Building a Natural Language Chess Engine with Pretraining and Instruction Fine-Tuning

Stanford CS224N Custom Project

Bowen Jiang
(Mentored by Jesse Mu)
Department of Bioengineering
Stanford University
name@stanford.edu

Abstract

Although pretrained large language models (LLMs) can generate convincing natural language about games like chess, they lack positional and contextual knowledge and as such are poor game-playing agents. In this project, I utilize language pretaining; instruction fine-tuning, an additional training regimen with chess-specific tasks presented in natural language; and chain-of-thought prompting, a natural language description of problem reasoning prepended to the answer of a problem, to improve the performance of LLMs at chess move generation (validity/legality and quality of moves). I show that fine-tuned GPT-2-XL, a 1.5B parameter LLM, performs favorably well at move generation compared to ChatGPT with few-shot learning; I also validate the additional benefits of chain-of-thought prompting compared to plain prompts in ChatGPT while highlighting tradeoffs between the quality of natural language and the quality of chess when more verbose prompts are used in the smaller GPT-2-XL.

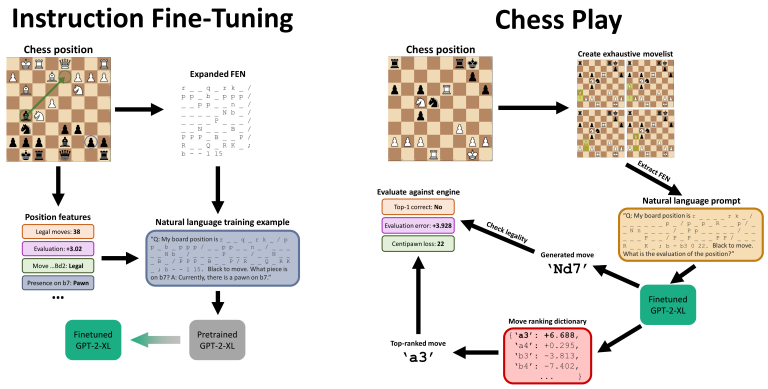


Figure 1: Graphical abstract of model during training and test time (chess play).

1 INTRODUCTION

Chess is a landmark domain for deep learning research because of its latent structure which lends itself well to a diversity of modeling paradigms. Although the majority of advances in state-of-the art chess play have come from deep reinforcement learning models like AlphaZero (Silver et al., 2018), natural language processing (NLP) researchers have also attempted to build systems for chess, leveraging and adding to a rich history of chess commentary, annotation and literature (Winter, 2022). Jhamtani et al. (2018) utilized a database of 300k move-commentary pairs across over 11k games to train a chess commentary engine which would generate text describing a chess move. Kamlish et al. (2019) used the same dataset to create a chess engine by training a sentiment classifier on the move comments, finding that the resultant model could outperform random agents and a depth-one chess engine. Both of these early works utilized chess-specific models which utilize extracted board features and long short-term memory (LSTM) networks to build representations for or generate text.

Recently, large language models (LLMs) such as ChatGPT which utilize the attention-based Transformer architecture introduced by Vaswani et al. (2017) have become the gold standard for NLP applications. The paradigm for NLP problem-solving has also broadly

shifted from the construction of application-specific networks towards the use of large multi-billion-parameter models with general language modeling pretraining, which have achieved state-of-the-art performance on the majority of NLP tasks. In addition, the most recent largest LLMs demonstrate emergent abilities, which Wei et al. (2022a) defines as a an ability which is "not present in smaller models but is present in larger models", that include tasks beyond natural language like performing arithmetic or writing code. In many cases, models are able to perform these tasks with no additional gradient updates, when provided with just a few exemplars (few-shot learning) or even just a question in natural language (zero-shot learning). In the chess domain, ChatGPT demonstrates some knowledge of chess and its symbolic representations, and is capable of generating moves in algebraic notation and evaluating chess positions in a zero-shot and few-shot manner. However, ChatGPT and LLMs in general are largely incapable of actually playing chess, as they will routinely suggest illegal or impossible moves in their outputs, incorrectly identify pieces on squares or misname common strategies, sometimes with hilarious effect (Rozman, 2023).

My contribution to this area investigates the extent to which general language pretraining, as well as advances in LLM prompting and fine-tuning, can be used to create a small natural language chess engine. Instruction fine-tuning, introduced by Wei et al. (2021), aims to enhance an LLM's zero-shot performance at unseen tasks by fine-tuning it with an additional dataset of related natural language problem-solving tasks. Chain-of-thought prompting was first invented by Wei et al. (2022b), and consists of prepending a description of problem-solving steps to the answer of few-shot exemplars to improve LLMs' few-shot learning performance. In this project, I transform features extracted from a chess board representation into natural language question-answer pairs in utilizing both instruction fine-tuning and chain-of-thought prompting strategies, to this date **a novel application** of these techniques. I then train a 1.5B-parameter LLM on these question-answer pairs and investigate its performance at position evaluation and move generation, which I benchmarked against elementary chess heuristics, random agents, and ChatGPT with few-shot learning.

2 EXPERIMENTS

2.1 DATASET AND PREPROCESSING

Instruction	Description	Count	Example
`movecount`	Return the number of legal moves in a given position	1	Q: My board position is {Expanded FEN}. White to move. **How many legal moves does White have?** A: From this position, White has 20 legal moves.
`square`	Return the name of a piece/pawn if present on a randomly-chosen square	3	Q: My board position is {Expanded FEN}. White to move. **What piece is on a8?** A: Currently, there is a rook on a8.
`legal`	Tell whether a randomly-chosen move in algebraic notation is legal	4	Q: My board position is {Expanded FEN}. White to move. **Is Kc2 an acceptable next move?** A: No, Kc2 is illegal.
`nextpos`	Return the board state in expanded FEN notation after a move has been made	1	Q: My board position is {Expanded FEN}. White to move. **White plays Kxc4. What is the next board position?** A: The next board position is {Expanded FEN}.
`response`	Produce a short description of a randomly-selected legal move	3	Q: My board position is {Expanded FEN}. White to move. **What is a possible response for White?** A: Moving the rook from a1 and checking the king. White plays Re1+.
`ischeck`	Tell whether a player is in check in the given position	1	Q: My board position is {Expanded FEN}. White to move. **Is White in check?** A: No, White is not in check.
`ismate`	Tell whether a player has forced checkmate, and if so, in how many moves	1	Q: My board position is {Expanded FEN}. White to move. **Is there a forced mate on the board?** A: No, White does not have forced mate.
`eval`	Return the engine evaluation of a given position	1	Q: My board position is {Expanded FEN}. White to move. **What is the evaluation of the position?** A: This position is +7.63 from White's perspective. White has a significant advantage with best play. Black has a small material disadvantage.
`best`	Return the engine-suggested next best move of a given position	0.02	Q: My board position is {Expanded FEN}. White to move. **What is the next best move for White?** A: White moves the bishop to c4. White should play Bc4.
`best-ctp`	Return the engine-suggested next best move of a given position with a longer manually-generated description	-	Q: My board position is {Expanded FEN}. White to move. **What is the next best move for White? Let's work through this step by step.** A: White takes more space on the kingside and prepares to harass the Black bishop with f5. White should play f4.

*Table 1: Details on chess-specific instructions generated from each board position. Here, **Count** refers to the number of prompts generated per board position;* `best` *training examples were generated at random in approximately 1 out of every 50 positions.* `response`, `eval`, `best` *and* `best-ctp` *examples had more diverse natural language descriptions which were created with probabilistic rulesets or through manual annotation.*

I utilized the `lichess.org` open database to obtain chess games for model training. In total, this database contains over 4.2 billion rated games played on the `lichess.org` website starting in January 2013. The games are stored in PGN format, containing metadata such as player skill level, time control, date and time of the game and the result, as well as a movelist representation of the game

encoded in algebraic notation. In about 6% of games, there are additional move-level evaluations of the game provided by a chess engine. I created training data using the file of games from April 2017 (which in total number about 11.35 million). Ultimately, the first 1000 games were used to generate data for instruction fine-tuning and evaluation-specific fine-tuning; an additional 100 games were used to create validation and test sets. A total of 12857 games were used to create train, validation and test sets for next-best-move prediction, and ~70 games were used to find interesting positions for the manual annotation of a small number of next-best-move examples.

For instruction fine-tuning, I manually created natural-language scaffolds for nine chess-specific tasks (eight of which were used for general training), which are detailed in Table 1. For each task, a prompt-response pair in natural language comprise a single training example. The prompt consists of the task-specific question, along with a chess board position encoded in a modified Forsyth-Edwards Notation (FEN) to improve tokenization. In this modified notation, each square on the chess board receives a corresponding character (unlike standard FEN which uses an integer ranging from 1 to 8 inclusive to enumerate adjacent empty squares); each square character is also space-delimited, and the board representation is explicitly separated from position metadata by a semicolon (Appendix Figure A1). The Python `chess` library was used to extract features from the board position to produce the response, which was converted into natural language manually with a small task-specific ruleset. Training examples for all tasks were generated from every board position for each game in the dataset according to the quantities given in Table 1, which ultimately yielded 954999 examples (across all tasks) in the training set and 92785 in the validation set (of which 5000 were selected for model development to improve the pace of computing validation loss). The evaluation-specific fine-tuning dataset consisted only of prompt-response pairs for the position evaluation task, which numbered 58523 examples in the training set and 5528 in the validation set. A total of 4000 examples were used to train models for next best move prediction, 80 of which (2%) were manually generated with chain-of-thought styled prompts in a semi-supervised fashion and the remainder of which were generated using a simple ruleset. To maximize the diversity of positions seen, positions for generating next-best-move examples were randomly sampled from games at the rate of 1 position per 50 plies.

2.2 MODEL AND TRAINING

Model. In my experiments, I used the HuggingFace implementation of GPT-2-XL, a Transformer-based decoder-only language model with 1.5 billion parameters in 48 layers with 25 attention heads each (Radford et al., 2018). The pretrained instances of this model were trained on a causal language modeling objective using the ~40GB WebText dataset, consisting of the text from non-Wikipedia links within all Reddit posts with ≥ 3 karma. The corresponding tokenizer has a vocabulary size of 50267 and was not finetuned for chess-specific use.

Instruction, evaluation-specific and next-best-move fine-tuning. The instruction fine-tuning procedure used was adapted from Wei et al. (2021), except that no task-specific dataset balancing was performed beyond the setting of training example proportions as detailed in Table 1. All models were finetuned for a maximum of 6k steps with the Adafactor optimizer (Shazeer and Stern, 2018). For models receiving their first round of chess-specific training, the learning rate was set at 3e-5; models receiving additional training were finetuned with a learning rate of 3e-6. A linear warmup of 1k steps was used with no additional learning rate scheduling. Training example lengths were capped at 256 (except for next-best-move generation, which had maximum lengths of 1024), for which a batch size of 2 was used with 8 gradient accumulation steps, resulting in an effective batch size of 16. Validation loss was computed every 250 steps, and was used to determine when early stopping should be performed. Model checkpoints were saved every 500 steps. With the full 6k steps performed, fine-tuning took approximately 6 hours on a VM with 32GB RAM and 1x NVIDIA A10G GPU (24GB).

2.3 BASELINES AND EVALUATIONS

Models trained. In total, I trained seven model instances in my experiments:

- IF: No pretrained language modeling weights, one round of instruction fine-tuning with eight tasks
- EVAL: No pretrained language modeling weights, one round of evaluation-specific fine-tuning
- IF+EVAL: No pretrained language modeling weights, one round of instruction fine-tuning followed by an additional round of evaluation-specific fine-tuning
- PT+IF: Pretrained language modeling weights, one round of instruction fine-tuning with eight tasks
- PT+EVAL: Pretrained language modeling weights, one round of evaluation-specific fine-tuning
- PT+IF+EVAL: Pretrained language modeling weights, one round of instruction fine-tuning followed by an additional round of evaluation-specific fine-tuning
- PT+IF+NEXT: Pretrained language modeling weights, one round of instruction fine-tuning with eight tasks followed by an additional round of next-best-move fine-tuning (unseen task)

Evaluating position evaluation quality. A test set of 500 randomly-selected examples was used to evaluate the models' ability to evaluate chess positions. Position evaluations are given by a float representing the pawn advantage from the perspective of the White player. An evaluation of 0 indicates a drawn position; a score of +1 represents a position equivalent to a one-pawn advantage for White, and a score of -3 reflects a position equivalent to a three-pawn advantage for Black. In the case where there is a forced checkmate on the board, the pawn-advantage score is undefined; instead, an integer is used to indicate the number of moves until

checkmate is reached, where positive integers indicate a forced checkmate for White and negative integers for Black. To avoid complications with evaluations on two different scales, the 500 positions selected do not include any positions where forced mate is a possibility.

For each example, a prompt was constructed using the `eval` task template and given to the model to complete. 5 replicates per prompt were generated through a combination of top-k and nucleus sampling with k = 50, p = 0.95 and a maximum length of 110. A simple ruleset was used to extract evaluations from the model outputs; all successfully-parsed evaluations were then averaged to give a final evaluation for each position. The error in position evaluation was determined by taking the difference between the averaged position evaluation generated by the language model and the position evaluation given by the lichess database for that move (likely generated by Stockfish 14 at depth > 20, although this cannot be verified).

For this task, I defined the MAT baseline as the position evaluation calculated by using a material-only heuristic, namely the difference between the sum of the values of all White and Black pieces on the board. The AlphaZero relative piece values as described in Tomašev et al. (2020) - pawn = 1, knight = 3.05, bishop = 3.33, rook = 5.63, queen = 9.5 - were used for this calculation.

Evaluating chess-playing quality with exhaustive position evaluation. Because of the additional time complexity, only two models (IF+EVAL and PT+IF+EVAL) and the first 100 of the 500 positions in the above test set were used to further characterize models' chess playing abilities. For each example, the Python `chess` library was used to generate all legal moves for the player to move. Successively, each of the legal moves was made, and the resultant position was then used to create an `eval` task prompt for the model to complete. As above, 5 replicates per prompt were sampled (with the same hyperparameters), parsed and averaged to give an evaluation of the position after each legal move. The legal moves were then sorted based on the evaluations of their resultant positions; for White, the best moves have the largest positive evaluations, whereas for Black, the best moves have the largest negative evaluations. Stockfish 15.1 at depth 24 was used to generate ground truth best moves for each position. The models were evaluated on top-1 accuracy, the proportion of positions in which the model-suggested move matched the Stockfish move exactly; top-3 accuracy, the proportion of positions in which the Stockfish move was one of the top 3 moves ranked by the model; and average centipawn loss, the average difference in the evaluation of the position after the Stockfish move versus after the top model-suggested move. (Note that the equivalent centipawn loss is a multiple of 100 greater than the corresponding pawn advantage used in the position evaluation experiments above.) To avoid artefacts arising from generating evaluations at different positions in the game, and blundered or missed mates (whose evaluation is undefined on the centipawn scale), the centipawn loss of a given move was clipped to between 0 and 1000; a maximum possible centipawn loss of 2000 has been used by existing systems for game evaluation (Indrajatin, 2022), but a lower ceiling was defined here to reduce the variance of this metric. This was justified with the findings of Coulombe (2017) that average centipawn loss correlates poorly with the ELO rating of human chess players.

For this task, I defined the RAND baseline in which the "top" move for was selected at random from the set of legal moves for a given position. Top-1 accuracy in this context represents the expected proportion of positions in which the generated move matched the Stockfish move exactly; top-3 accuracy is the proportion of positions in which a randomly-selected set of 3 moves contains the Stockfish move. To calculate average centipawn loss, I used Stockfish 15.1 at a depth of 15 to evaluate the positions after each legal move was made; the centipawn losses for each position (relative to the top Stockfish move) were then averaged.

Evaluating chess-playing quality with next-best-move generation. Once again, the first 100 of the 500 positions in the above test set were used to further characterize models' chess playing abilities. A single model was evaluated (PT+IF+NEXT), but under two generation schemes differing in their prompts: the outputs of PT+IF+NEXT_PLAIN were generated with a `best` prompt, whereas the outputs of PT+IF+NEXT_CTP were generated with a `best-ctp` prompt (including the additional phrase "Let's work through this step by step."). As above, 5 replicates per prompt were sampled (with the same hyperparameters, except using a new maximum generation length of 400), and a simple ruleset was used to extract the first move in algebraic notation from each generation. The models were evaluated on legality proportion, the proportion of generated moves which were legal for the given position; top-5 accuracy, the proportion of positions in which the Stockfish move was one of the 5 moves generated by the model; and average centipawn loss, the average difference in the evaluation of the position after the Stockfish move versus all legal moves generated for that position, once again clipped to the range [0, 1000]. Top-1 accuracy was calculated conditional on the known top-5 accuracy (the expected number of positions in which the top Stockfish move would be selected from the set of legal moves). Additionally, I computed the frequency in the dataset of the top 1 and top 3 most commonly suggested moves (legal or illegal) in algebraic notation relative to all generated moves to serve as a proxy of the diversity of the generated distribution.

For this task, I defined two baselines: ChatGPT+10shot_PLAIN, an instance of ChatGPT conditioned on 10 `best` task exemplars and evaluated on the 100 test positions with `best` prompts; and ChatGPT+10shot_CTP, a separate instance of ChatGPT conditioned on the same 10 positions but with `best-ctp` task exemplars containing manual position annotations, and evaluated on the 100 test positions with `best-ctp` prompts. A single output was generated using the default ChatGPT interface, and all suggested moves from this single output were parsed. In this context, k for top-k accuracy was potentially variable amongst different examples, but top-1 and top-5 accuracies were extrapolated with the assumption of uniformly sampling at random 1 or 5 legal moves from the output for each position; all other evaluation metrics remained the same.

Statistics. Simple pairwise tests were used to compare models based on their evaluation metrics; p-values and the statistic type are mentioned for their corresponding test. Significance was set at $FWER = 0.05$ and the individual test threshold was adjusted with Bonferroni correction.

3 RESULTS

3.1 MODEL TRAINING

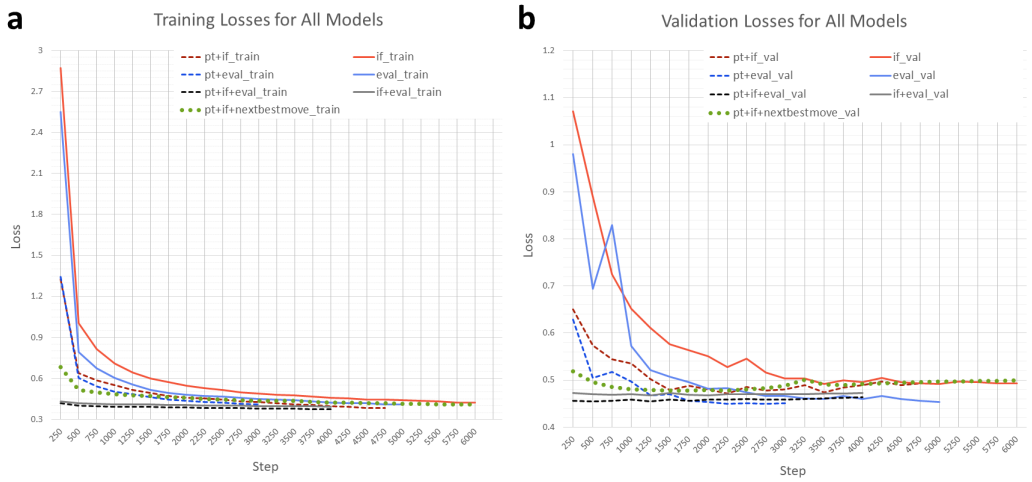


Figure 2: Training and validation loss curves for all seven fine-tuned model instances.

The training and validation curves as shown in Figure 2 indicated that both learning and convergence had taken place for all seven model instances trained in my experiments. As expected, the two models trained from scratch (IF and EVAL) had the largest initial training and validation losses and required the longest time to converge (6k gradient steps). The models being finetuned on chess-specific data for the first time (PT+IF and PT+EVAL) began training much more rapidly, and both converged by 3k gradient steps. The IF+EVAL and PT+IF+EVAL models, which were finetuned on a second round of chess-specific data, experienced only marginal or stochastic improvements in validation loss over the the first 2000 gradient steps. Finally, PT+IF+NEXT experienced a moderate amount of learning, as the task of next best move prediction was not included in the original instruction fine-tuning dataset, but also converged within 2000 gradient steps. The rapid convergence in validation loss on the causal language modeling objective at a rather high floor of around 0.45 validation loss across all models (when only a fraction of an 1 epoch of training data had been seen) suggested that they quickly gained the ability to produce coherent responses (i.e. learning the ruleset that generated the natural language training examples), but were not able to make meaningful progress on matching the distribution of chess-specific tokens without overfitting.

Model	Parsable evals	Evaluation Mean	Evaluation SD	Error Mean	Error SD
IF	195/2500 (7.8%)	1.39	3.22	2.81	12.76
EVAL	1308/2500 (52.32%)	-3.60	**4.50**	-1.71	11.45
IF+EVAL	496/2500 (19.84%)	-3.17	4.22	**-0.92**	12.62
PT+IF	**2500/2500 (100%)**	1.16	2.89	2.77	10.58
PT+EVAL	**2500/2500 (100%)**	0.78	2.93	2.39	9.50
PT+IF+EVAL	**2500/2500 (100%)**	0.22	2.74	1.83	10.39
MAT	–	**-0.46**	3.27	1.15	**9.00**
GT	–	-1.61	10.18	–	–

*Table 2: Performance of six models at position evaluation, compared to the material heuristic and ground truth distributions. **Parsable evals** enumerates the number of sampled outputs for each model from which an evaluation string could be parsed with the same ruleset used to generate training set data. **Evaluation mean** is the average evaluation across all 500 positions, where the evaluation for each position was determined by taking the average of all evaluations that were successfully parsed from each of the 5 outputs generated per position. The mean was calculated from all positions containing at least one parsable evaluation. **Evaluation SD** gives standard deviation of all evaluations generated for a given position averaged across all positions containing at least two parsable evaluations; for the MAT and GT baselines, **Evaluation SD** was simply the standard deviation of all position evaluations averaged across all 500 positions. **Error mean** is the average difference between the evaluation for each position averaged across all model outputs, and the evaluation for the position given by lichess comments. **Error SD** is the standard deviation of the per-position evaluation error.*

3.2 POSITION EVALUATION

Table 2 shows the results of six models' evaluation of 500 chess positions, as averaged from the outputs of 5 sampled completions of a `eval` task prompt containing the board position. As evidenced by the proportion of parsable evaluations, general language

pretraining appears instrumental to generating consistently-parsable evaluations. However, pretraining appears to also reduce the alignment between model outputs and ground truth evaluations (GT) as seen in the increased error mean between average position evaluation generated by the pretrained models and those of Stockfish, although this effect appears to be ameliorated with increased chess-specific fine-tuning. The material-only evaluation baseline appears to work best for aggregate evaluation of positions as evidenced by the minimal distance to the mean of Stockfish evaluations; while the absolute mean error in predictions is bested slightly by IF+EVAL, this difference is not statistically significant (p = 0.78, two-tailed Student's *t*) due to the large standard deviation in evaluation error. There is thus no evidence to suggest that any of the models outperforms material-based heuristics for evaluating chess positions.

3.3 EXHAUSTIVE MOVE-RANKING

Table 3 shows the results of two models' move suggestions by exhaustively evaluating and ranking all legal moves in 100 positions, compared to a random move selection baseline. All models have a similar top-1 accuracy; both language models have a worse top-3 accuracy than the random baseline, but this difference is not significant (p = 0.12, 0.25 for IF+EVAL and PT+IF+EVAL respectively, Fisher's exact test). Similarly, IF+EVAL and PT+IF+EVAL achieve a slightly worse but statistically insignificant average centipawn loss as compared to the random baseline (p = 0.43, 0.57, two-tailed Student's *t*). In conclusion, the models demonstrate performance roughly equivalent to randomly selecting moves to play.

Model	Top-1 Accuracy	Top-3 Accuracy	Mean Centipawn Loss
IF+EVAL	**4%**	11%	303.34
PT+IF+EVAL	**4%**	13%	295.94
Random	**3.80%**	**19.53%**	**276.18**

Table 3: Performance of two models at exhaustive move evaluation compared to the random baseline and ground truth distributions.

3.4 NEXT-BEST-MOVE GENERATION

Model	Legal	Top-1 Freq.	Top-3 Freq.	Top-1 Acc.	Top-5 Acc.	Mean Centipawn Loss
PT+IF+NEXT_PLAIN	130/500 (26%)	**14/500 (2.8%)**	38/500 (7.6%)	**0.8**	**4**	303.31
PT+IF+NEXT_CTP	105/500 (21%)	17/500 (3.4%)	**40/500 (8%)**	0.6	3	254.35
ChatGPT_10shot_PLAIN	23/113 (20.35%)	12/113 (10.62%)	30/113 (26.55%)	0	0	226.64
ChatGPT_10shot_CTP	**68/183 (37.15%)**	16/183 (8.74%)	39/183 (21.31%)	0	0	**210.57**
Stockfish 15.1 depth 24	–	3/100 (3%)	8/100 (8%)	–	–	–

Table 4: Performance of the PT+IF+NEXT model and ChatGPT at next-best-move generation under with two different styles of prompting/text generation conditioning compared to the Stockfish 15.1 depth 24 baseline. ***Legal*** *refers to the proportion of all moves generated across 100 positions that were legal for the given board position.* ***Top-1 Freq.*** *refers to the count of the most commonly-generated move +out of all total moves generated.* ***Top-3 Freq.*** *refers to the sum of frequencies of the three most commonly-generated moves out of all total moves generated.*

Table 4 shows the results of using the PT+IF+NEXT model and two separate ChatGPT baselines for suggesting the next best chess move to play in 100 positions. As has been previously documented, all models struggle to consistently produce legal moves, with even the best-performing model suggesting a legal move in fewer than half of all generated outputs. The benefits of chain-of-thought prompting on the legality of outputs are mixed; in the GPT-2-XL models, there was no evidence to suggest it improved the legality of moves (p = 0.97, one-tailed proportion *t*), but chain-of-thought-styled few-shot examples almost doubled the proportion of legal moves generated by ChatGPT for the same positions (p = 1.17e-3, one-tailed proportion *t*), although it is unclear if this benefit is attributable to more expressive board commentary or the explicit "Let's work through this step by step" prompt. ChatGPT_10shot_CTP also consistently produced longer, more verbose descriptions and suggested a greater number of moves per position (and thus more total moves) as compared to the instance conditioned on shorter examples; however, both ChatGPT baselines consistently generated paragraphs of text describing the position (albeit often incorrectly) regardless of prompt type. PT+IF+NEXT never generated natural language outside of the ruleset in the PLAIN objective, although it attempted to do so (creating short 1-2 sentence descriptions of the position or move) in at least 22 of the 500 total samples (4.4%) generated under the CTP objective (as detected by ruleset-based parsing debugging), significantly more than the expected proportion from the training data (p = 6.47e-4, two-tailed proportion *t*).

All models suffered from low top-k accuracy which rendered direct model comparisons impossible, especially with the ChatGPT baselines for which only one response (although possibly multiple candidate moves) was sampled per position. There was no significant effect of chain-of-thought prompting on ChatGPT, which in both few-shot regimens failed to produce a single move that matched the top Stockfish recommendation. In total, the top-5 accuracy of moves generated by PT+IF+NEXT in both the PLAIN and CTP objectives was about equal to the top-1 accuracy of the random agent, although a direct comparison is complicated by

conditionality constraints because of the possibility of PT+IF+NEXT generating illegal or nonsensical moves. Using mean centipawn loss as the metric, the accuracy of (legal) moves generated by PF+IF+NEXT_PLAIN almost exactly matched that of the top move suggested by IF+EVAL, both of which underperformed relative to the random agent. However, chain-of-thought prompting appeared to improve the model alignment to the Stockfish baseline, albeit without statistical significance ($p = 0.11$, one-tailed t-test). The best-performing models were the two instances of ChatGPT, both of which achieved an average centipawn loss of less than 250. Again, a slight but statistically insignificant benefit was observed with the use of chain-of-thought prompting ($p = 0.28$, one-tailed t-test). The single best-performing model, ChatGPT_10shot_CTP, did obtain a mean centipawn loss that was substantially lower than the random baseline ($p = 0.014$, one-tailed t-test); however, after adjusting for multiple hypothesis testing, this difference was not statistically significant.

As the final part of evaluating the models' playing utility, I compared the frequencies of moves generated by each model across all 100 positions in the test set to the true distribution of optimal moves. The proportions of all moves that belonged to the sets of top 1 and top 3 most commonly-suggested moves by PT+IF+NEXT for both generation objectives perfectly matched the expected frequencies of top 1 and top 3 most frequent optimal moves as determined by Stockfish. On the other hand, there was significant evidence to suggest that the distribution of moves generated by ChatGPT was less diverse than the true distribution of optimal moves; in particular, the top 3 most commonly-suggested moves under both generation objectives constituted a greater proportion of total generated moves than the corresponding 3 most common Stockfish moves for the same positions ($p = 2.09e\text{-}4$, $2.01e\text{-}3$ for PLAIN and CTP respectively, one-tailed proportion t).

In conclusion, the tested language models are able to generate chess moves from a given board position with varying degrees of success. There is some limited evidence suggesting the ability of the highest-performing models to slightly outperform a random agent at move selection, albeit with a less diverse move distribution. Chain-of-thought prompting appears to affect models to different degrees based on their size; although engineered prompts may help larger models may generate more legal moves that are better aligned with engine evaluations, there appears to be a diversifying effect on the outputs of smaller models that reduces legality. Overall, the exact magnitude of the effects of chain-of-thought prompting on move accuracy are still unclear.

4 RELATED WORK

My work is well-contextualized at the intersection of several NLP research areas such as natural language generation, finetuning, prompting, and emergent abilities of large language models. Here I describe three major avenues of research that relate most closely to this project.

The choice of a Transformer-based architecture for a chess engine has precedent in several studies (Noever et al., 2020; DeLeo and Guven, 2022); in particular, the approach of DeLeo and Guven (2022) is quite similar to mine, utilizing a BERT-like model that operates over games represented as sequences of FEN strings instead of individual algebraic move tokens, and is trained with Stockfish self-play data. My project builds on this previous work by incorporating additional modalities into both model inputs and outputs, forcing the model to simultaneously learn representations for FEN positions, moves in algebraic notation, and natural language. As of yet, a language model trained to play chess in natural language has not been published.

My instruction fine-tuning approach is largely based on the work of Wei et al. (2021), who show that fine-tuning a language model with a diverse set of NLP tasks greatly improves its zero-shot and few-shot learning abilities on unseen tasks. However, they note that "performance improvements from instruction tuning emerge only with sufficient model scale," consistent with other studies that have characterized the emergent abilities of large language models (Wei et al., 2022a). My project attempts to adapt the instruction fine-tuning strategy to a parameter-efficient regime through the use of domain-specific training tasks for "targeted tuning" and the inclusion of the target task in the fine-tuning regimen to compensate for decreased model size (two orders of magnitude smaller than the model finetuned by Wei et al. (2021)) and complexity of the target domain, the vast majority of the information about which is contained in non-natural-language representations.

Another closely-related avenue of research into the use of building language model reasoning capabilities is chain-of-thought prompting. Wei et al. (2022b) showed that a small number of question-answer pair exemplars which contain natural-language descriptions of the problem-solving process prepended to the answer can be used to prompt LLMs, exceeding the performance of standard prompting and occasionally even supervised task-specific fine-tuning. In my project, I attempt to evaluate the effect of chain-of-thought prompting on the few-shot learning performance of ChatGPT in the chess domain, which is known to be highly error-prone. Additionally, I experiment with the use of chain-of-thought prompting styled training data to fine-tune smaller language models, with a particular focus on combining gradient updates with a semi-supervised learning strategy to probe the extent to which this technique can be used at smaller, pre-emergent model scales.

5 DISCUSSION

In this paper, I have explored several strategies to produce chess-playing capabilities in a small language model by using a combination of symbolic chess board representation, natural language, and constructed datasets informed by newly-developed LLM prompting strategies. I first observed that general language pretraining, chess-specific instruction fine-tuning and task-specific fine-tuning all helped improve the consistency and quality of board evaluation predictions, although none of my tested models outperformed a simple material-based evaluation heuristic or random agents for move selection. The fact that all of the models converged within a fraction of an epoch and exhibited very high standard deviations for average position evaluation, evaluation error and centipawn

loss suggested that they were able to quickly learn the simple natural language scaffolds in the training examples, but generated the critical chess-related tokens almost at random and did were not able to learn them further. Next, I showed that a small language model fine-tuned on a next-best-move generation task reaches a playing performance that compares favorably to ChatGPT with few-shot learning at average centipawn loss, proportion of sampled moves that were legal, and overall diversity of moves sampled. For both fine-tuned models and ChatGPT, using chain-of-thought prompting or chain-of-thought styled test examples appeared to slightly improve the accuracy of moves, and greatly improved the proportion of sampled moves that were legal in the larger model. However, chain-of-thought prompting appeared to slightly harm the smaller fine-tuned models in the move legality and top-k accuracy metrics. In summary, the high inter-output variances made it difficult to draw statistical conclusions about the advantages possessed by either small language models or LLM few-shot learners; however, the experiments with language pretraining and prompt construction indicate the existence of a tradeoff between parsability (i.e. the quality of natural language) and chess-playing ability in the model outputs, at least at the few-billion parameter scale.

As for limitations of this study, there are several major areas that could be improved to make the findings more robust. Firstly, the quality of the natural language in the training data severely lagged behind that of the chess-specific features, which could be readily generated with a strong ground-truth chess engine. The limited diversity of the rule-based systems used to create the natural language scaffolds was likely a major factor in the quick convergence of the models, and may have stifled their ability to learn stronger representations of chess-relevant tokens. Additionally, I lacked a high-quality dataset of annotated chess moves with which to enrich the chess-related natural language vocabulary of the next-best-move generation task. Previous chess NLP studies employed the `gameknot.com` dataset first described in Jhamtani et al. (2018); however, the codebase for extracting these comments is obsolete, and manual review of sample comments revealed significant shortcomings such as off-topic comments (Jhamtani et al. (2018) identified a total of 6 categories of comments, several of which do not describe the board position or move at hand), weak labeling (single comments describing multiple consecutive board positions), poor move quality (annotated moves made by players of lower skill levels), and poor comment quality (short, uninformative and ungrammatical comments). I decided to employ a semi-supervised learning strategy to supplement a very small number of hand-generated comments with automatically-generated comments using an engine, but again this likely had the effect of reducing the distribution diversity for this particular task. Another limitation is the relatively small number of games that were used to create training data. Each position in these games was used to generate prompts for each task, but doing so likely biased the dataset towards the opening and early middlegame due to the right-tailed distribution of chess game lengths. Additionally, there are likely complex patterns of statistical dependence associated with the training examples generated from consecutive positions in each game which were not considered in the initial formation of the training dataset. Finally, it is worth noting that the models I was able to train were significantly smaller than the LLMs for which the instruction fine-tuning and chain-of-thought prompting strategies were originally introduced. Wei et al. (2022a) characterized several emergent tasks and the effectiveness of augmented prompting strategies, finding that many of them only began to show noticeable effects in models at least an order of magnitude larger than GPT-2-XL (1.5B parameters). It is possible, therefore, that larger models may show greater responses to my finetuning strategies, although I was not able to experiment with LLMs beyond prompting. Additionally, I did not perform any experiments with models of varying size to characterize differences in performance on smaller scales, although initial milestone experiments suggested that GPT-2 (125M parameters) would produce outputs from which it is substantially harder to parse evaluations.

These initial results and limitations leave open many interesting possibilities for future research. The initial fine-tuning experiments using a combination of plain and chain-of-thought styled training examples showed that sampling from the distribution using chain-of-thought prompts could produce more verbose outputs at a higher frequency than the longer outputs occupied in the training data. Follow-up experiments in this direction may focus on extending or refining this semi-supervised learning strategy with a training objective that attends to correctness of information instead of purely causal language modeling, the inclusion of more hand-labeled data, and perhaps even a self-training paradigm in which the more verbose outputs of a model are used alongside ground truth examples to iteratively train new models and sample outputs. It may also be possible to explore applying prompting techniques directly to newer LLMs, as very recent exploration indicates that GPT-4 through Bing may have significantly better chess-playing ability, including the ability to hallucinate calls to Stockfish, although preliminary outputs seem not to contain natural language justifications for these moves (@zswitten, 2023). An alternative approach might be to use a different model architecture, such as an encoder-decoder model, in which the representation of a chess question input is built separately from the model output. However, it may be that the most useful interactions between natural language and AI chess engines is not in creating a natural language chess-playing agent, but using a language model to reason over the chess capabilities of existing chess agents. This avenue of future work - creating natural language conditioned on a chess move - would explore the inverse objective to the one targeted in this study, and has research precedent (Jhamtani et al., 2018). A motivating problems in this space is the interpretation of moves made by superhuman engines like AlphaZero, for which a multimodal learning strategy may be promising. In this paradigm, a learning objective would attempt to harmonize the representations created by chess-specific and natural-language-specific heads in a shared latent space, with the ultimate goal of being able to classify, interpret or generate information in one modality using the other.

6 CONCLUSION

In this project, I experimented with several strategies to create a natural language chess engine, including language pretraining, instruction fine-tuning and chain-of-thought prompting. I found that a combination of pretraining and short fine-tuning regimens with chain-of-thought styled training examples was as effective at boosting the chess-playing performance of a small language model as few-shot learning for ChatGPT, although both models failed to significantly outperform random baselines. This study will hopefully inspire future work on language model reasoning over complex game systems using a combination of prompting, fine-tuning, semi-supervised learning or multimodal learning strategies.

References

Patrick Coulombe. 2017. Predicting rating from centipawn loss. https://web.chessdigits.com/articles/predicting-rating-from-centipawn-loss. Accessed: 2023-03-17.

Michael DeLeo and Erhan Guven. 2022. Learning chess with language models and transformers. In *Data Science and Machine Learning*. Academy and Industry Research Collaboration Center (AIRCC).

Indrajatin. 2022. What is the highest average centipawn loss you can achieve in a game? https://lichess.org/forum/general-chess-discussion/what-is-the-highest-average-centipawn-loss-you-can-achieve-in-a-game. Accessed: 2023-03-19.

Harsh Jhamtani, Varun Gangal, Eduard Hovy, Graham Neubig, and Taylor Berg-Kirkpatrick. 2018. Learning to generate move-by-move commentary for chess games from large-scale social forum data. In *Association for Computational Linguistics (ACL)*.

Isaac Kamlish, Isaac Bentata Chocron, and Nicholas McCarthy. 2019. Sentimate: Learning to play chess through natural language processing. In *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*.

David Noever, Matt Ciolino, and Josh Kalin. 2020. The chess transformer: Mastering play using generative language models. In *Association for the Advancement of Artificial Intelligence (AAAI)*.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2018. Language models are unsupervised multitask learners. In *OpenAI blog*.

Levy Rozman. 2023. Chatgpt just beat chess. https://www.youtube.com/watch?v=rSCNW1OCk_M. Accessed: 2023-03-19.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *International Conference on Machine Learning (ICML)*.

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

Nenad Tomašev, Ulrich Paquet, Demis Hassabis, and Vladimir Kramnik. 2020. Assessing game balance with alphazero: Exploring alternative rule sets in chess. In *arXiv*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Jason Wei, Maarten Bosma, Vincent Y. Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M. Dai, and Quoc V. Le. 2021. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations (ICLR)*.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, Ed H. Chi, Tatsunori Hashimoto, Oriol Vinyals, Percy Liang, Jeff Dean, and William Fedus. 2022a. Emergent abilities of large language models. In *Transactions on Machine Learning Research (TMLR)*.

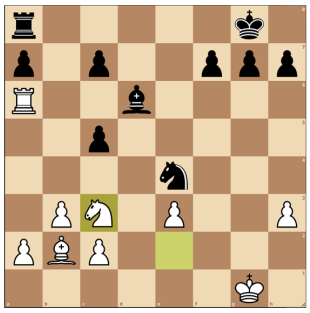
Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2022b. Chain-of-thought prompting elicits reasoning in large language models. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Edward Winter. 2022. Chess annotations. https://www.chesshistory.com/winter/extra/annotations.html. Accessed: 2023-03-01.

@zswitten. 2023. Ok this scared me a little: Bing/sydney can play chess out of the box. https://twitter.com/zswitten/status/1631107663500304384?lang=en. Accessed: 2023-03-17.

A Appendix

Board Position



Traditional FEN Notation

```
r5k1/p1p2ppp/R2b4/2p5/4n3/1PN1P2P/PBP5/6K1 b - - 3 25
```

Expanded FEN Notation
(line breaks added for legibility)

```
r _ _ _ _ _ k _ /
p _ p _ _ p p p /
R _ _ b _ _ _ _ /
_ _ p _ _ _ _ _ /
_ _ _ _ n _ _ _ /
_ P N _ P _ _ P /
P B P _ _ _ _ _ /
_ _ _ _ _ _ K _ ;
b - - 3 25
```

Figure A1: Side-by-side comparison of a chess board position, representation in traditional FEN, and representation in expanded FEN.