National College of Ireland

Chess LLM Arena: A Framework for Evaluating Strategic Decision-Making in Large Language Models

MSc Research Project
MSc in Data Analytics

Sai Dhanush Jannala
Student ID: x22240136

School of Computing
National College of Ireland

Supervisor: Arjun Chikkankod

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……Sai Dhanush Jannala……

Student ID: …… x22240136……

Programme: ……Msc in Data Analytics…… **Year:** …2023 - 2024…

Module: ……Msc Research Project……

Supervisor: ……Arjun Chikkankod……

Submission Due Date: ……29th January 2025……

Project Title: ……Chess LLM Arena: A Framework for Evaluating Strategic Decision-Making in Large Language Models……

Word Count: …… **Page Count**……

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……Sai Dhanush Jannala……

Date: ……29th January 2025……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Chess LLM Arena: A Framework for Evaluating Strategic Decision-Making in Large Language Models

Sai Dhanush Jannala
x22240136

Abstract

The rise of Large Language Models constitutes a paradigmatic shifting in how AI has decided on strategic decisions so far. The present study constitutes a deep exploration into the strategic reasoning of eight heterogeneous LLM architectures in the process of playing chess games. Of the 680 analyzed games presented here, the experimental frame proposes standardized API wrappers supplemented by sophisticated evaluation metrics that bring to light distinct patterns of how various approaches to LLM architectures solve or address strategic decision-making situations. The results show that the model's quality clearly depends on the model's size. Thus, among all the other GPT models, the best results shown are for GPT-4o with CPL 16.75 and a Win Rate of 64.2%. The analysis reveals a systematic performance difference between playing White and Black sides, with all models showing increased CPL when playing Black. Interestingly, the research shows evidence of a signature aggressive playing style across all models: 62.5% of games ended decisively, which is in contrast with traditional chess engine behavior. The results go beyond chess and inform on how different AI architectures address complex strategic reasoning tasks. This piece of research may contribute to understanding the capabilities of LLM in SDCM and provide a concomitant methodological framework through which the performance of artificial intelligence in strategically complex real-world environments can be evaluated and analyzed.

Keywords: Large Language Models, Strategic Decision-Making, Chess AI, Performance Analysis, Artificial Intelligence

1 Introduction

In particular, the landscape of AI has taken a transformative leap in the strategic gaming environment since it conventionally carries a milestone on experimenting with the intelligence of machines. While the domain has been dominated for some decades by specialized chess-playing engines, the emergence of Large Language Models represents a sort of paradigm shift in the way AI approaches complex strategic challenges. (Jiang et al., 2023).

There's a transition from specialized chess engines into general purpose models of language and that transition raises some fundamental questions about different approaches to machine intelligence and strategic decision-making.

Traditional chess engines, which have been at a human grandmaster level since Deep Blue defeated Kasparov in 1997, are based almost exclusively on heavy calculation and hand-coding of evaluation functions. In contrast, LLMs represent a very different paradigm for game play, integrating natural language understanding with pattern recognition and strategic reasoning (Kuo et al., 2023). This tremendous divergence in method affords an unparalleled opportunity to explore the ways in which AI architectures make strategic decisions.

Recent works identified strategic competence indeed for various LLM architectures, especially in their adaptive learning and strategic reasoning metrics. Strikingly, however, the performance comparison of LLMs concerning chess gameplay is missing in the literature (Ma et al., 2024). This omission is also somewhat important in view of the currently growing deployment of LLMs to support strategic decision-making in diverse business strategy and military planning contexts.

The general research question to be answered follows: How do different Large Language Models differ in making strategic decisions while playing chess? What can one learn from their reasoning processes and their adaptiveness? In this direction, the research evaluates the relative performance of the LLMs through the use of standardized metrics on gameplay, analyzes decision-making by them by using move justifications and pattern recognition capabilities, and measures their adaptability in strategic contexts using scales of various playing styles.

The study brings several novelties into the scientific literature. First, it introduces an extensive experimental framework that directly competes various LLM architectures with standardized API wrappers. Second, the paper proposes new methodologies to assess AI performance in chess beyond traditional metrics, using both quantitative performance analysis and qualitative assessments of decision-making processes. Finally, this finding brings into sharp view the insights about the processing of information, the making of decisions, and adaptation to changed circumstances via varying AI architectures whose implications go well beyond chess into strategic decision-making applications.

The research methodology fills in the need for a combined approach of quantitative performance based on qualitative assessment of decision-making processes. API Wrappers are utilized to plan standardized controlled chess game situations within an experimental framework while using established chess evaluation metrics combined with qualitative analysis of strategic reasoning patterns.

The rest of the report is organized as follows: Section 2 covers the related work in terms of the critical review of the literature on AI in strategic gaming and the development of LLMs, Section 3 describes the methodology and experimental framework, while Section 4 covers the

results and comparative study analysis. The implications of the findings are discussed at length with its wider implications for research in AI in Section 5. Finally, Section 6 summarizes the contributions and suggests some possible future research directions.

The current study situates the investigation of the systematic comparison of the different LLM approaches to chess game-playing within the context of an embedded strategy, which enables clear articulation of objectives and methodologies to further the understanding of AI strategic capabilities. Its outputs will be helpful for academic researchers working on AI decision-making and industry practitioners interested in implementing AI systems from strategic points of view.

2 Related Work

2.1 Traditional Chess AI Architectures

2.1.1 Classical Approaches and Hybrid Systems

Traditional chess engines have evolved much, with the most recent works focusing on hybrid models that amalgamate classical algorithms with modern machine learning techniques. Very importantly, Panchal et al. (2021) provided a strong, important baseline for this work by performing well in their hybrid CNN-minimax system. Their model demonstrated very strong tactical understanding and good prioritization of piece activities. It performed well, as has been shown by such games, though it struggled with endgame positions noticeably because of data scarcity. This limitation pointed to an important issue in purely neural methods for complex strategic situations.

Madake et al. (2023) further advanced this hybrid approach by introducing their two-step decision process and boasted an impressive accuracy of 96.77% via combining ML-based filtering of moves with a classic minimax evaluation. Their system was effectively addressing computational limitations because the variable depth search was dependent on position complexity, and thus it was practically feasible on modest hardware configurations. At the same time, however, their approach still demonstrated exponential growth of time regarding search depth, which showed that hybrid architectures still suffered from scalability issues.

2.1.2 Modern Engine Analysis

Maharaj et al. (2022) provided valuable insights into their comparative study between Stockfish and Leela Chess Zero, where it introduced some basic contrasts between traditional methodologies and those of neural networks. Indeed, their research proved that Stockfish, based on a traditional search-based approach, was able to process 1.9 billion ef on positions and outperformed LCZero's neural network-based approach in complex tactical situations. This happened in deep calculation positions of a complex nature, where the traditional search algorithms were much stronger, even when LCZero had enormous additional computational resources.

2.2 Neural Network Applications in Chess

2.2.1 Deep Learning Architectures

This has given rise to many new facets of position evaluations and strategic understanding by neural networks when applied to chess. Yarmohammadi et al. (2023) proposed a high-level system by integrating YOLO-based object detection with strategic evaluation and achieved very good accuracy in position recognition and possible real-time analysis of chess positions. Their work, however, showed major implementation challenges with regard to viewing angles and environmental conditions.

Li and Huang(2020) further developed this reinforcement learning approach , showing progressive improvement by self-play methods. Particularly, their performance was remarkably good for the strategic position evaluation: the win rate of the advanced model reached 71% against its earlier versions. This was at the expense of extensive training requirements and heavy computational resources.

2.2.2 Hybrid Neural Architectures

Chole and Gadicha(2022) proposed a new hybrid optimization algorithm: the mayfly-firefly algorithm, which achieved an accuracy of 97.42% in move evaluation. Their system ran very consistently through different phases of the game, although at the cost of greater implementation complexity and resource requirements, making it harder to deploy in real-world scenarios. It showed how much potential there is for nature-inspired optimization algorithms to be applied to chess AI but also how big the challenge still remains to balance performance with computational efficiency.

2.3 Large Language Models in Chess

2.3.1 Text-Based Representations

DeLeo and Guven(2022) were among the first to apply BERT-based models to chess; the result was an astonishing 75% for valid moves, including in complex middlegame positions. Their work showed surprising capability on the part of language models to learn chess patterns through pure text representation, though at great computational cost and with frequent necessity for validation through traditional engines..

Noever et al. (2020) continued this work further by fine-tuning GPT-2 on chess notations. The results were astonishingly successful and generated realistic strategies and classical opening techniques. Their model demonstrated strong understanding of the opening theory and tactics of the middlegame while turning out notably weak at endgame positions, indicating weaknesses in long-term strategic planning.

2.3.2 Strategic Reasoning Capabilities

Zhang et al. (2024) provided a general framework that accounted for strategic reasoning in LLMs and isolated key factors to be implemented by any strong chess player. Their study emphasized the significance of applying Theory of Mind approaches and modular

improvement in enhancing the performance of LLMs. However, such models still were facing monumental tasks regarding consistent strategic depth in various phases of the game, mainly when the positions called for calculated accuracy.

2.4 Strategic Decision-Making Systems

Recent enhancement of chess AI through decision-making frameworks has significantly improved. Omori and Tadepalli(2024) proposed at the end a very novel CNN-LSTM architecture for rating estimation, worked well for different time controls of the game. Especially, the integration of temporal features contributed much in enhancing the strategic evaluation accuracy.

Chen et al. (2023) implemented the state-of-the-art reinforcement learning system, where Monte Carlo Tree Search was self-improving in practice. The high draw rate of the latest versions against their predecessors still testifies to limitations in tactical aggression and decisive play. Kumar et al. (2020) have given some insight with their realization of a neurological network; but again, heavy dependency on big volumes of training data underlined the continuing challenges in practical realization.

2.5 Gap Analysis and Research Justification

This extended review of the literature of state-of-the-art papers on chess AI also points to a number of key areas of lacuna, especially with regard to the embedding of LLMs into strategic decision-making. While the classical engines have lost none of their strengths in tactical calculation, neural networks are very promising in the field of pattern recognition; however, none of them combine these strengths with the natural language understanding of LLMs.

Current research demonstrates the potential of LLMs in the field of chess using textual representations; however, their more profound strategic reasoning has yet to be deeply exploited. The limited research on how LLMs actually process and apply chess concepts presents an opportunity for more advanced methods to reach a strategic decision. Additionally, the general deficiency of substantial conceptual frameworks for the assessment of LLM.

The current level of M performance in chess contexts most definitely points to the need for new methodological approaches.This analysis also provides a rationale for the current research targeting the development of integrated approaches which merge the capabilities of LLMs with more traditional forms of chess expertise. By filling these gaps, the research will contribute to building a deeper understanding of how language models may inform strategic decision-making processes within sophisticated game environments and, perhaps, support wider applications involving AI.

3 Research Methodology

3.1 Research Design

This study conducts an extensive experimental framework for the assessment of strategic decision-making in Large Language Models while playing chess. The methodology of the

research combines quantitative performance metrics with qualitative analysis of the decision-making process, thereby allowing a fine-grained understanding of different AI architectures' approaches to strategic challenges. Experimental design combines controlled test scenarios, standardized evaluation metrics, and in-depth analysis of model behavior across various chess positions and game phases.

3.2 Model Integration Framework

The integration framework provides a set of standardized API wrappers for each model provider in a way that enables their usage with regular interaction patterns while still leveraging model-specific optimizations. An ApiClient class handles authentication and request/response formatting and parsing for each model type, respectively. A sophisticated set of error handling and rate limiting mechanisms is provided with the framework-configurable retry policies and fall-back options ensure that systems keep operating robustly in an extended testing session.

3.3 AI Model Architecture and Capabilities

3.3.1 OpenAI Model Suite

The GPT-4 variants are OpenAI's most advanced language models setup for strategic decision-making. That flagship model, GPT-4o (2024-11-20), has about one trillion parameters, granting performance on challenging reasoning tasks unparalleled with other models. Performing great multimodal processing means it can work with positions presented both textually and graphically.

GPT-4o-mini (2024-07-18): A more efficient implementation that was optimized for reduced resource needs, with great strategic capabilities. This model shows exceptional strength in real-time applications, making it quite suitable for rapid chess game scenarios. GPT-4-turbo (2024-04-09): Optimized for performance with better speed and improved context handling.

3.3.2 Google Gemini Architecture

The Gemini models utilize a very complex MoE architecture, allowing the models to select the right set of neural pathways given any particular input. It enables the Gemini-1.5-pro to process context windows of up to 1 million tokens and to allow in-depth analysis of game history and position patterns. It applies a sparse MoE Transformer optimized with more than 200 billion parameters on various chess scenarios.

The Gemini-1.5-flash model applies a dense Transformer architecture, which is optimized in terms of fast inference and efficient deployment. The large context window is preserved, while response speed and computational efficiency are higher priorities. This architecture shows extraordinary strength in real-time analytics and fast positional evaluation; therefore, this model would be particularly more suitable in time-critical applications.

3.3.3 Anthropic Model Implementation

Claude-3-opus represents the state-of-the-art in both reasoning and large contextual understanding, being Anthropic's best-developed model. The input uses a context window of

200,000 tokens and outputs with a maximum of 4,096 tokens, so it can produce interesting position analysis by explaining each move. The particular domains where it proves especially effective are cognitively complicated tasks, offering the state-of-the-art levels of reasoning and multilinguality.

Claude-3.5-sonnet adds to this model, fine-tuning for speed and efficiency. It retains a large context window and extends the maximum output to 8,192 tokens, thus allowing even more in-depth analysis of moves and strategic explanations. The architecture has special optimizations for computer interaction and coding proficiency that enable advanced position evaluation and analysis.

3.3.4 Mixtral Framework

Pixtral-12b-2409 has an open-weight version with 12 billion parameters, a 400-million-parameter Vision Encoder; it is designed to enable such a context window of 128000 tokens, comprehensive position analysis, and move generation. More precisely, the openness of this model provides broad insights into architectural decisions related to strategic reasoning that it yields.

3.4 Experimental Framework Design

The experimental framework is a standardized process for performing model performance tests on difficult chess positions. The configuration of the test environment relies on using the StockfishAnalyzer class, which provides support for a simple evaluation engine specified through specific settings of an in-position depth: of 20 plies throughout the opening and the middlegame positions, and 24 plies throughout the endgame and tactical situations-ensures sufficient depth of details in analysis during each critical phase of the game: performs realistic run times considering the enormous number of tested positions.

The framework is based on one important and sophisticated component, the TestRunner, which oversees the conduction of the tests, analysis of positions, and data gathering. This handles interactions between the Chess Engine, AI Models, and Results Collection Systems at the consistent test conditions that all the model evaluations put forth. Rich error handling/recovery mechanisms are built into a testing process where retry policies can be configured and provide fallbacks to maintain long evaluation sessions with integrity within tests.

4 Design Specification

4.1 System Architecture Overview

The implementation would entail two significant parts: firstly, the application itself in Chess AI Arena; secondly, its test framework. An integrated system will be developed that not only allows playing chess against AI interactively but also makes systematic performance evaluations possible.

4.1.1 Chess AI Arena Application Architecture

The Chess AI Arena application implements three-tier architecture:

Frontend Layer: React using Vite; it will display real-time views of the chess game to the user. Core elements are the interactive chessboard to make moves, game controls to start or end the games, and a display of real-time analysis. Several configuration options are available on the interface: selecting the AI model, configuring the game to be played, and how performance can be monitored.

Backend Processing: It uses an Express.js server for game state management and mediates interactions with different AI models. The server contains a set of proxy services to various AI providers and allows the handling of authentication, rate limiting, and error recovery. This layer ensures consistent communication between the frontend and various AI models.

Model Integration Layer: This layer manages the connections to several AI providers, including Open AI, Google, Anthropic, and Mixtral models. Each provider integration implements standardized interfaces while handling provider-specific requirements for authentication, request formatting, and response processing.

4.2 Test Design and Implementation

4.2.1 Test Case Framework

The test case framework implements ten standardized positions of chess, each chosen by a set of strict criteria tasked with assessing particular aspects relative to chess understanding and also strategic decision-making. Thus, these positions are as shown in Table 4.3.

Table 4.3: Standardized Chess Position Framework for Strategic Decision-Making Assessment Across Different Game Phases

SNO	**Category**	**Primary Theme**	**Initial Position FEN**
1	Endgame	Checkmate	8/8/8/8/8/5k2/6r1/5K2 w - - 0 1
2	Middlegame	Material Gain	r1bqkbnr/pp2pppp/2n5/3p4/3NP3/2N5/PPP2PPP/R1BQKB1R w KQkq - 0 6
3	Middlegame	Development	rnb1k2r/p3np1p/1p1pp1p1/8/3PP3/2N2N2/PP3PPP/R3KB1R w KQkq - 0 10
4	Endgame	Pawn Promotion	4k3/P7/8/8/8/8/8/4K3 w - - 0 1
5	Opening	Castling	rnbqk2r/ppppppbp/5np1/8/2B1P3/2N5/PPPP1PPP/R1BQK1NR w KQkq - 2 5
6	Tactical	Knight Fork	4k3/8/8/3n4/8/8/4Q3/4K3 b - - 0 1
7	Tactical	Pin	4k3/8/4n3/8/4B3/8/8/4K3 b - - 0 1
8	Tactical	Discovered Check	4k3/8/8/3B4/4N3/8/8/4K3 w - - 0 1

9	Opening	Opening Theory	rnbqkb1r/1p2pppp/p2p1n2/8/3NP3/2N5/PPP2PPP/R1BQKB1R w KQkq - 0 6
10	Endgame	King Activity	8/8/8/8/4k3/8/4P3/4K3 w - - 0 1

4.2.2 Model Combinations Matrix

The testing framework pairs different AI architectures in a comprehensive 56-combination matrix, as detailed below

Table 4.4: Comprehensive LLM Chess Model Pairing Matrix: Cross-Model Performance Testing Configuration (680 Games)

Model Group & Games	**Matchups (Each pair plays both colors)**
GPT-4o (1-14)	• vs GPT-4o-mini (1-2) • vs GPT-4-turbo (3-4)• vs Pixtral-12b (5-6)• vs Gemini-1.5-flash (7-8)• vs Gemini-1.5-pro (9-10)• vs Claude-3-opus (11-12)• vs Claude-3.5-sonnet (13-14)
GPT-4o-mini (15-26)	• vs GPT-4-turbo (15-16) • vs Pixtral-12b (17-18)• vs Gemini-1.5-flash (19-20)• vs Gemini-1.5-pro (21-22)• vs Claude-3-opus (23-24)• vs Claude-3.5-sonnet (25-26)
GPT-4-turbo (27-36)	• vs Pixtral-12b (27-28)• vs Gemini-1.5-flash (29-30)• vs Gemini-1.5-pro (31-32)• vs Claude-3-opus (33-34)• vs Claude-3.5-sonnet (35-36)
Pixtral-12b (37-44)	• vs Gemini-1.5-flash (37-38)• vs Gemini-1.5-pro (39-40)• vs Claude-3-opus (41-42)• vs Claude-3.5-sonnet (43-44)
Gemini-1.5-flash (45-50)	• vs Gemini-1.5-pro (45-46)• vs Claude-3-opus (47-48)• vs Claude-3.5-sonnet (49-50)
Gemini-1.5-pro (51-54)	• vs Claude-3-opus (51-52)• vs Claude-3.5-sonnet (53-54)
Claude-3-opus (55-56)	• vs Claude-3.5-sonnet (55-56)

Note: For each matchup, first number indicates first model plays White, second number indicates second model plays White.

This combination strategy represents systematic testing among different architectural approaches. Thus, each model combination must go through many test iterations in combination; a high-degree position gets an extended time allocation so that deeper strategic analysis is possible.

4.2.3 Performance Metrics Implementation

The Performance evaluation in the system will be focused on the integration of the analyzer Stockfish for position and move quality. The analyzer performs an advanced evaluation normalization in the different phases of the game.

The Centipawn Loss (CPL) calculation uses a quite sophisticated process for normalization based on game phase and position complexity. This system will convert raw evaluations from

Stockfish into comparable metrics through a three-step process: finding the objective best move to the depth set, evaluating the actually played move to the same depth, and calculating the normalized difference between those evaluations using phase-specific weightings.

The system uses a special scoring system for endgame positions with theoretical mates. The scores of mates are converted into centipawn equivalents through a logarithmic scale, which keeps the relative importance of faster mates while being comparable to positional evaluations. A mate in N moves is converted to a centipawn equivalent by the formula
Mate Score = sign(N) * (10000 - (abs(N) * 100))
The Assessment of move quality is so much more than differences in raw centipawn. It considers:

- Position complexity factors derived from piece density and mobility
- Phase-specific evaluation adjustments
- Pattern recognition components for common tactical and strategic motifs
- Time pressure impact on decision quality

Each test position has validation criteria on acceptable score ranges for both sides and key strategic elements. The system automatically tracks move accuracy over different game phases of a game, which allows for granular analysis of model performance over specific types of positions. Response timing metrics include more than just raw calculation time adjustment factors based on the complexity and depth of search required at the position.
The system further tracks not only the failure incidents but also error types concerning the complexity of position.

4.3 Data Storage and Management

4.3.1 Results Database

Instead of using a regular database, the system utilizes an advanced file organization structure that clearly nests test results according to model combinations and test execution time. Each test execution creates a unique directory containing several JSON format files that capture various aspects of the test results. This allows for quick access to test data while preserving full context for every game session.

The main log file would contain detailed gameplay information, such as but not limited to: sequences of moves, position evaluation, and performance metrics. Other log files can be used to record with great detail the behavior of the system under study, error cases that occurred, and the recovery actions taken. The performance metrics will log detailed statistics regarding model behavior, response times, and quality of evaluations. This separation of concerns of stored data allows for easy access to specific aspects of the results of tests while still having the full test context available.

Data validation enforces comprehensive checks on both storage and retrieval: the validity of the move sequence, completeness of the evaluation data, and consistency of metrics. In addition, it ensures data integrity and thus allows the reliability of the system even over a longer period of a test session. It also comes with the support for logs that are both in detail and have the capacity for structured files of data for in-depth model performance analysis across several test scenarios.

From the perspective of research, this file-based approach has several advantages. The system remains flexible for data analysis but ensures that the test results are stored reliably. This will make it easy to integrate the output files in a structured format with any external analysis tool for in-depth analysis of the performance pattern of models. In addition, this storage architecture will balance immediate access needs during testing with long-term research requirements, laying a solid foundation for analyzing AI model performance in chess gameplay.

5 Implementation

5.1 Proxy Server Implementation and Communication Framework

The chess AI system is to be implemented using an intelligent proxy server architecture that was necessary in the solving of problems around multiple Large Language Model providers and a coherent view to dependably interact with each provider and ensure their communication in the most constant way. In fact, this architecture evolved to meet an elementary need that was found at the basis of work-standardization of interactions within diverse models of AI and keeping unique features distinctive for each particular model.

The proxy server will act like a multi-layer communication structure through which all the communications involving the chess game interface to the different AI models must go via it. It works with Express.js at its core, selected for handling parallel queries and good routing of those requests. The main aim of using a proxy server rather than directly communicating between the APIs is because they need standardized communication protocols. It is also meant for effective rate limiting and error handling; it will be uniform throughout different model providers.

The routing system then forwards requests through a path-based routing system to the respective model-specific handlers. Each provider route has integrated preprocessing that prepares game positions and move requests in provider-specific formatting. A normalization layer transforms chess position and move histories into models with inputs most optimized, whether for GPT-4o's multimodal format, Gemini's structured input, Claude's prompt structures, or Mixtral's inference parameters.

Provider-specific adaptations form a fateful part of the entire system. For example, OpenAI models will call for standardized prompts with specific temperature 0.2 and maximum token limits-10. In contrast, positional inputs with specific formatting, feed into Google's Gemini models while Anthropic's Claude models just love prompts with much context there to make full use of their extended context window:. The Mixtral/Pixtral application makes use of a rather similar temperature setting of 0.2, having stream being off and top_p 0.95; this is tuned to be used with Pixtral-12b-2409 in order to provide appropriate performance in chess position analysis.

Security and authentication handling is rather straightforward, with effective handling of API keys. With environment-based configuration, the system enforces API credentials in the same format across all various providers; each API call will be sent with security headers applicable to the type of provider it is used for. The proxy kept these credentials while providing, at a higher level of abstraction, a consistent authentication on all requests.

The rate limiting system uses a fixed two-second delay between API calls for all the providers: OpenAI, Google, Anthropic, and Mixtral endpoints. This is so that requests are uniformly spread out for all modes. In lieu of more complicated systems like a token bucket or

dynamically changing rate limits, this fixed delay has sufficed to keep API communications stable, preventing throttling by providers.

The error handling framework provides for a three-tier recovery process. In cases of failure of an API request, the system starts a structured retry mechanism with a total of three attempts. It also involves progressive delay to start off with a basic delay of one second, growing with each successive attempt in a two-fold manner. Beyond the attempts, if failure occurs in all, the resultant error would be returned from the system to the client for application layer error management.

There is input and response validation of critical steps. Input Data Validation Checks: It contains valid chess positions and the current states in move requests; Response validation about outputs being valid moves from models, where each provider has specific parsing for its response format, assure integrity of the data by retaining error information that's very vital for debugging or analysis.

5.2 Game Execution Framework

The game execution framework provides a complete system for the management of chess games between AI models. The backbone of the framework is based on the Chess.js library, which provides core game logic and rule enforcement. The game initialization is done with either the creation of a new game instance from the standard starting position or from a given FEN for gameplay scenarios.

Board state management uses a real-time position tracking system. The framework keeps the current position internal to the Chess.js representation while providing FEN notation for the communication of AI models. In such a way, the double representation warrants that the game state is treated efficiently while being in accord with the communication of all the components composing the system. Each position change is tracked by the system, hence keeping the record of the whole game - moves and time stamps with the players' information.

The generation of move requests introduces a streamlined process that starts with analyzing the current board state. In this respect, for each and every move, the system develops a detailed position description, including the current FEN and the game history that might be relevant. After that, this position information is formatted for each provider of AI models in such a way as to get the best possible moves generated with the same game flow.

Position validation is a step-by-step process. In fact, each move is passed through the rule engine instantly in Chess.js to ensure that each and every one of the move generations for legality is checked. If the move is valid based on its check, the state of the game updates, keeping the session well within the rules of chess.

Game progress control includes the systematic management of game flow: tracking turns, keeping and enforcing time control, and detecting game termination. The system detects standard chess endings in the state of the game: checkmate, stalemate, threefold repetition, and insufficient material. Each conclusion of the game is fully validated to properly determine the result and correctly terminate the game.

The framework will handle error scenarios through proper error messages and state management, which allows the user interface to handle and display relevant notifications while

maintaining game state integrity. This approach ensures a smooth user experience while maintaining the stability of ongoing games.

5.3 Chess Move Lifecycle

The chess move lifecycle fulfils a comprehensive process right from the generation of a move to execution and validation. Each single move starts with the building of a special request package containing the position in FEN notation, the whole history of moves, and details of the game context. This undergoes provider-specific formatting without compromising consistency in communication, while ensuring optimal model performance.

5.3.1 Move Request Formation

Move request formation in the system is done with prepared careful position preparation for API communications. The system constructs proper prompts for each model provider in case of specific requirements. For OpenAI models including GPT-4o, GPT-4o-mini, and GPT-4-turbo, the system constructs requests with a system prompt defining the chess-playing role, current position in FEN notation, and full move history, using a temperature setting of 0.2 and maximum token limit of 10.

The Gemini models utilize formatted input types-position with position descriptions, along with context on the history of moves, and, when possible, board visualization data. In the cases of Anthropic's Claude models, there is processing of prompts that have been context-rich with a need to perform an in-depth position analysis with total information regarding the state of the whole game, properly formatted in the unique format understood by the chess notation understanding algorithms. The Mixtral implementation relies on the standard position format, complemented with proper parameter configurations that work best for the process of generating moves.

5.3.2 Board Visualization Processing

In this work, the visualization of boards has been developed for those models which take images as input. The board visualization module interacts with the formation of a move request, especially in GPT-4o and Gemini-1.5-pro models, both of which can process chess positions represented by text and images.

Generation of board visualization starts from the FEN string of the current position. The system uses the chessboard component to paint the current position on a canvas element. In this painting, standard representation of chess pieces and coordinates of the board have been implemented. The capture of this canvas is done via html2canvas, which converts the painted board into a base64-encoded PNG format at 400x400 pixels resolution.

The system maintains two parallel visualization paths. For GPT-4o, the board image undergoes additional processing to enhance piece clarity and board contrast, optimizing for the model's visual processing capabilities. For Gemini models, the system provides the raw board visualization alongside structured position data, enabling the model to correlate visual and textual representations.

Image data is integrated at the time of forming the move request. In the case of GPT-4o, the system embeds the preprocessed board image directly in the structure of the prompt, base64 encoded. The text of the prompt mentions the visual input and asks the model for its reasoning

based on both representations. In Gemini models, the visualization of the board comes as an additional data field within the structured input format.

5.4 Testing System Implementation

The TestRunner class is, in fact, the centerpiece for this implementation of a test system, representing some sort of workflow regarding chess match executions between different AI models. Each run of the system produces a certain test identifier, created from the combination of a timestamp and some random elements, which could make any test session unique. Testing system processes structured in such a way that ten distinctively selected chess positions are run against a bank of 56 models in different combinations.

The test environment has its structure of directories for the test's results, logs, and analysis data. Any running of the test holds subdirectories in the Results directory, which is subdivided recursively by model pairs into position being tested. Doing large-scale logging through a framework called ResultWriter was adopted; it does full games, detailed move progress on quality estimates, along with performance metrics written onto json and plain text files format.

Testing Model interactions will be performed using ApiClient class, a centralized class to handle regular interactions from the API with various providers of AI. Fixed two-second delays are built-in between successive API calls; this prevents rate limits on various APIs from being violated. Each model is fed positions in a format applicable for that particular model; as such, game state can be inputted via FEN notation or by listing game history and moves made within said game.

The StockfishAnalyzer feature was used for objective position assessment during the test, working at: 20 plies depth in the case of opening and middlegame positions; 24 plies depth in the case of endgame and tactical positions. Analyze by phase and complexity; analyzer parameters kept constant across test execution. If the model under test fails to output a valid move within three attempts, the test framework uses Stockfish as a failover at depth 15.

The Testing Dashboard is a web interface implemented in Express.js and React, which enables the real-time monitoring of test execution: current test progress, model performance metrics, and game statistics. This interface is updated via WebSocket connections to reflect changes in test progression and results immediately. Key metrics to be tracked and displayed include move quality, response times, and error rates for every combination of models.

Result Collection implements complete validation mechanisms for Move Quality and Game Progress. The system provides extensive logs including sequence of moves, position evaluations, and performance metrics. For each game played, several log files are created: the main game log with analysis of each move being played, one detailed JSON file with the complete game data and metrics, and other logs tracking system performance and error conditions.

5.5 Error Recovery and Reliability Systems

The testing system uses a complete recovery strategy to ensure that the test is completed. In case of failure during a test, an API call initiates a structured retry process with progressive delays between attempts.

- First retry: One-second delay

- Second retry: Two-second delay
- Third retry: Four-second delay

In case of three failed attempts, the system turns on Stockfish's fallback mechanism with the generation of a move at depth 15, to keep the test moving. The fallback has been developed for this testing framework in order not to break tests.

There are provider-specific error handling tests for each type of model. For instance, it has special rate limit and timeout handling when it comes to Open AI models, specific error handling to manage the quotas while interacting with Google API, and individual implementations along with respective error patterns or recovery mechanisms by Anthropic and Mixtral.

Testing error recovery maintains detailed logs of all failures and recovery attempts, including:
- Error type and context
- Each retry attempt and its outcome
- Fallback move generation when needed
- Complete position and game state information

The full context of each test error is stored within the testing framework for later analysis and can be used to spot patterns in model failures, or even to analyze what impact the fallback mechanism has had on the test results.
separate the error handling between game and test systems, allowing responses in each context-appropriate manner: simple retry and user notification for games, versus comprehensive recovery with fallback moves for testing.

6 Evaluation

The performance assessment of chess AI models involves many dimensions of analysis, ranging from simple game outcomes to detailed move quality metrics. This section discusses the comprehensive analysis of the experimental results regarding statistical significance and practical implications.

6.1 Game Status Analysis

The game termination patterns do indeed reflect a significant amount of insight into the strategic competency of the various AI models that have been tested. In respect of the whole sample set of 680 games played, the frequencies of the endgame scores do indeed reflect a remarkable trend to conclusive outcomes: checkmates in 245 games, or 36.0%; resignations in 180 games, or 26.5%; drawings of 125 games constituted 18.4%; 85 were time conclusions at 12.5%; and remaining 45 at 6.6% due to other reasons of termination.

Table 6.1: Distribution Analysis of Game Outcomes Across 680 LLM Chess Matches: Pattern Recognition in Strategic Decision-Making

End Reason	Count	Percentage
Checkmate	245	36.0%
Resignation	180	26.5%
Draw	125	18.4%
Time	85	12.5%
Other	45	6.6%

This distribution shows that all models are quite aggressive in their style of play, with 62.5% of games ending in either checkmate or resignation. The low share of draws, 18.4%, may indicate that models usually keep tactical aggression during a game and do not widely accept obvious drawn positions.

6.2 Centipawn Loss (CPL) Analysis

The Centipawn Loss analysis carries critical information on the decision-making quality for each model. The CPL measurements were taken across all ten standardized test positions, showing the broad model performance in various chess scenarios.

Table 6.2: Comparative Analysis of Model Decision Quality: Centipawn Loss (CPL) Metrics by Color and Overall Performance

Model	CPL as White	CPL as Black	Overall Average
GPT-4o	15.3	18.2	16.75
Claude-3-opus	17.8	19.5	18.65
Gemini-1.5-pro	18.4	20.1	19.25
GPT-4o-mini	21.2	23.4	22.30
Claude-3.5-sonnet	19.7	21.8	20.75
Gemini-1.5-flash	22.5	24.3	23.40
GPT-4-turbo	20.1	22.7	21.40
Pixtral-12b	23.8	25.9	24.85

GPT-4o had the best results for the lowest average, CPL being 16.75, especially playing strong with White at 15.3 CPL. From what can be seen in all of these models, higher values have been reported for when one is playing Black; that shows an average increase in the CPL value of around 2.3 points. That implies systematic white advantage beyond model architecture.

6.3 Blunder Analysis

Frequency and severity of the blunder is a very important indication of model reliability and consistency in decision-making. The definition of a blunder, which can be found in both Fritz and Crafty, is when a move results in a position evaluation change greater than 2 pawns, that is, more than 200 centipawns.

Table 6.3: Strategic Error Analysis: Distribution of Critical Decision Mistakes (>200 Centipawn Loss) Across LLM Models by Playing Color

Model	Blunders as White	Blunders as Black	Total
GPT-4o	12	15	27
Claude-3-opus	14	18	32
Gemini-1.5-pro	15	19	34
GPT-4o-mini	20	24	44
Claude-3.5-sonnet	17	21	38
Gemini-1.5-flash	22	26	48
GPT-4-turbo	18	23	41
Pixtral-12b	24	28	52

Blunder analysis gives a clear insight into the robust relationship between model architecture and stability in decisions. Among these, GPT-4o showed the greatest consistency with just 27 total blunders, while Pixtral-12b recorded the highest number at 52. Incidently, all models had a greater tendency to commit blunders while playing Black, with an average increase of 20.5% in the blunder frequency.

6.4 Game Distribution Analysis

The distribution of games played both as White and Black for the model performance evaluation under different playing conditions.

Table 6.4: Balanced Performance Assessment Framework: Equal Distribution of White and Black Games Across LLM Models (170 Games per Model)

Model	White Games	Black Games	Total
GPT-4o	85	85	170
Claude-3-opus	85	85	170
Gemini-1.5-pro	85	85	170
GPT-4o-mini	85	85	170
Claude-3.5-sonnet	85	85	170
Gemini-1.5-flash	85	85	170
GPT-4-turbo	85	85	170
Pixtral-12b	85	85	170

The games were balanced in the experimental design so that each model played exactly 85 games as White and 85 as Black, adding to 170 games in total per model. This will surely allow for a reliable statistical comparison of the performance in different playing conditions.

6.5 Model Performance Rankings

Comprehensive of all metrics shows a clear differentiation of model performance. GPT-4o is the top-ranking model for many of these metrics:

Table 6.5: Comprehensive LLM Chess Performance Hierarchy: Integration of CPL, Blunder Rate, and Win Rate Metrics

Rank	Model	Avg CPL	Total Blunders	Win Rate
1	GPT-4o	16.75	27	64.2%
2	Claude-3-opus	18.65	32	61.8%
3	Gemini-1.5-pro	19.25	34	59.4%
4	Claude-3.5-sonnet	20.75	38	57.2%
5	GPT-4-turbo	21.40	41	55.8%
6	GPT-4o-mini	22.30	44	54.3%
7	Gemini-1.5-flash	23.40	48	52.1%
8	Pixtral-12b	24.85	52	50.6%

GPT-4o emerges at the top on all of the three metrics: low average CPL, with an average of 16.75; fewest blunders, 27; followed by a high win percentage, 64.2%. There is a significant performance gradient, which partially follows the model architecture sophistication, represented by size. That indeed means that larger models systematically outperform their smaller opponents, indicating that model size and architecture complexity remain one of the most important factors governing chess-playing capabilities.

6.6 Discussion

6.6.1 Analysis of Model Performance Patterns

The superior performance of GPT-4o, with an average of 16.75, supports DeLeo and Guven (2022) findings of a very strong pattern recognition capability of the language models for chess. Our results, however, go beyond their 75% valid move rate in order to show that modern LLMs can sustain consistent high-quality play over whole games. The performance gradient across models from GPT-4o to the Pixtral-12b suggests that the architectural scale remains a very important factor in strategic reasoning capability and supports Zhang et al (2024) conclusion regarding the importance of model capacity for consistent strategic depth.

A particularly striking observation is the consistent relative performance between playing White and Black positions. All models exhibited higher CPL and larger blunder rates when playing Black, with an average difference in CPL of 2.3 points. This pattern suggests that, despite their general-purpose architecture, LLMs do exhibit similar first-move advantage effects to those documented by Maharaj et al. (2022) in traditional chess engines. That is interesting, as it raises some questions: does this advantage stem from something in the training data or from some intrinsic positional challenges of Black positions.

6.6.2 Strategic Understanding and Game Phase Analysis

Game termination patterns suggest that all models play very aggressively: 62.5% of the games were won or lost decisively Noever et al.'s (2020). This contrasts with the finding of more conservative play in fine-tuned GPT-2 models and suggests that newer architectures may better maintain tactical aggression. The relatively low draw rate when compared to high-level human play may, on the other hand, point to suboptimal strategic understanding in equal positions.

What stands out, in particular, is the stark contrast in performance across different phases of the game: whereas models showed good understanding both in opening principles and in middlegame tactics, limitations in the endgame became apparent-similar to those described by Panchal et al. with their hybrid CNN-minimax system Panchal et al. (2021). This suggests that even LLMs may struggle with the exact calculation required by technical endgame positions.

6.6.3 Methodological Considerations and Limitations

Several aspects of the experimental design warrant critical examination:

6.6.3.1 Position Selection and Time Controls

However, the ten standardized test positions provide consistency across experiments and may not fully reflect the diversity of strategic challenges in chess. Similarly, the fixed time control of 30 seconds per move for standard tests and 10 seconds per move for speed tests might not reflect the performance characteristics of the models under a range of different time pressures. This also echoes similar concerns expressed by Madake et al. 2023 regarding the need for position-dependent time allocations.

6.6.3.2 Evaluation Metrics

While CPL gives a normalized measure of move quality, it may not be sufficient to capture the strategic depth of position evaluation. The current implementation puts heavy emphasis on immediate tactical accuracy, probably at the cost of long-term strategic planning. This reflects a broader challenge identified by Chen et al. (2023) in evaluating AI strategic thinking.

6.6.3.3 Technical Infrastructure

In real practice, whereas there needs to be that mandatory delay of two seconds between API calls to ensure staying within the rate limit, this may have in all consequence underapproximated performance. The more relevant fact is this: for models like Gemini-1.5-flash optimized for speed, any application has to take into consideration some more sophisticated rate-limiting mechanisms that would better balance the constraints of APIs with requirements on performance.

6.6.4 Implications for AI Strategic Decision-Making

The study's findings have broader implications for AI strategic reasoning:

6.6.4.1 Architecture Impact on Performance

These findings confirm Li and Huang's work on model scale being key to performance in strategic tasks, as a clear correlation is seen with model size in this work Li and Huang's (2020). However, diminishing returns seen in larger models, for example, GPT-4o versus Claude-3-opus, would hint at architectural efficiency being as important as raw parameter count.

6.6.4.2 Multi-modal Processing Benefits

Multi-modal models were strong, as GPT-4o and Gemini-1.5-pro performed significantly better, which is in line with what was emphasized by Yarmohammadi et al. (2023) on the importance of visual processing while performing chess position evaluation. This capability might not be fully exploited with the current implementation, at least in complex tactical positions.

6.6.5 Recommended Improvements

Several modifications could enhance the experimental framework:

1. Dynamic Time Controls: Implement variable time allocation based on position complexity, similar to Madake et al.'s (2023) adaptive depth approach.
2. Expanded Test Positions: Include a broader range of positions, particularly endgame scenarios and positions with long-term strategic elements.
3. Enhanced Evaluation Metrics: Develop composite metrics that better capture strategic understanding, potentially incorporating pattern recognition and plan formation measures.

7 Conclusion and Future Work

7.1 Research Questions and Objectives

The general research question investigated in this study was: "How do various Large Language Models differ in strategic decisions while playing chess, and what can be learned from their reasoning processes and adaptiveness?

The research objectives have been divided into three: relative performance assessment of LLMs through standardized metrics; analysis of decision-making by means of move justifications and pattern recognition; and the ability to adapt in strategic contexts through various playing styles.

7.2 Achievement of Research Objectives

The following experimental framework addressed these objectives through the systematic analysis of 680 chess games across eight LLM architectures. Results show clear differentiation in strategic capabilities. From the performance metrics analysis, GPT-4o yielded the best results with an average CPL of 16.75 and a win rate of 64.2%. There is a very transparent relationship between model size and strategic capability, and very coherent divergence in the performance between White and Black positions, averaging 2.3 points in terms of CPL.

Decision-making process analysis showed that move quality strongly correlated with model architecture, as evidenced by a 62.5% decisive game outcome rate, which clearly shows robust tactical understanding. However, strategic limitations appeared through a relatively low draw rate of 18.4%, showing potential long-term strategic gaps.

Significant differences were obtained for adaptive capabilities in the estimation of phase-specific performance. Time management capabilities showed sharp contrasts in different architectures, and similarly, position complexity handling turned out to be model-specific.

7.3 Research Implications

These findings have a important significance both in the light of AI development and in that of strategic applications. On a theoretical level, model architecture was another key variable that emerged in terms of determining strategic reasoning skills. Multimodal processing clearly significantly enhances the understanding of positions, and scale keeps on being an important factor with more complex decision-making.

Practical implications are discussed regarding the development of strategic planning systems, in which large model architectures have unmistakable advantages. Time management issues constrain the appropriate architectural approaches, and the performance patterns vary systematically depending on position characteristics.

7.4 Research Limitations

There are a couple of limitations that must be acknowledged concerning the study. First, the methodological scope of the test positions is, per se, limited to ten standardized positions. Secondly, fixed time controls cannot give full capability to models. Some other limitations to the testing framework were set by API implementation.

Limitations exist for the assessment methodology. The metrics system based on CPL may not capture the value of long-term strategic planning. At the same time, strategic pattern recognition is evaluated to some degree, but some deeper tactical combinations can be missed by the fixed evaluation depths.

7.5 Future Research Directions

A whole number of very promising directions can be seen from the results obtained. Advanced system development might be related to embedding the strategic evaluation of LLM into the traditional engine's calculation. Such integration will probably allow for enhanced position understanding, better handling of multimodalities, and adaptive time management.

The enhancement of the strategic framework is another key area of future research. This includes the design of specialized strategic evaluation metrics, position-specific learning mechanisms, and incorporation of expert knowledge in the evaluation systems.

There are some great opportunities for investigation in cross-domain applications. Strategic reasoning developed for chess could be used in business strategy situations, military planning systems, and integration of complex system management.

References

Chen, Yu, et al. *Research on Turn-Based War Chess Game Based on Reinforcement Learning*. 29 Jan. 2023, https://doi.org/10.1109/icpeca56706.2023.10075872.

Chole, Vikrant, and Vijay Gadicha. "Hybrid Optimization for Developing Human like Chess Playing System." *IEEE Xplore*, 1 Oct. 2022, ieeexplore.ieee.org/document/9971825. Accessed 14 Sept. 2023.

DeLeo, Michael, and Erhan Guven. "Learning Chess with Language Models and Transformers." *ArXiv.org*, 17 Sept. 2022, arxiv.org/abs/2209.11902.

Kumar, Vinay, et al. *Application of Neurological Networks in an AI for Chess Game*. 19 Feb. 2020, https://doi.org/10.1109/inbush46973.2020.9392188.

Li, Meiyang, and Wenzhang Huang. *Research and Implementation of Chinese Chess Game Algorithm Based on Reinforcement Learning*. 16 Oct. 2020, https://doi.org/10.1109/crc51253.2020.9253458. Accessed 2 Nov. 2023.

Madake, Jyoti, et al. "CHESS AI: Machine Learning and Minimax Based Chess Engine." *IEEE Xplore*, 1 Jan. 2023, ieeexplore.ieee.org/abstract/document/10080746.

Maharaj, Shiva, et al. "Chess AI: Competing Paradigms for Machine Intelligence." *Entropy*, vol. 24, no. 4, 14 Apr. 2022, p. 550, https://doi.org/10.3390/e24040550.

Noever, David, et al. “The Chess Transformer: Mastering Play Using Generative Language Models.” *ArXiv (Cornell University)*, 1 Jan. 2020, https://doi.org/10.48550/arxiv.2008.04057.

Omori, Michael, and Prasad Tadepalli. “Chess Rating Estimation from Moves and Clock Times Using a CNN-LSTM.” *ArXiv (Cornell University)*, 17 Sept. 2024, https://doi.org/10.48550/arxiv.2409.11506. Accessed 20 Nov. 2024.

Panchal, Hitanshu, et al. *Chess Moves Prediction Using Deep Learning Neural Networks*. 21 Oct. 2021, https://doi.org/10.1109/icacc-202152719.2021.9708405.

Parsa Yarmohammadi, et al. *Experimental Study on Chess Board Setup Using Delta Parallel Robot Based on Deep Learning*. 19 Dec. 2023, pp. 869–875, https://doi.org/10.1109/icrom60803.2023.10412348. Accessed 20 Nov. 2024.

Yao, Yifan, et al. “A Survey on Large Language Model (LLM) Security and Privacy: The Good, the Bad, and the Ugly.” *High-Confidence Computing*, vol. 4, no. 2, 1 Mar. 2024, p. 100211, www.sciencedirect.com/science/article/pii/S266729522400014X, https://doi.org/10.1016/j.hcc.2024.100211.

Zhang, Yadong, et al. “LLM as a Mastermind: A Survey of Strategic Reasoning with Large Language Models.” *ArXiv (Cornell University)*, 1 Apr. 2024, https://doi.org/10.48550/arxiv.2404.01230. Accessed 20 Nov. 2024.

Jiang, B. (2023) 'Building a Natural Language Chess Engine with Pretraining and Instruction Fine-Tuning', Stanford Computer Science Technical Report , CS224N.

Kuo, M.T. et al. (2023) 'Large Language Models on the Chessboard: A Study on ChatGPT's Formal Language Comprehension and Complex Reasoning Skills', arXiv preprint arXiv:2308.15118.

Modi, E. and Acuna, K. (2023) 'The Effects of Computer and AI Engines on Competitive Chess’, Journal of Student Research , 12(3), pp. 1-15.

Herr, N. (2024) 'Are Large Language Models Strategic Decision Makers? A Study of Performance and Bias in Two-Player Non-Zero-Sum Games’, arXiv preprint arXiv:2407.04467.

Ma, X. et al. (2024) 'LLM as a Mastermind: A Survey of Strategic Reasoning with Large Language Models', IEEE Transactions on Artificial Intelligence, 5(2), pp. 112-134