LLMs and Chess

Purva Parmar
MTech AI, Sr. No.: 24013
Indian Institute of Science
purvaparmar@iisc.ac.in

Ayan Biswas
PhD Engineering, Sr. No.: 24474
Indian Institute of Science
ayanbiswas@iisc.ac.in

Abstract

We explore the capabilities of Large Language Models (LLMs) in playing chess. We test various open-source LLMs with different parameter sizes (1B-8B) against the Stockfish chess engine to analyze their performance. We also do some Supervised Fine-Tuning, which only modestly improves the performance. Our findings indicate that smaller LLMs are notoriously bad at playing chess, and we look at various strategies to improve them, if only modestly.

1 Introduction

1.1 Motivation

The motivation for this project stems from the unique challenges chess presents as a reasoning task for large language models (LLMs). While LLMs have demonstrated remarkable success in mathematical reasoning tasks [1], chess introduces distinct complexities due to its lack of well-formulated natural language representations for move rationale. Unlike mathematics which often has unambiguous notation for reasoning steps, chess merely has algebraic notation for moves and positions [2], without standardized natural language descriptions for the strategic thinking behind each move. This gap makes chess an interesting testbed for examining LLMs' ability to handle complex, non-linguistic reasoning tasks. Understanding how LLMs process chess moves and strategies could provide valuable insights into their capacity for structured reasoning in domains that don't naturally align with linguistic patterns.

1.2 Project Overview

The core objective of our project is to investigate and potentially improve the chess-playing capabilities of open-source large language models (LLMs) with parameter sizes ranging from 1B to 8B. The project leverages the Python-chess library for the game environment, utilizing Stockfish 17.1 as the opponent to benchmark performance. A key aspect involves supervised fine-tuning of the LLMs on datasets of elite chess games to enhance their understanding of legal and strategic moves. Evaluation is conducted through comprehensive metrics that assess both move legality and quality, ensuring a rigorous analysis of the models' chess-playing proficiency. The overarching goal is to advance the performance of open-source LLMs in complex reasoning tasks, using chess as a structured and measurable domain.

2 Implementation

The tools and resources used are described in Table 1

Course Project E2 335 - Topics in Artificial Intelligence (Indian Institute of Science, 2025).

Table 1: Tools and Resources Used

Category	Details
Python Modules	Python-Chess, Stockfish 17.1, HuggingFace Transformers [3], PEFT, Datasets, BitsAndBytes [4], PyTorch
Computational Resources	Nvidia RTX A4000 (16GB), 3× Nvidia Tesla V100 (32GB)
Models Tested	Mistral-7B-Instruct-v0.3 [5, 6], Meta-Llama-3-8B-Instruct [7], openchat-3.5-0106 [8], gemma-7b-it [9], deepseek-coder-1.3b-instruct [10], phi-2 [11], Phi-4-mini-instruct [12], TinyLlama-1.1B-Chat-v1.0 [13]

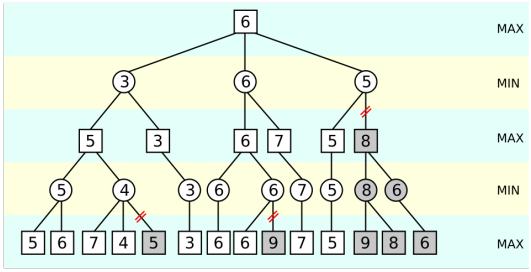


Figure 1: Alpha-Beta Pruning in the Search Tree (Image Source: Wikipedia)

2.1 Chess Engine Background

Stockfish [14] serves as our benchmark chess engine. Given a board state, it finds out the best move by a brute-force search up to a user-specifiable depth. It employs several key techniques:

- **Search Algorithm**: Depth-first search with alpha-beta pruning
- **Evaluation Function**: Combines material counting with positional factors and neural network evaluation
- **Optimizations**: Opening book for early game moves and endgame tablebases for perfect play in known endgames

The search process can be represented algorithmically as:

Algorithm 1 Stockfish Move Selection

1: Generate all legal moves from current position
2: Build search tree of possible move sequences
3: Evaluate positions using scoring function
4: Prune unpromising branches (alpha-beta pruning, Figure 1)
5: Select move leading to most favorable outcome

2.2 System Workflow

We make LLMs play with White against Stockfish (Skill Level 5) in an environment and workflow as described in the flowchart of Figure 2. After each game we keep a record of some game-level metrics as defined in subsection 3.2.

2.3 Prompt Design

We implemented two primary prompt styles:

- **State-only Prompt**:

```
Current chess position (FEN): [board.fen()]
```

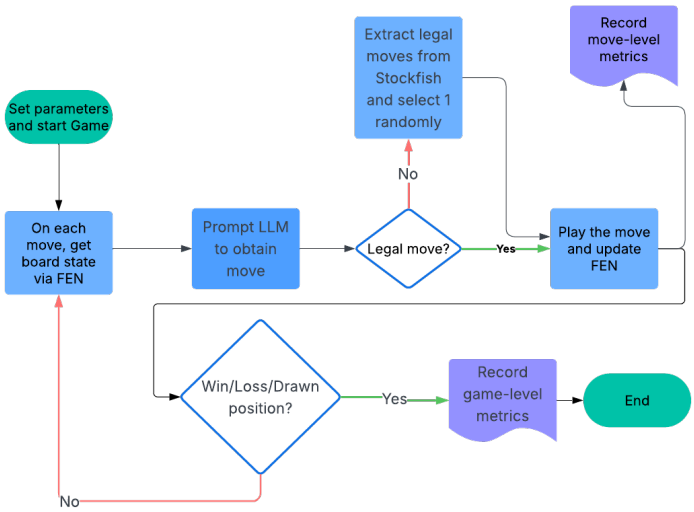


Figure 2: System workflow showing interaction between LLM and chess environment

```
What is the best move in this position?
Respond with just the move in standard algebraic notation.
```

- **Legal Moves Prompt**:

```
Current chess position (FEN): [board.fen()]
Legal moves: [legal_moves_list]
What is the best move from these options?
Respond with just the move.
```

2.4 Experimental Parameters

Table 2: Experiment Parameters

Parameter	Value
Models Tested	8 open-source LLMs (1B-8B params)
Prompt Methods	State-only / Legal move choice
Quantization	4-bit (for 7B+ models)
Temperature	0.3, 0.7 (only 0.7 for later tests)
Top-p	0.9
Games per Setting	20
Stockfish Level	5, 10 (max 20)
Move Cap per Game	200

3 Metrics and Evaluation

We established comprehensive metrics at both move and game levels:

3.1 Move-Level Metrics

- **Move Accuracy**: Whether the LLM's move appears in Stockfish's top 3 recommended moves

- **Move Quality**: The evaluation change (from Stockfish) normalized by top moves:

$$\text{Move Quality Metric} = \Delta Q = E^{\text{best}}_{\text{Stockfish}} - E_{\text{LLM}}$$

where E represents the position evaluation.

3.2 Game-Level Metrics

- **Legal Move Compliance**: Percentage of legal moves among the moves suggested
- **Average Move Accuracy**: Percentage of moves matching Stockfish's top 3
- **Average Move Quality**: Mean normalized move quality
- **Game Result**: Win (+1), Draw (0), Loss (-1)

4 Supervised Fine-Tuning

4.1 Dataset

We used the `austindavis/lichess-elite-uci` dataset [15] containing 33 million Lichess games between high-rated players (2200+ Elo) in Universal Chess Interface (UCI) notation. This dataset is processed to go move by move, and generate (Prompt containing FEN, "Best" Move Response) pairs to create the `chess-sft-dataset` with $\approx$ 1.2M examples

4.2 Parameter-Efficient Fine-Tuning

PEFT is a general term for techniques which aim to reduce the number of training parameters to make training more efficient.

- **Adapters:** We freeze the original model weights, add some extra layers ("adapters") and only train these extra layers. Could be completely extra layers inserted into the model or layers representing weight updates to existing layers.
- **LORA:** The LoRA (Low-Rank-Adaptation) approach [16] decomposes weight updates for the matrix $W \in \mathbb{R}^{d\times k}$ as:

$$\Delta W = BA \quad \text{where} \quad B \in \mathbb{R}^{d\times r},\ A \in \mathbb{R}^{r\times k},\ r \ll \min(d,k)$$

and the final update for any input x is scaled as $h = Wx + \frac{\alpha}{r}BAx$
- **Prefix Tuning:** We add learnable tokens (prefixes) to input sequences of each transformer layer and only train the prefixes.

Given hardware constraints (max 32GB GPU memory), we employed:

- 8-bit quantization via BitsAndBytes
- Low-Rank Adaptation (LoRA) targeting attention blocks
- LoRA rank = 64, α = 16
- Learning rate = 2e-5 with warmup ratio 0.1
- Train for 3 epochs

4.3 Failed Attempt at PPO Training

We experimented with the `trl` library's `PPOTrainer` to implement reinforcement learning (RL)-based fine-tuning using the Proximal Policy Optimization (PPO) algorithm [17]. The reward signal was derived from Stockfish's position evaluation, but the approach encountered several critical issues:

- **Resource Requirements**: The method necessitated simultaneous deployment of both a trainable model and a reference model (for KL divergence computation), resulting in prohibitive GPU memory demands.

- **Technical Incompatibility**: Persistent failures occurred when attempting to combine 8-bit quantization and Parameter-Efficient Fine-Tuning (PEFT) via LoRA, despite multiple configuration attempts.
- **Implementation Barriers**: The `PPOTrainer` documentation proved insufficient, with available examples demonstrating deprecated API usage patterns that failed to generalize to our use case.

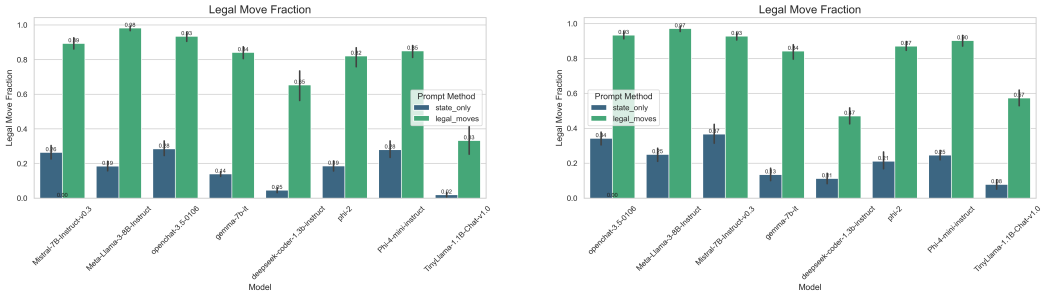
Due to these unresolved challenges, we discontinued PPO-based training and prioritized supervised fine-tuning (SFT) as our primary methodology.

5 Results

Model	State Only						Legal Moves					
	Legal Rate		Accuracy		Quality		Legal Rate		Accuracy		Quality	
	Base	SFT	Base	SFT	Base	SFT	Base	SFT	Base	SFT	Base	SFT
Mistral-7B-Instruct-v0.3	0.2643	0.3666	0.0029	0.0024	-3.5416	-8.9060	0.8933	0.9275	0.0066	0.0082	-5.6860	-3.6570
Meta-Llama-3-8B-Instruct	0.1851	0.2504	0.0172	0.0091	-3.1891	-6.5383	0.9825	0.9714	0.0068	0.0197	-2.7609	-6.7492
openchat-3.5-0106	0.2841	0.3428	0.0057	0.0079	-4.8116	-9.6796	0.9337	0.9341	0.0264	0.0258	-3.9600	-4.9167
gemma-7b-it	0.1402	0.1344	0.0026	0.0064	-9.9985	-6.6663	0.8414	0.8421	0.0150	0.0167	-8.3490	-0.3015
deepseek-coder-1.3b-instruct	0.0457	0.1130	0.0024	0.0065	0.0997	-3.9515	0.6535	0.4705	0.0139	0.0126	-7.6026	-5.9498
phi-2	0.1855	0.2119	0.0052	0.0092	-3.2015	-8.1493	0.8210	0.8712	0.0272	0.0229	-5.1665	-10.3873
Phi-4-mini-instruct	0.2800	0.2463	0.0253	0.0140	-2.1947	-5.6345	0.8495	0.9027	0.0135	0.0158	-11.0636	-3.2619
TinyLlama-1.1B-Chat-v1.0	0.0192	0.0784	0.0007	0.0026	-5.1180	-11.1808	0.3330	0.5740	0.0088	0.0179	-8.5309	-5.3038

5.1 Legal Move Compliance

Fine-tuning improved legal move compliance somewhat (Figure 3a to Figure 3b)



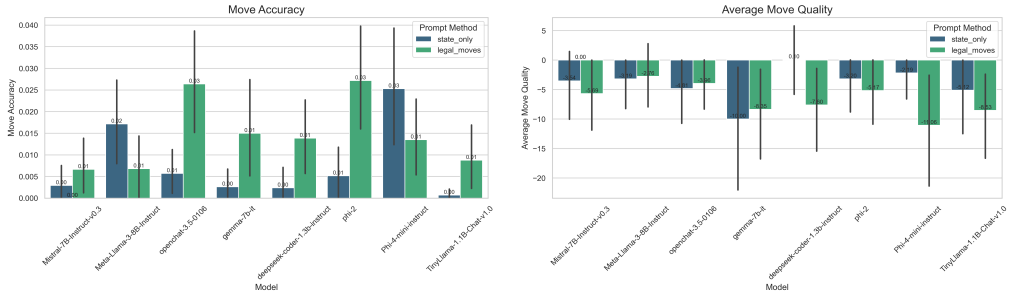
(a) Before Fine-Tuning (Mistral around 26%)

(b) After Fine-Tuning (Mistral around 36%)

Figure 3: Comparison of legal move rates before and after fine-tuning

5.2 Move Accuracy and Quality

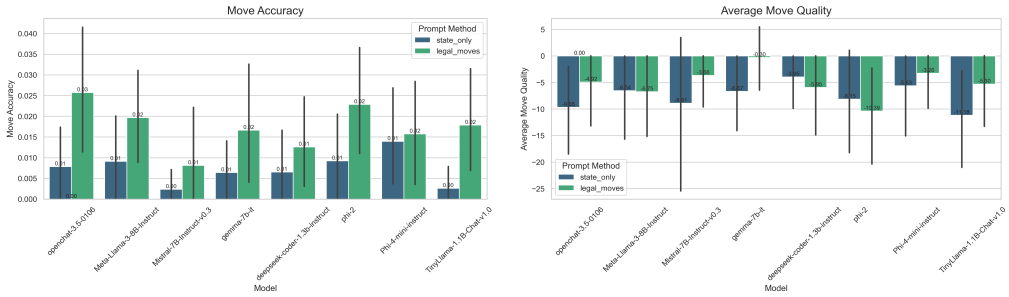
However, move quality metrics showed limited improvement (Figure 4, Figure 5):



(a) Move Accuracy

(b) Move Quality

Figure 4: Strategic move selection before SFT



(a) Move Accuracy

(b) Move Quality

Figure 5: Minimal improvement (infact, decline in some cases) in strategic move selection after SFT

6 Advanced Prompting Techniques

To address limitations, we explored chain-of-thought prompting:

```
prompt = f"""Analyze this chess position:

{self.board.fen()}

Instructions:
1. Analyze the position
2. Consider material and king safety
3. Choose the best move

Format your response as:
THOUGHT: <your analysis>
MOVE: <selected move in SAN format> (e.g., e2e4, g7g8q, 0-0, 0-0-0)"""
```

We also tried two more specialised versions of this chain-of-thought method:

6.1 Pattern-Aware Prompting

Chess has certain strategic patterns called pin, fork and skewers. We preprocessed the FEN (Forsyth–Edwards Notation) to recognise these patterns and include them in the prompt. We used the following approach of prompting:

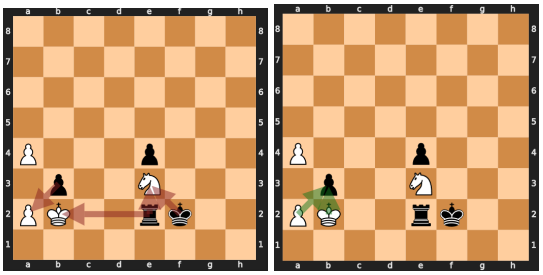
```
def build_prompt(self, board: chess.Board, legal_moves: List[str]) ->
    str:
    fen = board.fen()
    patterns = ChessPreprocessor.detect_patterns(board)

    prompt = f"""Analyze this chess position:

    {ChessPreprocessor.expand_fen(fen)}
    Detected Patterns:
    {self._format_patterns(patterns)}

    Legal Moves: {', '.join(legal_moves)}

    Instructions:
    1. Analyze the position using detected patterns
    2. Consider material and king safety
    3. Choose the best move from legal options

    Format your response as:
    THOUGHT: <your analysis>
    MOVE: <selected move in SAN format> (e.g., e2e4, g7g8q, 0-0,
    0-0-0)"""

```

```
    return prompt
```

6.2 Attack and Threat Mapping

We developed attack and threat maps (Figure 6) and included them in the prompt:

```
prompt = (
    f"Analyze this chess position:\n\n"
    f"FEN:\n{board.fen()}\n\n"
    f"Attack Map:\n{attack_map_str}\n\n"
    f"Threat Map:\n{threat_map_str}\n\n"
    f"Instructions:\n"
    f"1. Carefully assess the threats from the Threat Map and the
    attacks from the Attack Map.\n"
    f"2. Consider material, king safety, and positional advantages.\n"
    f"3. Suggest the best move from the legal options.\n\n"
    f"Format your response as:\n"
    f"THOUGHT: <your analysis>\n"
    f"MOVE: <selected move>\n"
)
```



(a) Threat map (on White) (b) Attack map (by White)

Figure 6: Visualizing board dynamics for LLM understanding

6.3 Results from chain-of-thought approaches

These small LLMs mostly output very incoherent thoughts and often generate a lot of irrelevant moves in the thought. Moreover, it does not always format the thought and answer separately as desired in the prompt, so it became difficult for us to parse out the suggested move. As a result, the rate of legal moves dropped to near-zero.

7 Contemporary Approaches in Chess and Language Models

7.1 Strategy and Tactics Annotation with MATE

Recent work by researchers from UCLA, Microsoft, MBZUAI, University of Toronto, and Peking University (Wang et al.) addresses LLMs' limitations in complex reasoning through their MATE dataset. This corpus contains 1 million chess positions annotated by experts with detailed explanations of both strategic concepts (e.g., space control, king safety) and tactical motifs (e.g., forks, pins). The team fine-tuned LLaMA-3-8B on four dataset variants: MATE-N (no explanations), MATE-S (strategy only), MATE-T (tactics only), and MATE-ST (combined strategy and tactics). Their results demonstrate that the MATE-ST model achieved 95.3% accuracy in move selection, significantly outperforming general-purpose models like GPT-4, Claude, and Gemini. This work establishes that structured language annotations can dramatically enhance LLMs' reasoning capabilities in chess, providing a framework applicable to other complex decision-making domains [18].

7.2 ChessGPT: Integrated Policy and Language Learning

The ChessGPT project pioneers a multimodal approach combining policy learning (gameplay) with language modeling (strategic analysis). Their dataset integrates 25 million samples across three categories: game data (17.5 million Lichess games, 3 million engine-analyzed positions, 3.2 million puzzles), language data (73,000 blog posts, 8,000 books, 140,000 forum discussions, 410,000 Reddit conversations), and mixed media (245,000 annotated games, 83,000 YouTube videos with transcripts). The architecture features ChessCLIP for board-text alignment through contrastive learning and a GPT-style model trained on the combined corpus. Evaluations show exceptional performance: 90%+ accuracy in move notation conversion (UCI/PGN to FEN), 53-63% accuracy in strategic multiple-choice tasks, and policy performance reaching 70%+ accuracy in checkmate puzzles and Elo 2700-level move selection. This work demonstrates strong alignment with human evaluation metrics while opening new pathways for LLM applications in instructional domains [19].

7.3 World State Tracking Through Chess

Research by Toshniwal et al. leverages chess as an ideal testbed for studying world state tracking in LLMs - the ability to maintain internal representations of entities, actions, and their temporal relationships. Unlike natural language's ambiguity, chess provides a deterministic, fully observable environment for probing model understanding. Their methodology trains transformers on UCI move sequences and evaluates world modeling through targeted prompting. Key findings reveal that while transformers can accurately track piece positions and legal moves given sufficient data, their Randomly Annotated Piece (RAP) technique - which artificially labels piece types during training - significantly improves performance in low-data regimes. This work provides fundamental insights into how LLMs build and maintain internal world representations, with implications extending beyond chess to any domain requiring state tracking [20].

8 Conclusion

Our investigation reveals that while LLMs can somewhat improve on legal chess moves through fine-tuning, strategic understanding remains challenging with limited move quality improvement. Advanced prompting techniques show promise but require careful implementation, aligning with contemporary works that demonstrate potential through specialized training approaches. Future directions could explore the integration of vision-based models for enhanced board state understanding, reinforcement learning from chess engine feedback mechanisms, annotating chess engine search trees via natural language, and larger-scale training with strategy annotations to bridge the gap between legal move generation and true chess mastery. These advancements could unlock more sophisticated reasoning capabilities in language models for structured decision-making tasks.

9 Acknowledgement

We thank Prof. Aditya Gopalan for the course E2 335 - Topics in Artificial Intelligence held at the Indian Institute of Science, Bengaluru in the January 2025 Semester and the guidance he provided throughout the project. We would also like to acknowledge the ECE Department for providing GPU resources through the TATA ELXSI AI Lab and the Arjuna ECE Cluster to aid in the project.

10 Code Repository

We have consolidated all the codes used in this project in this GitHub repository: https://github.com/TheReconPilot/LLM-Chess [21]

References

[1] Zhihong Shao et al. *DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models*. 2024. arXiv: 2402.03300 [cs.CL]. URL: https://arxiv.org/abs/2402.03300.

[2] Chess.com. *Chess Notation*. Official guide to algebraic notation. 2021. URL: https://www.chess.com/article/view/chess-notation.

[3] Thomas Wolf et al. "Transformers: State-of-the-Art Natural Language Processing". In: *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Ed. by Qun Liu and David Schlangen. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. DOI: 10.18653/v1/2020.emnlp-demos.6. URL: https://aclanthology.org/2020.emnlp-demos.6/.

[4] Tim Dettmers et al. *QLoRA: Efficient Finetuning of Quantized LLMs*. 2023. arXiv: 2305.14314 [cs.LG]. URL: https://arxiv.org/abs/2305.14314.

[5] Mistral AI. *Mistral 7B Instruct v0.3*. 2024. URL: https://docs.mistral.ai/models/.

[6] Albert Q. Jiang et al. *Mistral 7B*. 2023. arXiv: 2310.06825 [cs.CL]. URL: https://arxiv.org/abs/2310.06825.

[7] AI@Meta. *Llama 3 Model Card*. 2024. URL: https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md.

[8] Guan Wang et al. "OpenChat: Advancing Open-source Language Models with Mixed-Quality Data". In: *arXiv preprint arXiv:2309.11235* (2023).

[9] Gemma Team et al. *Gemma: Open Models Based on Gemini Research and Technology*. 2024. arXiv: 2403.08295 [cs.CL]. URL: https://arxiv.org/abs/2403.08295.

[10] Daya Guo et al. *DeepSeek-Coder: When the Large Language Model Meets Programming – The Rise of Code Intelligence*. 2024. arXiv: 2401.14196 [cs.SE]. URL: https://arxiv.org/abs/2401.14196.

[11] Microsoft Research. *Phi-2: The surprising power of small language models*. 2023. URL: https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models/.

[12] Microsoft et al. *Phi-4-Mini Technical Report: Compact yet Powerful Multimodal Language Models via Mixture-of-LoRAs*. 2025. arXiv: 2503.01743 [cs.CL]. URL: https://arxiv.org/abs/2503.01743.

[13] Zhang et al. *TinyLlama-1.1B-Chat-v1.0*. 2024. URL: https://github.com/jzhang38/TinyLlama.

[14] Stockfish Development Team. *Stockfish 17.1*. https://stockfishchess.org/blog/2025/stockfish-17-1/. Mar. 2025. URL: https://stockfishchess.org/blog/2025/stockfish-17-1/.

[15] Austin Davis. *Lichess Elite Database (UCI Format)*. https://huggingface.co/datasets/austindavis/lichess-elite-uci. Dataset containing elite Lichess games in UCI move format. 2023.

[16] Edward J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021. arXiv: 2106.09685 [cs.CL]. URL: https://arxiv.org/abs/2106.09685.

[17] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG]. URL: https://arxiv.org/abs/1707.06347.

[18] Shu Wang et al. *Explore the Reasoning Capability of LLMs in the Chess Testbed*. 2025. arXiv: 2411.06655 [cs.CL]. URL: https://arxiv.org/abs/2411.06655.

[19] Xidong Feng et al. *ChessGPT: Bridging Policy Learning and Language Modeling*. 2023. arXiv: 2306.09200 [cs.LG]. URL: https://arxiv.org/abs/2306.09200.

[20] Shubham Toshniwal et al. *Chess as a Testbed for Language Model State Tracking*. 2022. arXiv: 2102.13249 [cs.CL]. URL: https://arxiv.org/abs/2102.13249.

[21] TheReconPilot. *Experiments with LLMs and Chess*. https://github.com/TheReconPilot/LLM-Chess. 2025.