Learning Chess Blindfolded: Evaluating Language Models on State Tracking

Shubham Toshniwal, Sam Wiseman, Karen Livescu, Kevin Gimpel

Toyota Technological Institute at Chicago

Abstract

Transformer language models have made tremendous strides in natural language understanding tasks. However, the complexity of natural language makes it challenging to ascertain how accurately these models are tracking the world state underlying the text. Motivated by this issue, we consider the task of language modeling for the game of chess. Unlike natural language, chess notations describe a simple, constrained, and deterministic domain. Moreover, we observe that the appropriate choice of chess notation allows for directly probing the world state, without requiring any additional probing-related machinery. We find that: (a) With enough training data, transformer language models can learn to track pieces and predict legal moves with high accuracy when trained solely on move sequences. (b) For small training sets providing access to board state information during training can yield significant improvements. (c) The success of transformer language models is dependent on access to the entire game history i.e. "full attention". Approximating this full attention results in a significant performance drop. We propose this testbed as a benchmark for future work on the development and analysis of transformer language models.[1]

1 Introduction

Recently, transformer-based language models have stretched notions of what is possible with the simple self-supervised objective of language modeling, becoming a fixture in state of the art language technologies [Vaswani et al., 2017, Devlin et al., 2019, Brown et al., 2020]. However, the black box nature of these models combined with the complexity of natural language makes it challenging to measure how accurately they represent the world state underlying the text.

In order to better measure the extent to which these models can capture the world state underlying the symbolic data they consume, we propose training and studying transformer language models for the game of chess. Chess provides a simple, constrained, and deterministic domain where the exact world state is known. Chess games can also be transcribed exactly and unambiguously using chess notations (Section 2). Most importantly, the form of chess notations allows us to probe our language models for aspects of the board state using simple prompts (Section 3) and without changing the language modeling objective or introducing any new classifiers.

Due to the simplicity and precision of chess, we can evaluate language model predictions at a more fine-grained level than merely comparing them to the ground truth. For example, even if the next move prediction doesn't match the ground truth move, we can still evaluate whether the move is legal given the board state, and if it is illegal, the error can be automatically analyzed (Appendix 6). Moreover, since world state transitions are deterministic and known, we can evaluate models using counterfactual queries as well. Our proposed evaluation sets and metrics are described in Section 3.2.

While chess represents a controlled domain, it is by no means trivial for a language model. To illustrate the challenges of language modeling for chess, consider the left board shown in Figure 1b, where white is next to move. In order to generate a valid next move, the language model needs to (a) infer that it is white's turn, (b) represent the locations of all pieces, both white and black, (c) select one of the white pieces which can be legally moved, and finally (d) make a legal move with the selected piece. Thus, a language model has to learn

---

[1]Code and data available at `https://github.com/shtoshni92/learning-chess-blindfolded`

(a) Square naming

(b) Board state before (left) and after (right) the bishop at `f1` is moved to `b5`. UCI notation represents the move as `f1b5`.
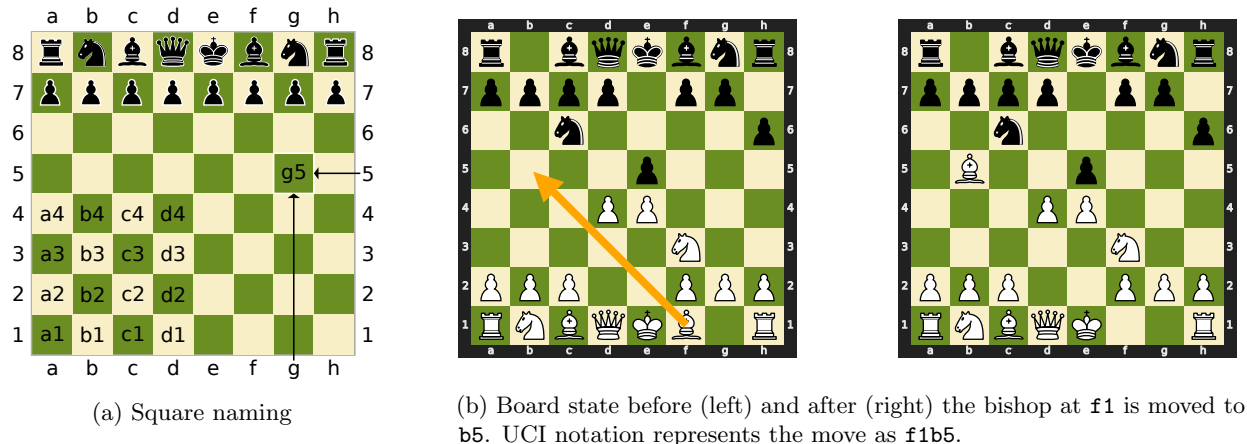
Figure 1: Chess Notation

to track the board state, learn to generate moves according to the rules of chess, and on top of that learn chess strategies to predict the actual move.

We find that when given enough training data, transformers can learn to both track piece locations and predict legal moves with high accuracy. However, when trained on small training sets, predictive ability suffers. In this more challenging setting, introducing parts of the board state as tokens in the training sequences (Section 3.1) improves piece tracking significantly (Section 6).

Our results also provide some key insights on transformer language models: (i) They are robust to changes in input distribution where additional tokens, related to board state, are added to input sequence *only during training* (Section 3.1). In contrast to LSTMs, transformers achieve this robustness even with smaller training sets (Section 5.3). (ii) Even though chess is Markovian, the model relies on having access to the whole history, and the performance drops when limiting this access (Section 5.3.1).

To summarize, our contributions are to:

- Propose chess as a testbed for evaluating world state tracking capabilities of language models which can be used for development and analysis of these models.

- Show that with the appropriate chess notation, we can probe language models for aspects of the world state using simple prompts (Section 3).

- Show that given enough training data, transformer language models can learn to track piece locations and predict legal moves with high accuracy.

- Demonstrate that transformer language models are robust to certain changes in input distribution, and that access to world state during training improves performance with small datasets.

2   Chess Preliminaries

We represent moves using Universal Chess Interface (UCI) notation, which combines the starting square and the destination square to represent a move.[2] The move in Figure 1b is represented as `f1b5` in UCI where `f1` indicates the starting square and `b5` denotes the ending square. While the SAN notation is the standard choice for gameplay, we prefer UCI (see Appendix A for why we pick UCI over SAN).

For training language models, we first tokenize games represented in UCI notation using a simple regular expression based tokenizer, which considers a board square symbol such as `b1` as a single token. This gives us a vocabulary of 77 token types, which includes the 64 squares, piece type symbols, and other special symbols (see Table 8 in the Appendix).[3] For example, the move sequence "e2e4 e7e5 g1f3" is tokenized to "e2, e4,

---

[2]For more details see `https://en.wikipedia.org/wiki/Universal_Chess_Interface`

[3]In initial experiments we used a delimiter token to indicate move boundary. However, removing it did not degrade performance and made training faster due to reduced sequence length.

Table 1: Token sequences corresponding to the move sequence `e2e4 e7e5 g1f3` for different notations during training and inference. Notice that regardless of the RAP probability used during training, at inference time the token sequences have no piece types.

Notation	Training	Inference
UCI	e2, e4, e7, e5, g1, f3	e2, e4, e7, e5, g1, f3
UCI + RAP 15	e2, e4, P, e7, e5, g1, f3	e2, e4, e7, e5, g1, f3
UCI + RAP 100	P, e2, e4, P, e7, e5, N, g1, f3	e2, e4, e7, e5, g1, f3
UCI + AP	P, e2, e4, P, e7, e5, N, g1, f3	P, e2, e4, P, e7, e5, N, g1, f3

`e7`, `e5`, `g1`, `f3`". We then train an autoregressive language model on these move sequences, using the standard maximum likelihood objective.

3 Language Model Prompts as Board State Probes

One attractive property of having a language model trained on chess games represented in UCI notation (as described in the previous section) is that the notation *itself* allows us to probe the trained model's state tracking abilities. In particular, by feeding the trained language model a prefix of a game as a prompt, we can determine — using the language model's next-token predictions — what the model understands about the board state implied by this prefix. For example, consider the prompt "`e2e4 e7e5 g1f3 b8c6 d2d4 h7h6 f1`," where the underlined move sequence leads to the left board state in Figure 1b. A language model's next-token prediction (after consuming the prompt) can be interpreted as the ending square predicted for the bishop at `f1`, which can be used to determine the level of board state awareness of the model. If, for instance, the model predicts `g1`, this may indicate that the model does not recognize that the piece type at `f1` is a bishop, as such a move is not possible for a bishop. If, on the other hand, the model predicts `g2`, that may indicate that the model is not aware that another piece is currently at `g2`.

3.1 Randomly Annotated Piece type (RAP)

While predicting the token representing the ending-square of a move given a prompt allows us to assess the model's state tracking abilities, it also to some extent conflates the model's understanding of the board state with its understanding of chess strategy. If we could easily probe for where the model thinks a piece *currently* is (rather than where it is likely to end up) given a game prefix, this would allow us to more directly probe the model's state tracking abilities. In particular, we would like to give a language model a prompt such as "`e2e4 e7e5 g1f3 b8c6 d2d4 h7h6 N`", where `N` represents knight, and expect it to generate a valid starting position for a knight of the correct color. While UCI notation does not ordinarily include these piece type tokens, to allow for testing the model with such prompts, we propose to randomly include these piece types tokens in moves during training with some fixed probability p. We refer to this strategy as "randomly annotated piece type" (RAP) and use the nomenclature "UCI + RAP p" to indicate that with $p\%$ probability, piece type is part of the move notation during training. Note that for $p = 0$, the notation reduces to UCI.

When *testing* with these starting square prediction prompts, we only include piece type for the prompt, not for any moves in the history. Thus, using RAP during training allows us to probe, at test time, where the model thinks each piece is, given any game history's prefix; by simply providing the desired piece type (e.g., `N`) the model outputs the predicted starting square for a piece of that type. For example, given the prompt "`e2e4 e7e5 g1f3 b8c6 d2d4 h7h6 N`", a prediction of `f3` or `b1` shows that the model is aware of where the knights are.

We also experiment with an "oracle" variant of RAP where piece types are added both during training and testing. We refer to this notation as "UCI + AP " where AP stands for "always piece type". For our running example the equivalent prompt in this notation would be "`Pe2e4 Pe7e5 Ng1f3 Nb8c6 d2d4 Ph7h6 N`".

In terms of the language modeling training objective, addition of RAP represents a distribution change between training and inference. Table 1 illustrates how the use of RAP changes the token sequence during training but not during inference. While there's a distribution mismatch, we hypothesize that addition of

Table 2: Examples of each probing task, as well as the corresponding exact move (ExM) and legal move (LgM) correct answers, are shown below. All examples assume the language model was fed the prefix `e2e4 e7e5 g1f3 b8c6 d2d4 h7h6` (see Figure 1b), and that the actual next move was `f1b5`. While there is only one valid prompt token for both End-Actual and Start-Actual tasks, there are many valid prompt tokens for the other tasks, and we show just one possibility for each. Start-tasks (bottom sub-table) assume the model was trained on games described in UCI+RAP notation.

Task	Prompt Token	Correct Answers (ExM)	Correct Answers (LgM)
End-Actual	f1	{b5}	{e2, d3, c4, b5 ,a6 }
End-Other	f3	N/A	{d2, g1, h4, g5, e5}
Start-Actual	B	{f1}	{f1, c1}
Start-Other	N	N/A	{f3, b1}

RAP can aid the model in learning to track the pieces by providing additional supervision which, in turn, can improve language modeling performance as well.

3.2 Board State Probing Tasks

In this subsection we describe the probing tasks introduced above more concretely. In each probing task we feed the model a prefix of a game followed by a single prompt token, and the model is evaluated based on the highest probability next-token under the model given this context. We show an example of each probing task in Table 2 (which we further describe below), assuming the model has been fed the move sequence prefix `e2e4 e7e5 g1f3 b8c6 d2d4 h7h6`, which is visualized as the left board in Figure 1b. The actual next move played in the game is `f1b5`, which takes the white bishop at square `f1` to square `b5`, as shown in the right board of Figure 1b.

3.3 Ending Square Tasks

In this set of tasks, the model is given a game prefix and prompted with the starting square of the next move (`f1` in the example of Table 2). The model's next-token prediction represents its prediction for the ending square of this move, which tests the model's ability to track the board state and follow the rules of chess, as well as strategic awareness.[4] We consider two task variants:

1. **End-Actual**: Given a move sequence prefix, the model is prompted with the starting square of the actual piece moved next in the game.

2. **End-Other**: Given a move sequence prefix, the model is prompted with the starting square of any piece on the board that can be legally moved according to the rules of chess.

We evaluate End-Actual predictions in terms of both exact move (ExM) accuracy (whether the model predicted the true ending square, `b5` in our running example) and legal move (LgM) accuracy (whether the model predicted a legal ending square for the piece starting at the square in the prompt). For LgM evaluation, we also calculate the R-Precision which is the Precision@R where R is the total number of legal ending squares Manning et al. [2008]. In our running example, there are 5 legal ending squares, and R-Precision will be calculated for the model's top-5 predictions. ExM accuracy evaluation is similar to the typical evaluation of language models on natural language data, while LgM is less stringent and focuses on testing just the model's understanding of chess rules and the board state. Note that for End-Other, only LgM evaluation is available. See Table 2 for examples.

3.4 Starting Square Tasks

In this category of task, the model is again given a game prefix, but prompted with just the piece *type* of the next move, such as `B` for bishop in the example in Table 2. The model's next-token prediction thus represents

---

[4]Strategic capabilities of a chess language model are strongly tied to the quality of training games.

its prediction for where the prompted piece type currently is on the board. This task tests the model's ability to track pieces.[5] Note that only models which have seen piece types during training, i.e. "UCI + RAP" models, can actually be tested on this task. Also, no piece types are used in the game prefix. We again have two variants of this task:

1. **Start-Actual**: Given a move sequence prefix, the model is prompted with the piece type of the actual piece moved next in the game.

2. **Start-Other**: Given a move sequence prefix, the model is prompted with the piece type of any piece on the board that can be legally moved according to the rules of chess.

We again evaluate Start-Actual both in terms of ExM accuracy (whether the model predicts the starting square of the piece actually moved next in the game), as well as in terms of LgM accuracy (whether the model predicts the starting square of a legally movable piece of the given piece type) and LgM R-Precision (precision of the model's top-R predictions with respect to all of the R starting squares of legally movable pieces of the given piece type). For Start-Other, only LgM evaluation is applicable; see Table 2 for examples.

4   Experimental Setup

Data   We use the Millionbase dataset which is freely available and has close to 2.9 million quality chess games.[6] After filtering out duplicate games, games with fewer than 10 moves, and games with more than 150 moves (for the complete game to fit into one transformer window), we are left with around 2.5 million games. From this filtered set we randomly select 200K games for training, 15K games each for dev and test, and another 50K games to create board state probing evaluation sets described in Section 3.2. The dev and test sets are used for perplexity evaluations. The dev set perplexity is used for choosing hyperparameters. From the 200K training set, we create subsets of size 15K and 50K which we refer to as "Train-S" and "Train-M", while the full training set is referred to as "Train-L". For detailed statistics, see Table 10 in Appendix. All the data processing steps requiring chess knowledge, including parsing chess databases, are carried out using python-chess [Fiekas, 2012].

To create the board state probing evaluation sets, we use the 50K games reserved for this task. We only consider prompts for non-pawn pieces since the dynamics of pawns are fairly limited. We ensure that the game prefixes selected are never seen in the training data. The final evaluation set consists of 1000 instances with prefix length (in number of moves) in the range $51 \leq l \leq 100$.

Model Details   We use the GPT2-small architecture for our base language model [Vaswani et al., 2017, Radford et al., 2019]. GPT2-small is a 12-layer transformer model with 12 attention heads and an embedding size of 768 dimensions. The context size of the model is limited to 512, which is sufficient to cover the longest game in our training set. Note that we only borrow the model architecture; the models themselves are *trained from scratch.* [7]

For the UCI + RAP p models, we tune over $p \in \{5, 15, 25, 50, 75, 100\}$ based on perplexity on the validation set. Note that for perplexity evaluation, logits corresponding to piece type tokens are masked out since piece type tokens are only available during training. We find that $p = 25$ performs the best for Train-S and Train-M, while $p = 15$ is best for Train-L (Figure 2). Larger values of p lead to greater mismatch between training and inference, while small values likely do not provide enough training signal.

We also experiment with other transformer and non-transformer models in Section 5.3. Among the transformer models, we experiment with two "partial" attention models (i.e., models where every token doesn't attend to all other tokens in the context) , namely, Reformer Kitaev et al. [2020] and Performer Choromanski et al. [2021]. We set the number of layers and attention heads to 12 for both architectures, as in GPT2-small. We also train LSTM language models with and without RAP. For details on hyperparameters and tuning, see Appendix E.1.

---

[5] In certain cases, this task also tests understanding of chess rules. For example, in Figure 1b only the rook at `h1` can be moved.
[6] Download link available at `https://rebel13.nl/rebel13/rebel%2013.html`
[7] Colab notebook to play chess against the base language model `https://github.com/shtoshni92/learning-chess-blindfolded/blob/master/GPT2_Chess_Model.ipynb`

Training Set	Model	Dev set	Test set
Train-S	UCI	23.6	23.6
	UCI + RAP	15.9	15.9
	UCI + AP	16.1	16.2
Train-M	UCI	11.6	11.6
	UCI + RAP	10.4	10.4
	UCI + AP	10.1	10.0
Train-L	UCI	7.7	7.7
	UCI + RAP	7.4	7.4
	UCI + AP	7.2	7.2

Table 3: Canonical validation and test set perplexity. By canonical we mean that one move, say `f1b5`, counts as one token.
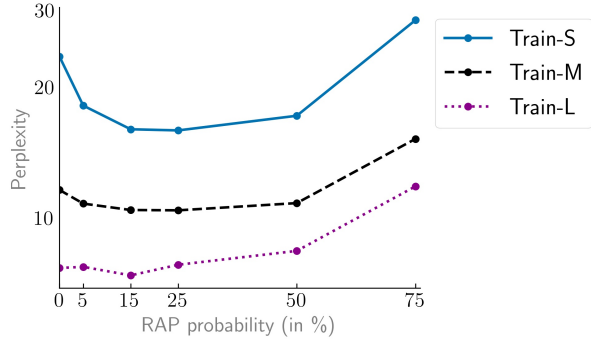

Figure 2: Validation set perplexities as a function of RAP probabilities for the different training set sizes. RAP 0 is the standard UCI notation. RAP 100 is not shown as perplexities are too high.

Training Details   Models are trained for 10 epochs with a batch size of 60. Validation is performed at the end of every epoch and training stops whenever the validation loss starts increasing. For optimization we use Adam [Kingma & Ba, 2014] with learning rate of 5×10^{-4} and L2 weight decay of 0.01. The learning rate is warmed up linearly over the first 10% of training followed by a linear decay. To accelerate training, we use mixed precision training [Micikevicius et al., 2018]. All experiments are carried out using the PyTorch Lightning framework built on top of PyTorch [Falcon, 2019, Paszke et al., 2019]. We use the transformers library [Wolf et al., 2019] for all models[8] except for the Performer for which we use a popular unofficial implementation. [9]

5   Results

We first present language modeling results, where we show significant improvements with the addition of RAP (Section 5.1). Next, we show results on the board state probing tasks for the base language model, where we demonstrate that the model trained on the large training set can learn to track pieces and predict legal moves with high accuracy (Section 5.2). Finally, we present results on the probing task with partial attention transformer architectures and LSTMs, where we show a performance drop in comparison to the base model with full attention (Section 5.3).

5.1   Language Modeling

Table 3 presents the perplexity results on the validation and test sets. Figure 2 plots the validation set perplexities as a function of RAP probability for different training set sizes. The addition of RAP and AP leads to a decrease in perplexity for all training sizes, particularly for small training sets. For small training sets, RAP probabilities as high as 50% can improve the validation perplexity, but for larger training sets, lower RAP probabilities are preferred. The reductions in perplexity for RAP are surprising given that the extra tokens added via RAP are not present in the validation and test sets, and thus there is a data distribution shift. Models trained with UCI + AP achieve the lowest perplexities on larger training sets. Both RAP and AP aid the model in piece tracking, as we will see in later results, and in the case of chess this can significantly improve the language modeling results as well. Note that for calculating the perplexity of UCI + RAP models, we mask out the logits corresponding to piece type tokens since they are never present during inference.

---

[8]Reformer implementation in `transformers` library is still a work in progress. The presented results are with the 4.2.2 version.
[9]https://github.com/lucidrains/performer-pytorch

Table 4: Accuracies and R-Precisions (%) for predicting starting squares ("Start-Actual" and "Start-Other" tasks). S, M, L in the first column refer to the training set sizes.

| | Notation | LgM | | | | ExM |
| | | Actual | | Other | | |
		Acc.	R-Prec.	Acc.	R-Prec.	Acc.
S	UCI + RAP	91.3	90.2	89.3	89.2	78.8
	UCI + AP	99.2	99.1	98.8	98.8	86.9
M	UCI + RAP	98.2	98.0	98.6	98.7	88.0
	UCI + AP	99.9	99.8	100.0	100.0	90.2
L	UCI + RAP	100.0	100.0	99.6	99.5	91.8
	UCI + AP	99.9	99.9	99.7	99.7	91.1
Random Legal		-	-	-	-	86.0

Table 5: Accuracies and R-Precisions (%) for predicting ending squares ("End-Actual" and "End-Other" tasks). S, M, L in the first column refer to the training set sizes.

| | Notation | LgM | | | | ExM |
| | | Actual | | Other | | |
		Acc.	R-Prec.	Acc.	R-Prec.	Acc.
S	UCI	74.0	61.1	65.5	57.7	26.7
	UCI + RAP	88.4	75.5	80.4	72.1	33.3
	UCI + AP	87.0	77.0	78.8	72.3	36.1
M	UCI	92.9	80.6	85.8	78.5	42.2
	UCI + RAP	94.9	82.2	87.9	78.0	45.9
	UCI + AP	94.7	82.4	88.3	79.1	47.3
L	UCI	97.7	85.6	91.9	83.8	52.0
	UCI + RAP	97.0	86.1	93.1	83.9	54.7
	UCI + AP	98.2	87.3	95.2	86.3	56.7
Random Legal		-	-	-	-	19.6

5.2 Board State Tracking

Tables 4 and 5 show results when predicting starting squares and ending squares, respectively. There are several observations to note. First, **transformers can learn to identify where pieces are located**. This is shown by the LgM accuracies in Table 4. UCI + RAP can predict legal starting positions with perfect accuracy and R-Precision. However, this capability requires Train-L, and the accuracy drops to 91.3% for Train-S. The gap between UCI + RAP and its "oracle" counterpart, UCI + AP, also reduces with an increase in training set size with UCI + RAP achieving parity for Train-L. When asked to identify the location of a piece other than the one selected to be moved next, this accuracy drops only slightly to 99.6%. Typically, the piece location tracking is slightly better for the piece type that is actually moved than for other piece types.

The difference between the location of the piece in the exact move (ExM) and the location of either piece of the given type (LgM) is substantial, at more than 8% absolute. However, this difference relates to chess strategy rather than board state tracking.

Second, **transformers can learn to predict legal moves**. This is shown by the LgM accuracies in Table 5, for which both UCI and UCI + RAP exceed 97% accuracy. However, while the top predictions of the models have high accuracy, their ability to predict all legal moves is significantly lower, with R-precision of about 85%. This is to be expected, since the model is trained on only actual games, where the emphasis is on "meaningful" moves rather than any legal move. Due to similar reasons, there's a significant drop in performance when predicting ending squares for starting squares other than the one in the actual game.

Table 6: Accuracy and R-Precision (%) for predicting ending squares ("End-Actual" and "End-Other" tasks) with varying attention window sizes. LSTM + RAP refers to LSTM trained with UCI + RAP.

| | Model | LgM | | | | ExM |
| | | Actual | | Other | | |
		Acc.	R-Prec.	Acc.	R-Prec.	Acc.
S	GPT2	74.0	61.1	65.5	57.7	26.7
	GPT2 ($w = 50$)	69.5	57.4	60.4	53.2	23.1
	Reformer	71.0	57.2	61.5	53.5	24.8
	Performer	65.4	54.3	57.9	49.5	20.5
	LSTM	60.2	51.0	52.5	46.4	20.9
	LSTM + RAP	59.5	50.5	52.4	46.0	21.9
M	GPT2	92.9	80.6	85.8	78.5	42.2
	GPT2 ($w = 50$)	86.0	74.9	80.9	71.3	35.8
	Reformer	86.4	73.2	76.6	68.6	32.4
	Performer	89.2	76.3	80.5	71.5	36.0
	LSTM	73.8	61.6	67.2	59.8	32.0
	LSTM + RAP	77.5	64.9	69.7	61.7	32.1
L	GPT2	97.7	85.6	91.9	83.8	52.0
	GPT2 ($w = 50$)	95.8	84.5	90.5	82.7	51.6
	Reformer	88.0	74.9	77.0	68.1	33.5
	Performer	95.8	84.5	90.5	82.7	51.6
	LSTM	93.4	79.5	86.1	76.0	45.2
	LSTM + RAP	92.8	80.4	87.3	77.1	46.0

The "other" starting square would, by design, have legal continuations, but lack any "meaningful" ones (see examples in Appendix F).

We find consistent gains in almost all metrics with the addition of RAP during training, with the gains being particularly impressive for small training sets. Thus, not only are the transformers robust to distribution shift due to RAP (available only during training), they are in fact able to utilize this additional information. Error analysis of illegal predictions shows that the addition of RAP improves piece tracking related errors (Section 6).

The relatively low ExM accuracies of the models can be attributed to the inherent difficulty of the task. Randomly selecting an ending square from all legal ending squares has an accuracy of only around 20%, implying that on average there are roughly 5 legal choices, which might explain the difficulty of the task.

5.3 Compressing the Game History

The base transformer language model, based on GPT2, attends to the entire history (i.e., it uses "full attention"), which results in complexity quadratic in the length of the sequence. We might wonder whether attending to this entire history is necessary for the impressive state tracking performance observed in the previous section. We accordingly explore models that do not attend to the entire history in Table 6.

We first experiment with a variant of the GPT2 model that limits its attention to a window of only the 50 most recent tokens ("GPT2 ($w = 50$)"). In Table 6 we see worse performance for this model across data sizes, but especially for small- and medium-sized datasets.

In Table 6 we also consider a language model based on the LSTM [Hochreiter & Schmidhuber, 1997], which considers only its current hidden state and cell state in making its predictions, and does not explicitly attend to the history. Here we find an even more significant drop in performance, in all settings. (Interestingly, we also find that training LSTM language models on sequences with RAP improves performance, but only for larger training sets; transformer language models generally improve when trained with RAP data).

The results of GPT2 ($w = 50$) and of the LSTM language model suggest that attending to the full game history is, unsurprisingly, useful for board state tracking in chess. This finding further suggests that the
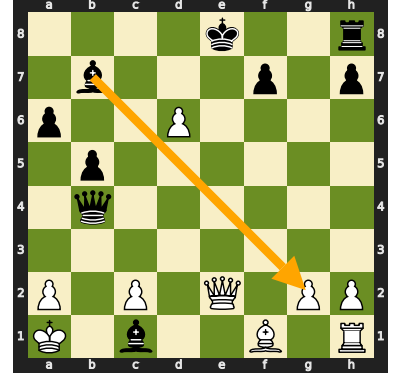
(a) *Syntax*: Queen can move like all other piece types except for knight.
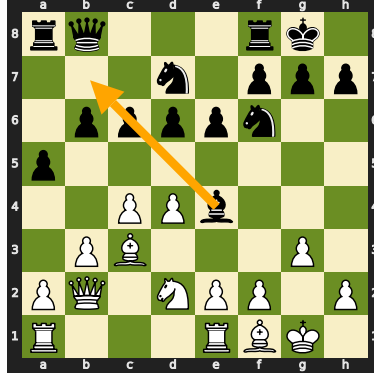
(b) *Path Obstruction*: The pawn at c6 is blocking the bishop.

(c) *Pseudo Legal*: The black king remains in check.

Figure 3: Instances of the three prominent categories of illegal ending square predictions.

task of board state tracking in chess can serve as an excellent testbed for recently proposed transformer variants [Kitaev et al., 2020, Katharopoulos et al., 2020, Choromanski et al., 2021, *inter alia*] that attempt to make use of long histories or contexts, but *without* incurring a quadratic runtime.

Table 7: Error counts for ending square prediction.

	Model	Actual			Other		
		Syntax	Path Obst.	Pseudo Leg.	Syntax	Path Obst.	Pseudo Leg.
Train-S	UCI	168	48	40	173	90	80
	UCI + RAP	20	58	38	17	96	81
	UCI + AP	1	99	29	3	126	81
	Performer	235	56	53	243	70	106
Train-M	UCI	16	30	25	15	54	72
	UCI + RAP	3	30	18	7	56	55
	UCI + AP	0	36	17	3	59	53
	Performer	41	27	40	42	45	108
Train-L	UCI	1	10	12	4	26	49
	UCI + RAP	0	19	11	3	29	36
	UCI + AP	0	13	5	3	13	31
	Performer	8	18	16	9	23	63

5.3.1 Partial Attention Transformers

We experiment with the recently proposed Reformer Kitaev et al. [2020] and Performer Choromanski et al. [2021] architectures. Reformer replaces the "full attention" with attention based on locality-sensitive hashing, while Performer approximates the "full attention" with random features (for details see Appendix E.1)) [10].

The results, in Table 6, suggest that the Performer generally outperforms the Reformer, except in the small dataset-setting. Furthermore, we find that neither of these architectures significantly outperforms the GPT2 ($w = 50$) baseline, except for Performer in the medium-sized data setting. These models do, however, typically outperform the LSTM models. These results demonstrate the challenge of modeling chess with a partial attention. We hope that future work will use this task as a way of benchmarking more efficient transformer architectures.

---

[10]In practice, these models often use a combination of the proposed approximate global attention and simple local attention.

6   Error Analysis

In this section we analyze errors on the ending square prediction task. Incorrect predictions for this task can be (exhaustively) categorized into four categories:

- *Unreachable*: The predicted ending square cannot be reached by any possible piece type at the starting square regardless of the board state.

- *Syntax*: The predicted ending square cannot be reached by the piece type present at the starting square regardless of the board state. This error indicates failure at tracking the piece type present at the starting square.

- *Path Obstruction*: The predicted ending square cannot be reached because there are other pieces obstructing the path. This error indicates failure at tracking other pieces on the board or a lack of understanding that for all piece types except the knight, the path must be clear. For example, in Figure 3b, the pawn at `c6` blocks the bishop's move from `e4` to `b7`.

- *Pseudo Legal*: The move is illegal because the moving player's king is in check at the end of the move.

Table 7 shows error counts for the ending square prediction task. For brevity we omit unreachable errors since they are rare (< 5 for all models).

Errors across all categories decrease with more training data. For syntax errors this reduction is particularly dramatic, decreasing by roughly an order of magnitude when moving from Train-S to Train-M. In contrast, both path obstruction and pseudo legal errors decline more gradually. Determining whether a path is blocked or if the king is in check requires a computation involving multiple piece locations which all need to be computed from the move history. These trends suggest that identifying the piece type at a starting square requires data but is learnable, while keeping track of all *other* pieces on the board remains challenging even with large training sets.

UCI + RAP consistently outperforms UCI in syntax errors, the differences being largest for the small training sets. This validates our hypothesis that RAP can aid the model in piece tracking (Section 3.1). Across other error categories we don't see consistent trends, suggesting piece tracking improvements do not necessarily translate to other error categories. The Performer generally makes more errors than the transformers, especially in the syntax category. The partial attention in the Performer may be limiting its ability to attend to the most relevant prior positions to determine the piece type at the given starting square.

Predicting ending squares for the actual move made ("Actual") is easier than for a randomly chosen legal move ("Other"). However, the syntax errors are comparable between the two settings, while there are many more path obstruction and pseudo legal errors for the Other instances. The higher error rate for these categories could be because:

- Avoiding path obstruction and check are difficult functions to learn and may therefore be being "mimicked" from training data rather than being learned as a general algorithmic function.

- The model is trained on only actual games with emphasis on meaningful moves rather than legal moves. We observe that some of the Other instances lack any "meaningful" continuations (Appendix F).

- There's a distribution shift between piece types moved in actual moves vs randomly chosen legal moves. For example, the End-Actual task has only about 13% prompts for moves made by king in comparison to the 33% for the End-Other task (Appendix D). We find that moves made by king have a higher chance of resulting in pseudo legal errors in comparison to other piece types (Appendix F.1).

7   Related Work

Simulated Worlds.   There have been several prior efforts in relating simulated worlds to natural language. The bAbI framework simulates a world modeled via templates to generate question answering tasks [Weston et al., 2015]. The recent TextWorld framework facilitates generating, training, and evaluating interactive text-based games [Côté et al., 2018]. Hermann et al. [2017] and Hill et al. [2017] develop and use 3D world simulations for learning grounded language. These efforts are similar to our work in the sense that the true world state is, by construction, available, but our setup differs in that it provides a natural way of probing the state tracking of a model trained with an LM objective.

Cloze Tasks for Natural Language Models. There has been a great deal of work on cloze tasks for evaluating natural language models [Hermann et al., 2015, Hill et al., 2016]. These tasks range from testing general text understanding [Paperno et al., 2016] to targeting particular aspects of natural language, such as commonsense/pragmatics [Mostafazadeh et al., 2016, Ettinger, 2020], narrative understanding [Mostafazadeh et al., 2017], and factual knowledge [Petroni et al., 2019]. Creating these tasks often requires human curation, and the evaluation is typically limited to exact match.[11] Our proposed tasks are a form of cloze tasks, but can be precisely automated so that they require no human curation, and can be evaluated at a fine-grained level.

Probing. One of the goals of this work is to probe the language model's board state tracking capability. A typical solution used by prior work is to train a probing model on top of a pretrained model [Ettinger et al., 2016, Alain & Bengio, 2017, Adi et al., 2017, Tenney et al., 2019, Hewitt & Liang, 2019]. This setup is time-consuming as it requires training probing models for all tasks. Moreover, the complexity of the probing model can also affect the conclusions [Pimentel et al., 2020]. In our case, by using an appropriate choice of notation, probing for board state can be accomplished via simple prompts (Section 3).

Deep Learning for Chess. Deep networks have been used in prior work to predict the next move given the true game state David et al. [2016], Oshri & Khandwala [2015]. For example, using only self-play and the rules of chess, AlphaZero achieves superhuman performance starting from random play [Silver et al., 2018]. The focus of this prior work is the quality of game play given the true board state, while we use chess as a testbed for evaluating a language model's board state tracking capability. Recently there has also been work focusing on transformer language models for chess [Presser & Branwen, 2020, Cheng, 2020, Noever et al., 2020]. This work is similar to ours in the sense that the input is limited to the move sequence without the true board state, but the focus is again the quality of game play rather than the model's awareness of the underlying state.

8   Conclusion

We propose the game of chess as a testbed for evaluating how well language models capture the underlying world state. We show that with an appropriate choice of chess notation, a language model can be probed for different aspects of the board state via simple prompts. The simple and precise dynamics of chess allow for (a) training models with varying amount of explicit state, and (b) evaluating model predictions at a fine-grained level. Results show that transformer language models are able to track the board state when given enough data, but with limited data, providing access to board state information during training can yield consistent improvement.

Wider Implications for Natural Language Processing. Our results shed light on the following properties of transformers: (a) they are robust to RAP-like changes in input distribution, and (b) for high performance the models require access to the entire context, as well as large training sets (Section 5.3.1). Future work can use the first finding to introduce the world state, or more specifically the output of linguistic analyzers such as coreference, via RAP-like tokens during pre-training and fine-tuning of transformers. RAP-like tokens can also be used for debugging/diagnosing a model's understanding, similarly to the starting square prediction tasks. The second finding implies that the proposed benchmark can guide the search for new transformer architectures that are adept at understanding long text, and that can learn from small training sets. The proposed framework allows for probing and understanding new architectures that address these challenges.

Acknowledgements

---

[11]Automated cloze tasks without human filtering can yield instances which even humans can't answer [Hill et al., 2016].

References

Adi, Y., Kermany, E., Belinkov, Y., Lavi, O., and Goldberg, Y. Fine-grained Analysis of Sentence Embeddings Using Auxiliary Prediction Tasks. In *ICLR*, 2017.

Alain, G. and Bengio, Y. Understanding intermediate layers using linear classifier probes. In *ICLR Workshop*, 2017.

Bender, E. M. and Koller, A. Climbing towards NLU: On Meaning, Form, and Understanding in the Age of Data. In *ACL*, 2020.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. Language Models are Few-Shot Learners, 2020.

Bruni, E., Tran, N. K., and Baroni, M. Multimodal Distributional Semantics. *JAIR*, 49(1), 2014.

Chen, W., Chen, X., Zhang, J., and Huang, K. Beyond Triplet Loss: A Deep Quadruplet Network for Person Re-identification. In *CVPR*, 2017.

Cheng, R. Transformers Play Chess, 2020. URL https://github.com/ricsonc/transformers-play-chess.

Choromanski, K. M., Likhosherstov, V., Dohan, D., Song, X., Gane, A., Sarlos, T., Hawkins, P., Davis, J. Q., Mohiuddin, A., Kaiser, L., Belanger, D. B., Colwell, L. J., and Weller, A. Rethinking Attention with Performers. In *ICLR*, 2021.

Côté, M.-A., Kádár, A., Yuan, X., Kybartas, B., Barnes, T., Fine, E., Moore, J., Tao, R. Y., Hausknecht, M., Asri, L. E., Adada, M., Tay, W., and Trischler, A. TextWorld: A Learning Environment for Text-based Games. *CoRR*, abs/1806.11532, 2018.

David, E., Netanyahu, N. S., and Wolf, L. DeepChess: End-to-End Deep Neural Network for Automatic Learning in Chess. In *International Conference on Artificial Neural Networks (ICANN)*, 2016.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL*, 2019.

Ettinger, A. What BERT is Not: Lessons from a New Suite of Psycholinguistic Diagnostics for Language Models. *TACL*, 8(0), 2020.

Ettinger, A., Elgohary, A., and Resnik, P. Probing for semantic evidence of composition by means of simple classification tasks. In *1st Workshop on Evaluating Vector-Space Representations for NLP*, 2016.

Falcon, W. PyTorch Lightning, 2019. URL https://github.com/PyTorchLightning/pytorch-lightning.

Fiekas, N. python-chess: a pure Python chess library, 2012. URL https://python-chess.readthedocs.io/en/latest/index.html.

Ha, D. and Schmidhuber, J. Recurrent World Models Facilitate Policy Evolution. In *NeurIPS*, 2018.

Hermann, K. M., Kočiský, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. Teaching Machines to Read and Comprehend. In *NeurIPS*, 2015.

Hermann, K. M., Hill, F., Green, S., Wang, F., Faulkner, R., Soyer, H., Szepesvari, D., Czarnecki, W. M., Jaderberg, M., Teplyashin, D., Wainwright, M., Apps, C., Hassabis, D., and Blunsom, P. Grounded Language Learning in a Simulated 3D World. *CoRR*, abs/1706.06551, 2017.

Hewitt, J. and Liang, P. Designing and Interpreting Probes with Control Tasks. In *EMNLP-IJCNLP*, 2019.

Hill, F., Bordes, A., Chopra, S., and Weston, J. The Goldilocks Principle: Reading Children's Books with Explicit Memory Representations. In *ICLR*, 2016.

Hill, F., Hermann, K. M., Blunsom, P., and Clark, S. Understanding Grounded Language Learning Agents. *CoRR*, abs/1710.09867, 2017. URL http://arxiv.org/abs/1710.09867.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9, 1997.

Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. Scaling Laws for Neural Language Models, 2020.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention. In *ICML*, 2020.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *ICLR*, 2014.

Kiros, R., Salakhutdinov, R., and Zemel, R. S. Unifying Visual-Semantic Embeddings with Multimodal Neural Language Models. *CoRR*, abs/1411.2539, 2014.

Kitaev, N., Kaiser, L., and Levskaya, A. Reformer: The Efficient Transformer. In *ICLR*, 2020.

Linzen, T., Dupoux, E., and Goldberg, Y. Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies. *TACL*, 4, 2016.

Manning, C. D., Raghavan, P., and Schütze, H. *Introduction to Information Retrieval*. Cambridge University Press, 2008.

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaiev, O., Venkatesh, G., and Wu, H. Mixed Precision Training. In *ICLR*, 2018.

Mostafazadeh, N., Chambers, N., He, X., Parikh, D., Batra, D., Vanderwende, L., Kohli, P., and Allen, J. A Corpus and Cloze Evaluation for Deeper Understanding of Commonsense Stories. In *NAACL*, 2016.

Mostafazadeh, N., Roth, M., Louis, A., Chambers, N., and Allen, J. LSDSem 2017 Shared Task: The Story Cloze Test. In *2nd Workshop on Linking Models of Lexical, Sentential and Discourse-level Semantics*, 2017.

Noever, D., Ciolino, M., and Kalin, J. The chess transformer: Mastering play using generative language models, 2020.

Noji, H. and Takamura, H. An Analysis of the Utility of Explicit Negative Examples to Improve the Syntactic Abilities of Neural Language Models. In *ACL*, 2020.

Ororbia, A., Mali, A., Kelly, M., and Reitter, D. Like a Baby: Visually Situated Neural Language Acquisition. In *ACL*, 2019.

Oshri, B. and Khandwala, N. Predicting Moves in Chess using Convolutional Neural Networks. 2015.

Paperno, D., Kruszewski, G., Lazaridou, A., Pham, N. Q., Bernardi, R., Pezzelle, S., Baroni, M., Boleda, G., and Fernández, R. The LAMBADA dataset: Word prediction requiring a broad discourse context. In *ACL*, 2016.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS*, 2019.

Petroni, F., Rocktäschel, T., Riedel, S., Lewis, P., Bakhtin, A., Wu, Y., and Miller, A. Language Models as Knowledge Bases? In *EMNLP-IJCNLP*, 2019.

Pimentel, T., Valvoda, J., Hall Maudslay, R., Zmigrod, R., Williams, A., and Cotterell, R. Information-Theoretic Probing for Linguistic Structure. In *ACL*, 2020.

Presser, S. and Branwen, G. A Very Unlikely Chess Game, 2020. URL `https://slatestarcodex.com/2020/01/06/a-very-unlikely-chess-game/`.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language Models are Unsupervised Multitask Learners. 2019.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419), 2018.

Tenney, I., Xia, P., Chen, B., Wang, A., Poliak, A., McCoy, R. T., Kim, N., Durme, B. V., Bowman, S. R., Das, D., and Pavlick, E. What do you learn from context? Probing for sentence structure in contextualized word representations. In *ICLR*, 2019.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is All you Need. In *NeurIPS*, 2017.

Weston, J., Bordes, A., Chopra, S., Rush, A. M., van Merriënboer, B., Joulin, A., and Mikolov, T. Towards AI-Complete Question Answering: A Set of Prerequisite Toy Tasks, 2015.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771, 2019.

Ziegler, Z. M., Melas-Kyriazi, L., Gehrmann, S., and Rush, A. M. Encoder-Agnostic Adaptation for Conditional Language Generation. *CoRR*, abs/1908.06938, 2019.

A SAN Notation

Standard Algebraic Notation (SAN) combines the piece type moved and the destination square to denote a move.[12] For example, the move in Figure 1b is represented as `Bb5` in SAN where `B` represents the piece type bishop and `b5` represents the destination square.

Standard Algebraic Notation (SAN) Ambiguity SAN notation doesn't use the starting square of the piece in its move representation. This limits the ability to prompt a SAN-based language model with specific piece type instances. For example, given the prompt "e4 e5 Nf3 Nc6 d4 h6 B" (the underlined move sequence leads to the left board state in Figure 1b), it's not clear whether the token `B` refers to the bishop at `f1` or `c1`. Due to this limitation on the specificity of probing queries, we do not use SAN for our experiments.

B Model Vocabulary

Table 8: Model Vocabulary

Type	Examples	Count
Square names	`e4`, `d1`	64
Piece type	`P, K, Q, R, B, N`	6
Promoted Pawn Piece type	q, r, b, n	4
Special symbols	BOS, EOS, PAD	3
Total		77

Table 8 shows the vocabulary used. No delimiter token is used to denote the move boundary. Tokens of promoted pawn piece type are used when a pawn gets promoted. For example, `e7e8q` denotes the move where a pawn from e7 moves to e8 and becomes a queen.

C Effect of Model Size

In this section, we present results for training larger transformer models to evaluate the impact of increase in model size with increase in training set size.

Table 9 presents results with transformer models of sizes varying from GPT2-small to GPT2-medium. We also introduce a new configuration, referred to as GPT2-intermediate, which serves as an intermediate between GPT2-small and GPT2-medium. For Train-S, GPT2-small outperforms both GPT2-intermediate and GPT2-medium on almost all evaluations. However, with increasing in training data, GPT2-intermediate and GPT2-medium are are able to outperform GPT2-small on most evaluations.

These results are along the expected lines of larger training sets alleviating the overfitting problem with larger models [Kaplan et al., 2020]. Note that we stick with the default GPT2 configuration for all our experiments. Tuning the regularization hyperparameters such as dropout, can further improve results for bigger models trained with small training sets.

D Data Statistics

Table 10 presents the statistics of the language modeling dataset used. The average game length for all splits is around 75 moves. Figure 4 presents the histogram of lengths of tokenized UCI games in Train-L.

Table 11 presents the piece type counts for the different board state prompts. All the prompts have the same game prefix i.e. the previous moves, though, the move prefix is different - starting square of the move is used for the ending square predictions while the piece type used for the move is used for the starting square prediction. As the game prefix is the same, End-Actual and Start-Actual use the same piece type for each

---

[12]For more details see `https://en.wikipedia.org/wiki/Algebraic_notation_(chess)`

Table 9: Accuracy and R-Precision (%) for predicting ending squares ("End-Actual" and "End-Other" tasks) for different model sizes. S, M, L in the first column refer to the training set sizes. GPT2-small = {12 layers, 12 heads, 768 embedding size}; GPT2-intermediate = {16 layers, 12 heads, 768 embedding size}; and GPT2-medium = {24 layers, 16 heads, 1024 embedding size}.

| | Model | LgM | | | | ExM |
| | | Actual | | Other | | |
		Acc.	R-Prec.	Acc.	R-Prec.	Acc.
S	GPT2-small	74.0	61.1	65.5	57.7	26.7
	GPT2-inter.	72.3	60.7	64.5	58.6	24.8
	GPT2-med.	67.8	58.2	62.5	55.7	24.5
M	GPT2-small	92.9	80.6	85.8	78.5	42.2
	GPT2-inter.	92.9	81.8	84.8	77.8	41.5
	GPT2-med.	93.7	81.8	86.2	77.1	41.7
L	GPT2-small	97.7	85.6	91.9	83.8	52.0
	GPT2-inter.	97.5	86.6	94.7	85.2	54.0
	GPT2-med.	98.2	87.4	94.6	85.8	57.0

Split	# of games (in 10^3)	# of moves (in 10^6)
Train-S	15	1.1
Train-M	50	3.7
Train-L	200	15.0
Dev	15	1.1
Test	15	1.1

Table 10: Statistics of the language modeling data.
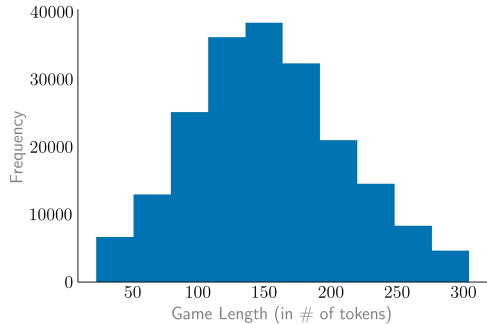
Figure 4: Histogram of tokenized game lengths for Train-L.

prompt. For the End-Other task, we pick a random starting square among all starting squares from which a legal move can be made, except the starting square used for the actual move. For the Start-Other task, we pick a random piece type among all piece types which can be legally moved, except the piece type which is actually moved in the game. The different strategies for picking the random starting square and random piece type explains the different piece type distributions for End-Other and Start-Other. Figure 5 shows the histogram of length of game prefixes (in number of moves) used in board state prompts.

E   Model Hyperparameters and Training time

Table 12 presents the hyperparameters used for the different models. For the base language model based on GPT2-small we use the default hyperparameters. For other baselines we perform separate hyperparameter grid search for Train-S and Train-M, and use the Train-M hyperparameters for Train-L. Only exception to this rule is the Reformer model, which we found particularly difficult to train, for which we explain the details next.

Reformer model uses a combination of local and LSH-based self attention layers. We borrow the attention layer configuration used for enwiki8 experiments in the original paper. [13] For both the local and LSH attention, we use a chunk length of 50 tokens - the model divides the sequence into chunks with the causal attention limited to tokens within a chunk and one before. The transformers library implementation suggests not pre-specifying the number of hash buckets. The implementation sets the number of buckets on the fly based

---

[13]https://cdn.huggingface.co/google/reformer-enwik8/config.json

Piece type	End/Start-Actual	End-Other	Start-Other
Rook (R)	358	273	197
Knight (N)	144	136	126
Bishop (B)	164	170	161
Queen (Q)	204	103	129
King (K)	130	318	387
Total	1000	1000	1000

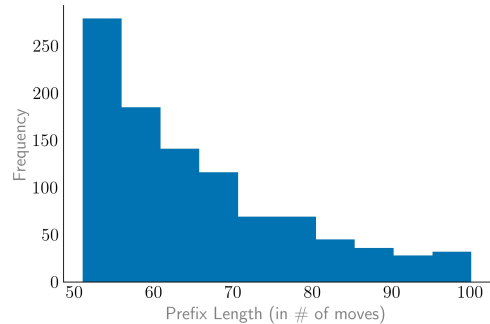Table 11: Piece type counts for ending square prediction prompts.

Figure 5: Histogram of prefix lengths of board state prompts.

Table 12: Hyperparameters used for the different models. Bold values are selected for all the training set sizes, otherwise, training set specific hyperparameter values are specified via parenthesis.

| Hyperparameters	GPT2	LSTM	Reformer	Performer
# of layers	12	3 (S), 4 (M, L), 5	12	12
# of attention heads	12	0	12	12
Embedding size	768	**768**, 1024	768	768
Hidden size	768	768, **1024**	768	768
Dropout probability	0.1	0, 0.1, **0.2**, 0.5	0.05 (0 for LSH attn.)	0.1
# of hash buckets	-	-	8	-
# rounds of hashing	-	-	1 (L), 2 (S), 4 (M)	-
Axial position shape	-	-	[14, 25]	-
Axial position embedding size	-	-	[256, 512]	-
Generalized attention	-	-	-	**Yes**, No
Feature redraw frequency	-	-	-	1000
# of local attention heads	-	-	-	0 (M, L), 6 (S)
Local/LSH attn chunk size	-	-	50	-
Local attn window size	-	-	-	50
# of parameters (in millions)	85	24 (S)/32 (M, L)	83	86

on the sequence length, which in this case it sets to 8 hash buckets. The original paper experiments with the number of hashing rounds and shows consistent improvement with more hashing rounds. However, we didn't find that to be the case, and hyperparameter tuning sometimes preferred lower number of hashing rounds. We found it particularly difficult to train the model on Train-L where the training loss started increasing after only a couple of epochs which triggered early stopping. To alleviate this: (a) we experimented with a different learning rate decay mechanism, namely, the inverse square root decay schedule which lead to slightly better final results [14], and (b) perform a separate hyperparameter tuning for Train-L. Note that all other experiments use the learning rate schedule described in Section 4 and use the hyperparameters for Train-M.

E.1 Training Time

Experiments with transformers take around 4 hrs for Train-S, less than 10 hrs for Train-M, and less than 24 hrs for Train-L on a single GeForce RTX 2080 Ti. For LSTMs it takes less than 2 hrs for Train-S, less than 4 hrs for Train-M, and less than 8 hrs for Train-L on a single GeForce RTX 2080 Ti.

[14]https://fairseq.readthedocs.io/en/latest/_modules/fairseq/optim/lr_scheduler/inverse_square_root_schedule.html
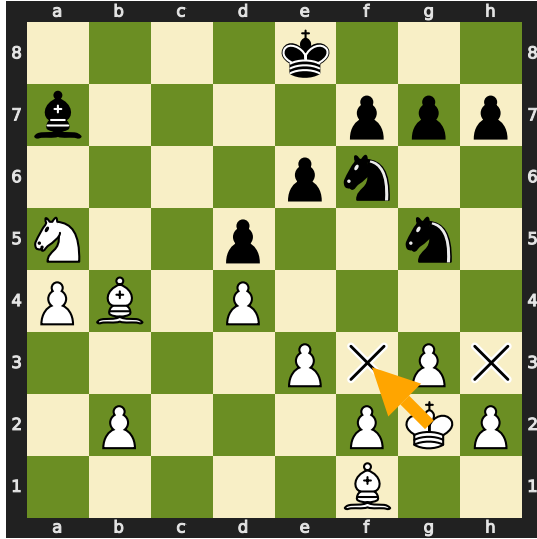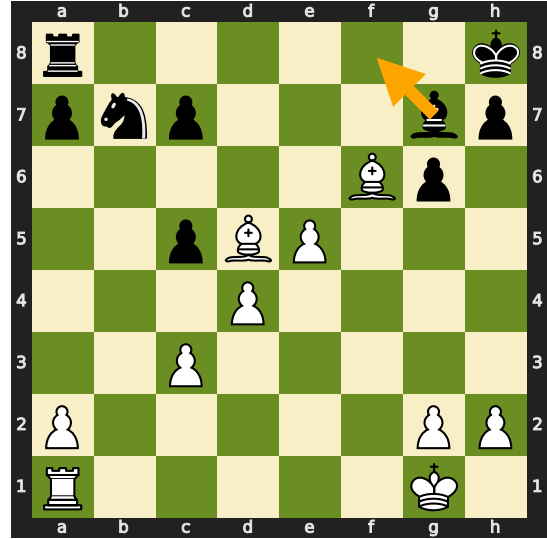
F   Detailed Error Analysis
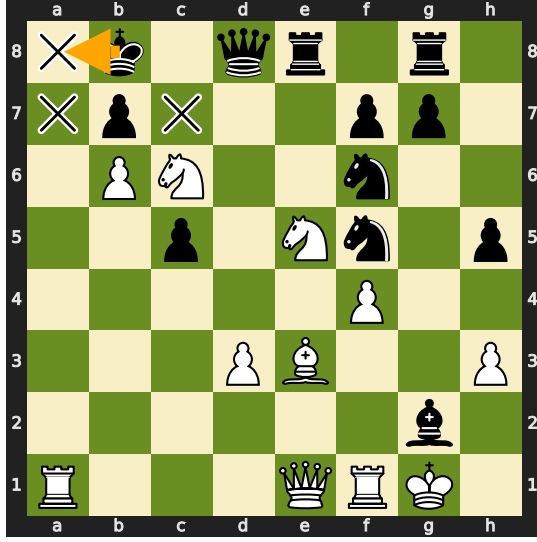
(a) *Check + King*: Black king is in check and the predicted ending square is already covered by the white rook on `a1`.
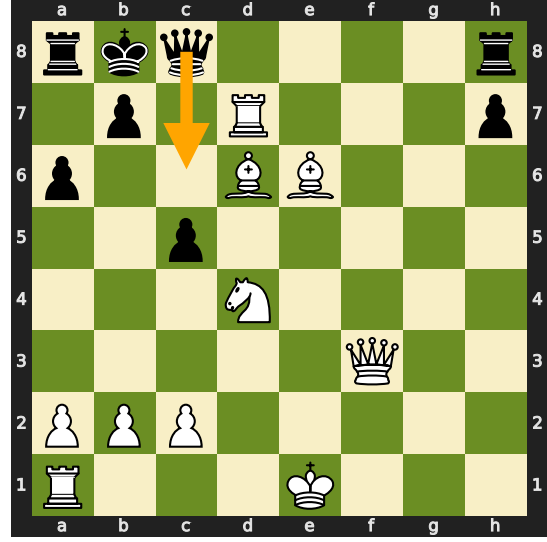
(b) *Check + Other*: Black king is in check and the only legal move for the black queen is `c7` but the model predicts `c6`.

(c) *No Check + King*: The predicted ending square `f3` for the white king is guarded by the black knight at `g5`.

(d) *No Check + Other*: The predicted ending square `f8` for the black bishop exposes its king to the white bishop at `f6`.

Figure 6: Four combinations of the king being in check or not, and if the king is moved or not, that can result in Pseudo Legal errors.

In this section we conduct a more in-depth analysis of errors made by the UCI model trained with Train-L for the ending square prediction task. We limit our focus to the two main error categories, namely, Pseudo Legal and Path Obstruction.

Table 13: Pseudo Legal error counts for different categories. For the total column we remove instances with errors of other category.

Category	End-Actual		End-Other	
	Errors	Total	Errors	Total
Check + King	1	27	2	20
Check + Other	7	26	16	33
No Check + King	4	101	31	296
No Check + Other	0	835	0	619
Total	12	989	49	968

Table 14: Piece type counts for Path Obstruction error category. For the total column we remove instances with errors of other category.

| Piece type	End-Actual		End-Other	
	Errors	Total	Errors	Total
Rook (R)	3	355	17	267
Knight (N)	1	144	1	131
Bishop (B)	1	162	3	164
Queen (Q)	4	202	4	99
King (K)	1	124	1	284
Total	10	987	26	945

F.1 Pseudo Legal Errors

We conduct our analysis by categorizing instances according to: (a) if the king was in check before the current move, and (b) if the king is being moved in the current move. Figure 6 presents one instance for each of these four categories. Table 13 presents the breakdown of errors for the End-Actual and End-Other instances. The key takeaways from the error categorization are: (a) Error counts for "Check + King" and "No Check + Other" are relatively low and similar across the two classes of prompts. (b) "Check + Other" i.e. the king is in check and some other piece is moved, has high count for both the splits. The particularly high count for End-Other could be explained by the lack of "meaningful" moves for certain prompts of this kind. For example, in figure 6b the prompt asks for the queen at c8 to move, and the only legal continuation is for the queen to bring itself to the firing line at c7. (c) "No Check + King" is another common error category. The significantly higher error count for End-Other could be due to a combination of the higher frequency of such prompts and the out-of-distribution prompts.
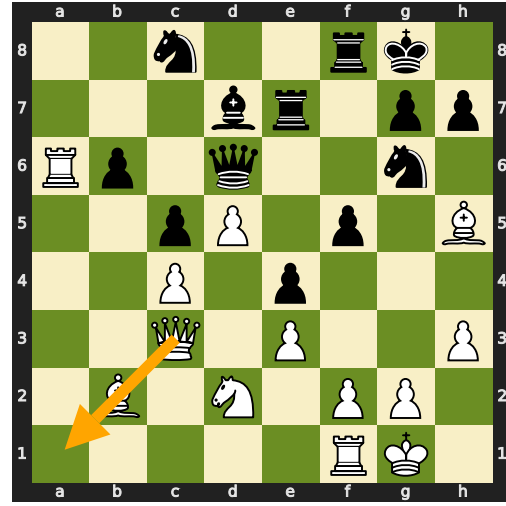
F.2 Path Obstruction

Table 14 presents the path obstruction errors for different piece types for the End-Actual and End-Other task. Figure 7 represents instances of path obstruction error for different piece types. The error counts show that piece types with more dynamic movement except knight i.e. rook, bishop, and queen, have more path obstruction errors (a knight just needs to avoid landing on its own piece to avoid path obstruction errors). These piece types also show a significant increase in frequency of path obstruction errors for End-Other in comparison to End-Actual. As in pseudo legal errors, this could again be due to the out-of-distribution nature of these prompts. Figure 8 shows that the average path length for predicted moves with path obstruction error is significantly higher than the legal predictions for both the kinds of prompts (knight and king have a constant path length). [15]
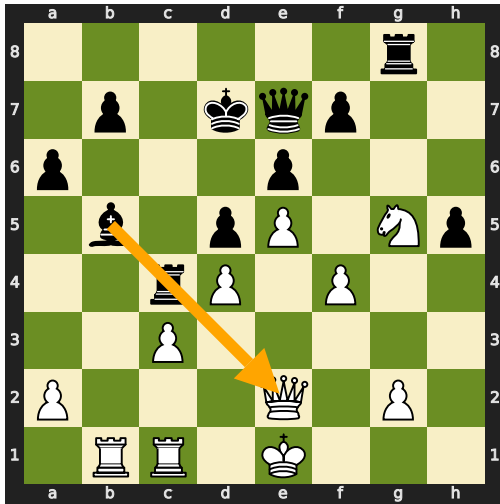
---

[15] Path length is measured in number of king moves i.e. number of moves it will take a king to go from starting to ending square.
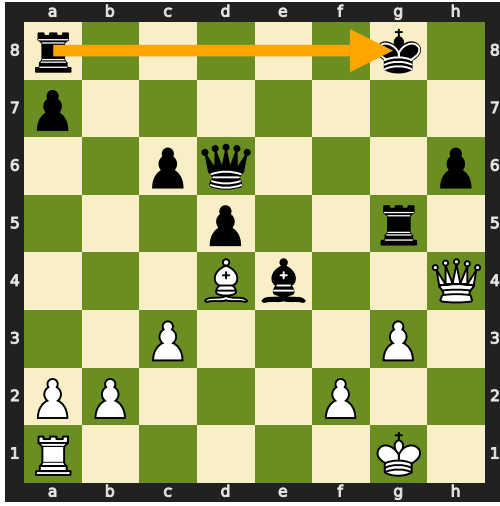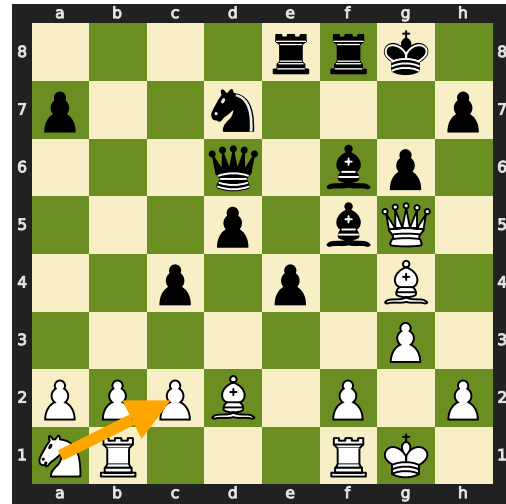
(a) Rook forgets about its own king at g8!

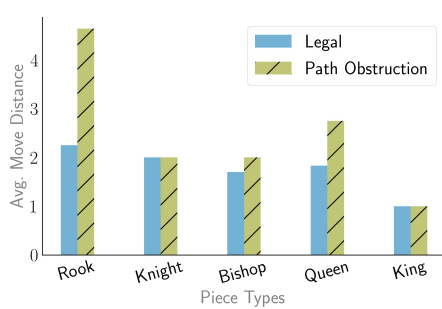(b) Bishop at b2 stands in the way of the queen.

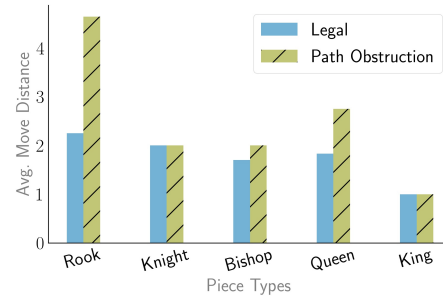(c) Bishop forgets reality in pursuit of fantasy queen kill!

(d) A trapped, frustrated knight is out to kill its own pawn!

Figure 7: Instances of Path Obstruction errors with different piece types.

(a) End-Actual

(b) End-Other

Figure 8: Comparison of average path length of predicted moves for different piece types when the move is legal vs ones with path obstruction error.