CHESSFORMER: A UNIFIED ARCHITECTURE FOR CHESS MODELING

Anonymous authors
Paper under double-blind review

ABSTRACT

Chess has played a uniquely important historical role as a testbed domain for artificial intelligence. Applying new architectures to improve absolute chess performance, and more recently to predict human moves at specified skill levels, has therefore garnered attention in the machine learning literature. Current approaches to these problems employ transformer models with widely varying architectural designs, and use unintuitive tokenization schemes that are not amenable to interpretability techniques, which hinders their applicability for teaching and human-AI interaction. We introduce Chessformer, a novel chess transformer model design that consists of an encoder-only model which processes chessboard squares as input tokens, instead of moves or the entire position; a dynamic positional encoding scheme that allows the model to flexibly adapt to the unique geometries present in chess; and an attention-based policy output design. We show that Chessformer advances the state of the art in all three major chess modeling goals: it significantly improves the chess-playing performance of a state-of-the-art chess engine, it surpasses the previous best human move-matching prediction performance with a much smaller model, and it enables substantial interpretability benefits. Our unified approach constitutes a broad advance across several important tasks in chess AI, and also demonstrates the benefits of carefully adapting transformers' tokenization systems, output systems, and positional encodings to reflect the structure of a domain of interest.

1 INTRODUCTION

A central goal of artificial intelligence (AI) is to build systems that are simultaneously high-performing and human-compatible. Capable yet intelligible models not only solve tasks but can also collaborate with and teach human users by understanding their strengths, weaknesses, and learning trajectories. Chess provides a particularly well-suited model system for this dual goal: while modern engines are decisively superhuman, their behavior is often opaque even to experts, leaving a persistent gap on the human side of the problem.

The current literature on developing strong chess engines and accurate human move prediction is disjointed and at times unprincipled. Prior systems for human emulation span convolutional stacks over board images (McIlroy-Young et al., 2020), skill-aware attention atop convolutional features (Tang et al., 2024), and language modeling over move histories (Zhang et al., 2025); other efforts distill strong oracles (Ruoss et al., 2024) with one-dimensional position representations that are potentially misaligned with the underlying action space, complicating both efficiency and analysis.

We introduce Chessformer, a unified architecture that advances the state of the art on three fronts at once: it produces substantial gains over prior methods in raw chess engine ability, it achieves state of the art human move-matching performance, and it admits downstream interpretability much more naturally than previous models. Concretely, Chessformer is an encoder-only transformer that treats the 64 board squares as tokens, pairs this square-token body with an attention-based "source-destination" policy head, and equips the trunk with Geometric Attention Bias (GAB), a novel dynamic positional-bias generator that adapts attention to the geometry of a position. It has also been adopted in a leading open-source engine (anonymized as Apollo) in configurations that defeated Stockfish, a perennial world champion, in computer-chess competitions.

Empirically, we find that GAB is a key driver of these gains. Ablations show consistent improvements over absolute and relative position encodings across Elo, puzzle accuracy, and policy and value accuracy, with sizable performance-per-compute advantages. These results reinforce a broader lesson: adapting tokenization, output heads, and positional encodings to the domain's structure allows transformer models to more flexibly adapt to both task mastery and human-compatibility. In chess, this yields a single model family that pushes engine strength, advances human move matching, and makes square-level attributions and attention pattern interpretability straightforward, all while being over an order of magnitude more parameter and compute efficient than prior art. That Chessformer achieves these simultaneously is surprising, both because chess is a very well-studied problem which is considered a gold standard for AI, and because past approaches have focused on only one of these at a time. In short, Chessformer demonstrates that principled, domain-conforming architectural choices can deliver state-of-the-art chess performance and human emulation, while also enabling interpretability analyses that can power downstream human-compatible applications, offering a template for other structured decision-making domains. We open-source our full pipeline, including all code and training data, at (link removed for anonymity).

2 Related Work

Traditionally, computational approaches to chess have focused on maximizing absolute playing strength by developing hand-crafted search heuristics and position evaluations. This approach gave rise to Deep Blue (Campbell et al., 2002), which in 1997 became the first computer to defeat a reigning human World Chess Champion under official tournament conditions, and Stockfish (Romstad et al., 2023), which is widely considered the strongest engine available today. AlphaZero (Silver et al., 2018) introduced a different recipe based on Monte Carlo Tree Search (MCTS) and reinforcement learning, training neural networks through self-play to predict state values and policy distributions over subsequent actions. Its open-source re-implementation Leela Chess Zero (Pascutto & Linscott, 2019) refined this approach with new neural network architectures and search strategies and often ranks as a close runner-up to Stockfish in computer chess competitions. Even without search, transformer-based agents can achieve grandmaster-level strength when strong oracles are distilled into them (Ruoss et al., 2024).

A more recent line of work aims to develop chess systems that are not only strong but also human-compatible, in the sense that they understand and are attuned to the behavior of humans, by modeling human play across skill levels. This was first explored by Maia (McIlroy-Young et al., 2020), which employed a set of convolutional neural networks, each trained to model players at a specific rating range. Jacob et al. (2022) augmented these with search to better replicate strong play. Maia-2 (Tang et al., 2024) simplified Maia's approach with a unified model, introducing a skill-aware self-attention layer that tokenized the channels of the output of a stack of convolution layers. Allie (Zhang et al., 2025) viewed this behavior replication task through the lens of language modeling, training a decoder-only transformer model on a move-based representation of the game trajectory to achieve state-of-the-art human move matching. These human emulation methods were exploited by Hamade et al. (2024) to investigate human-AI cooperation in chess. Modeling individual behavior, McIlroy-Young et al. (2021) demonstrated that human players can be reliably identified from just a few of their games, while McIlroy-Young et al. (2022) improved move-matching performance on individual players by finetuning on their games.

Chess has also served as a model system for broader problems. Farebrother et al. (2024) used chess to demonstrate that classifying rather than regressing improves scalability in deep reinforcement learning, while Feng et al. (2025) investigated creative generative AI in the testbed of chess puzzles. In mechanistic interpretability, which studies the mechanisms underlying the behavior of AI models, McGrath et al. (2022) used linear probes to identify concepts learned by AlphaZero, while Jenner et al. (2024) found evidence of learned planning in a transformer model trained by the Leela Chess Zero project.

3 Methodology

Here we describe our Chessformer architecture and general training setup. Fundamentally, Chessformer is an encoder-only transformer that processes the 64 chessboard squares as tokens and aug-

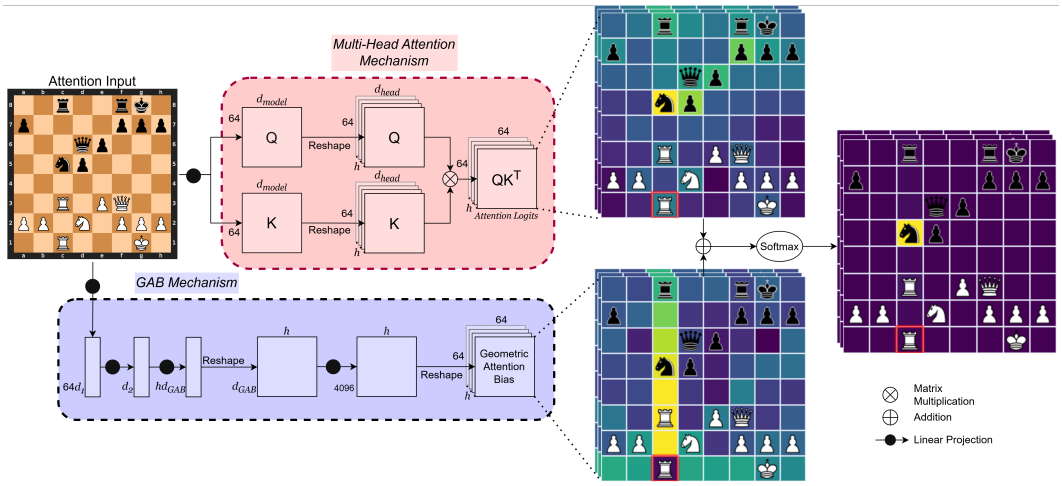

Figure 1: Chessformer attention mechanism. Chessformer adopts the natural visual representation of the chessboard, processing the 64 board squares as tokens. It augments the dot-product attention mechanism with the *Geometric Attention Bias* (GAB), a novel position encoding that generates biases for attention logits from a compressed representation of the board state. A consistent theme of Chessformer models is harmonious collaboration between the semantic dot-product attention component and the positional GAB component. In the example shown, white's rook on c1 is the querying square, highlighted in red. Higher attention scores are colored yellow, while lower attention scores are colored purple. The dot-product logits highlight important pieces, while the GAB biases focus on squares that are a rook's move away. Together, they point white's rook to the pinned black knight on c5.

ments self-attention layers with a novel dynamic position encoding called the Geometric Attention Bias. We evaluate this architecture on two chess modeling tasks: emulating human play and maximizing raw playing strength as the distillation target of a strong engine. More concretely, for both tasks, we assemble a dataset of chess games and train Chessformer models in a supervised fashion to take in board states from those games and predict both game outcomes and played moves (or for engine distillation, distributions over moves corresponding to playouts chosen during engine self-play).

3.1 Board Representation: Squares as Tokens

Existing transformer-based methods in chess use a variety of tokenization schemes: ALLIE represents a position through the trajectory of past moves, MAIA-2 tokenizes the channels of the output of a stack of convolution blocks, and Ruoss et al. (2024) applies rotary position embeddings (Su et al., 2024) on top of a tokenized representation of the 64 board squares and Forsyth-Edwards Notation (FEN) (Edwards, 1994) representation of the position. We argue in Appendix D that these formulations are misaligned with the underlying domain and may therefore hinder performance. Our Chessformer architecture instead adopts the natural visual representation that treats individual chessboard squares as tokens, allowing model representations to correspond to more uniform units and fixing the relative positions of tokens in two dimensions to enable more effective position encodings. It also allows tokens to specialize to their corresponding squares rather than represent the entire board state, which greatly reduces the load on parameters.

Concretely, positions are represented as a sequence of 64 one-hot or zero vectors of dimension 12 indicating which of the 12 pieces are present on that square, and the board is flipped with the side to move. To obtain the input for a given board state, we concatenate representations of the current and past n positions, where n is a non-negative integer controlling the amount of history information conditioned upon, and repeat the earliest position if some or all of these past positions are not available. Unless otherwise stated, $n = 7$. For the engine distillation setup, we also concatenate auxiliary information that was necessary for compatibility with the Apollo engine infrastructure,

though this did not noticeably affect performance in initial experiments; see Appendix A.2. The final board representation consists of 64 tokens, one for each square.

Our setup is most similar to that of Ruoss et al. (2024), which tokenized the 64 board squares in addition to other information about the position, for a total of 77 tokens. However, that work adopted rotary position embeddings (Su et al., 2024) on a linearized representation of the board squares, enforcing a one-dimensional structure on the board that is not grounded in the domain. This is especially deleterious given the central role that position plays in chess. For example, among relationships between squares, their architecture maximally decays the attention strength between opposite corners, even though those corners lie on a main diagonal that is critical for checkmate patterns. We propose a novel position encoding called GAB that rectifies this by allowing self-attention to dynamically and flexibly model piece movement in two dimensions.

3.2 Geometric Attention Bias (GAB)

Self-attention in Transformer architectures is permutation-invariant, so positional information must be introduced to the model through some kind of position encoding. In language and vision settings, simple Euclidean distance is arguably the predominant notion of position, and thus static position encoding schemes, like rotary position embeddings and absolute and relative biases (refer to Appendix C for details), power state-of-the-art models in these domains. However, chess follows its own special geometry, in which the six piece types move in particular ways. In addition, positional relationships can vary heavily with the board state. As a simple example, relationships corresponding to the movement of a particular piece are only sensible if that piece is present on the board. But chess is rife with more complex interactions; for example, connections between distant squares are weaker in locked positions. A more versatile positional encoding is necessary.

To capture the variable geometry of chess, we propose an adaptive position encoding called Geometric Attention Bias (GAB). GAB uses a compressed representation of the board state to generate biases for each attention head from a set of "templates". To compress the board state, tokens first undergo a linear projection of dimension d_1 and are flattened, followed by a linear projection of dimension d_2 with GELU activation and layer normalization. To generate the attention biases, we apply another linear projection of depth hd_3 followed by activation and normalization, and reshape to $h \times d_3$. We apply a final linear projection, shared by the whole model to reduce parameter count, to form biases $h \times 4096$ which are reshaped to $h \times 64 \times 64$ and added to the dot-product logits before softmax. This final projection can be viewed as dynamically mixing a set of d_3 attention bias templates. Pseudocode for generating GAB biases can be found in Figure 3.

Our approach has a number of benefits. First, it models positional information globally rather than through individual tokens. This aligns well with our later finding that Chessformer models mainly adapt the GAB component of the self-attention computation to global positional features like the game stage (opening, middlegame, endgame), rather than local features like the locations of individual pieces. Second, representing attention logits as the sum of a semantic component generated by dot-product attention and a positional component generated through GAB allows self-attention layers to assign relevant aspects of the attention computation to each part. The choice of a dynamic position encoding enables attention heads to be repurposed based on the board state, a behavior we explore in Section 6. Finally, formulating the interaction additively allows us to use existing memory-efficient attention kernels (Dao et al., 2022) during training and inference.

3.3 Output Heads

All models we train have two output heads: a value head that predicts the game outcome (*win*, *draw*, and *loss*), and a policy head that predicts the move (or in the case of oracle self-play games, the distribution of moves corresponding to playouts) chosen during the game. To generate the game outcome prediction, we apply mean pooling to the encoder body's output, followed by a layer normalization layer. We then apply a linear projection to dimension 128, followed by a ReLU nonlinearity, followed by a linear projection of dimension 3 whose output is taken as the logits for the game outcome prediction target.

Prior work has modeled move distributions in a variety of ways. ALLIE autoregressively predicted Universal Chess Interface (UCI) tokens corresponding to each of the 1968 possible moves in UCI

Table 1: Main results for human move-matching accuracy. The ZEUS family of models achieves new state-of-the-art performance with a fraction of the parameter count.

Agent	Accuracy (%)	Parameters	History	Search
ZEUS-79M	57.1 ± 0.1	79M	✓	✗
ZEUS-23M	56.6 ± 0.1	23M	✓	✗
ZEUS-5M	55.4 ± 0.1	5M	✓	✗
ALLIE-ADAPTIVE-SEARCH	55.9 ± 0.1	355M	✓	✓
ALLIE-POLICY	55.7 ± 0.1	355M	✓	✗
MAIA-2	52.0 ± 0.1	23M	✗	✗
MAIA*	51.6 ± 0.1	92M	✗	✗
GPT-3.5	53.7 ± 0.1	175B	✓	✗

notation, while MAIA-2 used an MLP layer. We propose a policy head based on self-attention that reflects the "from-to" structure of the underlying action space, encoding moves by the starting square and destination square of the moved piece. Given the 64 tokens returned by the encoder body, 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, resulting in a 64x64 matrix representing all possible traversals from one chessboard square to another. In effect, logits are computed as a bilinear function of the processed tokens. This is sufficient to represent all moves except castling and promotions, implementation details for which are reported in Appendix A.3.

Though we did not observe a large performance benefit from this policy head formulation, we found it to substantially improve the interpretability of MLP activations, which we attribute to its alignment with the domain; see Section 6. We hope that this design choice will help position our proposed architecture as a useful test case for future mechanistic interpretability research.

4 PREDICTING HUMAN PLAY

4.1 DATASET

We construct a training dataset consisting of blitz games played on the online chess platform Lichess from January 2023 to July 2025. As the bulk of these games are from the middle of the skill distribution, we re-sample the games during training so that all skill levels are equally represented. The standard evaluation metric for human emulation is move-matching accuracy, which is the rate at which, given a board position encountered in a real game, a model correctly predicts the move played by a human player. For our main analysis, we adopt the dataset of 884,049 positions curated by Zhang et al. (2025), which we call the ALLIE test set. These were formed by sampling Lichess blitz games from 2022 and removing the first 10 moves from each game, as these can be easily memorized, as well as positions which occur after the first time a player has less than 30 seconds on the clock, which tend to be more noisy due to time pressure. During training, we retain the first 10 moves but discard moves made under time pressure in the same way. We describe the processing of training data in more detail in A.1.

4.2 TRAINING METHODOLOGY

We perform three main training runs at scales of 5 million, 23 million, and 79 million parameters, calling the largest of these ZEUS. We ablate the position encoding at the smallest scale, comparing to the absolute and relative biases described in Appendix C. We also train a smaller 3 million parameter ablation to show that GAB enables significantly smaller models to outperform these. For GAB models at the 5M and 3M scale, we replace the first linear projection and flattening layer of GAB with average pooling as it is parameter-intensive at small scales. Initial experiments showed this to have only a minor effect on performance, decreasing move-matching accuracy by approximately 0.2%.

Table 2: Position encoding ablations for human emulation.

Representation	Loss		Accuracy (%)		FLOPS	Params
	Policy	Value	Policy	Value		
Absolute	1.4176	0.7538	54.7 ± 0.1	62.6 ± 0.1	268M	4.58M
Relative bias	1.4200	0.7538	54.6 ± 0.1	62.6 ± 0.1	268M	4.58M
ZEUS-5M	**1.3873**	**0.7361**	$\mathbf{55.4 \pm 0.1}$	$\mathbf{62.8 \pm 0.1}$	276M	4.91M
ZEUS-3M	1.4192	0.7390	54.8 ± 0.1	62.6 ± 0.1	**164M**	**2.98M**

Skill in chess is modeled with the Elo system, where Elo ratings vary roughly from 0 for weak players to 3000 for the strongest human players. We condition human-emulating models on the skill levels of both players by prepending two "soft embeddings" of dimension 128, corresponding to the Elo ratings of the players, to each of the 64 tokens. Following ALLIE, we compute an embedding e_k for an Elo rating k as a linear interpolation between two learnable embeddings: a weak embedding (e_{weak}) corresponding to 0 Elo and a strong engine-level embedding (e_{strong}) corresponding to 5000 Elo. Formally, we set $e_k = \gamma e_{\text{weak}} + (1-\gamma)e_{\text{strong}}$, where $\gamma = \frac{5000-k}{5000}$. Representing the Elo ratings as scalar inputs would achieve the same representational capacity, but this design enables the flexibility to, for example, model individual behavior stylometry. There are several Elo rating systems currently in use that are generally incomparable. Ratings on Lichess, from which we source our training data, tend to be slightly inflated compared to the more standard FIDE ratings.

The final input for our human emulation models consists of 64 tokens, a concatenation of representations of the current and n past board states and two strength embeddings of dimension 128. This comes out to a depth of $12 \times (1+n) + 2 \times 128$, which is 352 for the $n = 7$ hyperparameter choice used in our main training and ablations runs. Despite the dimensions of this input being dominated by these embedding vectors, we did not find the choice of embedding dimension 128 to impact performance. Detailed information on our training setup and model hyperparameters can be found in Appendix A.1.

4.3 BOARD HISTORY

History-less representations of chess have a "Markov Property" in that future board states are independent of previous states given the current state, with the (typically negligible) caveat of the threefold repetition rule that a game shall end in a draw when a position repeats three times. Though optimal play in a given position is almost always independent of previous moves, it is not clear that this should extend to human play. Humans often form long-term plans or exhibit consistent weaknesses that may be discernible from their previous actions, which may make this information useful when replicating their play. Prior human emulation work varies in its use of history information, with move-token methods like ALLIE conditioning on the full game history, and square-based methods like MAIA and MAIA-2 omitting history information entirely. To determine the usefulness of history information in modeling human chess play, we condition models on the current and past n board states for varying values of n. Ablation details can be found in Appendix E

4.4 RESULTS

Table 1 reports overall move-matching accuracies of ZEUS models on the ALLIE test set, demonstrating a marked improvement in move-matching performance and parameter efficiency. Figure 5 shows that these gains hold across a wide range of skill levels, with the improvement increasing in the game rating. Impressively, our 79M-parameter and 23M-parameter models achieve respective move-matching accuracies of 57.1% and 56.6%, outperforming the state-of-the-art 355-million-parameter searchless (55.7%) and search-enabled (55.9%) ALLIE methods at significantly smaller scales. Our 5M-parameter model reaches a move-matching accuracy of 55.4%, *achieving results comparable to the state of the art at 70 times fewer parameters*. The MAIA-2 paper trained two models, one on rapid games and one on blitz games, and we evaluate the latter for a fair comparison. Results for MAIA* and GPT-3.5 are taken from Zhang et al. (2025). MAIA* was formed by choosing the MAIA model with Elo rating closest to the active player's Elo rating, and GPT-3.5, which outperformed more recent models like GPT-4, operates on a move-based board representation.

We plot the move-matching accuracies of ZEUS by the Elo ratings of the active and opponent players in Figure 4. Interestingly, stronger players appear easier to predict when paired against weaker players, possibly because winning paths are clear in the decisively winning positions that tend to occur in these uneven match-ups. Results for ablating the position encoding are presented in Table 2, while results for ablating the number of history positions n are reported in Table 8. The baseline ZEUS-5M model significantly outperforms ablations equipped with absolute and relative biases, while ZEUS-3M matches their performance at 30% fewer parameters and 40% fewer FLOPS. In other words, GAB enables a Pareto improvement across model scale, computation, and policy and value metrics.

5 OPTIMIZING PLAYING STRENGTH

We now test our Chessformer architecture—a 64-token encoder transformer with GAB biases and attention policy—on the task of optimizing raw searchless chess playing strength. To do so, we distill the Apollo (name changed for anonymity) engine into Chessformer models, and perform ablations on the position representations to assess their impact. We then analyze a 191-million parameter Chessformer model trained by that project, which we call Apollo-CF, comparing agents constructed from that model to prior work on searchless playing strength. Finally, we demonstrate that these gains in searchless playing strength can be extended to engine strength— first by performing a tournament between configurations of Apollo equipped with either the Apollo-CF Chessformer or the Apollo-CNN convolutional model previously used in tournaments— and then by describing several prominent computer chess tournaments in which Chessformer-equipped Apollo configurations defeated pools of engines that included Stockfish.

5.1 TRAINING METHODOLOGY

Apollo is an open-source recreation of AlphaZero, which iteratively teaches a randomly initialized neural network to interact with a domain by generating self-play games between MCTS-augmented versions of that neural network. This neural network outputs a policy distribution that predicts the distribution of moves chosen by the MCTS algorithm and a value that predicts the outcome of the game. In effect, a model continually generates strong search-enabled oracles whose play is then distilled back into that model.

Table 3: Results for raw playing strength. Chessformer results in significant gains in both Elo and puzzle solving rate, using fewer FLOPS than competing models.

Agent	Elo	Puzzles (%)	FLOPS
Apollo-CF-policy	2374 ± 37	93.5 ± 0.5	7.6B
Apollo-CF-value	2466 ± 36	97.2 ± 0.3	152B
AC-9M	2044 ± 42	86.2 ± 0.7	14.2B
AC-136M	2257 ± 36	92.7 ± 0.5	215B
AC-270M	2299 ± 36	94.2 ± 0.5	427B
Apollo-CNN-policy	2096 ± 40	82.1 ± 0.8	12.5B
Apollo-CNN-value	2168 ± 36	92.5 ± 0.5	249B

The most expensive component of the AlphaZero process by far is the generation of training games, which typically requires hundreds of model evaluations per position. We skip this step by fixing a dataset of self-play games from an April 2024 Apollo reinforcement learning run, keeping only those games that occurred once the model's strength had leveled off. That run used a transformer model of roughly 100 million parameters at 600 nodes per move with a square-based token representation and our GAB biases and attention policy. In this way, we move to the supervised setting, distilling a search-augmented version of one model into another.

We motivate our design choices under this setup, ablating the position encoding of a 4-million-parameter model with those described in Appendix C. We also train a Chessformer model with 2.5 million parameters to determine the extent to which our techniques can replace the need for model size. Our oracle distillation setup is described in more detail in Appendix A.2.

5.2 EVALUATION METHODOLOGY

We include in our analysis two types of agents: those that function based on policy information by picking the highest-ranked move in the policy distribution predicted by the model, and those that function based on value information, emulating a depth-1 search by evaluating the model for each legal move and selecting the move that maximizes the resulting 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 conservative estimate of the average number of legal moves available in a position.

We construct agents from the 191-million parameter model using both strategies, denoting an agent by its model name followed by the strategy it uses (e.g., Apollo-CF-policy and Apollo-CF-value). We also construct agents from both strategies using a 195-million-parameter convolutional neural network trained by the Apollo project, which we call Apollo-CNN. Our analysis also includes models from Ruoss et al. (2024) that 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 AC-9M, AC-136M, and AC-270M, respectively.

Our evaluation setup is adapted almost exactly from Ruoss et al. (2024). We compare agents on both tournament strength and puzzle solving ability. To measure the former, we play 200 games between each pair of agents and calculate relative Elo ratings using BayesElo (Coulom, 2008). We perform separate tournaments for the main and ablation runs, anchoring the Elo value of the absolute position encoding to 0 and the Elo value of the AC-270M to the value reported by Ruoss et al. To measure puzzle solving ability, we report the accuracy of these models on a test set of 10 thousand puzzles curated by that work. For our ablation runs, we also report accuracy and loss metrics on the value and policy heads. These were calculated on 1,403,999 test positions, also sourced from the Apollo reinforcement learning run, that did not intersect the training set.

5.3 Results

As shown in Table 3, the Chessformer policy agent Apollo-CF-policy matches or outperforms all other agents we consider in both tournament strength and puzzle-solving ability despite having the lowest computational cost. We also see performance on the puzzle set approaching saturation, suggesting that new metrics for evaluation might be needed.

Our ablation results, reported in Table 6, show a monotonic improvement in performance from absolute to relative bias to GAB encodings. GAB outperforms the baseline absolute position encoding by 1.96% on move-matching accuracy, 0.34% on game outcome prediction accuracy, 3.37% on puzzle-solving accuracy, and 80 Elo rating points in tournament strength. We note that the maximum possible policy and value accuracies are well under 100%, both because Apollo's self-play methodology is inherently stochastic and because there are positions with a variety of best moves. Impressively, GAB enables a model to perform on-par with the absolute position ablation at nearly half the parameters and computation.

5.4 Engine Strength

To demonstrate that our Chessformer models have the capacity to push engine strength, we perform a tournament between configurations of the Apollo chess engine paired with either the Apollo-CNN model, a 195-million-parameter convolution-based model previously used by Apollo at tournaments, or Apollo-CF, a 191-million parameter Chessformer. We play 2000 games between these configurations at three time controls, described in further detail in Appendix B. As shown in Table 7, the Apollo-CF Chessformer model consistently increases the playing strength of Apollo by over 100 Elo. To contextualize these gains, the difference in playing strength between Stockfish versions 16 and 17, corresponding to 14 months of continuous development progress, was measured at around 46 Elo with a similar testing setup (Stockfish Team, 2024). This is especially notable due to the difficulty of improving top engines; the Stockfish project continually employs thousands of CPU cores to test over 10,000 potential improvements per year (Stockfish Team, 2022).

Configurations of Apollo equipped with Chessformer models defeated Stockfish, a perennial champion, at several online computer chess tournaments hosted by (name removed to preserve anonymity). These victories included a single-elimination Cup tournament[1], where 32 engines faced off in brackets, and two Swiss-system tournaments[2], each of which had around 40 contestants. In the cup-style event, Apollo used an 82M-parameter Chessformer model, and beat Stockfish in the

[1] See `https://en.wikipedia.org/wiki/Single-elimination_tournament`.

[2] See `https://en.wikipedia.org/wiki/Swiss-system_tournament`.

final round. In the Swiss events, Apollo scored first place both times, once one point ahead of the field and once on tiebreak.

6 INTERPRETABILITY

A recent line of work has used chess as a testbed for mechanistic interpretability, which aims to identify the mechanisms by which an AI model computes its outputs. Our domain-grounded architectural choice eases this process by allowing square-specific attribution of activations and attention patterns. As a preliminary investigation of the interpretability benefits of our approach we present an overview of the mechanisms by which GAB and dot-product attention cooperate.

6.1 GAB MAPS

We inspect the attention heads of both ZEUS and Apollo-CF, finding that they tend to represent semantic information with the dot-product attention logits and positional information with the GAB biases. In general, we observe that GAB represents different movement types and measures of proximity on the chessboard. In contrast, the dot-product attention (DPA) maps appear to focus more on global semantic information such as important enemy pieces.

To quantitatively evaluate this observation, we measure how much the rows of the GAB and DPA attention maps vary both between positions and within a single position. For the between-position variability, we randomly sample 30,000 positions from Lichess blitz games played in June 2019. For each query square, we extract the corresponding row-centered, L2-normalized GAB attention rows across all positions and compute the average cosine distance between them (i.e., GAB–GAB similarity for that square). We repeat the same procedure for the DPA maps (DPA–DPA similarity), and then average over all query squares, as well as all heads and layers.

For within-position variability, we fix a position and compute the average cosine distance among all pairs of row-centered, L2-normalized GAB maps across different query squares for a given head and layer in that position (i.e., GAB–GAB similarity within the same position). We then average over all heads and layers. We do the same for DPA. In both cases, we compare GAB–GAB and DPA–DPA statistics; we do not measure the similarity between GAB and DPA directly.

We find, consistent with our hypothesis, that the query square's GAB bias exhibits substantial variation across positions, unlike a fixed positional encoding. This suggests that GAB adapts meaningfully to the position context. However, this variation is much smaller than that of dot-product attention, indicating that GAB is still relatively stable between positions (see Figure 4). At the same time, GAB is more variable across query squares than dot-product attention, meaning it is more square-specific within positions (see Figure 4). Taken together, these findings suggest that GAB primarily captures specific information relevant to the query square, whereas dot-product attention is less tied to specific squares and instead reflects semantic information about the position as a whole.

Following this intuition, there are several examples in our models of GAB biases adapting to global features of the board state, like the game stage. Figure 2 shows an example of this where the GAB component of the attention computation transitions from modeling general piece movement (left) to modeling king movement (right) (More in Appendix F.2).

6.2 IDENTIFYING INTERPRETABLE FEATURES

A prominent approach to transformer interpretability consists of training Sparse Autoencoders (SAEs) or transcoders on a model's MLP activations, then interpreting their basis vectors by examining the inputs which activate them the most to identify interpretable features and circuits within the model Lindsey et al. (2024). Much previous work applies mechanistic interpretability to the testbed of chess Karvonen et al. (2024).

Our architecture enables a much finer interpretation based on transcoder features. Instead of merely examining the positions which maximally activate an MLP feature, one can identify the *squares* at which a feature is maximally activated. To explore this useful property of our architecture, we train a cross-layer transcoder on ZEUS (training details in Section A.4). We find that our resulting features are interpretable at a high rate and rich in the space of concepts they cover. For example, many

features correspond to core tactical concepts like forks and pins, whose top activated squares are those involved in the tactical dynamics. More importantly, we find features that would be otherwise uninterpretable from only their top activated positions, but whose top activated squares exhibit a clear pattern. Space constraints preclude us from including all 8192 transcoder features, but we visualize and annotate the top activated positions of the first 20 features of both transcoder layers in Figure 7 (since feature order is arbitrary, these are essentially randomly selected). Interested readers can also explore all features and their top activated positions at *URL redacted for anonymity*. The abundance of interpretable features in ZEUS and the usefulness of square-level activation attribution in parsing their meanings establish our architecture as a significant step forward towards the goal of interpretable chess modeling.

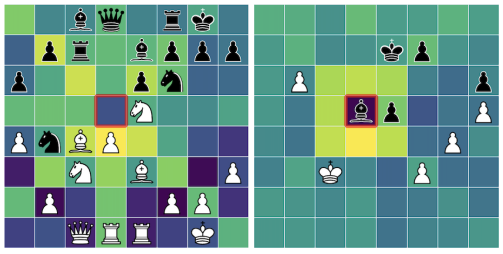

Figure 2: GAB bias maps in L14H11 of Apollo-CF in the early and late game. The GAB bias for this head transitions from modeling a wide range of movement in the early game to king movement in the late game.

Table 4: Cosine distance of attention rows within and between positions for Geometric Attention Bias (GAB) and dot-product attention (DPA). GAB exhibits higher variability within positions than DPA, but lower variability between positions.

Cosine distance	**GAB**	**DPA**
Between positions	0.230	0.770
Within positions	0.995	0.184

7 DISCUSSION

Our central contribution is Chessformer, an architecture for chess modeling that jointly advances the state of the art for raw engine strength and human-move prediction, while doing so in a naturally interpretable way. A Chessformer variant was integrated into a strong open-source engine and contributed to match wins over Stockfish in multiple computer-chess tournaments.

For human emulation, ZEUS reaches 57.1% move-matching accuracy and surpasses prior searchless (55.7%) and search-enabled (55.9%) Allie methods with fewer than one-fourth the parameters, indicating a significant improvement in architecture design. Ablations show a monotonic improvement from absolute to relative to GAB position representation, with GAB delivering +1.96% policy accuracy, +0.34% value accuracy, +3.37% puzzle-solving accuracy, and +80 Elo, and matching an absolute-encoding baseline at roughly half the compute. Together, these results establish that our methodology is doing substantive work, yielding performance-per-compute advantages and better Pareto points across Elo, puzzles, and policy and value accuracies.

Beyond chess, the general lesson is that it may be possible to make progress on both raw task performance and human-compatibility goals by allowing models to better conform to the domain at hand. In our case, this alignment closes the gap between modeling form and action space, enabling simultaneous gains in mastery (engine Elo; oracle/puzzle metrics) and human-compatibility (move-matching across skill levels) while allowing for interpretability testing that could enable progress on further downstream tasks.

Our methods have limitations that suggest concrete next steps. GAB is currently chess-specific, and its benefits may rely on domains where geometric relations are central. Extending this template to other structured decision problems invites exploration of further geometric modeling. The interpretability benefits—square-level maps, complementary roles of GAB versus dot-product attention—are promising but require deeper exploration.

REFERENCES

Murray Campbell, A. Joseph Hoane, and Feng hsiung Hsu. Deep blue. *Artificial Intelligence*, 134 (1):57–83, January 2002. ISSN 0004-3702.

Rémi Coulom. Whole-history rating: A bayesian rating system for players of time-varying strength. In *Computers and Games*, 2008.

Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. Flashattention: fast and memory-efficient exact attention with io-awareness. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA, 2022. Curran Associates Inc. ISBN 9781713871088.

Steven J. Edwards. Standard: Portable game notation specification and implementation guide, 1994. URL https://ia802908.us.archive.org/26/items/pgn-standard-1994-03-12/PGN_standard_1994-03-12.txt.

Jesse Farebrother, Jordi Orbay, Quan Vuong, Adrien Ali Taiga, Yevgen Chebotar, Ted Xiao, Alex Irpan, Sergey Levine, Pablo Samuel Castro, Aleksandra Faust, Aviral Kumar, and Rishabh Agarwal. Stop regressing: Training value functions via classification for scalable deep RL. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=dVpFKfqF3R.

Xidong Feng, Vivek Veeriah, Marcus Chiam, Michael D Dennis, Federico Barbero, Johan Obando-Ceron, Jiaxin Shi, Satinder Singh, Shaobo Hou, Nenad Tomasev, and Tom Zahavy. Generating creative chess puzzles. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=TNZse5q2Tr.

Andrew Grant. Openbench. https://github.com/AndyGrant/OpenBench. Accessed: 2025-11-22.

Wes Gurnee, Neel Nanda, Matthew Pauly, Katherine Harvey, Dmitrii Troitskii, and Dimitris Bertsimas. Finding neurons in a haystack: Case studies with sparse probing. *arXiv preprint arXiv:2305.01610*, 2023. URL https://arxiv.org/abs/2305.01610.

Karim Hamade, Reid McIlroy-Young, Siddhartha Sen, Jon Kleinberg, and Ashton Anderson. Designing skill-compatible AI: Methodologies and frameworks in chess. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=79rfgv3jw4.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7132–7141, 2018. doi: 10.1109/CVPR.2018.00745.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. In Ricardo Silva, Amir Globerson, and Amir Globerson (eds.), *34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018*, 34th Conference on Uncertainty in Artificial Intelligence 2018, UAI 2018, pp. 876–885. Association For Uncertainty in Artificial Intelligence (AUAI), 2018.

Athul Paul Jacob, David J Wu, Gabriele Farina, Adam Lerer, Hengyuan Hu, Anton Bakhtin, Jacob Andreas, and Noam Brown. Modeling strong and human-like gameplay with kl-regularized search. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162, pp. 9695–9728. PMLR, July 2022.

Erik Jenner, Shreyas Kapur, Vasil Georgiev, Cameron Allen, Scott Emmons, and Stuart Russell. Evidence of learned look-ahead in a chess-playing neural network. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 31410–31437. Curran Associates, Inc., 2024. doi: 10.52202/079017-0987. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/37d9f19150fce07bced2a81fc87d47a6-Paper-Conference.pdf.

Adam Karvonen. Emergent world models and latent variable estimation in chess-playing language models. In *First Conference on Language Modeling*, August 2024.

Adam Karvonen, Benjamin Wright, Can Rager, Rico Angell, Jannik Brinkmann, Logan Smith, Claudio Mayrink Verdun, David Bau, and Samuel Marks. Measuring progress in dictionary learning for language model interpretability with board game models. *arXiv preprint arXiv:2408.00113*, 2024. doi: 10.48550/arXiv.2408.00113. URL https://arxiv.org/abs/2408.00113.

Jack Lindsey, Adly Templeton, Jonathan Marcus, Thomas Conerly, Joshua Batson, and Christopher Olah. Sparse crosscoders for cross-layer features and model diffing. *Transformer Circuits Thread*, 2024. URL https://transformer-circuits.pub/2024/crosscoders/index.html. Preprint / technical note.

Thomas McGrath, Andrei Kapishnikov, Nenad Tomašev, Adam Pearce, Martin Wattenberg, Demis Hassabis, Been Kim, Ulrich Paquet, and Vladimir Kramnik. Acquisition of chess knowledge in alphazero. *Proceedings of the National Academy of Sciences*, 119(47):e2206625119, 2022.

Reid McIlroy-Young, Siddhartha Sen, Jon Kleinberg, and Ashton Anderson. Aligning superhuman ai with human behavior: Chess as a model system. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1677–1687, August 2020. ISBN 978-1-4503-7998-4.

Reid McIlroy-Young, Yu Wang, Siddhartha Sen, Jon Kleinberg, and Ashton Anderson. Detecting individual decision-making style: Exploring behavioral stylometry in chess. In *Advances in Neural Information Processing Systems*, volume 34, pp. 24482–24497, 2021.

Reid McIlroy-Young, Russell Wang, Siddhartha Sen, Jon Kleinberg, and Ashton Anderson. Learning models of individual behavior in chess. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1253–1263, 2022.

Aaron Mei. Understanding how chess-playing language models compute linear board representations. In *ICML 2025 Workshop on Methods and Opportunities at Small Scale*, 2025. URL https://openreview.net/forum?id=Z9OV9NygER.

Gian-Carlo Pascutto and Gary Linscott. Leela chess zero, March 2019.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners, 2019.

Tord Romstad, Marco Costalba, Joona Kiiski, and et al. Stockfish. https://stockfishchess.org, 2023. Accessed: 2025-11-29.

Anian Ruoss, Grégoire Delétang, Sourabh Medapati, Jordi Grau-Moya, Li Kevin Wenliang, Elliot Catt, John Reid, Cannada A. Lewis, Joel Veness, and Tim Genewein. Amortized planning with large-scale transformers: A case study on chess. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 65765–65790. Curran Associates, Inc., 2024. doi: 10.52202/079017-2102. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/78f0db30c39c850de728c769f42fc903-Paper-Conference.pdf.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

Stockfish Team. Stockfish testing framework. https://tests.stockfishchess.org/tests. Accessed: 2025-11-22.

Stockfish Team. Stockfish 15. https://stockfishchess.org/blog/2022/stockfish-15/, September 2022. Accessed: 2025-11-19.

Stockfish Team. Stockfish 17. https://stockfishchess.org/blog/2024/stockfish-17/, September 2024. Accessed: 2025-11-19.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomput.*, 568(C), mar 2024. ISSN 0925-2312. doi: 10.1016/j.neucom.2023.127063. URL https://doi.org/10.1016/j.neucom.2023.127063.

Zhenwei Tang, Difan Jiao, Reid McIlroy-Young, Jon Kleinberg, Siddhartha Sen, and Ashton Anderson. Maia-2: A unified model for human-ai alignment in chess. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 20919–20944. Curran Associates, Inc., 2024. doi: 10.52202/079017-0659. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/250190819ff1dda47cd23cecc0c5a69b-Paper-Conference.pdf.

Yiming Zhang, Athul Jacob, Vivian Lai, Daniel Fried, and Daphne Ippolito. Human-aligned chess with a bit of search. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu (eds.), *International Conference on Representation Learning*, volume 2025, pp. 4815–4836, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/0ef1afa0daa888d695dcd5e9513bafa3-Paper-Conference.pdf.

A Implementation Details

A.1 Human Emulation

All human-mimicking models were trained with the AdamW optimizer on the dataset described in Section 4. During training, we sample 32 positions per game at random, or take all of them if 32 positions are not available. All runs have 8 layers, head dimension 32, and an MLP expansion factor of 2. Our 3M, 5M, 23M, and 79M models have embedding dimensions of 192, 256, 512, and 1024, respectively. GAB is configured with $d_1 = 32$ and $d_2 = d_3 = 128$ for the 23M and 79M models and average pooling and $d_2 = d_3 = 64$ for the 5M model and all its ablations. All runs were trained for 1 million steps with a cyclic cosine annealed learning rate schedule. The 23M and 79M main runs were performed on 8 A100 GPUs, taking a few days and a week respectively, and all other runs were trained on 2 A100 GPUs for around a week.

Because low-rated games are overrepresented in our training dataset, we downsample this dataset during training to ensure that skill levels are equally represented. In particular, the rating spectrum is divided into 22 bins: 20 bins uniformly spanning 600 to 2600 Elo with 100-point intervals, plus two bins for players rated below 600 or above 2600. For each game, we compute the average Elo of the two players and assign the game to the corresponding bin. We organize the raw game data into chunks of 20,000 games. For each chunk, we iterate through games sequentially and distribute them into bins until each bin accumulates 10 games. The process terminates when either all games in the chunk are consumed or all bins reach 10 games. This encourages equal representation across skill levels, removing the bias toward low-rated games. We then uniformly at random select 32 positions per game, or take every position if that many are not available.

The ALLIE test set is suitable to measure overall performance, and all overall performance metrics we report on the human emulation task employ it. However, it lacks positions at very high and low skill levels therefore cannot be used to form statistically significant conclusions at these skill levels. For example, it has only 205 positions rated above 2850, which would result in a margin of error of over 7%. To rectify this, we augment the ALLIE test set with very highly and lowly rated Lichess blitz games from August and September 2025 to form the ALLIE-AUGMENTED dataset, which we use to compare modeling performance across skill levels— it is used in Figure 4, Figure 5, and Figure 6, and nowhere else.

To obtain the ALLIE-AUGMENTED test set from the ALLIE test set, we first take rating intervals of length 100 between 550 and 2950 in which ALLIE contains less than 20 thousand positions. These intervals are so chosen so that they correspond to the bins in Figure 5 and Figure 6, which round the game rating. We initialize empty buckets for each such interval and iterate through September and August 2025 Lichess blitz games, adding the game to the corresponding bucket so long as it has less than a thousand games. These games are added on top of the ALLIE test set to form the final ALLIE-AUGMENTED set of 1,087,778 positions.

A.2 Playing Strength

As described in Section 5, we train in the supervised setting on games generated from a past reinforcement learning run of the Apollo project. This strategy works well in practice and was used to train Apollo-CF, which surpassed the model that generated its training data, as well as each of the models used in the victories against Stockfish described in Section 5. Initial experiments showed that Chessformer models trained on self-play games produced by convolutional neural networks and other Chessformer models reach virtually identical playing strengths, suggesting that the strength of the oracle is far more important than the model underlying its search process.

To maximize compatibility with the Apollo infrastructure, we follow their input scheme as closely as possible. We form 64 tokens of depth 96 for the current and past 7 board states, following Section 3.1. We also concatenate indicators of whether each of the current and past 7 board states was a repetition, whether each of the 4 castling options are available, whether black is to move, and the number of plies since the last capture or pawn move, divided by 100. Finally, we concatenate a 0 and 1, which are relics that were originally intended to allow convolutional models to detect edges. This information is concatenated to each token, giving 64 pre-embedding tokens of depth $(96 + 8 + 4 + 1 + 1 + 2) = 112$. Initial experiments did not show this additional information to alter modeling performance.

Table 5: Human Move-Matching Training Configuration for Reproducibility

Parameter	Value
Training Setup	
batch_size_train	128
batch_size_val	16
gradient_accumulation_steps	4
num_workers	8
Optimization	
lr	0.00005
min_lr	0.00001
wd (weight decay)	0.000001
grad_clip_norm	3.5
warmup_steps	1,000
cosine_cycles	50,000
refresh_lr_scheduler	true
Mixed Precision	
use_amp	true
amp_init_scale	256
amp_max_scale	8,192
amp_growth_factor	1.5
amp_growth_interval	2,000
amp_backoff_factor	0.5
Loss Weights	
value_coefficient	0.1

All Chessformer models trained for this task used the Nadam optimizer with $\beta_1 = 0.9$, $\beta = 0.98$, $\epsilon = 10^{-7}$, and gradient clipping 10. Following Apollo's setup, checkpoints were calculated using Stochastic Weights Averaging (Izmailov et al., 2018), which incrementally boosted performance.

The 191-million parameter Apollo-CF model has hidden dimension 1024, MLP dimension 1536, a head size of 32, and 15 layers, and GAB is configured $d_2 = d_3 = 256$, and the first linear project and flattening layers are replaced with average pooling, which initial experiments showed greatly improves parameter efficiency at small scales at the cost of a slight performance degradation. It was trained for 6 million steps, with the learning rate initialized to 2×10^{-3} and dropped to 3×10^{-4} at 4.49 million steps and 3×10^{-5} at 5.47 million steps. The 195-million parameter Apollo-CNN is a squeeze-excitation ResNet (Hu et al., 2018) (He et al., 2016) with 512 filters and depth 40.

Table 6: Ablation results for raw playing strength. Accuracies and Elo values are reported with 95% confidence intervals

Representation	Loss		Accuracy (%)		Puzzles	Elo	FLOPS	Params
	Policy	Value	Policy	Value	(%)			
Absolute	0.363	0.567	56.6 ± 0.1	88.7 ± 0.1	61.0 ± 1.0	0 ± 18	210M	3.67M
Relative bias	0.346	0.565	57.5 ± 0.1	88.8 ± 0.1	63.2 ± 1.0	40 ± 18	210M	3.67M
GAB-4M	**0.330**	**0.562**	$\mathbf{58.5 \pm 0.1}$	$\mathbf{89.0 \pm 0.1}$	$\mathbf{64.2 \pm 1.0}$	$\mathbf{83 \pm 18}$	228M	4.01M
GAB-2.5M	0.360	0.567	57.0 ± 0.1	88.7 ± 0.1	61.5 ± 1.0	-4 ± 18	**131M**	**2.51M**

We train ablations with the three position representations described in Section 3 with 8 layers, embedding dimension 256, head dimension 32, and MLP dimension 256. GAB is configured with $d_1 = 8$ and $d_2 = d_3 = 32$. The smaller model has embedding dimension and MLP dimension 192, with all else held constant. Each was trained for 1.4 million steps on a single A100 GPU with a batch size of 2048 in approximately four days. The learning rate was held constant at $5 * 10^{-4}$.

A.3 Special Moves

A source and destination square are sufficient to represent all moves that can occur within the rules of chess, with some exceptions. 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 for squares in the last 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. Following the standard in computer chess, en passant captures are encoded as diagonal moves, and castling is encoded as the king moving two spaces horizontally.

```
def sm_bias(x: torch.Tensor) -> torch.Tensor:
    B = x.shape[0]
    y = sm1(x) # (B, 64, d_1)
    y = y.reshape(B, -1)  # (B, 64d_1)
    y = sm_act(sm2(y))  # (B, d_2)
    y = ln1(y)
    y = sm_act(sm3(y))  # (B, H*d_3)
    y = ln2(y).view(B, num_heads, gen_size) # (B, H, gen_size)
    b = torch.einsum("bhi,oi->bho", y, self.posenc_weight)
    return b.view(B, self.num_heads, 64, 64)
```

Figure 3: Torch-like pseudocode for GAB.

A.4 Transcoder Training

For interpretability purposes, we train a cross-layer transcoder on MLP activations collected from layers 3 and 4 (in other words, the 4th and 5th layers) of an earlier checkpoint of Zeus. The transcoder consists of encoders for each layer and decoders going between the two layers (including between each layer and itself), trained on reconstruction and sparsity loss. We train only on layers 3 and 4 because of a) compute constraints and b) our preliminary investigations finding that these layers tended to contain the most interpretable representations. It is common practice to train sparse autoencoders and transcoders on medium-depth layers of models, as these layers are often where the most interpretable representations are found Gurnee et al. (2023).

Let the base transformer have L_{base} layers, and let $S = \{\ell_0 < \ell_1 < \cdots < \ell_{K-1}\}$ be a subset of K layers on which we train the transcoder (for example, $S = \{3, 4\}$ when training only between layers 3 and 4). For each $\ell_k \in S$ we denote by $\mathbf{R}^{\text{pre}}_{\ell_k} \in \mathbb{R}^{B\times T\times D}$ the pre-MLP residual stream and by $\mathbf{M}_{\ell_k} \in \mathbb{R}^{B\times T\times D}$ the corresponding MLP output. After per-layer standardization over batch and tokens, we write

$$\mathbf{X}_k \in \mathbb{R}^{B\times T\times D}, \quad \mathbf{Y}_k \in \mathbb{R}^{B\times T\times D}, \quad k = 0, \ldots, K-1,$$

for the normalized inputs and targets. The transcoder operates on the index set $\{0, \ldots, K-1\}$, with indices i, j referring to layers $\ell_i, \ell_j \in S$.

$$\text{Encoder:} \quad \mathbf{Z}_i = \mathbf{X}_i\, \mathbf{W}_i^{(e)} + \mathbf{b}_i^{(e)} \in \mathbb{R}^{B\times T\times M}, \tag{1}$$

$$\mathbf{A}_i = \mathrm{ReLU}(\mathbf{Z}_i - \boldsymbol{\tau}_i) \in \mathbb{R}^{B\times T\times M}, \tag{2}$$

where $\mathbf{W}_i^{(e)} \in \mathbb{R}^{D\times M}$, $\mathbf{b}_i^{(e)} \in \mathbb{R}^M$, and $\boldsymbol{\tau}_i \in \mathbb{R}^M$ is a learned threshold broadcast over batch and tokens.

$$\text{Decoders (for all } 0 \le i \le j < K\text{):} \quad \widehat{\mathbf{Y}}_{i\to j} = \mathbf{A}_i\, \mathbf{W}_{i\to j}^{(d)} + \mathbf{b}_{i\to j}^{(d)} \ \in \ \mathbb{R}^{B\times T\times D}, \tag{3}$$

$$\widehat{\mathbf{Y}}_j = \sum_{i=0}^{j} \widehat{\mathbf{Y}}_{i\to j}, \tag{4}$$

where $\mathbf{W}_{i\to j}^{(d)} \in \mathbb{R}^{M\times D}$ and $\mathbf{b}_{i\to j}^{(d)} \in \mathbb{R}^D$.

$$\text{Reconstruction loss:} \quad \ell_j^{\text{MSE}} = \frac{1}{BTD} \left\| \widehat{\mathbf{Y}}_j - \mathbf{Y}_j \right\|_2^2, \tag{5}$$

$$\mathcal{L}_{\text{recon}} = \frac{1}{K} \sum_{j=0}^{K-1} \ell_j^{\text{MSE}}. \tag{6}$$

$$\text{Decoder-weighted sparsity penalty:} \quad \pi_{i,f} = \frac{1}{K-i} \sum_{j=i}^{K-1} \left\| \mathbf{W}_{i \to j}^{(d)}[:, f] \right\|_2, \quad f = 1, \ldots, M, \tag{7}$$

$$s_i = \frac{1}{BTM} \sum_{b=1}^{B} \sum_{t=1}^{T} \sum_{f=1}^{M} \tanh\left(c\, \pi_{i,f}\, (\mathbf{A}_i)_{b,t,f}\right), \tag{8}$$

$$\mathcal{L}_{\text{sparse}} = \lambda \cdot \left(\frac{1}{K} \sum_{i=0}^{K-1} s_i \right), \tag{9}$$

where $\lambda > 0$ and $c > 0$ are scalar hyperparameters.

$$\text{Total loss:} \quad \mathcal{L}_{\text{total}} = \mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{sparse}}. \tag{10}$$

We use a batch size of 22 games, a learning rate of 5e-5, and an expansion factor of 8. For training data, we use blitz games from lichess played during July 2019, filtered in the exact same way as in our base model training pipeline. We use the same data to sample the top activating tokens for each feature. At the end of training, our transcoder achieves a reconstruction MSE of 1.6% and a sparsity (ρ_0) of .90

B Implementation Details for Apollo Tournament

Table 7: Tournament performance of Apollo-CF Chessformer against Apollo-CNN convolution model. The time control is chosen so that N is roughly the number of playouts the Apollo-CNN model performs during each game. Elo values are reported with 95% confidence intervals.

N	**Elo Gain**	**Wins**	**Losses**	**Draws**
160k	112 ± 7	846	223	931
320k	111 ± 7	806	186	1008
640k	105 ± 7	779	190	1031

To gauge the impact of Chessformer on raw engine strength, we perform a tournament between versions of the Apollo chess engine configured with either the Apollo-CNN model, a 195-million-parameter convolution-based model previously used by Apollo at tournaments, or Apollo-CF, a 191-million parameter Chessformer. Because of the high computational load of long engine analyses, we use the distributed testing framework OpenBench (Grant) to run 2000 games between these configurations, so that each of 8 RTX 4090 and 4 A100 GPUs runs a single game at a time with engines alternating use of the hardware. Games are played in pairs starting from positions sampled from *UHO_Lichess_4852_v1.epd*, a book of 2.6 million unbalanced human openings curated by the Stockfish project.

To calculate the base time control T for each GPU, we benchmark the speed of Apollo-CNN and set T to an estimate of the amount of time the GPU would take to perform N MCTS playouts, varying the value of N to determine the effect of thinking time on the performance difference. The increment is set to $T/100$, and Apollo was free to use its time according to its time management algorithm. In other words, N is an estimate of the total amount of playouts the Apollo-CNN configuration performs over the game. On average, the speed of Apollo-CNN on these GPUs is approximately 20,000 playouts per second, so $T \approx N/20000$ seconds. Calculating the time control dynamically for each worker to adjust for variations in processing power is standard in modern distributed engine testing frameworks like Stockfish's Fishtest (Stockfish Team).

C Positional Encoding Baselines

Absolute Position Embeddings Perhaps the simplest choice of position encoding is the absolute position embedding, which consists of adding a learned embedding to each token and was notably used by GPT-2 (Radford et al., 2019). Formally, given a sequence of token embeddings $\mathbf{x}_1, \ldots \mathbf{x}_n$, one applies

$$\mathbf{x}_i \mapsto \mathbf{x}_i + \mathbf{c}_i \tag{11}$$

prior to the transformer sublayers.

Relative Position Biases Unlike absolute position embeddings, relative position encodings model positional information based on the relative displacement between tokens. One simple variant introduces relative biases f_k which are added to the attention logits:

$$e_{ij} = \frac{(\mathbf{x}_i W^Q)(\mathbf{x}_j W^K)^T}{\sqrt{d}} + f_{i-j} \tag{12}$$

We consider the two-dimensional analog of this technique, where a square on the chessboard is assigned coordinates (i, j), with i and j ranging from 0 to 7. The bias for querying square (i_1, j_1) and key square (i_2, j_2) is thus $f_{(i_2-i_1, j_2-j_1)}$, where $f_{a,b}$ is defined for $-7 \leq a, b \leq 7$. This adds 15×15 parameters per attention head.

D Tokenization

A number of tokenization schemes have been proposed for chess. We review some of these and attempt to give insight into why our recipe, a square-based representation with a strong position encoding, significantly outperforms them.

The Maia-2 architecture consists of a series of convolution blocks operating on a square-based visual board representation, followed by self-attention layers that process the depth-64 output channels as tokens. Though an interesting design choice, this does not align with the standard, well-tested recipe of transformers in domains like vision and language, which tokenizes inputs by partitioning them in space rather than through internal model representations.

Move-token formulations like Allie, on the other hand, rely on the established methodology of language modeling but lack another vital property: *specialization*. Processing inputs in parallel should allow a model to "divide and conquer", so that the overall computation is split into units that are processed with the same parameters. However, there is emerging evidence that move-token models do not specialize effectively and instead simply reconstruct the board state at each token (Mei, 2025), (Karvonen, 2024). It is also not intuitively clear why a trajectory-based representation should be natural in the Markovian domain of chess.

E Additional Analysis for Human Emulation

Here we provide additional analysis on our human emulation results. We first ablate n, the number of past positions concatenated to the current one to form the input, at the 5M scale. Interestingly, as shown in Table 8, there is a large increase in performance between $n = 0$ and $n = 7$, but no significant difference between $n = 7$ and $n = 31$. The improvement is concentrated at low game ratings and decreases steadily up to high ratings. This contrasts with the effect of model size on move-matching performance, shown in Figure 5 and Figure 6. The impact of scale on modeling performance is several times higher for strong play as it is for weak play.

To ensure that positions in which no or little history information is available remain in-distribution, we mask out during training a uniformly random amount of history information with probability 5%. In initial experiments, this was found to negligibly affect performance ($< 0.05\%$ reduction in move-matching accuracy) relative to always retaining all n past board states.

Despite marked improvements in parameter and compute efficiency, ZEUS-79M improves on the state-of-the-art move-matching accuracy by only 1.2%. We postulate that despite Chessformer's modeling capability, it runs into a performance ceiling at low and intermediate skill levels. Play at these skill levels is unsophisticated and easy to model, but inconsistent and stochastic enough that the accuracy appears to saturate at around 50%. ZEUS-79M shines however at modeling highly skilled play, advancing the searchless state of the art by up to 5% for very strong play. Prior work has struggled to emulate strong play, often relying on search to shore up weak human-aligned models. That ZEUS not only outperforms even search-enabled methods but achieves its largest gains at these very high skill levels suggests that our methodology jointly enables both human alignment and mastery.

Table 8: Move-matching accuracy by history length for human emulation. The value reported is n, the number of past positions excluding the current one fed inputted into the model.

History	Accuracy (%)
0	54.0 ± 0.1
7	$\mathbf{55.4 \pm 0.1}$
31	55.4 ± 0.1

To understand why this is the case, we decompose the error for human emulation into aleatoric uncertainty, the amount of uncertainty that is inherent to the task, and epistemic uncertainty, the amount of uncertainty that can be reduced through stronger models. Low-rated play tends to be unsophisticated, reducing the amount of improvement available from better modeling approaches, but stochastic and inconsistent, reducing the ceiling on predictability. In other words, it has high aleatoric uncertainty, explaining the low accuracies for these players but low epistemic uncertainty, explaining the minor gains provided by scale. In contrast, highly skilled play is more accurate and thus more consistent, but more difficult to predict due its sophistication; it has low aleatoric uncertainty but high epistemic uncertainty, giving plenty of room for improvement. History information likely improves move-matching performance by providing information about the player and reducing the aleatoric uncertainty of the task, particularly for low-rated play where it is higher.

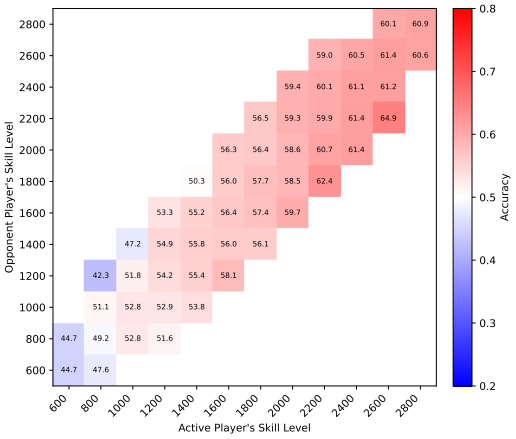

Figure 4: Move-matching accuracies of ZEUS for pairs of skill levels. Interestingly, players are more predictable when they are paired against weaker opponents. This figure uses the ALLIE-AUGMENTED test set, described in Appendix A.1

With this in mind, we believe that future human emulation work should focus on highly skilled play, where the performance ceiling appears to be much higher. One interesting question is maximum accuracy on this task increases monotonically with the game rating, aligning with our intuition about the consistency of strong players, or drop off for ratings above 2600 in line with our observed performance.

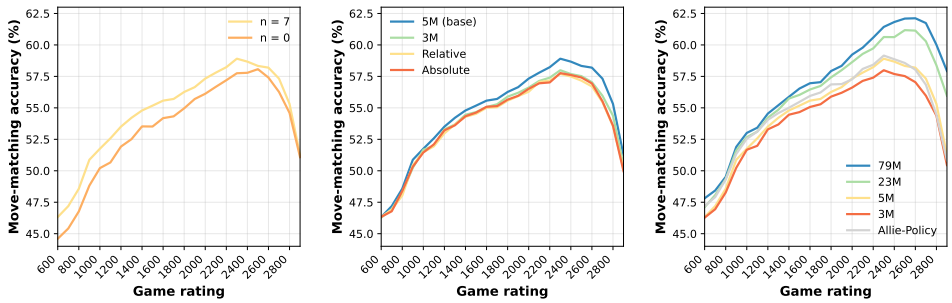

Figure 5: Human move-matching accuracies on the ALLIE-AUGMENTED test set by number of history positions n (left), position encoding (middle), and scale (right). History information helps most for weaker play, while scale and effective position encodings have a large effect for stronger play. We omit results for $n = 31$ history positions as they are virtually identical to those for $n = 7$, and also omit ALLIE-ADAPTIVE-SEARCH due to compute constraints. For all game ratings other than 2900, the margin of error is less than a percent.

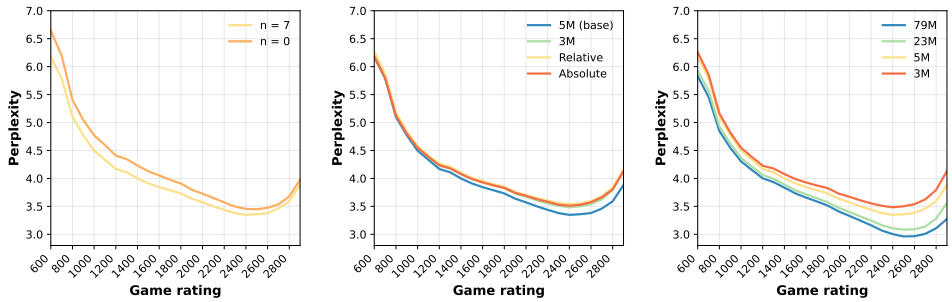

Figure 6: Human move-matching perplexity on the ALLIE-AUGMENTED test set by number of history positions n (left), position encoding (middle), and scale (right). We omit results for $n = 31$ history positions as they are virtually identical to those for $n = 7$.

F Additional Results

F.1 Top Activated Tokens for Transcoder

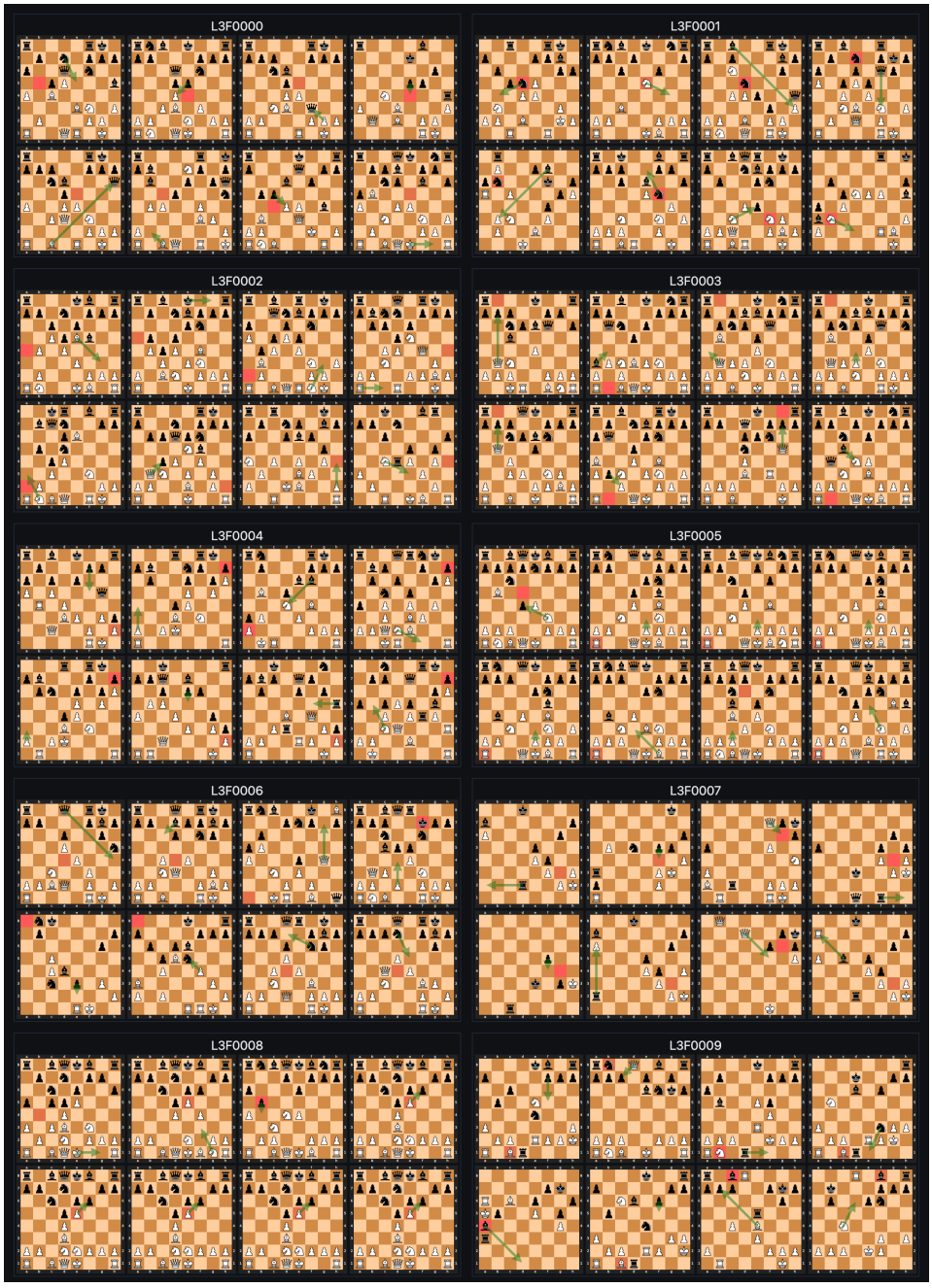

Figure 7: **Annotations for features 0-9 of layer 3.** L3F0000: Square that the active player can advance a pawn to in order to attack an enemy bishop. L3F0001: Active player's knight, usually under attack. L3F0002: Square on the side of the board that is controlled by the active player's rook or queen. L3F0003: Vacant square adjacent to a rook in the corner. L3F0004: An enemy pawn in the corner, in front of the active player's pawn, sheltering the opponent's king. L3F0005: Either a rook in the corner or a center pawn movement option. L3F0006: Not interpretable. L3F0007: A square diagonally adjacent to the opponent's king that is controlled by the active player's pawn. L3F0008: A square contested by both friendly and enemy pawns. L3F0009: Enemy minor piece on the edge of the board pinned to an enemy rook.

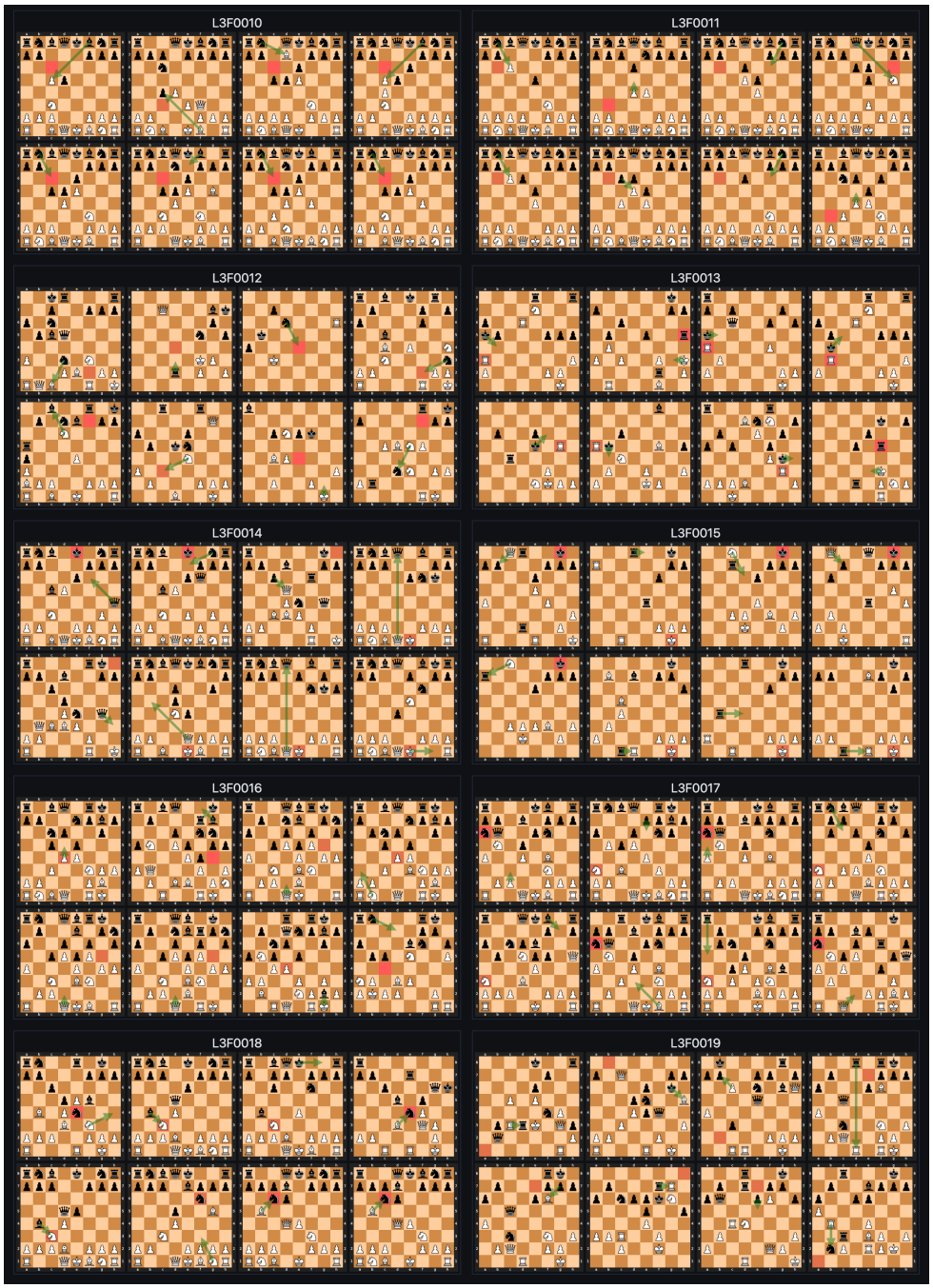

Figure 7: **Annotations for features 10-19 of layer 3 of ZEUS:** L3F0010: Queenside activation square for active player's knight in Queen's gambit structures. L3F0011: Square on b3, b6, f3, or f6 in the opening that have been weakened by the lack of a supporting pawn. L3F0012: Square that the active player's knight can move to to give check. L3F0013: Enemy rook checking the active player's king. L3F0014: Active player's king on the starting position, or an adjacent square if the king has castled. L3F0015: Enemy king in danger of being checkmated on the back rank. L3F0016: Square controlled by both friendly and enemy pawns. L3F0017: Enemy knight developed on the side of the board. L30018: Enemy knight attacked by active player's bishop and defended by enemy bishop. L30019: Not interpretable.

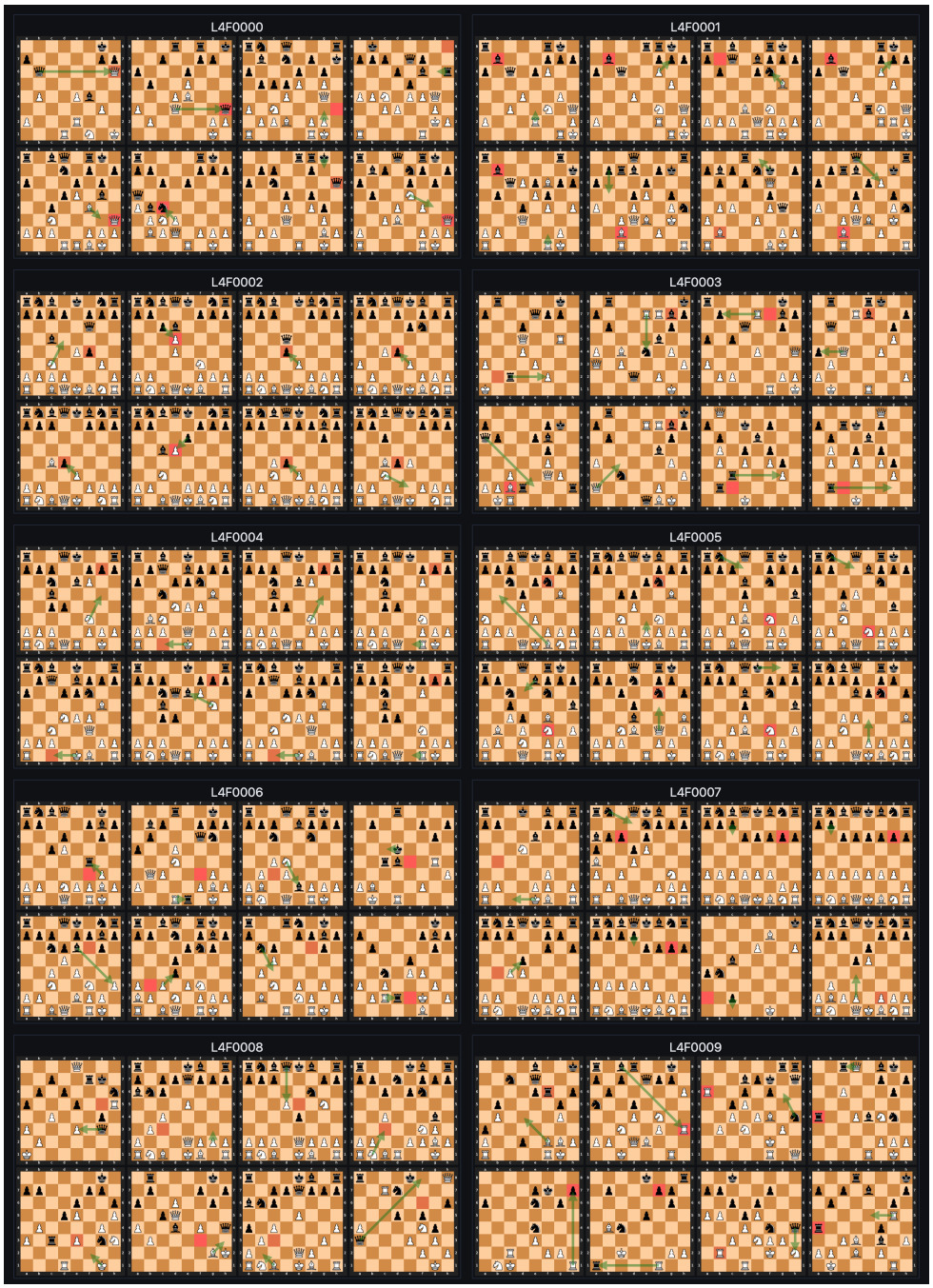

Figure 8: **Annotations for features 0-9 of layer 4 of ZEUS:** L4F0000: Not interpretable L4F0001: Active player's bishop on a strong diagonal, often paired up with a queen. L4F0002: Enemy center pawn targeted for capture in the opening. L4F0003: Square deep in opponent's territory attacked either by two rooks or a rook and a queen. L4F0004: Either long castling or tension between active player's f6 pawn and opponent's g7 pawn. L4F0005: Enemy knight pinned by the active player's bishop. L4F0006: Usually a square controlled by the active player's bishop. L4F0007: Not interpretable.

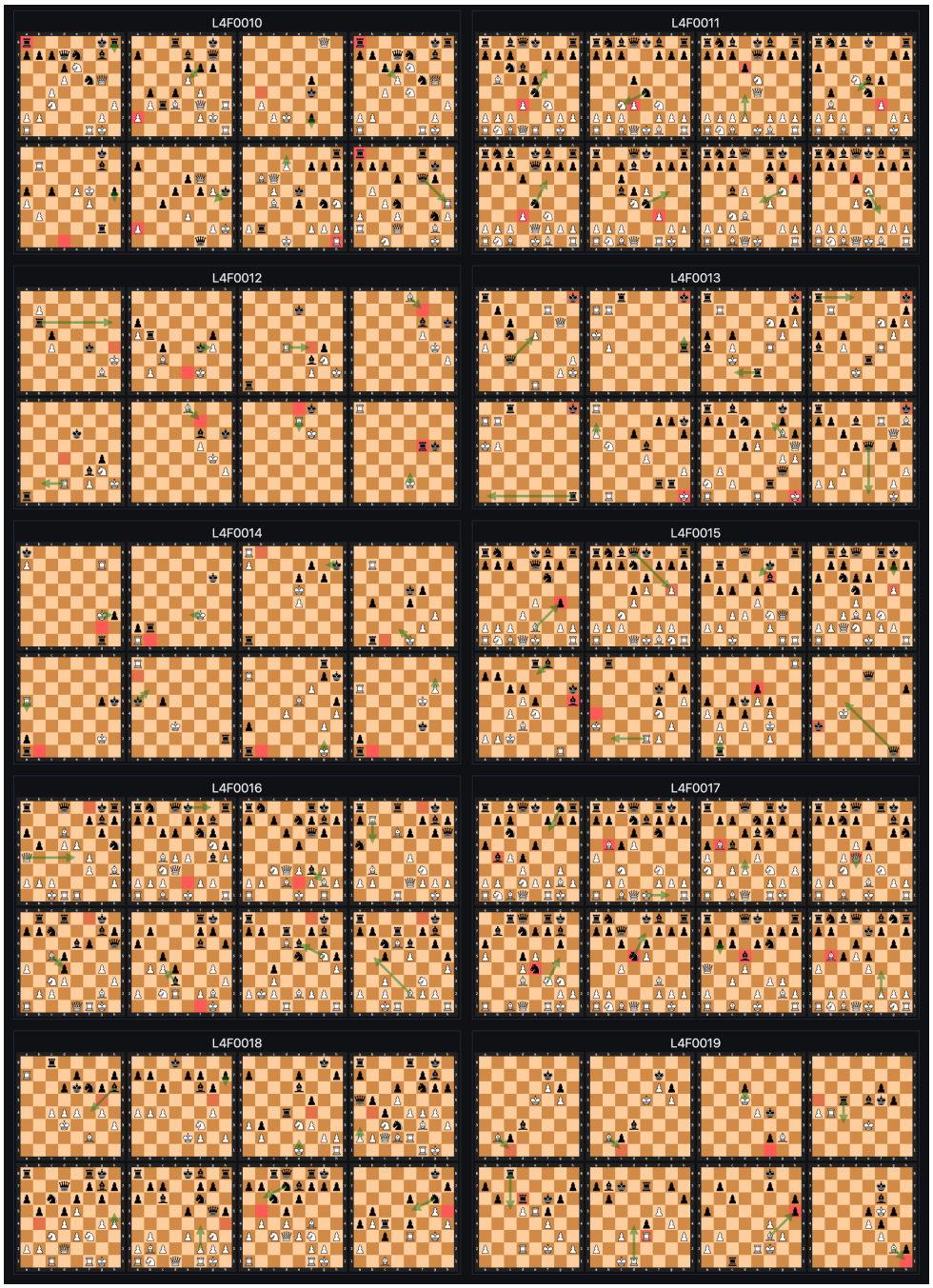

Figure 8: **Annotations for features 10-19 of layer 4 of ZEUS:** L4F0010: Not interpretable. L4F0011: Enemy pawn attacking or threatening to attack an active player's minor piece L4F0012: Not fully interpretable; miscellaneous key squares in endgames. L4F0013: Active player's vulnerable king in the corner.L4F0014: Square that is or will be controlled by enemy pawn, especially if it is close to promotion. L4F0015: Not interpretable. L4F0016: Square deep in opponent's territory controlled by active player's bishop. L4F0017: Active player's centralized piece in the middlegame. L4F0018: Key target square for enemy pawn push. L4F0019: Blockading square for enemy pawn.

F.2 Additional GAB and Attention Head Heat Maps

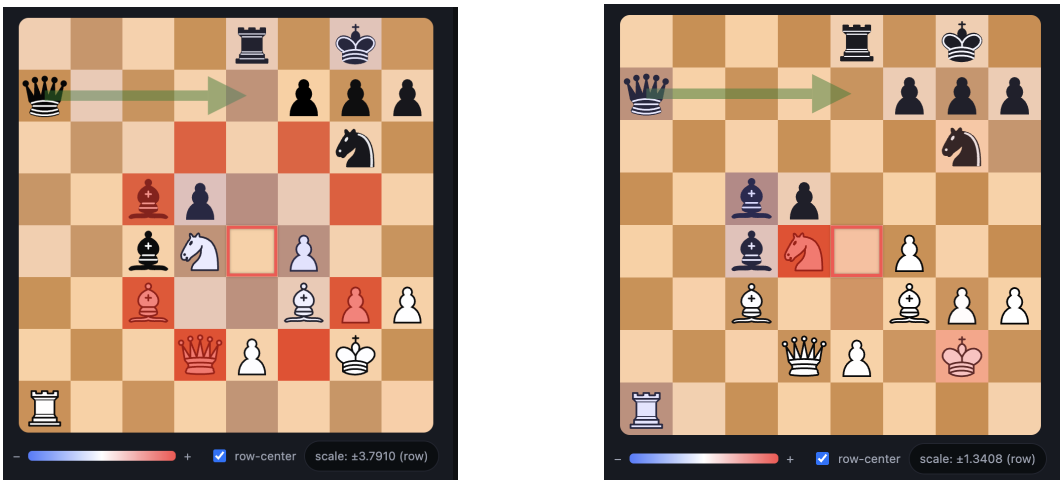

Figure 9: Layer 4 head 4 Zeus GAB and DPA maps, left and right respectively

Figure 10: Layer 4 head 5 Zeus GAB and DPA maps, left and right respectively

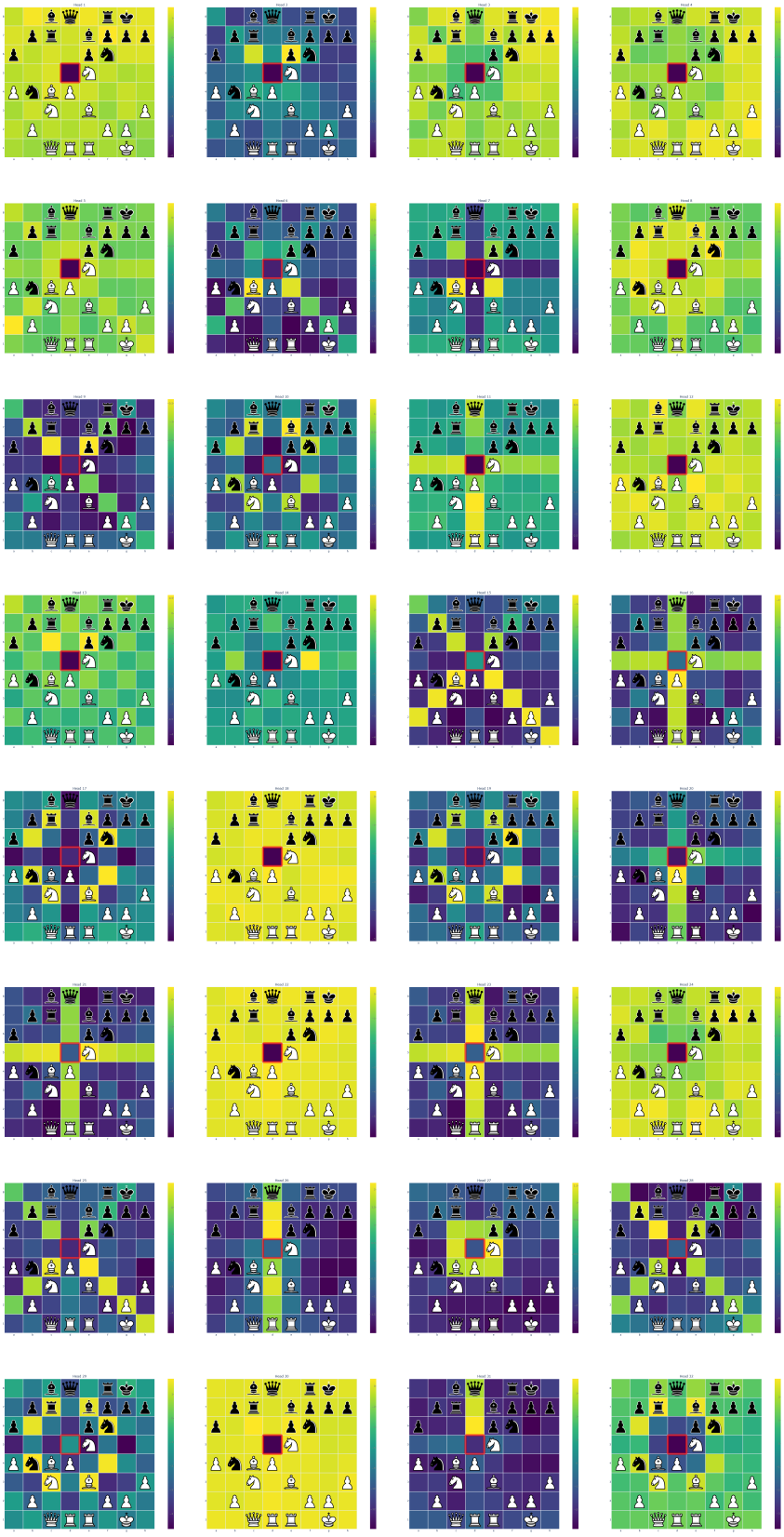

Figure 11: Additional Apollo GAB maps from layer 3

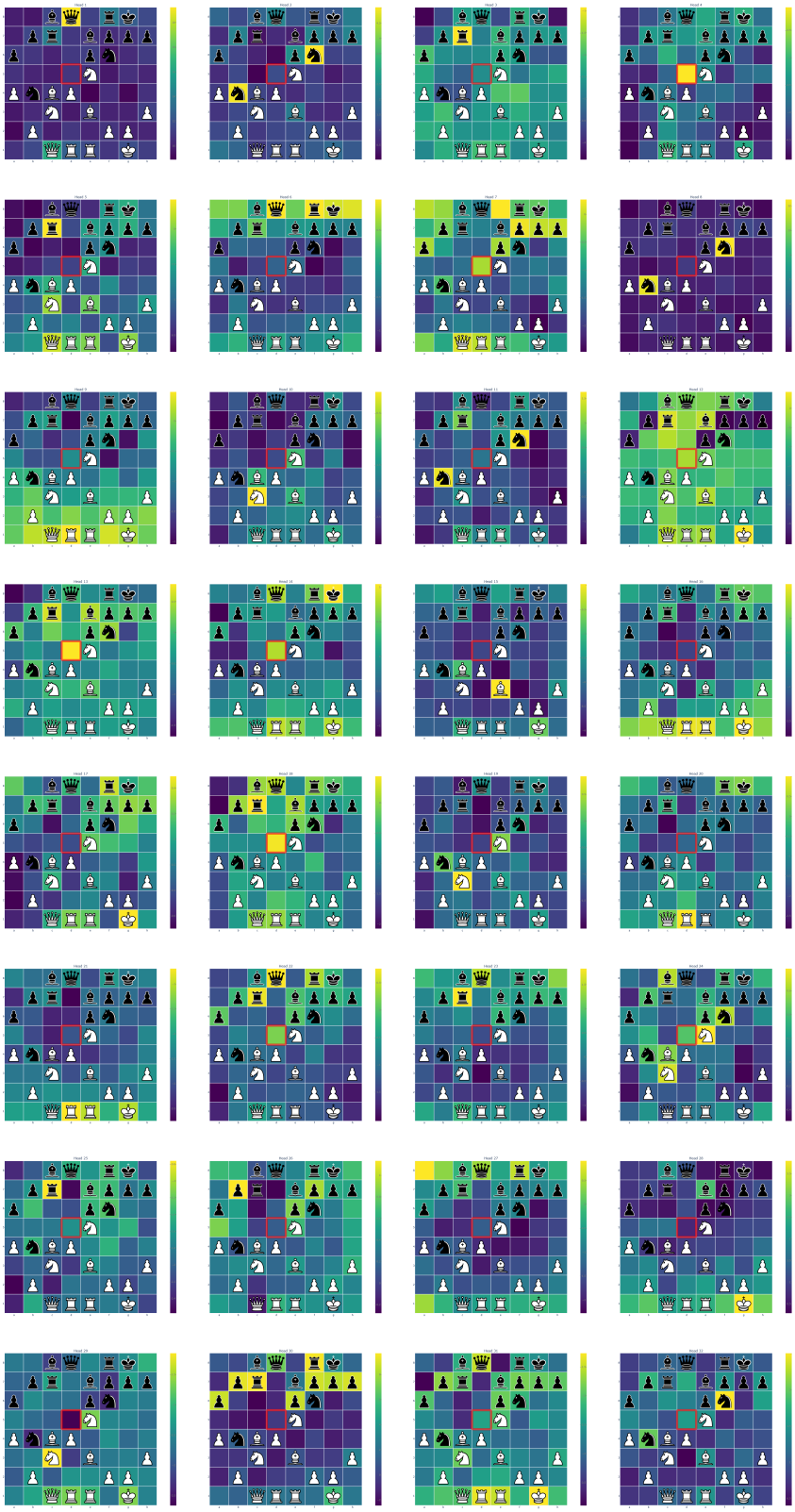

Figure 12: Additional Apollo DPA maps from layer 3