Understanding How Chess-Playing Language Models Compute Linear Board Representations

Aaron Mei [1]

Abstract

The field of mechanistic interpretability seeks to understand the internal workings of neural networks, particularly language models. While previous research has demonstrated that language models trained on games can develop linear board representations, the mechanisms by which these representations arise are unknown. This work investigates the internal workings of a GPT-2 style transformer trained on chess PGNs, and proposes an algorithm for how the model computes the board state.

1. Introduction

In recent years, board games have emerged as interpretable testbeds to study language model capabilities (Ruoss et al., 2024; Topsakal et al., 2024; Toshniwal et al., 2021). In particular, there have been many reports of language models possessing linear board state representations in games like chess, checkers, and othello (Li et al., 2023; Joshi et al., 2024). Notably, Karvonen (2024) trained a GPT-2 style transformer on chess PGNs and was able to show that the model contained a linearly decodable representation of the board.

However, the process by which the transformer constructs this representation remains unclear. Recent interpretability efforts, such as those by Davis and Sukthankar (2024), analyze attention patterns and identify where the model begins to commit to its next move, but stop short of examining how the model internally represents the board. In this work, we aim to elucidate a chess-playing language model's board reconstruction mechanism.

2. Setup and Background

2.1. Model and Data

We examine an 8-layered GPT model trained by Karvonen on 16 million human chess games. The model totals 25 million parameters with a 512-dimension hidden space and 8 heads per layer. The model input is a string of chess moves formatted as a PGN. Each game is preceded by a ';' token, and the model's vocabulary is restricted to the 32 characters required for chess notation. The model was solely trained to autoregressively generate the next character of the input.

The dataset we use is also obtained from Karvonen and consists of games from the Lichess open database. During probing and analysis, we use a set of 10,000 games not found in the model's training data. All games in this set are truncated to a length of 365 tokens, as this was the median length of a game.

2.2. Linear Probing

We train linear probes to analyze the existence of certain features in the model (Alain & Bengio, 2018; Belinkov, 2022; Nanda & Bloom, 2022). While a model is generating the next token, we can use its intermediate representation (e.g., the residual stream at a layer or the output from an attention head) to predict the desired feature (Rai et al., 2024). Since a probe has minimal predictive power of its own, an accurate probe is evidence that the model keeps a linearly encoded

[1]School of Engineering and Applied Science, University of Pennsylvania, Philadelphia, USA. Correspondence to: Aaron Mei <meiaaron99@gmail.com>.

Accepted at Methods and Opportunities at Small Scale (MOSS), ICML 2025, Vancouver, Canada.

representation of the feature. More concretely, given a vector $\mathbf{h} \in \mathbb{R}^d$ computed by the model during prediction and a target latent variable $y \in \mathbb{R}^C$, we learn weights $W \in \mathbb{R}^{C \times d}$ to minimize the loss between $W\mathbf{h}$ and y. We train probes with the AdamW optimizer and our hyper-parameters are reported in Appendix A.

3. Board Reconstruction

3.1. Linear Board Representation

Karvonen reports that the model possesses a linear representation of the board state, and we begin by reproducing those results. We follow their methods to train a linear probe to classify each square of the chess board as one of 13 possible states: blank, or one of six piece types (pawn, knight, bishop, rook, queen, king), each of which can be white or black. Then, the linear probe for square i at layer l is:

$$P_{i,j} = \text{Softmax}(W_{i,l} A_l) \tag{1}$$

where $P_{i,j}$ is the probability distribution over the 13 classes for square i at layer l, $W_{i,l}$ is the weight matrix, and A_l is the post-MLP residual stream from layer l. We find that the linear probe accuracy peaks at layer 5 with an accuracy of 99.0%. Crucially, we notice the accuracy jumps from 88.7% to 99.0% between layers 4 and 5. To identify where this improvement arises, we fit a second probe at layer 5 using the pre-MLP residual stream as input. This probe also reaches an accuracy of 99.0%, indicating that the board representation emerges from the attention mechanism. Board probe accuracies for all layers can be found in Appendix B.

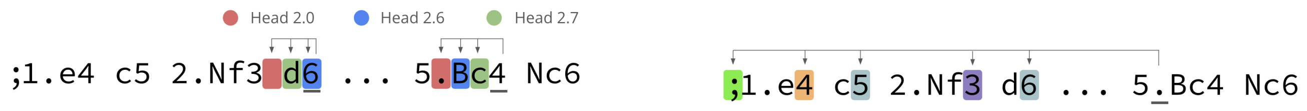
To test the robustness of the model's board construction algorithm, we create a dataset of games by picking random legal moves. We then evaluate the layer 5 probe trained on human games against this synthetic dataset. The probe accuracy remained unchanged, suggesting that the model's reconstruction algorithm generalizes outside of human move sequences.

3.2. Token-Level Encoding

The PGN strings that are input to the model represent game moves in standard algebraic notation. In general, each move is indicated by a letter denoting the piece type followed by the coordinates of the destination square. The coordinates are then further split into a letter a-h representing the file and a number 1-8 denoting the rank. For pawn moves, the letter indicating the piece is omitted (e.g., c5). The PGN move list is serialized as a sequence with the following pattern:

```
<Move-N>. <White-move> <Black-move> <Move-N+1>. <White-move> <Black-move> ...
```

Instead of encoding the (Black, White) color of a piece, it is common for GPTs to adopt a (Mine, Yours) representation. Nanda et. al. (2023) originally show this in othello playing language models, and Karvonen later finds that their chess GPTs exhibit the same pattern. Hence, for simplicity, we perform experiments only from the White perspective, as the opposite side is likely symmetric. For example, when probing for the board state, we use the residual stream vectors at each dot token. The equivalent token from the Black perspective is the whitespace after the white move.


Head 2.0
Head 2.6
Head 2.7
;1.e4 c5 2.Nf3 d6 ... 5.Bc4 Nc6
;1.e4 c5 2.Nf3 d6 ... 5.Bc4 Nc6

(a) (b)

Figure 1. Attention maps for different layers, with the underlined tokens representing the targets: (a) In layer 2, information about the color, piece type, and file is moved to the rank token. (b) Once piece information has been aggregated in the rank token, heads in later layers primarily attend to the move's rank token and the start of game delimiter with some exceptions (e.g., captures).

Since the rank number is the last token of a move, it is the prime candidate for collecting the information in a move. Indeed, in layer 2, we find (Cooney & Nanda, 2023) attention heads for reading move information (Figure 1a). After all of the move information has been synthesized into the rank token, heads in following layers primarily attend to these rank tokens (Figure 1b). Thus, when we refer to the token for a certain move, we specifically refer to the rank token of the move.

3.3. Previous Move Heads

As algebraic notation does not usually include information about the square the piece moved from, the model must infer this information from context. We find two heads that facilitate this; Head 3.2 specializes in finding previous moves for knights, bishops and kings, and Head 4.0 is a generic previous move head. That is, for moves in which the piece has previously moved, the rank token position attends to the rank token position of its previous move. To quantify this, we compare the true previous move token to the token with the maximal attention coefficient. The resulting retrieval accuracies are summarized in Appendix C.

3.4. Pawn Average Heads

From the attention map we observe that Head 5.1 attends primarily to pawn moves, so we search for a linear representation of pawn structure in this head's output.

Head value vector. For each pawn move t, let $\mathrm{from}(t)$ and $\mathrm{to}(t)$ be its origin and destination squares. Write $v_s \in \mathbb{R}^d$ for the learned vector of square s. The pre-projection value vector $z_t = W^V x_t$ computed by Head 5.1 empirically satisfies

$$z_t \;\approx\; -\,v_{\mathrm{from}(t)} \;+\; v_{\mathrm{to}(t)}. \tag{2}$$

Averaging. At move T[1], taking the average of z_t over all pawn moves $t \le T$ yields

$$Z_T \;=\; \frac{1}{n}\sum_{t\le T} z_t \;=\; \frac{1}{n}\Big(-\sum_{t\le T} v_{\mathrm{from}(t)} \;+\; \sum_{t\le T} v_{\mathrm{to}(t)}\Big),$$

where n is the number of pawn moves up to T. Tracking any fixed pawn p, every square (other than its initial square) that p moves *from* must previously have been a square it moved *to*. Thus the two sums nearly cancel, leaving

$$\frac{1}{n}\big(v_{\mathrm{final}(p)} - v_{\mathrm{initial}(p)}\big).$$

Summing over all moved pawns P yields

$$Z_T \;=\; \frac{1}{n}\Big(\sum_{p\in P} v_{\mathrm{final}(p)} \;-\; \sum_{p\in P} v_{\mathrm{initial}(p)}\Big). \tag{3}$$

Unfortunately, equation (3) doesn't give us a completely clean linear representation of the pawn structure, which would be $\sum v_{\text{final}(p)}$. How the model deals with the $1/n$ scaling and initial pawn positions is outside the scope of this work, but we offer some discussion in Appendix E.

Captures. When the opponent captures a pawn, the head attends to the dot token following the capturing move. Then, this capture token t can erase the pawn on the square by letting $z_t = -v_{\text{capture square}}$. In our analysis, we treat these captures as moves and include it in the Z_T sum.

3.4.1. Empirical Support

We provide a full analysis in Appendix D, but summarize the main points here:

- The computed average Z_T has a cosine similarity of ~ 0.9 with the true attention-weighted pawn move contributions to Head 5.1. This exceeds the cosine similarity of ~ 0.44 obtained by using only the last pawn move.
- A linear probe trained to identify the to and from squares of the value vector achieves an accuracy of 99.7%. We also find that negating the value vector z_t swaps the to and from squares. Finally, using just the to and from squares, we can reconstruct z_t with an R^2 of 0.956, confirming that $z_t \approx -v_{\mathrm{from}} + v_{\mathrm{to}}$ captures nearly all the signal.
- A square classification probe, similar to the board probe from equation (1), trained on Z_T achieves a 99.5% accuracy (baseline $\sim 91\%$). Applying the same probe to the true head contribution achieves 98.9%, indicating that the model's own computation encodes the pawn structure in the same way as Z_T.

[1]When we say "move T," we generally refer to the dot token following that move. For example, in the PGN string `1.e4 c5 2.Nf3 Nc6 3.Nc3 e5 4.Bc4 d6 5.d3 Be7 6.Nd5`, the T that contains all pawn moves up to and including `d3` is the index of the dot following `6`.

3.5. Exponential Moving Average Heads

In Layer 5, several heads cooperate to reconstruct the positions of the remaining pieces (except the king). Although a simple average (similar to the pawn-average head) would in principle work, empirical evidence suggests the model opts for a exponential moving average (EMA) strategy. We begin by fixing a piece type (e.g, knights).

Head value vector. For every knight move t, let $\mathrm{from}(t)$ and $\mathrm{to}(t)$ be its origin and destination squares, and let $v_s \in \mathbb{R}^d$ denote the learned square vector s for knights. We claim that the pre-projection value vector $z_t = W^V x_t$ computed by Head 5.6 satisfies:

$$z_t \;\approx\; -\alpha\, v_{\mathrm{from}(t)} \;+\; v_{\mathrm{to}(t)} \;+\; u_t, \tag{4}$$

where $\alpha \in (0,1)$ is a fixed decay factor and u_t carries auxiliary information.

Exponential moving average. Define $Z_0 = \mathbf{0}$ and update for each move T:

$$Z_T \;=\; \alpha\, Z_{T-1} + (1-\alpha)\, z_T. \tag{5}$$

Tracking a *single* knight over n moves, note that $v_{\mathrm{from}(t_k)} = v_{\mathrm{to}(t_{k-1})}$, so these terms cancel telescopically, leaving

$$\begin{aligned} Z_T &= (1-\alpha)\, v_{\mathrm{final}} \;-\; (1-\alpha)\, \alpha^{\,n-1}\, v_{\mathrm{initial}} + u \\ &\approx (1-\alpha)\, v_{\mathrm{final}} + u, \end{aligned}$$

with $u = \sum_t u_t$. Because $\alpha^{\,n-1}$ decays exponentially, the initial-position vector vanishes, so Z_T encodes the knight's current square. For a capture, we treat $\mathrm{from}(t)$ as the capture square and set $v_{\mathrm{to}(t)} = \mathbf{0}$.

Persisting companion pieces. The above logic works directly for "unique" pieces[2] (e.g., dark-squared bishop, light-squared bishop). For pairs such as knights or rooks, we posit that the head refreshes the companion piece via

$$u_t \;=\; (1-\alpha)\, v_{\mathrm{companion}(t)}, \tag{6}$$

where $\mathrm{companion}(t)$ is the square of the other knight/rook (or $u_t = \mathbf{0}$ if that piece has been captured). Because $\alpha(1-\alpha) + (1-\alpha)^2 = (1-\alpha)$, the companion's vector is effectively renewed. Therefore, up to normalization, we are left with a linear representation of both the piece's own position and that of its symmetric partner.

3.5.1. Empirical Support

We provide a full analysis in Appendix F, but summarize the main results here:

- We perform a grid search over the decay factor α and empirically pick the α that maximizes the cosine similarity with H_T. We obtain cosine similarities of at least 0.93 with most pieces having similarities over 0.97.
- Linear probes trained to identify the to and from squares of the value vector achieve high accuracies, and negating the value vector z_t swaps the to and from squares.
- Square classification probes attain high accuracies, and the performance persists when the same probes are applied to the true head contributions.
- Probes trained to extract the position of the companion knight/rook achieve accuracies of ~91% and ~80% respectively.

4. Conclusion

In this work, we explore how language models can compute linear representations of chess board states. We uncover key components of this process and identify the attention heads that implement them. Our central findings are the average and EMA mechanisms for accumulating a piece's move history. Combining these, we get a full board reconstruction algorithm.

Outside of some edge cases, the proposed strategies for board reconstruction are quite robust. This shows that a transformer without a priori knowledge of chess rules can develop an imperfect but logical algorithm to track the board state. Overall, we hope these findings can contribute to our understanding of how models navigate and develop heuristics in restricted domains.

[2]Although we treat the queen as a "unique" piece here, promotion to a queen is common which may require the model to track multiple queens. Investigating this possibility is left for future work.

5. Acknowledgments

We would like to thank Surbhi Goel for her guidance and for the insightful discussions on this project.

References

Alain, G. and Bengio, Y. Understanding intermediate layers using linear classifier probes, 2018. URL https://arxiv.org/abs/1610.01644.

Belinkov, Y. Probing classifiers: Promises, shortcomings, and advances. *Computational Linguistics*, 48(1):207–219, March 2022. doi: 10.1162/coli_a_00422. URL https://aclanthology.org/2022.cl-1.7/.

Cooney, A. and Nanda, N. Circuitsvis. https://github.com/TransformerLensOrg/CircuitsVis, 2023.

Davis, A. L. and Sukthankar, G. Decoding chess mastery: A mechanistic analysis of a chess language transformer model. In *Artificial General Intelligence: 17th International Conference, AGI 2024, Seattle, WA, USA, August 13–16, 2024, Proceedings*, pp. 63–72, Berlin, Heidelberg, 2024. Springer-Verlag. ISBN 978-3-031-65571-5. doi: 10.1007/978-3-031-65572-2_7. URL https://doi.org/10.1007/978-3-031-65572-2_7.

Joshi, A., Sharma, V., and Modi, A. CheckersGPT: Learning world models through language modeling. In Fu, X. and Fleisig, E. (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop)*, pp. 414–426, Bangkok, Thailand, August 2024. Association for Computational Linguistics. ISBN 979-8-89176-097-4. doi: 10.18653/v1/2024.acl-srw.48. URL https://aclanthology.org/2024.acl-srw.48/.

Karvonen, A. Emergent world models and latent variable estimation in chess-playing language models. In *Proceedings of the Conference on Language Modeling (COLM)*, 2024. URL https://openreview.net/forum?id=PPTrmvEnpW. Accepted at COLM 2024.

Li, K., Hopkins, A. K., Bau, D., Viégas, F., Pfister, H., and Wattenberg, M. Emergent world representations: Exploring a sequence model trained on a synthetic task. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=DeG07_TcZvT.

Nanda, N. and Bloom, J. Transformerlens. https://github.com/TransformerLensOrg/TransformerLens, 2022.

Nanda, N., Lee, A., and Wattenberg, M. Emergent linear representations in world models of self-supervised sequence models. In Belinkov, Y., Hao, S., Jumelet, J., Kim, N., McCarthy, A., and Mohebbi, H. (eds.), *Proceedings of the 6th BlackboxNLP Workshop: Analyzing and Interpreting Neural Networks for NLP*, pp. 16–30, Singapore, December 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.blackboxnlp-1.2. URL https://aclanthology.org/2023.blackboxnlp-1.2/.

Rai, D., Zhou, Y., Feng, S., Saparov, A., and Yao, Z. A practical review of mechanistic interpretability for transformer-based language models. *arXiv preprint arXiv:2407.02646*, 2024.

Ruoss, A., Del'etang, G., Medapati, S., Grau-Moya, J., Li, W. K., Catt, E., Reid, J., and Genewein, T. Amortized planning with large-scale transformers: A case study on chess. In *Neural Information Processing Systems*, 2024. URL https://api.semanticscholar.org/CorpusID:267523308.

Topsakal, O., Edell, C. J., and Harper, J. B. Evaluating large language models with grid-based game competitions: An extensible llm benchmark and leaderboard. *ArXiv*, abs/2407.07796, 2024. URL https://api.semanticscholar.org/CorpusID:271088744.

Toshniwal, S., Wiseman, S., Livescu, K., and Gimpel, K. Chess as a testbed for language model state tracking. In *AAAI Conference on Artificial Intelligence*, 2021. URL https://api.semanticscholar.org/CorpusID:248811258.

6. Appendix

A. Hyper-parameters and Training Details

Table 1. Key hyper-parameters for all probes.

Parameter	**Value**
Learning rate	1×10^{-3}
Weight decay	1×10^{-2}
Betas	$(0.9,\ 0.99)$

Table 2. Number of training epochs used for each linear probe by probe type

Probe	**Epochs**
Board	5
Pawn to/from/combined-square	3
Pawn value Reconstruction	5
Pawn board Reconstruction	5
Piece to/from/combined-square	10
Piece board Reconstruction	5
Companion	16

B. Board Probe Accuracies

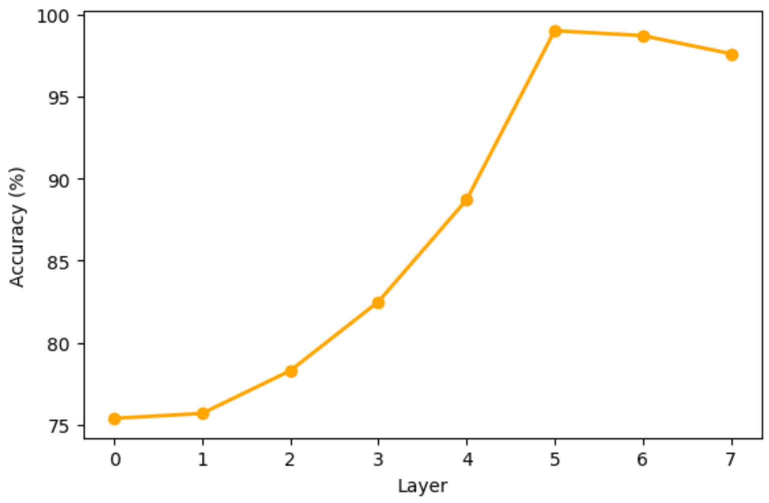


Figure 2. Layer-wise classification accuracy of the board probe. We observe a sharp rise in accuracies from 88% to 99% from layer 4 to layer 5.

C. Previous Move Head Accuracies

The difficulty of obtaining the previous move varies with the piece type. In particular, the model struggles with the rook and pawns. Rooks are difficult because they are symmetric, and their long range leaves many possible origin squares. As for pawns, due to their limited movement, the destination square already carries some information about the origin square. Hence, the model may not need to explicitly find the previous move to infer the from square.

Table 3. Retrieval accuracy of previous-move heads.

Piece type	**Head 3.2**	**Head 4.0**
Knight	0.8250	0.9834
Rook	0.0054	0.8303
Pawn	0.1021	0.5645
Queen	0.4575	0.9127
King	0.6364	0.9557
Bishop	0.9716	0.9991

D. Pawn Head Analysis

D.1. Cosine Similarity Analysis

To see if the model is truly taking an average over the pawn moves t, we compare Z_T to the true head contribution:

$$H_T = \sum_{t \leq T} \alpha_t z_t. \tag{7}$$

where α_t is the attention score for token t at move T. We emphasize that this only includes the contributions from the rank tokens, as they tend to have the largest attention weights. We exclude the delimiter (;) and all other non-rank tokens from the computation. The averaging scheme obtains a cosine similarity of roughly 0.9 with the true head contribution. This beats the baseline of just using the last pawn move's value vector, which obtains a cosine similarity of roughly 0.44.

D.2. Value Vector Analysis

Equation (2) claims that the value vector z is the sum $-v_{\text{from}} + v_{\text{to}}$. Then, a linear probe should be able to extract the to and from squares from z. Let $y_t \in \{0, 1\}^{64}$ be the one-hot encoding of the to-square of the move. We train a linear probe

$$\hat{y}_t = \text{softmax}\big(W\, z_t\big) \tag{8}$$

by minimizing the average cross-entropy loss. Repeating this process for the from squares, we find that both probes achieve an accuracy of over 99.7%, which strongly indicates that z_t encodes both the origin and destination square. Another assumption we make is that the to and from square vectors belong to the same subspace. This is important for cancellation during averaging.

If this is truly the case, negating z_t should reverse the roles of the to and from-squares. Hence, we combine the to and from-square datasets such that for each move t, we include both $(z_t, \text{to square}_t)$ and $(-z_t, \text{from square}_t)$. The probe obtains an accuracy of 99.9%, demonstrating that negating the value vector reliably swaps the to and from labels.

Finally, we want to confirm that the to and from vectors make up most of the information that z_t carries. Thus, we examine whether we can reconstruct the value vector using only the to and from-square information. We create a 128-dimensional input for each pawn move by concatenating the two 64-dimensional one-hot square vectors. Then, we train a linear probe to reconstruct the head value vector using a MSE loss. The reconstruction probe obtains a validation MSE of around 0.19 and yields an R^2 of 0.956.

D.3. Board Reconstruction

We want to verify that the average Z_T is capable of recovering the full pawn structure. Similar to the board probe presented in equation (1), we train a linear probe to classify whether a square is blank or contains a white pawn. This board probe achieves an accuracy of 99.5% (compared to a baseline of $\sim$91% if all squares are guessed to be blank), confirming that the averaging scheme can accurately recover the pawn structure.

Finally, to see if this average scheme is similar to what the model is computing, we can apply the probe (trained on the averages) to the true head contribution. This achieves an accuracy of 98.9%, which suggests that the average and the true contribution encode the pawn structure similarly.

E. Pawn Head Discussion

As mentioned earlier, $Z_T = \frac{1}{n}\left(\sum v_{\text{final}(p)} - \sum v_{\text{initial}(p)}\right)$ is not a perfect representation. We would like to scale by n and add v_{initial} for every single pawn.

We would also like to note that even a linear probe can somewhat overcome these limitations as the board probe could retrieve the pawn structure with relatively high accuracy. However, things get messier when multiple piece types are involved and the dimensions may become overloaded due to superposition.

Scaling One way that the model can remedy the scaling issue is by using the ';' token at the beginning of the game string. When there are fewer pawn moves, it can dump more attention into the ';' token and even out the scaling as pawn moves are made. We anecdotally observe an increase in overall attention on the pawn moves (and less on the ';' token) as pawn moves are made, but have not investigated this rigorously.

Moreover, there are a finite number of pawn moves (48) and not too many pawn moves per game (especially not until the endgame), so it is also possible that not having the exact scale is fine.

Initial Pawn Positions Another question that arises is how the model knows the "initial pawn positions." Since the initial positions are fixed, it is possible that the model could add a constant for each pawn (including unmoved pawns).

However, a linear probe is expressive enough to deal with this issue and adopts an alternative strategy. We train a board probe using averages of the 128-dimension to/from vectors in the value vector reconstruction section. We find that for most to-square dimensions, the probe injects a small positive bias onto all eight starting pawn squares. These small contributions are enough to mark the square as occupied, but when a pawn actually leaves the square, the subtraction term for the from square overwhelms that bias.

In addition, the probe's learned "occupied" and "blank" vectors are almost antipodal, with a cosine similarity of -0.999. Because the probe's task here is binary classification, it can afford to have a large negative direction for blank squares. However, this strategy may not be viable when multiple piece types (and not enough dimensions) are involved.

F. EMA Head Analysis

Table 4 summarizes the mapping between attention heads and their associated pieces.

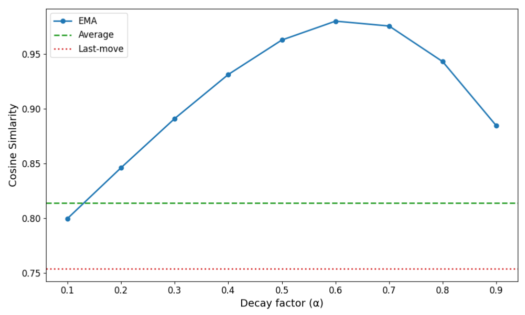
Table 4. Attention head assignments for each piece type. While each piece has a predominant head, sometimes other heads will also track the piece. For example, even though rooks are primarily tracked by Head 5.2, Head 5.6 will occasionally attend to rook moves as well.

Piece	Head
Dark-squared Bishop	5.6
Light-squared Bishop	5.3
Knight	5.6
Queen	5.2
Rook	5.2

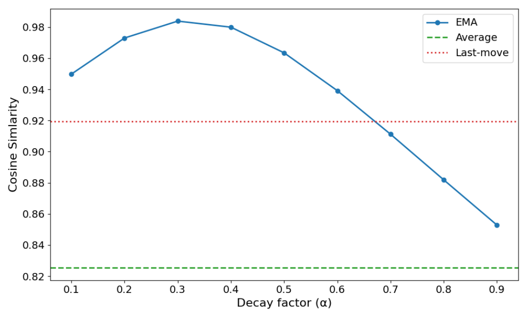
Finding the king's position is done in layer 4 and not 5. Tracking the king should be a relatively easy task due to the piece's uniqueness. Hence, we omit its analysis.

F.1. Cosine Similarity Analysis

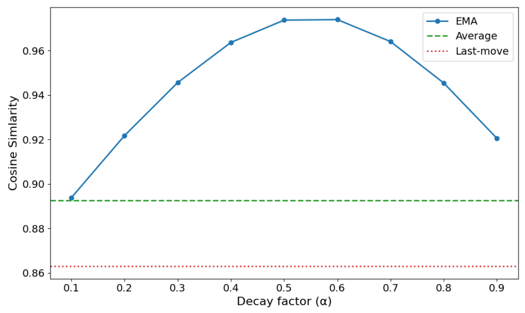
It is theoretically challenging to find the decay factor α. So, for each piece type and corresponding head in Table 4, we perform a grid search over $\alpha \in [0.1, 0.2, ..., 0.9]$ and empirically pick the α that maximizes the cosine similarity with the true contribution H_T. We benchmark the EMA against two baselines: the average of the value vectors, and the the value vector produced by the piece's most recent move. The comparison, presented in Figure 3, shows that the EMA attains markedly higher cosine similarities than either baseline.



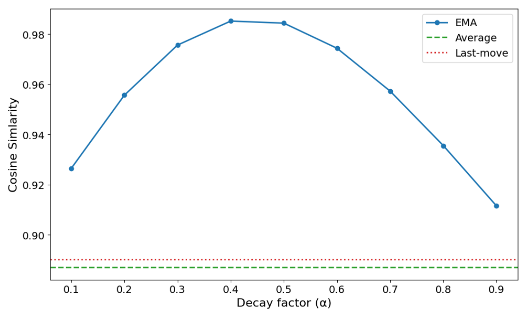
(a) Knight



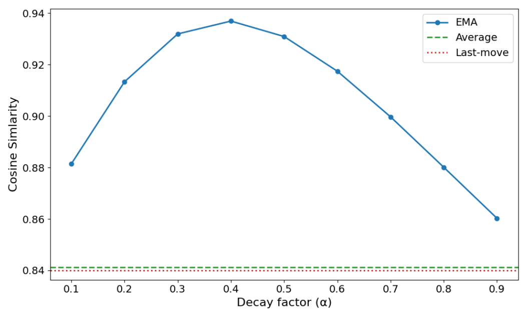
(b) Queen



(c) Rook



(d) Dark-squared Bishop



(e) Light-squared Bishop

Figure 3. Cosine similarity between the true head contribution H_T and three candidate proxies: (i) an exponential moving average proxy with decay α (ii) the average of all value vectors, and (iii) the value vector from the piece's most recent move. The EMA proxy outperforms both baselines, achieving a similarity of at least 0.97 for every piece type except the light-squared bishop (0.93).

F.2. Value Vector Analysis

Equation (4) claims that the value vector z is the sum $-\alpha\, v_{\text{from}(t)} + v_{\text{to}(t)} + u_t$. Once again, a linear probe should be able to extract the to and from squares from z. Furthermore, although the magnitudes of the to and from-square vector contributions differ, we still want to show that they point in opposite directions. Hence, we follow Section D.2 to train the to/from square probes and the combined probe. Table 5 shows that the to and from-square probes achieve high accuracies, confirming that the value vector z linearly encodes the destination and origin squares of a move. Additionally, the combined probe reaches essentially the same accuracy which supports the key requirement that the to and from-square directions reside in a shared subspace but have opposite directions.

F.3. Board Reconstruction

We want to verify that the exponential moving average Z_T (using the decay factors that maximize the cosine similarities) is capable of recovering the piece's positions. For each of the 64 squares, we train a linear probe to classify whether the square is blank or contains the piece. For details, refer to Section D.3. Because the majority of board squares are empty, a naive classifier that always predicts "empty" would still reach an accuracy of about 96.9-98.4%, so raw accuracy alone can

Table 5. Validation accuracies for the from-square, to-square, and combined probes. The high accuracies of the to and from-square probes show that z strongly encodes the destination and origin squares of the move. The high accuracy of the combined probe confirms that the to-square and from-square vector representations lie in the same subspace.

PIECE TYPE	FROM-SQUARE PROBE	TO-SQUARE PROBE	COMBINED PROBE
DARK-SQUARED BISHOP	0.9972	1.0000	0.9998
LIGHT-SQUARED BISHOP	0.9759	1.0000	0.9970
KNIGHT	0.9980	1.0000	0.9989
QUEEN	0.9147	1.0000	0.9680
ROOK	0.9504	0.9950	0.9681

Table 6. Board-reconstruction accuracy for each piece when the probe is driven by (i) the EMA-based vector Z_T with decay α chosen to maximize cosine similarity, and (ii) the true head contribution H_T. Near-perfect performance in both settings indicates that the EMA proxy encodes the same positional information as the model's actual computation.

PIECE	COMPUTED EMA (Z_T)	TRUE CONTRIBUTION (H_T)
DARK-SQUARE BISHOP	1.0000	1.0000
LIGHT-SQUARE BISHOP	0.9999	0.9993
KNIGHT	0.9990	0.9992
QUEEN	0.9999	0.9999
ROOK	0.9982	0.9979

overstate probe performance. Regardless, we obtain near-perfect accuracies, as outlined in Table 6.

Finally, to see if the EMA scheme is similar to what the model is actually computing, we test the probe (trained on the averages) on the true head H_T. As shown in Table 6, there is almost no drop in accuracy, which suggests that the EMA and the true contribution encode the piece positions similarly.

F.4. Companion Pieces

For symmetric pieces like knights and rooks, we hypothesize that when one of the pieces is moved, the position of the other piece is renewed. To test this, we train a probe to examine whether the square of the unmoved piece can be extracted from the value vector. The probe achieves an accuracy of 91% for knights and 80% for rooks. This means that while there is some information of the companion piece's location, the signal is not perfect.