# Building a Transformer Inference Framework for Chess Problems with LoRA and Tree-of-Thoughts

**Roy Yatskan**
Tel Aviv University

**Ksenia Polonsky**
Tel Aviv University

**Daniel Laufter**
Tel Aviv University

## Abstract

In this project, we develop a new inference framework called ChessToT for evaluating chess puzzles, inspired by minimax and Tree-of-Thoughts. We use ChessGPT, a 2.7B model pre-trained on puzzles and games, and enhance it with a short fine-tuning of synthetic examples via supervised LoRA. Our method consists of a two-step inference framework: first, the model suggests checkmating moves and then verifies if the resulting positions are indeed checkmates. Notably, our refined model surpasses the original ChessGPT's performance in checkmate-in-one puzzle solving. The code is available here.

## 1 Introduction

Recent advancements in natural language processing (OpenAI, 2023) have been driven by Large Language Models (LLMs) such as GPT-3 and the more recent GPT-4. These models have demonstrated strong performance in various tasks, including summarization, translation, question answering, and code generation.

LLMs make use of the transformer architecture, an important development in NLP(Vaswani et al., 2023). This architecture relies on self-attention mechanisms, allowing the model to understand relationships in the input text, even if elements are distant. This capability enhances the model's ability to process complex linguistic structures.

Chess, a widely played strategic game, has been a research interest in computer science for years, with landmark projects like IBM's DeepBlue to Google's AlphaZero (Silver et al., 2017; Campbell et al., 2002). With its defined rules and abundant open-source data, chess can serve as a valuable testing ground for LLMs. Current chess models, however, face challenges in explaining their reasoning or mimicking human play. We believe that future iterations of LLMs have the potential to address these issues, possibly giving rise to explainable chess engines.

ChessGPT, a 2.7 billion parameter model trained on a vast and diverse corpus of chess data, exhibits an understanding of the game beyond mere rule comprehension. The proficiency of this model surpasses basic legality understanding and internal chess representations, which were the primary focus of previous NLP research in chess in the BERT era. With the emergence of these advanced models demonstrating near-perfect accuracy in basic tasks, the research shifts focus towards solving more complex problems like chess puzzles and strategic game-play. In chess, both the classical algorithmic and human play style is through careful step-by-step evaluation, going through the player's possible moves and the opponent's response over multiple iterations.

In chess, players typically evaluate moves step-by-step, considering both their potential moves and the opponent's likely responses. This method aligns with the 'Tree-of-Thoughts' (ToT) inference method for LLMs, a progression from the earlier 'Chain-of-Thoughts'. This approach is well-suited for systematic planning tasks, such as chess. Building on this, we've developed a two-step algorithm inspired by the ToT framework, specifically designed for solving checkmate-in-one puzzles. While ToT is commonly associated with large models like GPT-3 or GPT-4, our work demonstrates that even smaller models like ChessGPT can benefit with the proper training and tweaks.

In one of the test sets for ChessGPT, the model was presented with checkmate-in-one puzzles. For each puzzle, the model is given a board position with has one potential checkmating move and tasked with identifying that move. Our project aims to enhance ChessGPT's performance on this specific task. To achieve this, we introduce an additional verification step where the model double-checks its suggested moves for a given puzzle.

To generate the initial $K$ (not necessarily unique) moves, we use the built-in beam search algorithm and get the top $K$ outputs. For the model to follow commands, we fine-tune it with examples carefully generated from Lichess. Then, using a heuristic we pick final move. For convenience, in this report, we sometime refer to the inference framework as **ChessToT**, the original ChessGPT model as **Model 1**, and the fine-tuned verification model as **Model 2**.

Applying the ToT framework to chess offers insights that could be applied to other decision-making tasks or even serve as a foundation for future LLM-based chess engines.

## 2 Related Work

Earlier research primarily focused on fine-tuning models such as GPT-2, BERT, and LSTM, which are comparatively weaker. Building upon this, (Feng et al., 2023) utilized a more powerful 2.7B model, which forms the basis for our fine-tuning and research. (Toshniwal et al., 2021) fine-tuned a GPT-2 using 2.9 million chess games, and tested its ability to understand a chess position represented as a sequence of moves. (DeLeo and Guven, 2022) fine-tuned BERT on a synthetic dataset generated by Stockfish, and represented moves in FEN format and showed that it could output legal moves and survive for a surprisingly long time against Stockfish in games. Our work is also inspired by Maia Chess (McIlroy-Young et al., 2020), a machine-learning project focusing on designing an engine that plays in a human-like style. Their goal was not to compete with Stockfish for being the SOTA engine but to serve as a playing companion for practice and learning. To do that, they used the Lichess dataset and trained multiple Neural Networks to predict human moves, each time using games played at different ELOs (ranks).

## 3 Background

This section introduces the basic chess notations and terminologies used in this study, providing the necessary grounding for readers unfamiliar with the specific chess nomenclature.

### 3.1 Chess Notations

Chess notations are systems that record the moves in a game of chess. They serve as a universal language for chess players worldwide and are used in this study to communicate moves and positions. There are several types of chess notations, three of which are particularly relevant to our research:

- **Forsyth-Edwards Notation (FEN)**: A standard notation that fully describes a specific board position in a chess game, capturing all information needed to restart a game from that position. However, it does not include exact details about players' moves previous to that position.
- **Standard Algebraic Notation (SAN)**: Employed for noting individual moves, SAN is a character string specifying the piece moved, its origin and destination squares, and special conditions like capture or check. For example, a SAN ending with # denotes a checkmating move.
- **Portable Game Notation (PGN)**: A standard notation for recording chess games that can be read by both humans and chess software. PGN is used for representing complete games and includes the moves played in SAN along with all relevant metadata.

### 3.2 General Chess Terminologies

Several chess-specific terminologies are used in this study:

- **Chess Puzzle**: A setup on the chessboard posing a specific goal to the player, such as achieving a checkmate or acquiring a material advantage.
- **Checkmate-in-One Puzzle**: A subset of chess puzzles where the task is to identify a single move leading to checkmate.
- **Centipawn**: A unit of measure used to evaluate a chess position, representing an advantage equal to one-hundredth of a pawn. For example, an advantage of 150 centipawns is equivalent to an advantage of 1.5 pawns. Chess engines commonly use this unit to assess the relative strength of one player over the other.The score is relative, and a higher score is better.

## 4 Methodology

We program the entire project in Python Jupyter Notebooks - github.com/ryatskan/NLP-Project

For the entire handling of the LLM models, from training to evaluation, we primarily rely on the Hugging Face libraries, among others.

Table 1: Summary of ChessGPT and Maia Metrics - higher is better

| Model | Centipawn Value | Similarity |
|---|---|---|
| **ChessGPT** | | |
| - First beam | -100.699 | – |
| - Seconf beam | -250.896 | – |
| **MAIA Models** | | |
| - MAIA 1100 | -30.116 | 54.0% |
| - MAIA 1400 | -13.860 | 44.7% |

### 4.1 Data Collection and Preprocessing

We sourced our data from Lichess (Lic, 2023), specifically using the game dump from December 2019. The preprocessing steps involve downloading and decompressing the PGN dump, converting it into a DataFrame for easier analysis, and using the python-chess library (pyt) to iterate through the games and extract specific positions. All preprocessing procedures are detailed in an accessible notebook, `generate_data.ipynb`.

## 5 Initial Exploration of ChessGPT

First, we explore the capabilities of the ChessGPT model and compare it head-to-head with chess engines.

### 5.1 Tournament

We matched the LLM model against the Maia engine, trained on lichess games of rank 1100. To further handicap the engine, its depth was set to 1. In a series of 100 games, the LLM model won 8, the engine won 59, and the remaining were draws. This experiment indicates that ChessGPT is not yet proficient enough to function as a competitive engine.

### 5.2 Centipawn & Similarity

For this section, the LLM model was evaluated against multiple Maia models as well as Stockfish, based on the quality of the moves they produce from randomly generated mid-game positions. These positions were randomly taken from the Lichess dump. For each of the 1000 positions, every model proposed a move, which was subsequently assessed by Stockfish for its *centipawn value*. In addition, we checked the similarity of ChessGPT's outputs with the Maia models. Our analysis revealed that the LLM model's performance was notably inferior to that of the Maia models. The detailed scores can be found in Table 1.

### 5.3 Checkmate-in-one

One of the key findings in the ChessGPT paper pertains to the checkmate-in-one puzzle evaluation. The paper reported an accuracy of 71.4% on the 'checkmate-in-one' task from Big-Bench and employing the lm-evaluation-harness library. In our project, we evaluated ChessGPT and tested it on a subset of Big-Bench, containing 3500 prompts. We analyzed the model's performance in two different scenarios. In the first scenario, where the model's first output of the beam search was taken as the final answer, we observed an accuracy rate of 62.77%, and 71.89% when the first output marked with # was considered. **Notably, there are scenarios where the model outputs the correct checkmating move, but not as the first in the beam search**. When we extended the analysis to consider any correct answer from the top five outputs from the model's beam search, the accuracy improved to 76.91%. This forms the motivation for the Part II.

## 6 Part I: Downstream LoRA fine-tuning

**Algorithm 1** Data Generation for Position Classification

**Input:** Chess game $G$, upper bound $U$, lower bound $L$
**Output:** Positions $[P_1, P_2, P_3]$ or None
Classify all positions in $G$ between $L$ and $U$ into disjoint subsets $[C_{mate}, C_{check}, C_{neither}]$
**if** any set $C_i$ is empty **then**
    **return** None
**else**
    For each $C_i$, randomly pick one position, with a bias towards latter positions: $P_i$
    **return** $[P_1, P_2, P_3]$
**end if**

In this phase, we will fine-tune the ChessGPT model on one task - given a position, classify it into three categories: a check, a checkmate, or neither. Even though we only need binary classification - if a position is a mate or not - we hope that incorporating a "check" label will help the model differentiate between check and checkmate positions.

### 6.1 Data Preprocessing

We extracted a dataset of 182,000 chess positions from a dump of 800,000 games on Lichess using Algorithm 1. To speed up the training process, we only included positions where white is to move.

These white positions can later be transformed to represent black positions by flipping the board and swapping colors. Our primary goal was to ensure that the model would generalize well to real-world positions rather than overfitting to training noise. To achieve this, we selected only three positions from each game for inclusion in our dataset. These chosen positions are randomly selected within a bound, with a bias to later positions. Without careful curation, the model might simply learn to associate a late-game position with a checkmate or that a checking move is always a checkmate.

### 6.2 Training

We trained the model on the examples using LoRA - Low-Rank Adaptation of Large Language Models (Hu et al., 2021), a lightweight training method that accelerates the training of large models while consuming less memory, at the expense of possibly worse results. The training took around 2 hours, trained on a single A100 40GB RAM GPU computer.

### 6.3 Results

We evaluated the performance of the Checkmate-in-One-ToT model. We tested the model on 2544 board positions' prompts. These positions were not taken from the games used for creating the training dataset. The results showed an accuracy of $88.6\%$ and high precision ($85.7\%$) and recall ($79.0\%$) rates. The confusion matrix in Fig.1 summarizes these results.

### 6.4 Qualitative Evaluation

We investigated the capability of Model 2 on carefully adjusted positions since the test set closely resembles the training set, and the results might still not reflect real positions. Notably, despite including a 'check' label, Model 2 struggles to accurately distinguish between 'check' and 'checkmate' positions. This task requires a nuanced evaluation of potential escape routes or the options of capturing the attacking piece. However, the model proves more capable in differentiating between mate and non-mate positions.

## 7 Part II: ToT Inference Framework

In the second phase, we use both the ChessGPT (Model 1) and the LoRA-fine-tuned model for position classification (Model 2). We combine both models to create an inference framework for checkmate-in-one puzzles. The algorithm for this approach is described in Algorithm 2, but the general idea is simple. First, we deploy Model 1 to generate moves, and then we use Model 2 to evaluate if the moves actually lead to a checkmating position. This strategy resembles how chess players typically consider potential moves and subsequently verify against the opponent's possible responses.

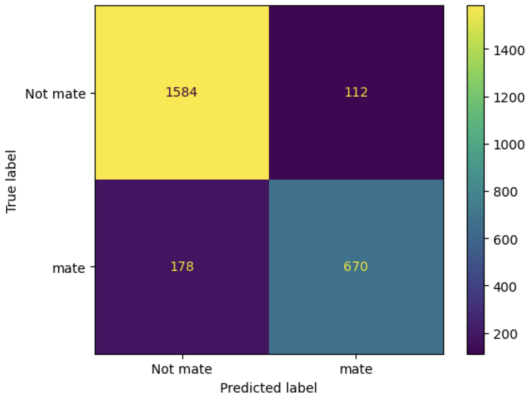


Figure 1: Confusion Matrix for Model 2 test of 2544 prompts; Accuracy for "mate" vs. the rest: 0.8860, Precision for "mate" vs. the rest: 0.8568, Recall for "mate" vs. the rest: 0.7901

**Algorithm 2** Checkmate-in-One-ToT

**Input:** $G$ game record in PGN representation, $K$ beam search size
Generate $K$ potential checkmating moves using Model 1: $[m_1, m_2, \ldots, m_K]$
**for** each move $m_i$ in $[m_1, m_2, \ldots, m_K]$ **do**
    Generate $K$ FEN positions by converting $G$ into a FEN board $B$ and applying $m_i$: $F_i$
    Standardize position for white to checkmate, if necessary
    Apply Model 2 to each $F_i$ and classify each $m_i$ into {mate, check, neither}: $R_i$
**end for**
**return** $m_{i^*}$ where $i^* = \min\{i \mid R_i = mate, 1 \leq i \leq K\}$ or $m_1$ if no such $i$ exists

### 7.1 Results

We start with running the first model on the subset of 3,500 prompts and getting five outputs for each prompt, representing five potential moves on the chessboard. After that, we transformed the initial prompts into chessboards, precisely we converted the PGN into FEN representation. For each of the FEN chessboards, we performed five possible

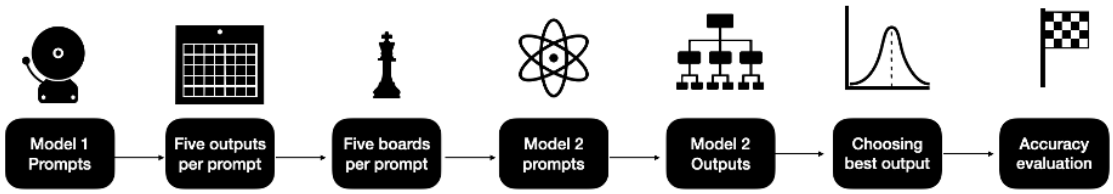


Figure 2: Graphical Representation of the Accuracy Evaluation Pipeline

moves, getting five post-move FEN chessboards for each initial prompt accordingly. We translated each chessboard into the corresponding prompt for Model 2, and the results were evaluated. Notably, the \$ and # signs were stripped from the outputs of model 1, which allowed us to avoid some unnecessary negatives when the move without the special character (\$, #) was considered accurate. Running the second model allowed us to assign one of the following labels to a corresponding move: "check", "mate", or "neither". We used the majority vote approach to assign the label based on the multiple outputs from Model 2; that is, the label with the highest count was assigned to the move. After the labels were given to all the moves, we took the first move with the "mate" label assigned to choose the best move.

Running this evaluation pipeline resulted in an improved accuracy of 77.2%, which is a significant improvement compared to the initial ChessGPT accuracy of 71.9% on the same dataset.

Some of the prompts, however, had a single move suggested in all outputs from model 1; therefore, applying the second model could not improve these lines. If we looked only at the prompts, which did not have a distinct answer (n=1492), we could summarize that ChessGPT accuracy was 36.3% based on the first output, 57.9% based on the final output (first winning output with "#" sign which was chosen by the model 1), and 69.5%, considering all five results accordingly. Therefore, we can see that selecting the correct answer out of five outputs provided by ChessGPT could bring a significant potential improvement.

Executing our pipeline and combining two models, only on the lines that do not have a single distinct output by ChessGPT, indeed results in an improved accuracy of 62.9% compared with the initial accuracy of 57.9% of ChessGPT output alone.

The data transformation within the pipeline is displayed in the example below.

| Accuracy / Model | ChessGPT | ChessToT |
|---|---|---|
| All | 0.719 | **0.772** |
| Non-Distinct | 0.579 | **0.629** |

Table 2: We tested the two models head-to-head regarding their accuracy in two scenarios - (1) all the n=*3500* examples, or (2) filtering only the examples where the beam search output was non-distinct (n=*1492*)

1. Model 1 - Prompts

| | input |
|---|---|
| 0 | 1. d4 d5 2. Nf3 Nf6 3. e3 a6 4. Nc3 e6 5. Bd3 ... |
| 1 | 1. e4 e5 2. Nf3 d6 3. d4 exd4 4. Nxd4 Nf6 5. N... |
| 2 | 1. c4 c5 2. Nf3 Nf6 3. g3 g6 4. Bg2 Bg7 5. O-O... |
| 3 | 1. g3 d6 2. Bg2 c5 3. d3 b6 4. Nf3 Nc6 5. O-O ... |
| 4 | 1. c4 e6 2. Nc3 Bb4 3. Ne4 d5 4. Ng3 dxc4 5. e... |
| ... | ... |
| 3495 | 1. e4 d6 2. Nf3 e6 3. d4 Nc6 4. Be2 Nf6 5. e5 ... |
| 3496 | 1. e4 Nf6 2. e5 Nd5 3. d4 d6 4. f4 Nc6 5. Nf3 ... |
| 3497 | 1. e4 e5 2. Nf3 d6 3. Bc4 h6 4. Nc3 Nc6 5. d4 ... |
| 3498 | 1. d4 d5 2. Bf4 Nf6 3. h3 h6 4. Nf3 Bf5 5. e3 ... |
| 3499 | 1. e4 e5 2. Bc4 Nc6 3. Nf3 Be7 4. d3 Nf6 5. h3... |

2. Model 1 - Outputs

| | input | label | outputs_0 | outputs_1 | outputs_2 | outputs_3 | outputs_4 |
|---|---|---|---|---|---|---|---|
| 0 | 1. d4 d5 2. Nf3 Nf6 3. e3 a6 4. Nc3 e6 5. Bd3 ... | Rg5# | 31... g6+ 32 | 31... Rf3# | 31... g6# 0 | 31... Rf2+ | 31... g6+ 32 |
| 1 | 1. e4 e5 2. Nf3 d6 3. d4 exd4 4. Nxd4 Nf6 5. N... | Qxe7# | Nf6+ gxf | Nf6+ {White | Nf6+ {White | Qxe7# 1- | Nf6+ {The |
| 2 | 1. c4 c5 2. Nf3 Nf6 3. g3 g6 4. Bg2 Bg7 5. O-O... | Bf6# | Bf6# 1- | Rxg4 {And black | Bf6# 1- | Bf6# 1- | Rxg4 Bxe7 |
| 3 | 1. g3 d6 2. Bg2 c5 3. d3 b6 4. Nf3 Nc6 5. O-O ... | Qxf1# | 30... Qxf1# | 30... Qxf1# | 30... Qxf1# | 30... Qxf1# | 30... Qxf1# |
| 4 | 1. c4 e6 2. Nc3 Bb4 3. Ne4 d5 4. Ng3 dxc4 5. e... | Rh5# | 35... Rh5# 0 | 35... Rh5# 0 | 35... Rh5# 0 | 35... Rh5# 0 | 35... Rh5# 0 |
| ... | ... | ... | ... | ... | ... | ... | ... |
| 3495 | 1. e4 d6 2. Nf3 e6 3. d4 Nc6 4. Be2 Nf6 5. e5 ... | Rh1# | 33... Rh1# 0 | 33... Rh1# 0 | 33... Rh1# 0 | 33... Rh1# 0 | 33... Rh1# 0 |
| 3496 | 1. e4 Nf6 2. e5 Nd5 3. d4 d6 4. f4 Nc6 5. Nf3 ... | Rg8# | Rg8# 1- | Rg8# 1- | Rg8# 1- | Rg8# 1- | Rg8# 1- |
| 3497 | 1. e4 e5 2. Nf3 d6 3. Bc4 h6 4. Nc3 Nc6 5. d4 ... | Qxh2# | 15... Qxh2 | 15... Qxh2 | 15... Qxh2 | 15... Qxh2 | 15... Qxh2 |
| 3498 | 1. d4 d5 2. Bf4 Nf6 3. h3 h6 4. Nf3 Bf5 5. e3 ... | Re1# | 68... Re1# 0 | 68... Re1# 0 | 68... Re1# 0 | 68... Re1# 0 | 68... Re1# 0 |
| 3499 | 1. e4 e5 2. Bc4 Nc6 3. Nf3 Be7 4. d3 Nf6 5. h3... | Qe2# | 24... Qe2# | 24... Qe2# | 24... Qe2# | 24... Qe2# | 24... Qe2# |

3. Model 1 – Moves extracted from the outputs

| | outputs_1 | outputs_2 | outputs_3 | outputs_4 | outputs_5 |
|---|---|---|---|---|---|
| 0 | Rf3# | g6# | Rf2+ | g6+ | g6# |
| 1 | Nf6+ | Nf6+ | Qxe7# | Nf6+ | Qxe7# |
| 2 | Rxg4 | Bf6# | Bf6# | Rxg4 | Rxg4 |
| 3 | Qxf1# | Qxf1# | Qxf1# | Qxf1# | Qxf1# |
| 4 | Rh5# | Rh5# | Rh5# | Rh5# | Rh5# |
| ... | ... | ... | ... | ... | ... |
| 3495 | Rh1# | Rh1# | Rh1# | Rh1# | Rh1# |
| 3496 | Rg8# | Rg8# | Rg8# | Rg8# | Rg8# |
| 3497 | Qxh2 | Qxh2 | Qxh2 | Qxh2 | Qxh2 |
| 3498 | Re1# | Re1# | Re1# | Re1# | Re1# |
| 3499 | Qe2# | Qe2# | Qe2# | Qe2# | Qe2# |

4. Post-Moves Conversion to FEN Boards

| | |
|---|---|
| 0 | 8/1p4Rp/2b1R3/2p5/2Pp1k2/r5PP/2B2P2/6K1 b - - ... |
| 1 | r3k2r/pppnb1pp/4QN2/6B1/3q4/8/PPP2PPP/R3R1K1 b... |
| 2 | 4b2k/2r2p1p/3p1B2/1PbBp1R1/2P3p1/6P1/7K/2R5 b ... |
| 3 | 5Q1k/p3rp1p/1p4p1/8/3p4/5P2/PB4PP/3R1RK1 b - -... |
| 4 | 4N3/6p1/7k/p1b5/7R/6R1/PPPB1PPP/6K1 b - - 2 36 |
| ... | ... |
| 3495 | 4rk1R/R7/3p3p/p3r3/1p6/6K1/PPP3PP/8 b - - 5 34 |
| 3496 | 5kR1/7R/p1p1pN2/1pPpP3/1P1P1p2/P1N2r2/3K4/8 b ... |
| 3497 | r1bqr2k/ppp4Q/2nb2p1/6NB/4p3/2PP3P/P1PB1PP1/R4... |
| 3498 | 2k1R3/8/2K5/1PN5/8/8/8/8 b - - 2 69 |
| 3499 | 2q1knr1/1ppbQ3/1p1p4/4pPB1/1P1nP3/r2P3P/2P3PK/... |

5. Model 2 - Prompts based on FEN boards

| | legals_0 |
|---|---|
| 0 | Q: Is this san position 8/1p4Rp/2b1R3/2p5/2Pp1... |
| 1 | Q: Is this san position r3k2r/pppnb1pp/4QN2/6B... |
| 2 | Q: Is this san position 4b2k/2r2p1p/3p1B2/1PbB... |
| 3 | Q: Is this san position 5Q1k/p3rp1p/1p4p1/8/3p... |
| 4 | Q: Is this san position 4N3/6p1/7k/p1b5/7R/6R1... |
| ... | ... |
| 3495 | Q: Is this san position 4rk1R/R7/3p3p/p3r3/1p6... |
| 3496 | Q: Is this san position 5kR1/7R/p1p1pN2/1pPpP3... |
| 3497 | Q: Is this san position r1bqr2k/ppp4Q/2nb2p1/6... |
| 3498 | Q: Is this san position 2k1R3/8/2K5/1PN5/8/8/8... |
| 3499 | Q: Is this san position 2q1knr1/1ppbQ3/1p1p4/4... |

6. Model 2 - Outputs

| | final_legals_0 | final_legals_1 | final_legals_2 | final_legals_3 |
|---|---|---|---|---|
| 0 | check | mate | check | check |
| 1 | check | check | check | mate |
| 2 | mate | neither | mate | mate |
| 3 | mate | mate | mate | mate |
| 4 | mate | mate | mate | mate |
| ... | ... | ... | ... | ... |
| 3495 | mate | mate | mate | mate |
| 3496 | mate | mate | mate | mate |
| 3497 | mate | mate | mate | mate |
| 3498 | mate | mate | mate | mate |
| 3499 | mate | mate | mate | mate |

## 8 Conclusions

In this project, we developed ChessToT, the first inference framework we know of for chess applications. The framework improves ChessGPT's performance in solving checkmate-in-one puzzles by adding a verification step to double-check generated moves.

We demonstrated Lichess's versatility by using it to generate case-specific examples for fine-tuning ChessGPT, instead of merely using it as a source of entire games as is common in most previous studies.

Moreover, we demonstrated that not only large models like GPT-3 or GPT-4 but also smaller models can benefit from inference frameworks with proper adjustments.

In summary, our project serves as a proof-of-concept for applying inference frameworks to chess and highlights Lichess as a resource for precise fine-tuning. Future work could extend similar frameworks to create LLM-based chess engines.

## Limitations

Our primary constraint was limited GPU resources. First, this limitation led us to focus on building a scaled-down model to serve as a proof-of-concept. The relative weakness of this model prevented us from developing a fully functional engine capable of outputting the best moves for a given position.

Second, our objective was to show that the simple integration of an inference system could improve the model's capabilities. However, due to the model's prompt-responsiveness limitations, adjustments were necessary. This led us to employ LoRA for fine-tuning.

Lastly, we trained the model suboptimally — opting for FEN notation over the PGN encoding that ChessGPT was originally trained on for chess puzzles, utilizing a small dataset, and employing comparatively weaker LoRA training. Consequently, our findings are not directly comparable to those reported in the ChessGPT paper for the Big-Bench checkmate-in-one test.

## References


python-chess: A chess library for python.

2023. Lichess.

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

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).

Xidong Feng, Yicheng Luo, Ziyan Wang, Hongrui Tang, Mengyue Yang, Kun Shao, David Mguni, Yali Du, and Jun Wang. 2023. Chessgpt: Bridging policy learning and language modeling.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models.

Reid McIlroy-Young, Siddhartha Sen, Jon Kleinberg, and Ashton Anderson. 2020. Aligning superhuman AI with human behavior. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery &amp Data Mining*. ACM.

OpenAI. 2023. Gpt-4 technical report.

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. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm.

Shubham Toshniwal, Sam Wiseman, Karen Livescu, and Kevin Gimpel. 2021. Learning chess blindfolded: Evaluating language models on state tracking. *CoRR*, abs/2102.13249.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention is all you need.

# A Appendix - Examples

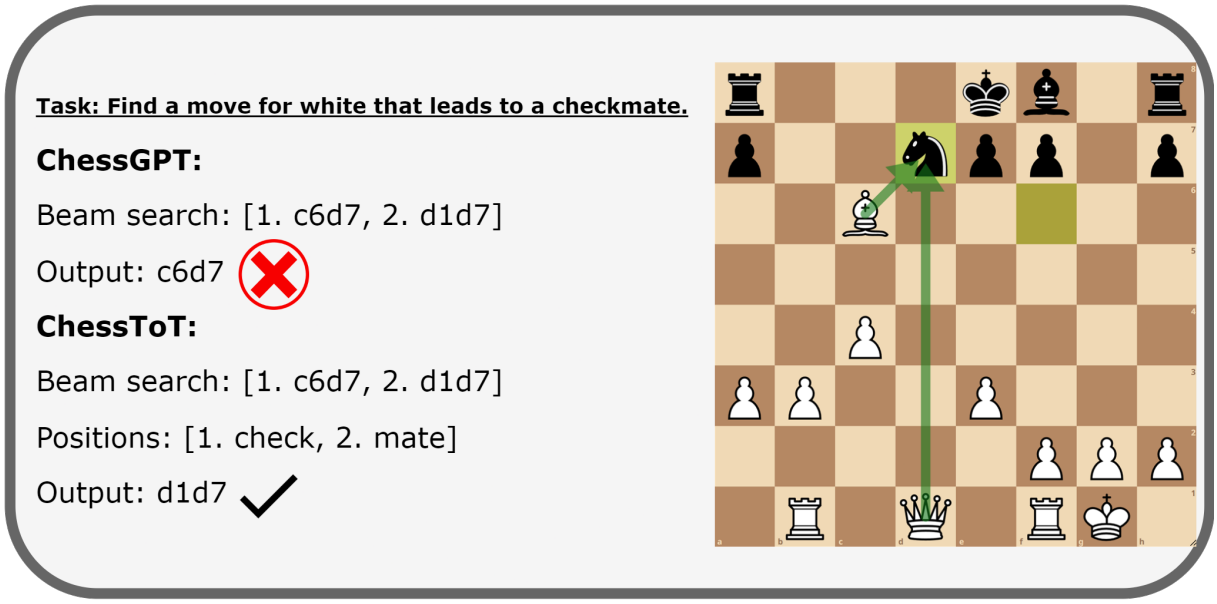


Figure 3: Example no. 196, where the ChessToT correctly chose the second move in the beam search.

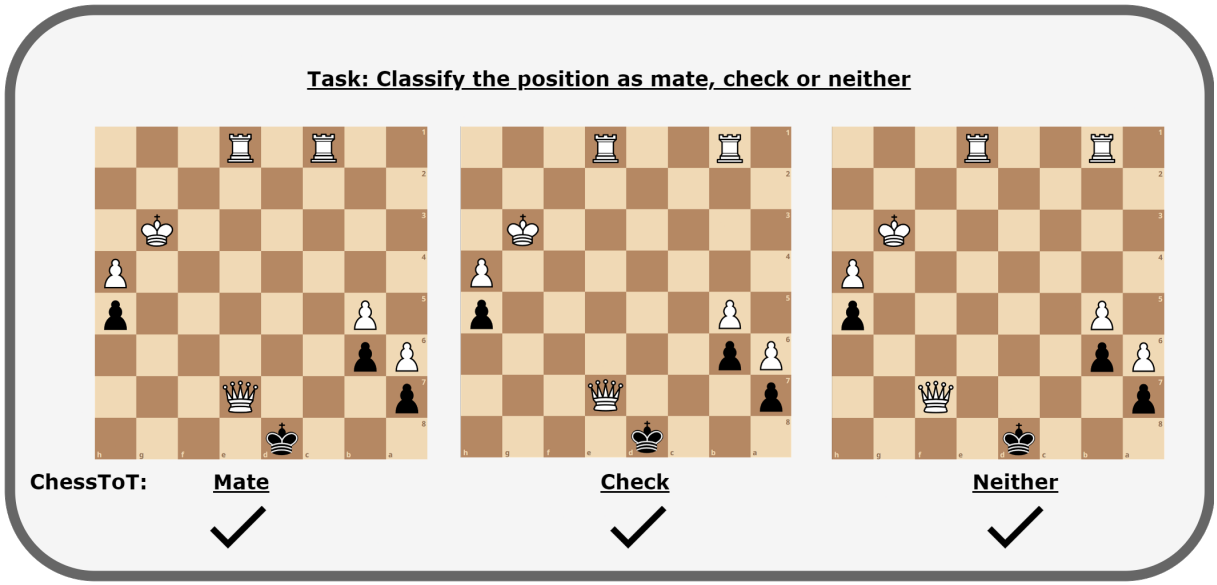


Figure 4: Demonstrating ChessToT's ability to discerns small board changes and adapt its output accordingly.