10-423 Generative Artificial Intelligence

Final Project Midway Report

March 22, 2026

Mihir Dhamankar mdhamank

Sachit Lumba slumba

Raaid Tanveer rtanveer

Introduction

Large language models (LLMs) have shown remarkable performance across a wide range of natural language tasks, including question answering, text generation, and dialogue systems. However, their ability to operate as agents in structured environments, such as two-player games like chess, remains an area of active research. In this project, we explore the feasibility of training an LLM to play chess competently by predicting the next move given a prior sequence of moves or a board state.

Our approach involves fine-tuning a pre-trained LLM, specifically Mistral's 7B v0.01 model, on a large dataset of chess games in Portable Game Notation (PGN) or Forsyth-Edwards Notation (FEN) format. We aim to improve upon the baseline performance of the pre-trained model, which generates valid moves only 9.7% of the time, by employing various techniques such as data cleaning, low-rank adaptation using LoRA and QLoRA, and reinforcement learning using Stockfish evaluation scoresjj as a reward model.

We draw inspiration from recent advancements in the field, such as the Chess Transformer [1], which demonstrates the potential of fine-tuning GPT-2 on chess games, and the use of Forsyth-Edwards Notation (FEN) for representing board states [2]. Additionally, we explore the incorporation of chess textbooks as an initial fine-tuning step to help the model learn game rules and strategies [3].

Our primary goal is to develop an LLM-based chess engine capable of consistently generating valid moves and playing competently against human opponents. We evaluate the model's performance using a tiered approach, starting with the proportion of valid moves generated and progressing to the assessment of move quality using Stockfish's evaluation function or the ELO rating system.

Through this project, we aim to contribute to the understanding of LLMs as agents in structured environments and demonstrate the effectiveness of low-rank adaptation techniques in enabling smaller models to achieve performance comparable to larger, fully fine-tuned models in the domain of chess.

Dataset and Task

Task.

We aim to determine if it is possible to train an LLM to play chess. In essence, this task reduces down to next-move prediction given a prior sequence of moves or a board state. We will examine if it is feasible to first even achieve a model that can output correct and valid moves, and then look for something that can play competently against an opponent.

Dataset.

-For this task, we use the KingBase 2019 dataset [4], that contains PGN's (Portable Game Notation) that describe over 2 million chess games. This is a substantial amount of data that we can use to hopefully derive some good results from.

PGN move notation describes games by specifying moves of black and white using algebraic notation in alternating order. Notably, this is a convenient format to specify as natural language input to language models.

Furthermore, we discovered that it might be better to convert the PGN format to FEN (Forsyth-Edwards Notation) since it can better represent information about the board state.

However, chess data doesn't resemble English semantically, which is what most pre-trained LLM's have been trained on - it is possible that this may pose an obstacle in getting such models to consistently output the desired format. However, due to the prevalence of chess on the internet, it is quite plausible that game data has been included in training datasets for most LLM's.

For this project we filtered out the drawing games so we can train our model only on the winning games.

Metrics.

We take a tiered approach to evaluating the performance of a model, where we first will evaluate whether a model is able to output well-defined and valid moves, before determining if it is able to output good ones. Thus, our first metric will simply look at the average proportion of valid moves outputted. After crossing this hurdle, there are a few options to choose from.

On a smaller scale, we can evaluate if a model makes good moves individually. Stockfish is a popular open-source chess engine that comes with a board state evaluation function i.e. one that assigns a score to a board state depending on how favorable it seems for a given player. We can use differences in this score before and after a move is made by a model to score it.

On the other hand, it may be possible, through the examination of multiple games played by a model, to determine its ELO score. The ELO rating system is one that is designed to calculate the relative strength of a player in a zero-sum game, and is widely used to maintain a ranking of chess players today. However, it is most accurate when observing a large amount of continued play, so it is a rather expensive metric. This might be more practically simplified to observing the proportion of games a model is able to win against an opponent of a certain ELO, simulated by a chess engine like Stockfish.

Related Work

Large Language Models as Agents in Two-Player Games [5]. This paper suggests that ideas from game theory and reinforcment learning can be used to train LLMs. An autoregressive language model predicting the best possible next token can be viewed as a two player game. Various strategies such as pre-training, SFT, RLHF, and in context learning correspond to different aspects of finding an optimal policy in the game. They claim that training a 2nd LLM to model the other player may provide some benefits. Chess is a 2 player game so we hope to be able to use some of the observations in this paper to our benefit.

LoRA+: Efficient Low Rank Adaptation of Large Models [6]. We intend to fine tune LLMs on chess data and LoRA seems like the best way to achieve this efficiently. This paper may provide useful methods to make sure this fine tuning is effective. The authors show that standard LoRA leads to suboptimal finetuning of large models with wide embeddings. This is because LoRA uses the same learning rate for the A and B adapter matrices. The authors propose LoRA+, where they use different learning rates for A and B, with B's learning rate much larger than A's, enabling efficient feature learning in LoRA finetuning. They claim LoRA+ achieves 1-2% higher test accuracy and up to 2x speedup compared to standard LoRA, at the same computational cost. They suggest changing the learning rate of B to be a constant factor of 16 times the learning rate of A.

Direct Preference Optimization: Your Language Model is Secretly a Reward Model [7]. This paper may be useful to reference when attempting to use Stockfish as a reward model to better tune our model. DPO avoids the complex reinforcement learning pipeline used in prior work like PPO-based RLHF (Stockfish feedback in our case). Instead, it directly optimizes the language model parameters using a simple binary cross-entropy loss. The performance is not impacted in evaluation. It is also more stable than PPO since it doesn't require sampling the LM or estimating value functions during training.

The Chess Transformer: Mastering Play using Generative Language Models [1]. The authors of this paper fine tuned a GPT-2 model to with chess games in PGN notation. Their "Chess Transformer" was able to generate plausible chess moves and game formations, including classic openings. They analyze how well the model performs with their full fine tuning. These findings will be a good comparison to our attempts at using LoRA to fine tune. We can also compare the effects of changing the base model based on their results.

Building a Natural Language Chess Engine with Pretraining and Instruction Fine-Tuning [8]. This paper firstly offers a good literature review for the papers that we should explore for this domain. Secondly, this paper uses an interesting approach where the chess board positions after every move are annotated with some extra information to assist the language model in understanding the board position, such as the number of valid moves at that position, what the valid moves are, where the pieces are located, etc.

Chess as a Testbed for Language Model State Tracking [9]. This paper highlights the importance of probing language models on their understanding of the state of the world based on a sequence of tokens that they have observed. Even if a language model observes a sequence of moves that have occurred e.g. `1. e4 2. d5 3. exd5`, it is difficult to say for sure whether the language model knows that at the end of move 3 the board state will be that the `e` pawn is now at `d5`, and the Black side's `d` pawn has been captured.

Given the findings of this paper in a language model's ability to play board games well given a large enough training set, and if we give enough information about the board state then even with a small dataset the language model will be able to learn to play a game well. Models such as ChatGPT have already been shown to be able to play games well, as shown in [9]. However, if we are to use smaller models to play games like Chess, we might need to pass in the board state using ASCII art, for example, in the instruction fine-tuning stage to allow the model to play the game better.

Learning to Play Chess from Textbooks (LEAP): a Corpus for Evaluating Chess Moves based on Sentiment Analysis [3]. This paper suggests using unstructured textbook data to do an initial fine-tuning pass whereby the model can capture rules of the game and strategies for playing chess before we fine-tune on the chess games data. The textbook data was acquired from Project Gutenberg, and in order to make our fine-tuned model play Chess better we might want to initially fine-tune on some Chess textbooks as well.

Learning Chess with Language Models and Transformers [2]. This paper implements a BERT model that learns to play Chess using the FEN notation. The FEN notation is special in that it represents all the information in the board that is necessary to resume playing the game from that encoding only. The implementation of this model is similar to the recommendations made in [3], in that both are advocating for using some richer form of board state represenation than just the moves used by the PGN format.

Overall Literature Review. From reviewing the literature regarding how pre-trained transformers can be fine-tuned to play games such as Chess, we see that there are many low-rank methods that can be used for the task (which have not been applied in the domain of Chess yet).

Furthermore, there are several options for constructing the data to fine-tune with, as well as the methodology for fine-tuning, such as:

- Using board state representations such as FEN, and annotating the board positions after every move with some useful information such as what available moves there are at that position.
- Fine-tuning with Chess textbooks first to enable the Language Model to learn strategies and rules of the game.
- Use low-rank or quantized low-rank adapters to reduce the number of parameters that need to be fine-tuned, and to reduce the compute required to fine-tune the models.

Our 10-423 Project aims to combine some, or all of these general approaches that we see are being used for fine-tuning models and playing board games using transformers to fine-tune a small language model to play Chess, that is comparable to larger models using full fine-tuning.

Approach

Baseline.

Mistral's 7B v0.1 pre-trained model was used as a baseline for the task of next move generation. From a preliminary test, we found that the model was able to generate valid moves around only **9.7%** of the time. This signifies that there is significant room for improvement on this task, first to ensure valid moves, and then to ensure good moves. However, it is likely that training on good moves will yield only valid moves.

Furthermore, in contrast to the baseline method of training using PGN files, we plan to be fine-tuning with FEN files that have a more complete representation of the board, in addition with some annotations of useful information about the boards such as number of valid moves available at the state, what those moves are etc.

Main Methods.

We are using Hugging Face's Alignment Handbook to fine-tune the model to play Chess. We will be trying to fine-tune using LoRA and QLoRA using different rank values, as well as witll

Experiments

Data Cleaning.

Our initial work involved analyzing the dataset and parsing it into a format that could be used for fine tuning. Since we planned to use a chat-like interface, we decided to use the built in apply_chat_template method in Huggingface's alignment handbook. This method converts a list of message objects in an alternating "user"/"assistant" format into a single string by applying the relevant chat template. We wrote our own chat format that uses "[white]" and "[black]" to denote different players. To use this pipeline, we wrote code to parse the raw PGN data into test and train .json files. This is the format we used:

```
System: white won the following chess game
White: 1.e4
Black: 1...e5
White:
```

When passed throught the apply_chat_template function, this becomes

```
<s> white won the following chess game </s> [white] 1.e4 [black] 1...e5 [white]
```

Baseline.

For our baseline generation, we attempted to predict every other move in the testing set with 1024 games using the base Mistral model. We started by feeding the model up to the 1st move, recorded the predicted result, then fed it up to the 3rd move, recorded the result, then up to the 5th move and so on. This simulates asking a chess bot to generate every even move one at a time (with the human playing the odd moves). Since many of the predicted moves were not valid and keeping track of valid board states would get complicated, we did not reincorporate the predicted moves back into each subsequent generation (we used the "correct" move from the dataset).

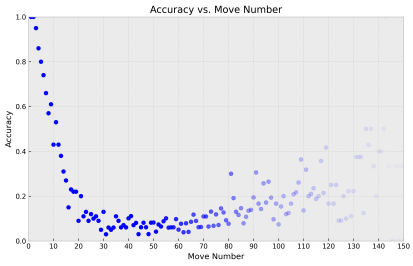

Figure 1: Baseline Model Accuracy

From this experiment we found that **20.4%** of the predicted moves were even valid. In this case valid means that the PGN notation is correct **and** that the move is a legal move to make at that point in the game. We noticed that the first move was almost always not well formed, let alone valid (e.g. "1...e5"). Seemingly after 3 correct moves in context the model was able to produce the correct move number (only even moves), but the moves themselves were not very accurate. Even more analysis can be done on the quality of moves themselves by comparing them either to the dataset moves or a chess engine's score.

Fine-tuning.

We fine-tuned the model on ~4000 games for 3 epochs, and found an average move validity of **75.7%**. Additionally, we see that the model begins performing far better in the mid-game, and is always over 50%.

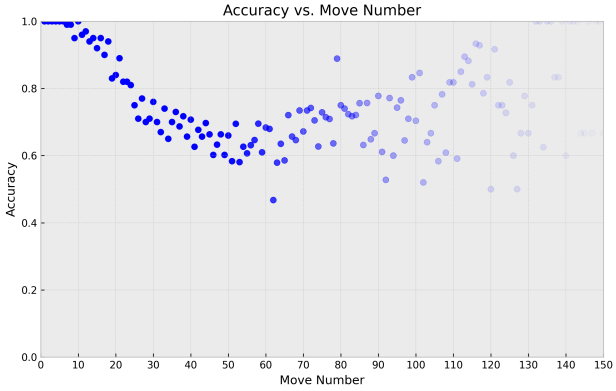

Figure 2: Fine-Tuned Model Accuracy

More Data. Training on ~8000 examples led to a lower validity of **71.9%**.

FEN-supplemented Input.

As an experiment with our data formatting, we inserted FEN board states after every move, in order to give the model a complete representation of the board state at the most recent time step and thus relieve it of having to memorize it after each made move.

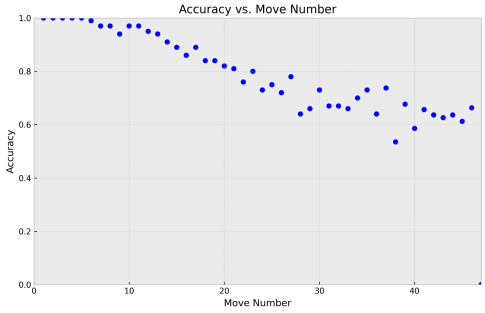

Figure 3: Baseline Accuracy with injected FEN notation

The model performed similarly after fine-tuning, but including the FEN notation end up being far more resource and time intensive, and hence we could only evaluate games till 47 moves. However, even in these few moves, we find an average validity of **77.9%**. The prior model's accuracy within the first 47 moves was 81.9%.

Evaluation.

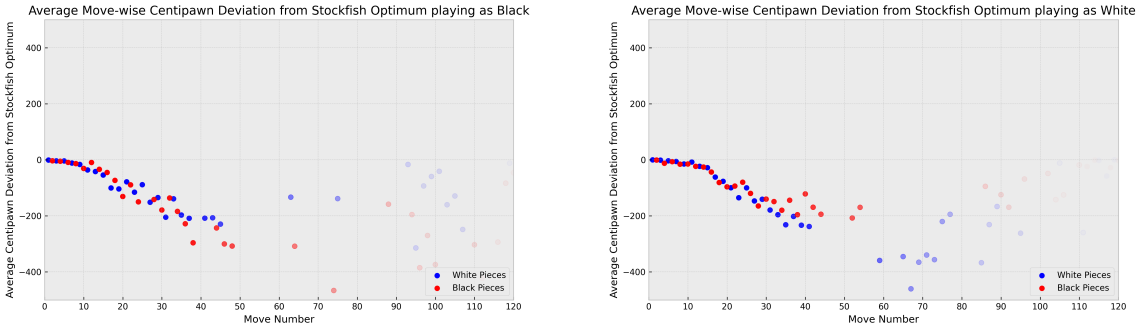

Figure 4: Average Centipawn Deviation from Stockfish

With our interactive evaluation loop that retries invalid moves with a higher temperature, we were able to play chess with the model. We noticed that it was unable to defend against obviously bad moves not in the training data. We easily defeated it by moving our queen out early and taking pieces even when they were defended. It also struggled with making moves when it was in check, indicating it may have not learned much about such rules of chess.

Note that in the Chess Transformers Paper [1], they were able to achieve a valid move percentage of 90%, so the model generated invalid moves 10% of the time with full fine-tuning on all 2 million KingBase-2019 games training for 8 hours total. We believe that our result of 75.7% valid move accuracy using QLoRA, around just 3500 training examples, and training for 3 epochs shows that it is possible to replicate the results of the Chess Transformers paper with fewer trainable parameters.

Code Overview

We wrote `dataparser/chess/extract.py` and `dataparser/chess/split_data.py` to convert the Kingbase dataset into a JSON file with individual chess moves and then further process it along with splitting it up into testing and training sets. We then used HuggingFace Alignment Handbook to fine tune the Mistral 7B model. We did not significantly change the training loop in `scripts/train_sft.py` other than changing how the dataset is loaded on line 89. We created a Jinja chat template to apply to each data point (created YAML files in `recipes/chess-llm/sft` and modified some existing code to make `scripts/setup_chess_tokenizer.py`). We adjusted several parameters such as LoRA alpha and r, number of epochs, and quantization amount. For evaluation we wrote a loop to simulate 100 games from the test set in parallel (`scripts/generate_new.py`). The model comes up with a next move suggestion for every game given a prefix of moves. We had to stop evaluation at 100 moves per game since execution time is quadratic with sequence length and running 100 games at a time was computationally expensive. We wrote code to make the model retry generating if a move is invalid (`scripts/generate_retry.py`). Rerunning evaluations with a maximum of 8 retries per move, we were able to calculate how much worse our models performed per move compared to StockFish (several files in `evaluate_results`). Finally, we wrote an interactive eval loop which lets the user play chess with the model (`scripts/play_chess.py`). The user can pick which side they want to play as or have the model play itself.

Timeline

Activity	Duration
Background Reading	2 hours
Reading Code Documentation	1 hour
Understanding Alignment Handbook	1 hour
Compiling/Running Alignment Handbook	10 hours
Modifying Code	5 hours
Running Experiments	4 hours
Writing Evaluation Scripts	2 hours
Compiling Results	1 hour
Writing Documents	2 hours

Research Log

1. First, we extracted the dataset from KingBase-2019. There were issues with downloading and processing the dataset in a reasonable amount of time. We ended up using Python's `multiprocess` library to parallelise processing of the data.
2. We then tried fine-tuning Mistral 7B. When using the T4 GPU's on AWS, CUDA was running out of memory when loading up the 7B parameter model. We ended up having to use an A100 GPU.
3. Even with all of the above, we were still sometimes running out of memory, so we had to switch to fine-tuning with quantization and low-rank approximations. Our rank was reduced from 16 to 8 to make the model fit into the GPU.
4. Our training job was taking too long, so we reduced the number of examples from 1.2 million to around 9000 examples sampled randomly.
5. We enabled packing different examples to the same line in order to reduce training time as well.
6. The initial fine-tuning results were promising. We then tried to experiment with adding more metadata such as the board FEN representation after each move. But this overflowed the context length, so we had to truncate the moves for each game so that we did not end up exceeding the context length. However, adding FEN did not end up improving our model by much, so we stuck with fine-tuning with moves.

Bibliography

[1] D. Noever, M. Ciolino, and J. Kalin, "The Chess Transformer: Mastering Play using Generative Language Models." 2020.

[2] M. DeLeo and E. Guven, "Learning Chess with Language Models and Transformers," in *Data Science and Machine Learning*, in DSML 2022. Academy, Industry Research Collaboration Center (AIRCC), Sept. 2022. doi: 10.5121/csit.2022.121515.

[3] H. Alrdahi and R. Batista-Navarro, "Learning to Play Chess from Textbooks (LEAP): a Corpus for Evaluating Chess Moves based on Sentiment Analysis." 2023.

[4] Chessdb, "KingBase 2019." 2019.

[5] Y. Liu, P. Sun, and H. Li, "Large Language Models as Agents in Two-Player Games." 2024.

[6] S. Hayou, N. Ghosh, and B. Yu, "LoRA+: Efficient Low Rank Adaptation of Large Models." 2024.

[7] R. Rafailov, A. Sharma, E. Mitchell, S. Ermon, C. D. Manning, and C. Finn, "Direct Preference Optimization: Your Language Model is Secretly a Reward Model." 2023.

[8] B. Jiang, "Building a Natural Language Chess Engine with Pretraining and Instruction Fine-Tuning."

[9] A. Yedidia, "Chess as a case study in hidden capabilities in ChatGPT." 28, 2023.