quality with GPT-4.

REFERENCES

- [1] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.
- [2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, pp. 254–259, 2017. [Online]. Available: https://doi.org/10.1038/nature24270
- [3] OpenAI, "GPT-4 technical report," arXiv:2303.08774, 2023.
- [4] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-attention with relative position representations," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, M. Walker, H. Ji, and A. Stent, Eds. New Orleans, Louisiana: Association for Computational Linguistics, Jun. 2018, pp. 464–468. [Online]. Available: https://aclanthology.org/N18-2074
- [5] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu, "Roformer: Enhanced transformer with rotary position embedding," *Neurocomput.*, vol. 568, no. C, mar 2024. [Online]. Available: https://doi.org/10.1016/ j.neucom.2023.127063
- [6] P. Izmailov, D. Podoprikhin, T. Garipov, D. Vetrov, and A. Wilson, "Averaging weights leads to wider optima and better generalization," in 34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018, ser. 34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018, R. Silva, A. Globerson, and A. Globerson, Eds. Association For Uncertainty in Artificial Intelligence (AUAI), 2018, pp. 876–885.
- [7] A. Ruoss, G. Delétang, S. Medapati, J. Grau-Moya, L. K. Wenliang, E. Catt, J. Reid, and T. Genewein, "Grandmaster-level chess without search," 2024. [Online]. Available: https://arxiv.org/abs/2402.04494
- [8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in NIPS, 2017.
- [9] J. Czech, J. Blüml, K. Kersting, and H. Steingrimsson, "Representation matters for mastering chess: Improved feature representation in alphazero outperforms switching to transformers," 2024. [Online]. Available: https://arxiv.org/abs/2304.14918
- [10] D. J. Wu, "Accelerating self-play learning in go," CoRR, vol. abs/1902.10565, 2019. [Online]. Available: http://arxiv.org/abs/1902. 10565
- [11] —, "Other methods implemented in katago," 2024.
 [Online]. Available: https://github.com/lightvector/KataGo/blob/master/docs/KataGoMethods.md
- [12] T. McGrath, A. Kapishnikov, N. Tomašev, A. Pearce, M. Wattenberg, D. Hassabis, B. Kim, U. Paquet, and V. Kramnik, "Acquisition of chess knowledge in alphazero," *Proceedings of the National Academy of Sciences*, vol. 119, no. 47, p. e2206625119, 2022. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.2206625119

- [13] E. Jenner, S. Kapur, V. Georgiev, C. Allen, S. Emmons, and S. Russell, "Evidence of learned look-ahead in a chess-playing neural network," 2024. [Online]. Available: https://arxiv.org/abs/2406.00877
- [14] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko *et al.*, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.
- [15] J. Wei, X. Wang, D. Schuurmans, M. Bosma, b. ichter, F. Xia, E. Chi, Q. V. Le, and D. Zhou, "Chain-of-thought prompting elicits reasoning in large language models," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 24 824–24 837. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/9d5609613524ecf4f15af0f7b31abca4-Paper-Conference.pdf
- [16] OpenAI, "Introducing openai o1-preview," 12 September 2024. [Online]. Available: https://openai.com/index/introducing-openai-o1-preview/
- [17] DeepMind, "Ai achieves silver-medal standard solving international mathematical olympiad problems," 25 July 2024. [Online]. Available: https://deepmind.google/discover/blog/ ai-solves-imo-problems-at-silver-medal-level/
- [18] H. Wang, S. Ma, L. Dong, S. Huang, D. Zhang, and F. Wei, "Deepnet: Scaling transformers to 1,000 layers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 10, pp. 6761–6774, 2024.
- [19] D. Misra, "Mish: A self regularized non-monotonic neural activation function," arXiv preprint arXiv:1908.08681, 2019.
- [20] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, and G. M. et al., "Palm: Scaling language modeling with pathways," *Journal of Machine Learning Research*, vol. 24, no. 240, pp. 1–113, 2023. [Online]. Available: http://jmlr.org/papers/v24/22-1144.html
- [21] N. Shazeer, "GLU variants improve transformer," CoRR, vol. abs/2002.05202, 2020. [Online]. Available: https://arxiv.org/abs/2002. 05202
- [22] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. V. Le, G. E. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *CoRR*, vol. abs/1701.06538, 2017. [Online]. Available: http://arxiv.org/abs/1701.06538
- [23] J. Puigcerver, C. R. Ruiz, B. Mustafa, and N. Houlsby, "From sparse to soft mixtures of experts," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=jxpsAj7ltE