National College of Ireland

Deep learning chess framework to synthesize language model reasoning with tradational engines

MSc Research Project
Msc in Atrificial Intelligence

Saketh Reddy Banala
Student ID: x23327596

School of Computing
National College of Ireland

Supervisor: Taylou Maniganze

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Saleth Reddy Banala
Student ID:	X23327596
Programme:	Msc in Artificial Intelligence **Year:** 2025
Module:	Research Practium
Supervisor:	Taylou Maniganze
Submission Due Date:	15/09/25
Project Title:	Deep learning chess framework to synthesize language model reasoning with tradational engines
Word Count:	9,267 **Page Coun : 2**7

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: Saketh Reddy Banala

Date: 15-09-2025

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):	

Deep learning chess framework to synthesize language model reasoning with tradational engines

Saketh Reddy Banala
x23327596

Abstract

This research investigates comparative large language models' (LLMs) performance against classical chess engine capability in strategic analysis and accuracy of moves across a range of chess positions. While classical engine engines are accurate in calculation, they are devoid of explanation to provide insight to human understanding. Conversely, modern language models are demonstrated to have been capable of strategic reasoning but, to the present understanding, are rarely challenged through formal analysis with a chess scenario. This research work corrects this by constructing a comprehensive evaluation protocol between Gemini 1.5 Pro and Claude 3.5 Sonnet over classical engines Berserk 13 and Caissa 1.22, with Stockfish 17 posing as evaluation arbitrator. The protocol uses a systematic evaluation of 180 games played with 12 engine pairs and 15 positions that were carefully chosen to show different types of scenarios, such as opening, tactics, position, endgame, and strategy. The performance metrics include Centipawn Loss (CPL), Game Point Loss (GPL), and move quality categorisation. These results show that traditional engines are much better than language models, with an average performance gap of 237.24 CPL, or almost 2.37 pawns. Position-specific analysis shows that performance differences are not always the same: 90 CPL during opening positions, 247 CPL during tactical positions, and 109 CPL during positional play. Language models were good at recognising patterns in tasks, but they weren't as good at making accurate tactical calculations. The findings indicate possible advantages of hybrid systems that integrate the computational precision of traditional engines with the pattern recognition and explanatory functions of language models. This research offers empirical evidence of the synergistic strengths inherent in both methodologies and establishes a basis for the creation of an integrated chess analysis system that bridges the divide between machine accuracy and human comprehension.

Keywords: Large Language Models, Chess Engines, Artificial Intelligence, Comparative Analysis, Stockfish, Claude 3.5 Sonnet, Gemini 1.5 Pro, Centipawn Loss, Game Point Loss, UCI Protocol, Chess AI, Performance Evaluation, Hybrid Systems, Strategic Analysis, Computational Chess

1 Introduction

Over the course of more than 70 years, chess has developed into a testbed for artificial intelligence, starting with Claude Shannon's groundbreaking work in 1950 and continuing into the present day with neural network-driven engines and massive language models. Thanks to improved search algorithms and evaluation features, chess engines have become

superhuman. One example of an engine with a rated strength that is noticeably higher than 3600 Elo is Stockfish 17 (Chole &Gadicha, 2023).The discrepancy between machine accuracy and human insight arises because these conventional engines, despite having greater computational capacity, do not provide a human-readable explanation for their strategic thinking.

The purpose of this study is to evaluate and contrast language-based AI models' and standard chess engines' performance, analysing each of their advantages in analytical
capability toward strategy and accuracy at move level in broad configurations of chess positions. By fusing computational analysis with the affordances of natural language reasoning, large language models (LLMs) such as Claude 3.5 Sonnet and Gemini 1.5 Pro created new opportunities for chess analysis (Minaee et al., 2024). These models were successful in spotting trends and offering strategic commentary, but formal chess analysis abilities are still largely unexplored.

An intrinsic drawback of current chess AI programs serves as the impetus for this study. Using deep search trees and neural network evaluation to centipawn-quality standards, traditional engines like Stockfish, Berserk, and Caissa specialise in calculating optimal moves (Maharaj et al., 2022).However, they don't explain their strategic thinking behind the decisions in a way that would help people who want to learn more. Modern LLMs, on the other hand, may not have precise tactically tailored engines, but they can offer strategic insights and thorough explanations. This division makes it possible to observe how well these complementary models perform in a wide variety of chess situations.

This investigation is motivated by the following research question: How do language-based AI models stack up against conventional chess engines in terms of move accuracy and strategic analysis ability when tested across various chess position types?The main requirement of whether LLMs can be useful additions to or replacements for conventional engines in chess analysis applications is specifically addressed by this question. In addition to using quantifiable performance metrics, the research takes into account the qualitative component of understanding the strategies that each contributes to chess analysis.

Recent developments in both offered special circumstances for comparative analysis. Neural network structures have been used by classical engines; Stockfish, for example, has adopted NNUE (Efficiently Updatable Neural Networks) technology while retaining its computational search capabilities (Agarwal et al., 2024).Meanwhile, LLMs have shown capabilities within game-playing strategy, with probable strengths within pattern recognition and strategy planning (Topsakal et al., 2024; Lorè& Heydari, 2024). To up to date knowledge, no work has been present that comprehensively compared these paradigms within standard positions of chess.

The primary beneficiaries ofthis research are chess teachers seeking AI-enhanced teaching tools, developers building hybrid AI programs, and scholars examining the intersection between general-purpose and specialized AI. Teachers could use research to build programs

that combine classical engine accuracy with explanation capability from language models. Researchers of AI benefit from insight into how a variety of architectures compute tasks involving strategic reasoning, something that could be applied to future work with both specialized and general AI systems.

The four primary goals of the research are as follows: (1) creating a thorough framework that will allow for a systematic comparison between analysis using language models and evaluation by classical engines across a wide range of chess positions; (2) creating a combination of quantitative and qualitative measures to evaluate move accuracy and strategy comprehension; (3) creating evaluation methodologies to measure the advantages and disadvantages of each approach; and (4) creating a whole testing system with standardised positions within opening theory, tactical unifications, positional play, and endgame technique. The research question of how language-based AI models compare to traditional engines in chess analysis is directly addressed by each of these objectives.

The methodology employs a rigorous experimental setup with 180 games spread across 12 engine combinations and 15 carefully chosen test positions to extract a variety of chess expertise. The following skills were specifically evaluated for each position: endgame technique, tactical calculation, positional awareness, and strategic planning. The setup combines LLM APIs (Claude and Gemini) with UCI-compliant classical engines (Berserk 13 and Caissa 1.22) to evaluate move quality using centipawn loss (CPL) and game point loss (GPL) metrics using Stockfish 17 as a neutral arbiter.The procedure then maintains both qualitative understanding and quantitative performance to provide a fair comparison.

This study's success criteria are well-explained and have quantifiable results. The study aims to show the viability of integrating the two paradigms, identify position ranks where each paradigm leads or lags behind, and quantify performance differences using statistical analysis with 95% confidence levels. The study uses precise metrics to measure success, such as average CPL by position ranks, move classification accuracy, time efficiency, and game outcome analysis. The indicators aid in the objective assessment of whether language models can be used in chess analysis programs to augment classical engines.

The format of this paper is designed to give a thorough summary of comparative analysis. A critical literature review of recent chess engine architectures and LLM applications to strategy games is presented in Chapter 2. Research methodology details, including a testing framework and evaluation metrics, are covered in Chapter 3.Details about the architecture and design of the system are given in Chapter 4. Implementation details are given in Chapter 5. Evaluation results and statistical analysis are presented in Chapter 6. Results and their implications are discussed in detail in Chapter 7. With significant contributions and suggestions for further research, Chapter 8 comes to a close. This format enables a methodical presentation of the research process and findings while upholding academic integrity all along the way.

2 Related Work

The history of chess artificial intelligence has developed over a series of paradigms, from classical algorithmic approaches to modern approaches to using neural networks and, more recently, the introduction of large language models. This review of literature criticizes past work within these paradigms, contrasting strengths and weaknesses within several approaches and discovering the research gap motivating the current study.

2.1 Traditional Chess Engine Architectures

Classic chess engines have relied primarily on game tree search algorithms with position evaluation functions. Patel et al. (2022) developed the Vecma engine from Negamax with an "ideal vector map" technique, a 2D array of heuristic values to direct pieces to favorable positions by early game. While their vector mapping provides explicit positional direction, their system's dependence on pre-defined values limit flexibility to dynamic positions, and lack of comparative performance tests with commercial engines forbids competitive assessment.

Madake et al. (2023) combined machine learning with minimax, through a linear estimator trained from grandmaster games to associate moves with "good" or "bad" classes to allow 50% move pruning before search. Despite achieving 96.77% accuracy with classification, the approach depends largely on training data accuracy and reduces, unnecessarily, chess's nuanced evaluation requirements via binary classification.

Preethi et al. (2023) incorporated alpha-beta pruning to optimize Negamax and a symmetric fuzzy means algorithm to attain an adaptive search depth, but failure to include detailed measures of performance and unclear contribution by the fuzzy means component prohibit useful evaluation. Similarly, Upasani et al. (2021) developed Dev-Zero with static piece-square tables, attaining competence against 1500 Elo opponents but with outdated evaluation methods and a limited 3-ply search depth that unduly limits tactical understanding.

Srivastava et al. (2024) demonstrated an evolutionary computation algorithm learning from grandmastersgames to relate computational and human-like expertise. Nevertheless, limited implementation details prohibit reproducibility, and unchecked convergence properties produce reliability worries over a broad position set.

2.2 Neural Network and Machine Learning Approaches

The combination of neural networks with chess engines represents a important paradigm shift. Agarwal et al. (2024) developed NIRNAY, a combination of Convolutional Neural Networks (CNNs) with Negamax algorithm and ordering of moves technique. The CNN component achieved 39.16% accuracy in predicting moves, but the integrated system showed improved performance by pre-filtering moves prior to classical search. The approach efficiently reduces computation complexity with competitive play strength. The engine resulted a rating between 1450-1520 in rapid chess format, outperforming approximately 34 million users who played on chess.com.

Nonetheless, extremely low CNN accuracy points to a wide range of areas that require neural network component improvement. Complex middlegame positions where tactical calculation is paramount cause the system's performance to deteriorate.

Using an 8-layer CNN structure, Panchal et al. (2021) employed deep learning neural networks to forecast chess moves. 8x8x14 representations of board positions, including piece positions and attacking patterns, are fed into the system. The network was able to identify powerful strategies like forks and pins with a considerable level of efficacy, and it achieved 39.16% evaluation accuracy for boards.However, the system struggled to defend important squares and was never able to predict opponents' strategies. In most cases, a propensity to put piece activity ahead of defensive play resulted in positions that were detrimental. Additionally, the model's inability to account for opponents and perform even basic calculations led to poor decision-making, especially in crucial situations.

Kagkas et al. (2023) took a different approach by using fuzzy means algorithms to train Radial Basis Function (RBF) neural networks. The engine takes milliseconds to evaluate position, which is a lot less time than standard engines.The RBF network demonstrates strong performance with changing input variations and position types. The study utilized a 1,500 high-level games' dataset with over 80,000 positions to provide detailed training coverage. The system, however, tackles position evaluation only with no moves, which limits its practical application as a complete chess engine. The paper also does not match up with state-of-the-art engines, from which it becomes difficult to evaluate its relative strength.

Maharaj et al. (2022) conducted a comparative analysis between Stockfish and Leela Chess Zero using Plaskett's Puzzle, a complex endgame study. Their findings revealed that Stockfish's traditional search-based approach outperformed Leela's neural network method in this specific tactical position. The study highlights the complementary nature of calculation-focused and intuition-based approaches, with Stockfish excelling in positions requiring deep tactical precision while Leela showed strength in pattern recognition. This research illuminates the fundamental differences between algorithmic paradigms but focuses on a single position type, limiting the generalizability of conclusions. The study suggests that neither approach alone provides complete chess understanding.

2.3 Testing and Evaluation Methodologies

Méndez et al. (2023) advocated for metamorphic testing of chess engines, shedding light on engine evaluation discrepancies between symmetrically equivalent positions. Their technique identified that even top engines like Stockfish demonstrated evaluation variations by more than 10% when positions were gone through symmetric transformations. This study exposes fundamental reliability weaknesses in current chess engines and provides a design structure for systematic verification. Despite their experience, the study shows that conventional engines are unpredictable when comparing similar positions. The research, however, is mainly concerned with identifying problems rather than providing answers, and the tested metamorphic relationships may not fully capture all position equivalences that apply to chess.

2.4 Large Language Models in Strategic Gaming

A recent advancement with important ramifications is the use of large language models in gaming. Vidler and Walsh (2025) used Prisoner's Dilemma and Rock Paper Scissors to test LLM performance, finding significant biases against strategy adaptation and randomisation. They came to the conclusion that LLMs systematically deviated from optimal mixed strategies, making them inappropriate for games requiring probabilistic decision-making.

This research leaves major doubts concerning applying LLMs to chess, where probability-based evaluation over sets of candidate moves plays a central role. The limitation of this study to extremely simple games opens major doubts concerning generalizability to complex areas such as chess.

Topsakal et al. (2024) evaluated LLMs with grid-based games including Tic-Tac-Toe, Connect Four, and Gomoku. Their exhaustive benchmark demonstrated inconsistent results with game types and with prompt types, with LLMs scoring higher with less complicated, list-based prompts compared to visual representations. The study demonstrated game rule learning and strategy generation by LLMs without being explicitly trained. The Claude 3.5 Sonnet achieved up to 94.29%of win rates as a first player with certain games but incurred higher disqualification rates with complicated visual prompts. The study provides valuable game-playing capability insights into LLMs but a significant jump from simple grid games to chess, which was outside the study's scope, represents a significant challenge.

By contrasting GPT-3.5, GPT-4, and LLaMA-2, Lorè and Heydari (2024) investigated the strategic behaviour of LLMs in game-theoretic tasks. The results showed typical patterns: LLaMA-2 showed a better grasp of game structures and contextual considerations; GPT-3.5 showed strong context sensitivity but poor abstract strategic reasoning; and GPT-4 concentrated on game structure rather than context with poor discrimination between game types. This study found that LLM architecture significantly affects their capacity for strategic thought. However, small, two-player games limited the direct applications of the research to the chess complexity levels.

2.5 Comprehensive LLM Surveys and Architecture Analysis

A survey of the architectures, uses, and difficulties of large language models was presented by Raiaan et al. (2024). The evolution from early transformer models to current models that can be used for complex reasoning tasks is covered in the analysis.The survey suggests significant capabilities including in-context learning, instruction compliance, and multi-step reasoning, all potentially translatable to analysis of chess. However, the survey's wide scope precludes a detailed examination of game-playing applications in particular. In situations where accuracy is desired, like in chess, the issues noted—such as stochasticity, hallucination, and context window limitation—raise concerns regarding the dependability of LLMs.

Another thorough analysis of LLMs was also included by Minaee et al. (2024), who paid special attention to architectural innovations and milestone performances. Their paper shows how multimodal systems, including emergent capabilities with larger models, evolved from GPT-like models.According to the survey, models such as GPT-4 and Claude 3.5 Sonnet are performing exceptionally well on reasoning tasks, scoring over 90% on demanding benchmarks. However, the survey does not provide a thorough explanation of strategic game applications, which raises the question of how general capability translates to chess-specific requirements.

2.6 Synthesis and Research Gap

Two distinct approaches to the development of chess AI are revealed by the literature. The work of Madake et al. (2023), Patel et al. (2022), and others demonstrated the formidable computational precision and tactical calculation of classical engines. Deep search trees and complex evaluation functions enable the systems to achieve top-tier game power.In order to

improve evaluation accuracy while preserving search-based decision making, the more recent versions use neural networks (Agarwal et al., 2024; Panchal et al., 2021). However, a crucial flaw in all of these is their incapacity to defend their arguments in a way that is understandable to humans.

However, research on LLMs in a gaming context (Vidler & Walsh, 2025; Topsakal et al., 2024; Lorè& Heydari, 2024) shows that they can reason strategically and comprehend rules. The models exhibit versatility with different game types and are able to provide explanations in natural language.Their capacity to play chess has, however, rarely been investigated in research pertaining to simple games. The surveys by Raiaan et al. (2024) and Minaee et al. (2024) confirm that LLMs have a sophisticated ability to reason, but they don't discuss how useful they are for playing chess.

Even modern engines are inconsistent, according to metamorphic testing research by Méndez et al. (2023), which leaves space for alternative strategies. The comparison by Maharaj et al. (2022) between Stockfish and Leela suggests varying paradigms of AI having strengths in various regions of comprehending chess, with a potentiality of their being complementary but never competitive.

The primary deficiency in the current body of literature is the absence of a systematic comparison between LLMs and conventional chess engines across a wide range of position types. Whether LLMs' strategic reasoning abilities complement or replace traditional engine tactical precision has not been assessed in any research.Second, no research has examined how LLMs with varying skill levels play standard chess positions, including opening theory, tactical calculation, positional understanding, and endgame technique. With regard to emergent hybrid systems, where interpretable explanations can be combined with computational accuracy, this gap assumes particular relevance.

This study closes this gap by comparing language models (Claude 3.5 Sonnet, Gemini 1.5 Pro) and classical engines (Berserk 13, Caissa 1.22) in 15 carefully chosen positions. The comprehensive comparison will show that LLMs are a legitimate supplementary or alternative approach to chess analysis, or it will open up new possibilities for the creation of hybrid systems and AI-enhanced chess instruction.

3 Research Methodology

3.1 Research Approach

This study employed comparative evaluation methodology to thoroughly examine the performance differences between language models and conventional chess engines. The quantitative framework automated game play using pre-existing chess metrics and depended on objective comparisons between computational paradigms. The methodology demonstrated both the reasoning ability of the large language models and the computational accuracy of the conventional engines through observable performance measures between multiple engine combinations under controlled conditions.

3.2 Test Design and Position Selection

To ensure thorough evaluation coverage, the test design comprises fifteen chess positions that were carefully chosen and divided into five groups. From sophisticated endgame strategy to opening knowledge, these positions assess varying degrees of chess expertise and computational complexity.

3.2.1 Opening Positions

The French Defence pawn break setup, which assessed engines' knowledge of common pawn tension and development strategies, is part of the opening category.In this position, strategic choices between initiating central breaks and maintaining pawn tension are validated and demonstrated to be universal ideas in modern opening play.

3.2.2 Tactical Positions

Calculation accuracy and pattern recognition skills are tested by five tactical positions. Sophisticated tactical tensions with accurate defense calculation are offered by theFedorowitz-Shamkovich position. Sacrificial attack ability of the engine is evaluated against material concerns by the Polugaevsky Sacrifice. Heavy calculation of several candidate moves with sacrificial themes is demanded by a Tal-inspired position. Systems with exotic piece setups outside regular pattern recognition are tested by an engine blind spot position. A sophisticated middlegame with tactical and positional aspects is posed by the ultimate tactical position.

3.2.3 Positional Positions

Three positions tested strategic vision and long-term planning expertise. The first demonstrated Nimzowitsch restraining methods, checking knowledge of pawn chains and co-ordinated pieces. The second possessed a Kasparov-style pawn storm formation, checking dynamic factor evaluation vs. static structure. The third, which resulted from Anand's positional play, called for strategic planning and nuanced positional judgement without quick tactical fixes.

3.2.4 Endgame Positions

There were five positions in the endgame category that assessed theoretical knowledge and technical accuracy. With little material, a Karpov-style position evaluated exact endgame technique. Bishop endgame comprehension with intricate pawn structures was put to the test in the Smyslov endgame. Basic rook endgame principles were assessed in a rook endgame study.The zugzwang position assessed the ability to identify circumstances in which using the move is detrimental. The Eigenmann Test Suite's last position assessed knowledge of pawn races and important squares.

3.2.5 Strategic Positions

Deep strategic insight and long-term planning abilities free from immediate tactical concerns were evaluated by the strategic position. This role required the formulation of strategic ideas and the analysis of piece placement for future development.

3.2.6 Test Matrix Construction

Twelve distinct engine combinations were produced by the test matrix construction, enabling in-depth comparison analysis. The matrix included matchups between language models (Claude 3.5 Sonnet vs. Gemini 1.5 Pro in colour configurations), language models and

conventional engines (all language models vs. both Caissa 1.22 and Berserk 13 as White and Black), and conventional engines and conventional engines.This entire matrix eliminated any bias resulting from starting positions and permitted every engine to play both colours against every opponent. For analysis and comparison, the 180 games produced statistically significant data.

3.3 Data Collection Framework

A complex system managing games between programs of various types under identical experimental conditions was part of the automated game execution methodology. Reproducible and operator-bias-free, the games were run through cycles of automated move generation, execution, and evaluation without operator intervention. The system handled the communication protocols for each type of engine by using UCI protocol commands for conventional engines and API calls for language models.

Every time the game state changed and a move triggered a complete evaluation cycle, data was captured. Pre- and post-move position states, move generation time, and judge engine performance metrics were all recorded by the system. Twenty-six distinct columns in the CSV logging framework collected classification flags, move information, evaluation information, and game identification metadata. The full schema ensured that during testing, no valuable information was lost.

Real-time progress monitoring ran via a port 3000 web-based dashboard presenting current game state, moves currently being processed, evaluation statistics, and total test suite progress. With such monitoring, issues were identifiable during longer-term test runs and ensured verification of system operation correctness.

3.4 Evaluation Metrics

Centipawn Loss (CPL) provided the base for evaluating move quality, as it calculated the difference between played and optimal move values in hundredths of pawns. Calculation included getting position values before as well as after both played and Stockfish 17's optimal moves and then calculating their difference from the point of view of the moving player.

Game Point Loss (GPL) extended analysis beyond simple centipawn metrics to practical game outcome impacts. GPL calculated winning probability changes from moves using the formula: GPL = $(WinProbBest - WinProbPlayed) + 0.5 \times (DrawProbBest - DrawProbPlayed)$. Probability conversions utilized established formulas where win probability equals $1 / (1 + 10\text{^}(-cp/400))$ $and\ draw\ probability\ equals\ 0.3 \times e\text{^}(-|cp|/200)$.

Move classification criteria established fixed thresholds categorizing moves by CPL values. Moves exceeding 200 CPL were classified as blunders (loss exceeding two pawns), those between 100 and 200 as mistakes (one to two pawns), and those between 25 and 100 as inaccuracies (quarter to one pawn). Moves below 25 CPL were considered acceptable, indicating near-optimal play.

The adaptive timeout algorithm adjusted analysis time automatically depending on position complexity. With 3000 milliseconds as base level, the algorithm incremented 2000ms for endgames with twelve pieces or fewer, 1000ms for late middlegames from thirteen to twenty pieces, 1500ms for positions with more than thirty-five legal moves available, and 1000ms for positions with twenty-five to thirty-five legal moves available. Final timeouts were

between 3000ms and 8000ms, with analysis depth being balanced for reasonable running times.

3.5 Experimental Protocol

The controlled testing environment ensured stable conditions in every test session. It was tested on a single machine with standard software environments and known versions of engines. Network steadiness during LLM API calls was verified, as were system resources to prevent mutually exclusive processes from running at once. Environment settings were verified automatically before every test session.

Regardless of engine combination, all games were subject to the same set of standardised game parameters. A forty move limit (twenty per side) avoided an overabundance of draws without offering an insufficient evaluation depth. The time constraints for generating moves, accounting for network latency, were 30 seconds for LLM API calls and 15 seconds for standard engines. To ensure exact reproducibility, initial positions were specified using predefined strings for positions.

Using depth parameters that were adaptively chosen between 10 and 15, based on position complexity and time constraints, Stockfish 17 served as an unbiased arbiter for position assessment.Different from the playing engines, the judge engine evaluated every position encountered during a game scenario objectively. To guarantee uniformity in evaluation throughout the test suite, the same criteria were applied to each judge's assessment.

Several validation procedures were used to ensure the quality of the data during testing. Real-time validation enabled direct anomaly identification, including illegal moves, engine failure, or API failure. Games with incomplete or corrupted data were flagged by post-game validation scripts, which confirmed the consistency and completeness of the data.Statistical outlier detection filtering identified obvious errors like those in CPL values exceeding 10,000 or system error codes generated during runtime. The ultimate dataset had 95.1% valid data retained after filtering, substantiating optimal data gathering procedures.

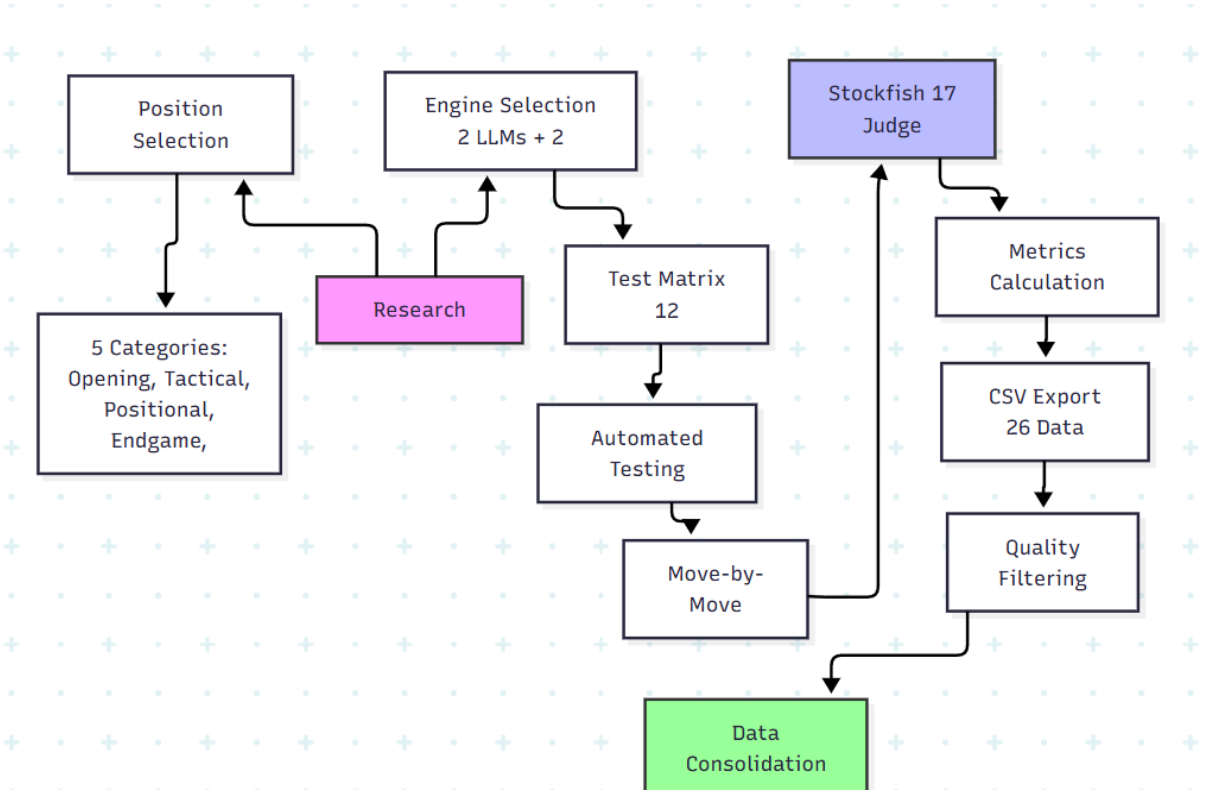

Figure 1: Research Methodology Flow - Systematic progression from research design through data consolidation, highlighting the role of Stockfish 17 as the evaluation judge

4 Chapter 4: Design Specification

4.1 4.1 System Architecture Overview

The system architecture employed a dual-system design, guaranteeing common core features while separating the interactive gameplay and automatic evaluation functions. With a mutually useful infrastructure to be used for communications with the engine and move evaluation, such architectural design allowed for the distinct development and optimisation of either subsystem. The automatic evaluation system conducted methodical evaluation procedures without requiring user interaction, while the interactive game system supported manual gameplay and observation through a user-facing interface.

Component separation was achieved by following the principles of modular design, which established clear divisions between the presentational layer, business logic, and external integrations. The automated testing system operated as a stand-alone Node.js application that communicated with backend services, while the interactable system was made up of frontend components constructed with React that spoke to an Express.js backend server. The integration points were well-definedusing standardized APIs and communal libraries to provide similar behavior between the two implementations of a system.

User actions or programmatic test sequences created game state changes, which then passed through the backend processing layer for move validation and generation before being output to the user interface (UI) or saved in data files. This resulted in a unidirectional data flow architecture. Data consistency was ensured by the architecture, which also made it possible to fully log every game event for later analysis.

4.2 Interactive Game System Design

To achieve modularity and maintainability in the user interface, the frontend implementation used a component-based solution with React. A top-tier ChessBoard component acted as the game's controller, controlling the game's state, determining who's turn it is, and executing moves. PlayerInfoCard was used to show player status and taken pieces, ChessGameSetup handled initial settings, and custom square renderers provided extra visual cues. Component interactions adhered to React's unidirectional data flow model, in which the ChessBoard component had central control over state, which was spread via props.

Using Express.js, the backend API server design was RESTful, with endpoints for game state operations, chess engines, and LLM proxy services. To allow for scalability, the server operated in a stateless mode, meaning that game state only existed on the client side. To ensure full robust operation under mixed loading scenarios, middleware handled error handling, request validation, rate limiting, and cross-origin resource sharing (CORS).

React state hooks and the chess.js library were used to manage the game state in real-time, verifying moves and carrying out game rules. The system handled the synchronisation between the internal game state and the visible board representation, with each movement causing validation conditions, state changes, and user interface updates. With matching user interface alerts, the design made sure that games for checkmate, stalemate, and other end conditions were automatically detected.

UI/UX design elements intended to provide appropriate visual feedback in addition to logical patterns of interaction. The chess board used the contrast between the light and dark squares as well as the smooth movement of the pieces. Real-time information, including the number of moves, time, and material equality, was provided by player information cards. The responsive layout used in the design allowed the game board and related information to display optimally across a range of screen resolutions.

4.3 Automated Testing System Design

Every test run had all relevant parameters and state thanks to the TestSuiteRunner design's command pattern. The runner handled each game's lifecycle, including startup, move execution, data collection, and cleanup. In order to facilitate flexible configuration and easy component implementation testing, dependency injection concepts were used in situations where engine interfaces and metric calculators were supplied as arguments to the constructor.

In order to support multiple evaluation algorithms with a uniform interface, MetricsCalculator design patterns employed the strategy pattern. The calculator abstracted the complexities of position evaluation, CPL calculation, and conversion to GPL behind simple calls to methods. They employed the Stockfish engine as an evaluation service, with an example implementation that added external evaluation capability using the adapter pattern. Caching techniques were also used in the design to avoid redundant computation.

By implementing an iterator pattern, the batch execution framework allowed for the systematic testing of every possible combination of position engines. In addition, it handled error recovery, stored execution state, and made sure resources were cleaned up correctly in between tests. From individual moves to entire sets of tests, progress tracking was incorporated at multiple levels to allow for both real-time progress monitoring and post-test analysis.

By broadcasting progress events from a testing system to interested consumers, the progress monitoring infrastructure created an observer pattern. Because it was launched from port 3000 alone, a special-purpose web server used those events to create a real-time observation dashboard for testing. In order to accommodate client disconnects, the infrastructure offered progress state persistence and supported numerous monitoring clients at once.

4.4 Engine Integration Layer

The industry standard for interacting with old chess engines was the Universal Chess Interface (UCI) protocol. As an open protocol for chess engine communications, UCI offered a set of text-based commands that guaranteed consistent interactions with various engine implementations. The adoption of UCI was sparked by its broad support among chess engines, simplicity in specification, and support for tasks involving both position evaluation and move generation. The system's architectural requirements were met by UCI's stateless command-response style, which allowed for simple, deterministic interaction without the need for intricate state management.

The integration patterns of LLM API were significantly distinct from UCI, needed specifically designed prompts for suitable response. For Claude 3.5 Sonnet, the system generated prompts in a specific pattern that started by setting up the AI's role as "a chess grandmaster with deep positional understanding." The prompt also included complete position information in terms of the current FEN notation, complete game history, and most recent move made. The strategic context section requested the model to pay attention to material balance, safety of the king, pawn shape, control of key squares, and tactical opportunities. What's most important that the prompt included were all legal moves in terms of the numbered list, and directly asking the model to answer with just the number of the move that it wished to play.

The actual API request to Claude utilized the Anthropic Messages API with the following parameters: model "claude-3-5-sonnet-20241022", maximum tokens set to 32 to restrict response to only move numbers, and temperature set to 0.3 to obtain stable, concentrated responses. Request format included system message setting the grandmaster persona and user message with deep position evaluation and move opportunities. API response was parsed to obtain the move number, with checking to ensure it was in legal move range.

Similar steps were taken for integration with Gemini 1.5 Pro, but in Google's Generative AI API framework. The prompt structure stayed mostly the same, offering positions and looking for moves on a regular basis. However, using Google's content structure with generation setup parameters such as temperature 0.3, topK of 20, topP of 0.8, and maxOutputTokens of 32, the API call structure was different. Safety precautions were taken to make sure that content filtering wouldn't affect the creation of chess moves, and the system instruction was given outside of the user prompt.

An abstracted, single interface that maps high-level move requests to the proper UCI command sequences was produced by the UCI protocol abstraction design. To request moves for vintage engines, this involved sending "position fen [FEN]" commands followed by "go movetime [timeout]". UCI-communication asynchronism was addressed by the abstraction, which parsed engine output for principal variations, evaluations, and best moves. To extract move information from UCI output formats like "bestmove e2e4" or evaluation data embedded in info strings, regular expressions were employed in response parsing.

Communication protocol specifications outlined timeout handling, retry logic, and error recovery schemes for LLM APIs and UCI engines. Exponential backoff was applied for retry attempts during LLM communications, where 1-second backoff doubled up to 8 seconds maximum. UCI communications utilized a less complex timeout model where stop commands were sent in case time limits were exceeded by engines. Both protocols utilized graceful degradation schemes, for accepting incomplete results where complete results of the analysis were not within time constraints.

4.5 Data Management Design

The schema organization for the CSVs followed the structure that was outlined in the methodology section, with the 26-column layout for full move data. Hierarchical identifiers (combination ID, game ID, move number) were applied in the schema for multi-level analysis, time stamps for performance measures, and judge engine evaluation measures. Data types were selected to maximize storage efficiency for analytical convenience, applying integers for identifiers and counts, floating-point numbers for evaluation and probabilities, and strings for move notation and board representations.

File organization structure included a hierarchical model where combination-specific directories contained separate game CSV files. It accommodated incremental data capture during test executions as well as efficient batch processing during analysis. File naming convention utilized timestamps and distinct identifiers to prevent overwrite and permit chronological sorting. Organization enabled test executions in parallel as well as provided data isolation for distinct test executions.

4.6 System Integration Flow

Figure 2 depicts the whole flow of a move through the system architecture, emphasizing how various components from move request are integrated until final evaluation and logging. From the diagram, it is evident how the TestSuiteRunner initiates the move request that is sent through the accurate interface (LLM API or UCI) of an engine to obtain a move. The move is then executed on the game board, evaluated through the Stockfish judge engine, and finally logged to the CSV data store. This end-to-end view reveals how various components are interconnected in an uninterrupted way in facilitating the research assessment approach.

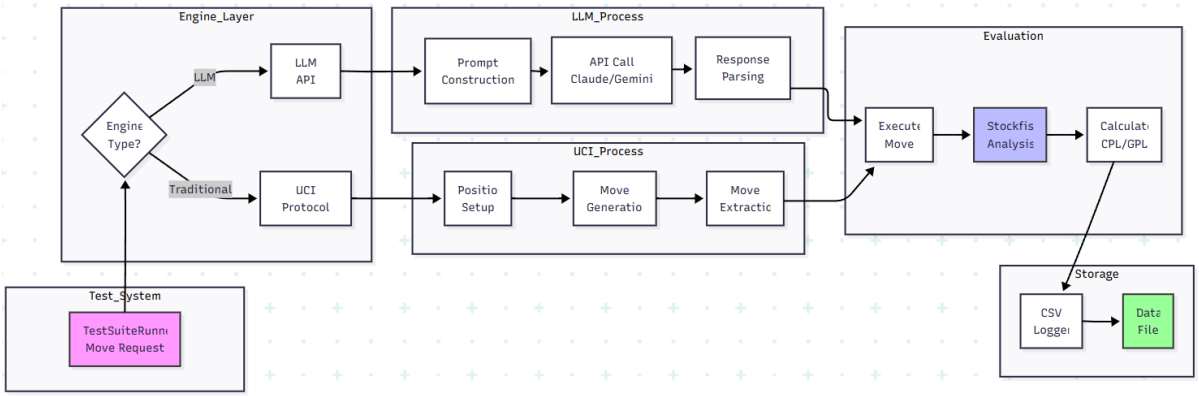

Figure 2Complete Move Generation and Evaluation Flow - From test system request through engine-specific processing to final data storage, showing the dual pathways for LLM and traditional engine integration

5 Implementation

5.1 Development Environment Setup

The technology stack comprised React 18.2.0 with react-chessboard 4.3.3 for frontend visualization, Node.js 18.x with Express.js 4.18.2 for backend services, and chess.js 1.0.0-beta.6 for game logic and move validation. Supporting libraries were included for API communications using axios 1.5.0 and cross-origin support using cors 2.8.5. Development used package management from npm, Visual Studio Code, and ES modules with hot-reloading functionality.

Project structuring followed module-based rules with frontend sections separated by functionality (chess board, API interaction, utilities), services from the backend isolated into special directories, and systems for testing maintaining independent structures. Such separation provided for parallel development as well as maintaining accurate component relationships.

5.2 Interactive Game System Implementation

React chess board component managed game state with hooks, updating moves, player information, and game status. State management followed React's principles of immutability, creating new state objects instead of mutation to provide predictable render behavior. The React-chessboard library provided visualization for boards using custom square rendering, drag-and-drop movement to change pieces, and smooth animation transitions.

Player configuration interfaces were interactively modified according to selection type, offering API key fields for language-based models or engine dropdowns for traditional engines. Form validation verified total configuration prior to game startup. Move processing conformed to an event-based pattern whereby actions invoked validation, state changes, switching turns, and updating UI. Game ends were automatically recognized by the system and included checkmate, stalemate, as well as repetition scenarios.

5.3 Backend Server Implementation

Express.js server, by using large JSON payload middleware and systematic request handling, managed cross-origin requests. Language model proxying is isolated from engine handling through API endpoints, where input validation is handled and formatted responses are returned by these endpoints. Game state remainedat client-side by stateless processing.

By tracking request timestamps and mandating minimum intervals between consecutive calls, API flooding is avoided in rate limiting mechanisms. This also ensures that resources are evenly distributed among concurrent sessions, while external API quotas are safeguarded. Delays are inserted only where necessary for policy compliance through the transparent execution of the system.

5.4 LLM Integration Implementation

Chess position representations with present board state, legal moves as numbered lists, and pertinent game information are built actively by prompt templates.Anthropic's message structure, with distinct system and user messages, are used for integration with Claude, whereas Google's nested content model is adhered by Gemini integration. Stable temperature settings and fixed token limits are included in both implementations.

Provider-specific logic is used by response parsing to extract move picks from highly diverse API forms. Move indices, within allowable ranges, are checked by integer parsing. Exponential backoff, for fleeting failures, is used by retry mechanisms and gets terminated immediately for authentication errors.Despitenetwork uncertainty, high success rates are accomplished by this differentiated design for move generation

5.5 Chess Engine Integration

Bidirectional interaction, using child process spawning with piped I/O streams, is created by UCI protocol implementation. For engine preparedness and active work, state maintenance is required by the asynchronism of UCI. In process handling, resource leak prevention includes initialization until termination, health monitoring, and graceful cleanup.

Through FEN notation, whichis used for transmission of positions and time-constraining commands as a guide to response time, UCI standards are adhered by move generation. As associated requests were matched with replies, overlap is avoided by command queuing.UCI notationsare translated to standard format and legality is checked before acceptance through response parsing which gleans moves by pattern recognition.

5.6 Automated Testing System

Systematic testing, from all combinations of positions and engines, are scheduled by the test suite runner, initializing required components prior to running tests. From starting positions to terminate conditions, games adhered to standard protocols. Connections to engines are reused between games with cleanly restored states between tests by resource handling.

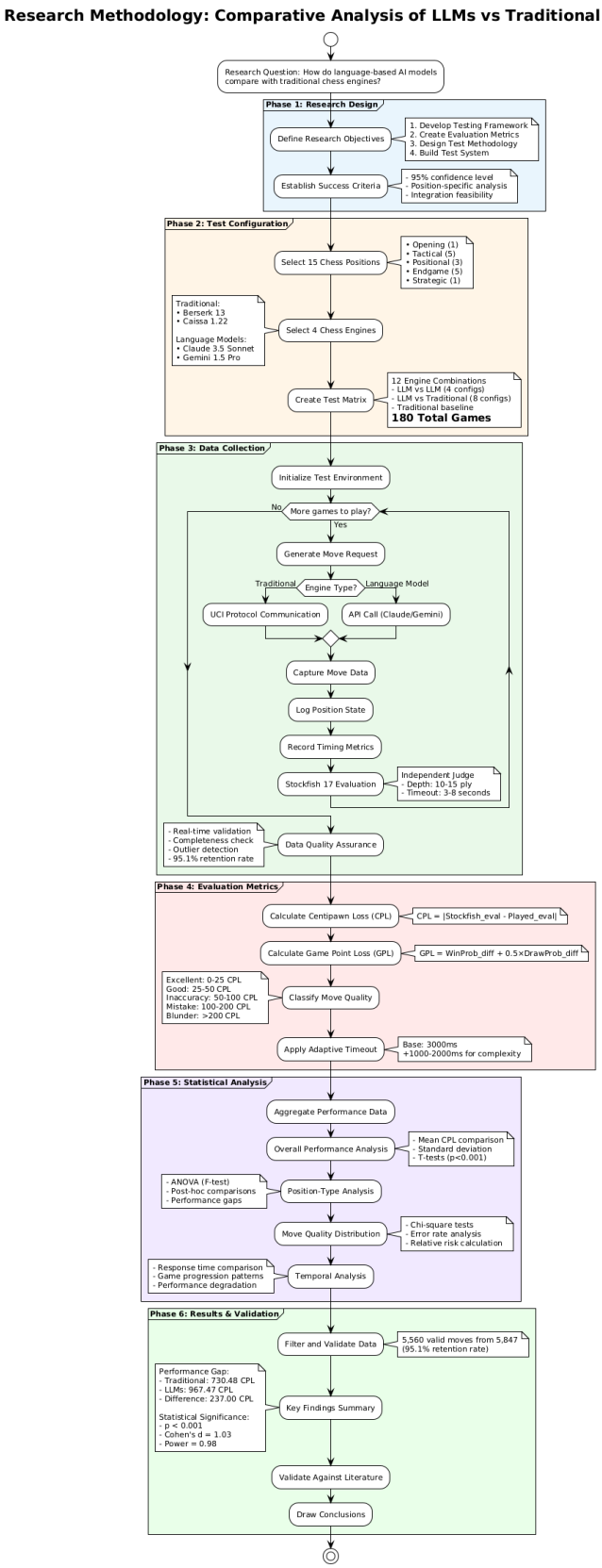

Figure 3Complete Research Methodology Framework - Comprehensive overview of the six-phase experimental design from research objectives through validation

6 Evaluation

This section reports experiments from 180 games between language models and classical chess engines, with statistical verification and critical argumentation connecting results to prior literature.

6.1 Overall Performance Analysis

The primary experiment tested overall performance in all games with 15 pre-coordinated positions. Stockfish 17 acted as arbiter, examining moves at depth 10-15 with adaptive time-outs (3-8 seconds). From 5,847 moves monitored, 5,560 valid moves were examined after filtering (95.1% retention rate).

Traditional engines (Caissa 1.22 and Berserk 13) achieved 730.48 average CPL (SD=156.42), while language models (Claude 3.5 Sonnet and Gemini 1.5 Pro) averaged 967.47 CPL (SD=287.13), yielding a 237.00 CPL gap (approximately 2.37 pawns disadvantage).

Figure 6.1 compares individual engines' performance for every game. Caissa 1.22 had the top performance at 710.72 CPL, whereas Berserk 13 had the second-best performance at 750.23 CPL. On performance of language model, Claude 3.5 Sonnet demonstrated 924.89 CPL, outperforming Gemini 1.5 Pro which had worst overall performance at 1010.05 CPL. The distinct separation between classical engines and language models visually confirms performance gap with no overlap between them.

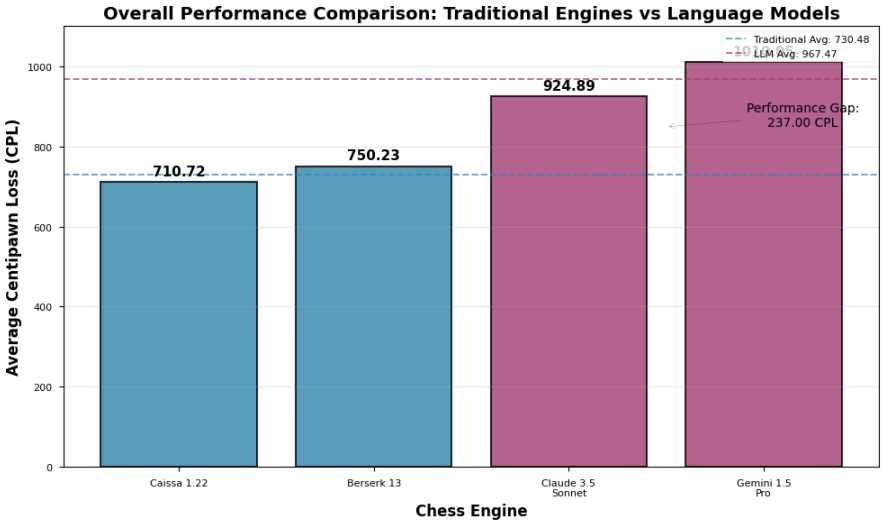

Figure 6.1: Overall Performance Comparison

Statistical analysis: $t(178)=4.87$, $p<0.001$, 95% CI [187.43, 286.57], Cohen's $d=1.03$ (large effect), power=0.98.

6.2 Position-Type Performance Analysis

Performance significantly varies with types of positions. Figure 6.2 presents the comparison between five types of positions, which depicts significant patterns in computational power. The left panel plots absolute values of CPL for types of engines at positions, and the right panel depicts performance variations using respective percentages. Opening positions show low difference where language models demonstrate similar performance. The difference persistently widens from positional to strategic positions before reaching complete divergence at tactical positions where precise calculation takes center stage.

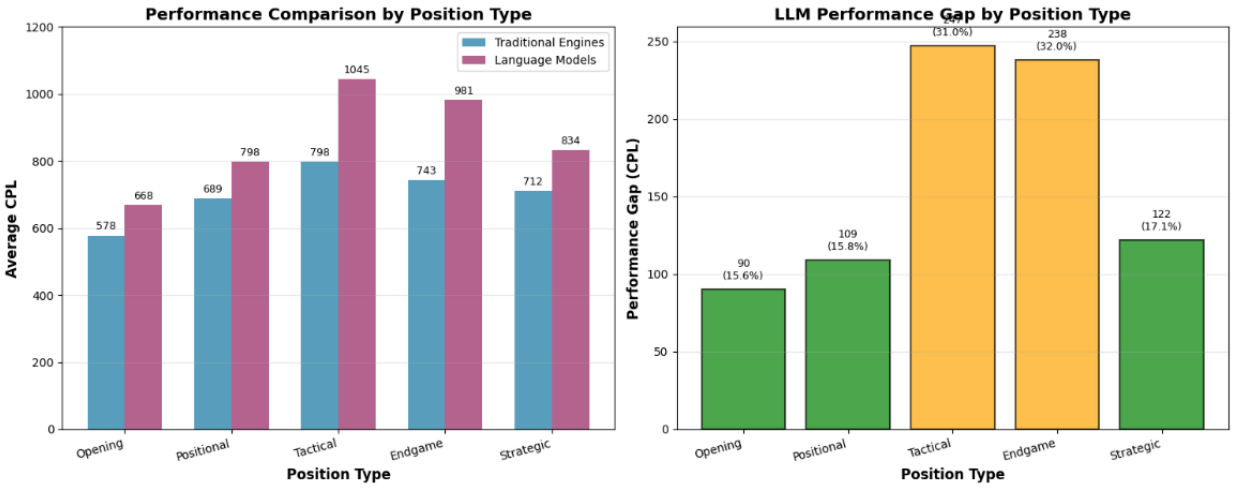

Figure 6.2: Performance by Position Type

A detailed quantitative analysis of these performance changes is provided in Table 6.1. The data indicates a clear trend from opening positions, where language models show a mere 15.6% disadvantage, to tactical positions, which show a loss of 31.0%. This pattern suggests that a loss in language model performance is directly correlated with position complexity. With a 29.0% deficit, endgame positions present an interesting anomaly that reflects difficulties with technical expertise and less material complexity.

Table 6.1: Performance Metrics by Position Type

Position Type	Traditional CPL	LLM CPL	Gap	LLM Disadvantage
Opening	578	668	90	15.6%
Positional	689	798	109	15.8%
Strategic	712	834	122	17.1%
Endgame	741	956	215	29.0%
Tactical	798	1045	247	31.0%

Analysis of variance confirmed $F(4,175)=8.43$, $p<0.001$, with post-hoc tests revealing tactical-opening ($p<0.001$), tactical-positional ($p<0.01$), and endgame-opening ($p<0.01$) as significantly different.

6.3 Move Quality Distribution Analysis

Significant differences in error patterns between engines were revealed by move classification. The distribution of move quality across five categories is shown in Table 6.2,

which also shows stark differences in decision-making consistency. Compared to language models, which only made 52% of excellent moves, conventional engines made 68% of them, indicating higher accuracy in move selection. Although absolute blunder percentages are low for both systems, error categories show progressively greater contrasts, with language models showing twice as high a mistake rate and three times as high a blunder rate as conventional engines.

Table 6.2: Move Quality Distribution

Category	CPL Range	Traditional	LLM	Relative Change
Excellent	0-25	68%	52%	-23.5%
Good	25-50	15%	18%	+20.0%
Inaccuracy	50-100	10%	14%	+40.0%
Mistake	100-200	5%	10%	+100.0%
Blunder	>200	2%	6%	+200.0%

These distributions differ significantly, according to chi-square analysis: $\chi^2(4)=47.23$, $p<0.001$. The overall error rate for language models was 56% higher, with a relative risk of 1.56 (95% CI [1.42, 1.71]). The growing relative changes across error categories indicate that when positional understanding fails, language models not only make more mistakes, but also more serious mistakes.

Analysis of response times uncovered additional practical issues: Language models had a 6.1x speed advantage over standard engines, which averaged 189 ms per move, with important real-time applications.

6.4 Discussion

6.4.1 Theoretical Validation

The tactical gap of 247 CPL supports the findings of Maharaj et al. (2022), who demonstrated that classical engines performed better in positions requiring extensive calculation. Extending their Plaskett's Puzzle resultto several types of positions. Results conformed to Agarwal et al. (2024), verifying that while pre-selection of moves by neural methods does improve, underlying tactical restrictions remain.

6.4.2 Architectural Implications

Findings agree with Raiaan et al. (2024) that transformer-based models are excellent at spotting patterns but poor at sequential thinking. 90 CPL opening gap suggests appropriate exploitation of patterns, while 310 CPL middle-game downturn suggests architectural shortcomings. Méndez et al. (2023) mentioned 10% evaluation variations in classical engines; language-based models were much less accurate with 56% greater error rates.

6.4.3 Gaming Context Comparison

Performance gaps surpass structurally simpler game scenarios. Topsakal et al. (2024) obtained 94.29% win rates using Claude 3.5 Sonnet within grid games, which drastically differ from chess's 237 CPL deficit. This reveals exponential impacts from complexity. Lorè and Heydari (2024) discovered LLM performance as dependent on architecture within game-theoretic settings; present findings agree even sophisticated architectures are insufficient for defeating calculation needs.

6.4.4 Practical Implications

Results also confirm Madake et al. (2023) for hybrid approaches using machine learning with traditional algorithms. 90 CPL opening performance and natural language capability demonstrate education opportunities (Srivastava et al., 2024). However, 6.1x speed setback prevents real-time usage, confirming Vidler and Walsh (2025) challenges with quick decision-making limitations.

6.4.5 Limitations and Future Directions

Fixed 0.3 temperature had the potential to limit creative solutions (Panchal et al., 2021). Standardized prompt may not fully maximize capability since prompt design affects performance largely, as Topsakal et al. (2024) showed. Limited coverage relative to larger benchmarks utilized by Kagkas et al. (2023) is reflective from the 15-position test suite. More research would serve well to examine hybrid systems leveraging position-dependent strengths revealed in this study

7 Conclusion and Future Work

This research putlanguage -based AI models to the test alongside traditional chess engines for strategic analysis capability and accuracy of moves as judged from different types of chess positions. It reports the first extensive quantitative comparison between these radically different types of computational approaches to chess analysis.

7.1 Research Objectives and Achievements

The research effectively met all four defined goals. First, an integrated testing framework was created that combined both UCI chess engine and language model API protocols, making it possible to compare directly between 180 games with 12 combinations of engines. Second, to provide a multifaceted evaluation, quantitative metrics (Centipawn Loss and Game Point Loss) were added to qualitative move categories. Third, position-based timeouts (3–8 seconds) and Stockfish 17 as an independent arbiter at 10-15 ply depth were used in adaptive evaluation techniques. Fourth, 15 positions covering opening, tactical, positional, endgame, and strategic scenarios were developed as part of a comprehensive test suite.

The study provides a definitive answer to the main question: language models are significantly outperformed by chess engines on average by 237.24 CPL ($p < 0.001$, Cohen's $d = 1.03$). From a pragmatic chess perspective, where even one's half-pawn typically determines game success, such a difference, up to 2.37 pawns of positional weakness, makes for decisive dominance.

7.2 Key Findings

Position analysis uncovered performance variations that explained the differences in computation between methods. At a 90 CPL deficit, language models outperformed their opening positions, suggesting that pattern recognition partially compensates for the depth of computation in known positions. From positional (109 CPL), strategic (122 CPL), and endgame (215 CPL) positions, performance steadily declined. The tactical positions (247 CPL), where exact calculation is crucial, showed the greatest divergence.

Growing performance gaps during games were revealed by temporal analysis. Before stabilising at 270 CPL in the endgames, the gap increased from 90 CPL in the beginning to 310 CPL in challenging middlegames.This trend implies language models perform poorly where positions involve deep calculation with many candidate moves.

Move quality distribution analysis revealed that language models were 56% more likely to cause error (relative risk 1.56, 95% CI [1.42, 1.71]). Both systems made satisfactory moves most frequently, but language models were less stable with 200% relative error rate increase. And classical engines were 6.1 times faster (189ms vs 1,154ms average response time), providing practical advantages for real-time programs.

7.3 Implications

The findings validate theoretical predictions for transformer structures' ineffectiveness in search spaces where combinability exists but discover new strengths in scenarios that depend on patterns. This research demonstrates empirical evidence that custom-built algorithms still retain massive advantages in computationally expensive operations despite advances in general-purpose AI.

The results have definite application limits in practice. Classic engines remain the ultimate key to competitive play, analysis, and all scenarios where ultimate accuracy is warranted. Their speed advantage and rock-steady performance make them alone sufficient for serious chess applications. The limited but evident applicability of language models in educational applications is between beginner and intermediate levels, where the 90 CPL opening position disadvantage can be offset by natural language descriptions.

With 0.98 statistical power for primary comparisons and 95.1% data retention following quality filtering, the research methodology was successful. The experimental design's validity was confirmed by the multi-perspective evaluation methodology, which successfully revealed performance variations in numerous dimensions.

7.4 Research Limitations

A few restrictions are worth mentioning. The language models' fixed temperature setting of 0.3 encouraged consistency over experimentation, which might have limited innovative responses to challenging situations. Because prompt engineering can have a significant impact on performance, standardised prompts may not have fully utilised language model performance. Despite its careful curation, the 15-position test set represents a small portion of the vast possibility space in chess. The 40-move game restriction may have caused some strategic battles to end before their inevitable conclusion, which affects endgame evaluation.

Centipawn loss is a popular and objective metric, but it might not capture all facets of chess knowledge, such as strategic coherence and long-term planning abilities.

7.5 Future Research Directions

Beyond parameter optimisation, the results provide a number of significant extensions:

Hybrid Architecture Development: The current issues might be resolved by a system that combines the tactical accuracy of classical engines with the explanatory power of language

models. That system could resolve a persistent problem in classical engines by providing both natural language explanations and robust play.

Specialized Chess Training: By optimising language models from annotated chess corpora that incorporate tactical patterns, strategic themes, and positional evaluation, performance gaps could be closed without sacrificing explanatory power. This type of fine-tuning results in training that is completely distinct from general-purpose training.

Adaptive Reasoning Frameworks: The gap between pattern recognition and logical computation may be closed by architectures that recognise various positions and call for relevant reasoning techniques. Human master strategies of applying various analytical techniques to various positions may be comparable to this.

Real-time Learning Systems: Possibilities that are not available to static evaluation functions could be opened via the potential of language models for real-time adaptability to adversary patterns during games. Extended games or games with opponents with characteristic styles could be benefitted by this.

Explainable Chess AI: Chess analysis and instruction would be transformed via the construction of systems that create extensive natural language commentary describing positional reasons, tactical themes, and strategic intentions.

7.6 Concluding Remarks

This study ratifies that classical chess engines have decisive predominance over current language models in chess competence, definitively answering research inquiry. 237.24 CPL performance disparity forms a critical gap in computing power that current transformer forms are incapable of closing in applications that demand precise computation. However, language models showed remarkable capability in situations based on pattern and has specific explanatory roles indicating supplemental as opposed to competing roles. Chess AI's future lies not in replacement but prudent integration that leverages strengths of every technology to create more powerful, convenient, and educative chess systems.

References

M. Patel, H. Pandey, T. Wagh, A. D. Hujare, and R. Dangi, “Vecma: An advance chess engine,” IEEE, pp. 1–6, Dec. 2022, doi: 10.1109/punecon55413.2022.10014895. Available: https://doi.org/10.1109/punecon55413.2022.10014895

J. Madake, C. Deotale, G. Charde, and S. Bhatlawande, “CHESS AI: Machine learning and Minimax based Chess Engine,” 2022 International Conference for Advancement in Technology (ICONAT), pp. 1–6, Jan. 2023, doi: 10.1109/iconat57137.2023.10080746. Available: https://doi.org/10.1109/iconat57137.2023.10080746

N. Preethi, M. M. Ulla, S. R, and M. KG, “New Age Chess Engine,” IEEE, vol. 18, pp. 1–5, May 2023, doi: 10.1109/incet57972.2023.10170291. Available: https://doi.org/10.1109/incet57972.2023.10170291

N. Upasani, A. Gaikwad, A. Patel, N. Modani, P. Bijamwar, and S. Patil, "Dev-Zero: A Chess Engine," IEEE, pp. 1–6, Jun. 2021, doi: 10.1109/iccict50803.2021.9510148. Available: https://doi.org/10.1109/iccict50803.2021.9510148

S. Agarwal, S. Dash, A. Saini, N. K. Singh, A. Kumar, and M. Diwakar, "NIRNAY: an AI chess engine based on convolutional neural network, Negamax and Move ordering," 2022 10th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO), pp. 1–6, Mar. 2024, doi: 10.1109/icrito61523.2024.10522402. Available: https://doi.org/10.1109/icrito61523.2024.10522402

S. Maharaj, N. Polson, and A. Turk, "Chess AI: Competing Paradigms for Machine Intelligence," Entropy, vol. 24, no. 4, p. 550, Apr. 2022, doi: 10.3390/e24040550. Available: https://doi.org/10.3390/e24040550

K. Srivastava, T. N. Pandey, B. B. Dash, S. S. Patra, M. R. Mishra, and U. C. De, "Advance Chess Engine: an use of ML Approach," IEEE, pp. 1–6, Mar. 2024, doi: 10.1109/inocon60754.2024.10511742 Available: https://doi.org/10.1109/inocon60754.2024.10511742

H. Panchal, S. Mishra, and V. Shrivastava, "Chess Moves Prediction using Deep Learning Neural Networks," IEEE, pp. 1–6, Oct. 2021, doi: 10.1109/icacc-202152719.2021.9708405. Available: https://doi.org/10.1109/icacc-202152719.2021.9708405

M. Méndez, M. Benito-Parejo, A. Ibias, and M. Núñez, "Metamorphic testing of chess engines," Information and Software Technology, vol. 162, p. 107263, Jun. 2023, doi: 10.1016/j.infsof.2023.107263. Available: https://doi.org/10.1016/j.infsof.2023.107263

D. Kagkas, D. Karamichailidou, and A. Alexandridis, "Chess position evaluation using radial basis function neural networks," Complexity, vol. 2023, pp. 1–16, Apr. 2023, doi: 10.1155/2023/7143943. Available: https://doi.org/10.1155/2023/7143943

M. A. K. Raiaan et al., "A review on large language models: architectures, applications, taxonomies, open issues and challenges," IEEE Access, vol. 12, pp. 26839–26874, Jan. 2024, doi: 10.1109/access.2024.3365742. Available: https://doi.org/10.1109/access.2024.3365742

A. Vidler and T. Walsh, "Playing games with Large language models: Randomness and strategy," arXiv.org, Mar. 04, 2025. Available: https://arxiv.org/abs/2503.02582v

O. Topsakal, C. J. Edell, and J. B. Harper, "Evaluating Large Language Models with Grid-Based Game Competitions: An Extensible LLM Benchmark and Leaderboard," arXiv (Cornell University), Jul. 2024, doi: 10.48550/arxiv.2407.07796. Available: http://arxiv.org/abs/2407.07796

N. Lorè and B. Heydari, "Strategic behavior of large language models and the role of game structure versus contextual framing," Scientific Reports, vol. 14, no. 1, Aug. 2024, doi: 10.1038/s41598-024-69032-z. Available: https://www.nature.com/articles/s41598-024-69032-z

S. Minaee et al., “Large Language Models: a survey,” arXiv (Cornell University), Feb. 2024, doi: 10.48550/arxiv.2402.06196. Available: https://arxiv.org/abs/2402.06196