Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

ShashGuru: Bridging Chess Engines and Large Language Models for Human-Interpretable Analysis

Relatore:
prof. Paolo Ciancarini

Presentata da:
Alessandro Libralesso

Correlatori:
ing. Andrea Manzo
dott. Alessandro Palla

Sessione I
Anno Accademico 2024/2025

Contents

Acknowledgments

I would like to express my deepest gratitude to my supervisor, **professor Paolo Ciancarini**, for his continuous support, valuable feedback, and encouragement throughout the development of this thesis.
I am also sincerely thankful to my co-supervisors, **Andrea Manzo** and **Alessandro Palla**, for their insightful guidance and helpful suggestions, which greatly enriched the quality of my work.
I would also like to thank **Intel** for providing the server infrastructure that enabled the development and experimentation involved in this thesis.

A mia mamma, la mia famiglia e i miei amici, che mi hanno supportato in ogni passo del mio percorso, la più sincera delle gratitudini. A Sara: ogni giorno ringrazio di averti conosciuta.

Abstract

This thesis presents ShashGuru, a chess analysis tool that integrates the computational strength of modern chess engines with the natural language generation capabilities of Large Language Models. The system combines ShashChess, an open source chess engine derived from Stockfish, with LLaMA 3.1-8B, an open source LLM, to provide users with both precise evaluations and human-readable explanations of chess positions.
The work addresses the limitations of traditional chess engines, whose analyses often lack intuitive explanations, and LLMs, which struggle with accurate chess reasoning without structured guidance. ShashGuru bridges this gap by leveraging engine-derived data to guide the LLM. The system is implemented as a web application with a Vue.js frontend and a Flask backend, offering interactive analysis and follow-up question support.
A user evaluation involving chess players of varying skill levels demonstrated ShashGuru's effectiveness, with participants favoring its analyses over raw LLM outputs obtained without the help of a chess engine. The results highlight the tool's potential to enhance chess training by delivering accurate, context-aware insights in an accessible format. Future work could explore fine-tuning the LLM for chess-specific tasks or expanding the system to other strategy games.

Abstract (italiano)

Questa tesi presenta ShashGuru, uno strumento di analisi scacchistica che integra la potenza computazionale dei moderni motori scacchistici con le capacità di generazione di linguaggio naturale dei Large Language Models (LLM). Il sistema combina ShashChess, un motore scacchistico personalizzato derivato da Stockfish, with LLaMA 3.1-8B, per fornire agli utenti sia valutazioni precise che spiegazioni comprensibili delle posizioni scacchistiche.

Il lavoro affronta le limitazioni dei motori scacchistici tradizionali, le cui analisi spesso mancano di spiegazioni intuitive, e quelle degli LLM, che senza una guida strutturata faticano a ragionare accuratamente nel contesto degli scacchi. ShashGuru colma questa lacuna sfruttando i dati forniti dal motore per guidare il modello linguistico.

Il sistema è implementato come applicazione web, con un frontend in Vue.js e un backend in Flask, offrendo un'analisi interattiva e supporto a domande di approfondimento.

Una valutazione condotta con giocatori di scacchi di diversi livelli ha dimostrato l'efficacia di ShashGuru, con una preferenza netta da parte dei partecipanti per le sue analisi rispetto a quelle generate da LLM non assistiti. I risultati evidenziano il potenziale dello strumento nell'ambito della formazione scacchistica, grazie alla sua capacità di fornire indicazioni accurate e contestualizzate in un formato accessibile. Lavori futuri potrebbero esplorare l'ottimizzazione del modello linguistico per compiti specifici negli scacchi o l'estensione del sistema ad altri giochi strategici.

Introduction

Motivations

Chess has long served as a benchmark for Artificial Intelligence. Currently, engines such as Stockfish exhibit superhuman performance, offering highly sophisticated evaluations of positions including numerical values and long and accurate principal variations. However, despite their strength, these systems produce outputs that are difficult for human players to interpret as they rely on neural networks to evaluate positions, acting as a black box.
Conversely, Large Language Models (LLMs) excel at generating human-readable text but struggle with precise chess reasoning without structured guidance.
Recent advancements in AI have demonstrated the feasibility of integrating these technologies. However these approaches often rely on costly fine-tuning, limiting their scalability and accessibility.
Bridging the gap between the computational precision of chess engines and the explanatory power of LLMs holds significant potential for enhancing chess training, analysis, and accessibility. This thesis presents ShashGuru, a novel chess analysis system that integrates ShashChess, a chess engine derived from Stockfish, with Meta's LLaMA 3.1-8B model.
All the code is open-sourced at `https://github.com/Bosslibra/ShashGuru`.

Thesis Structure

This thesis is organized into four main chapters, each contributing to a comprehensive understanding of the research undertaken. Chapter 1 presents the theoretical foundations and a review of relevant literature, providing essential context for the work. Chapter 2 gives an overview of the existing software used in this study, which is incorporated without modification. Chapter 3 describes the design and development of the novel software implementation introduced as part of this research. Chapter 4 reports the results of the study and discusses their broader implications.
Two appendices are included for reference:

- Appendix A lists the questions used in the user evaluation.
- Appendix B provides an explanation of chess terminology for readers unfamiliar with the game.

Chapter 1

Background

1.1 Short History of Chess

The game of Chess has more than a thousand years history [1], as well as one of the most extensive literature of any existing board game. It has always fascinated great minds, who have tried to uncover every facet of it. For instance, both Turing and Shannon have described computing machines able to play chess [2].
It is not surprising, then, that since the creation of the first computers, there were already attempts to develop programs capable of playing chess [3]. However, it was necessary to wait for the arrival of DeepBlue, developed by IBM [4], to finally defeat the best chess players. In 1997, it managed to beat Garry Kasparov, the reigning world chess champion at the time.

Chess is particularly well suited to be played by computers, as it is a perfect information, zero-sum game [5, 6].
These properties make the game representable as a tree, where each node is a position and each edge represents a possible move from that position [7]. The trees thus created are explored using tree search algorithms, such as Minimax [8].
However, the number of possible moves in a game of chess is far too large to explore the entire tree, so evaluation functions are used to estimate which move might be best in a given position.

For most of the history of chess engines, progress came from developing more accurate evaluation functions (created manually with the help of experts and based on human knowledge) or from optimizing tree search, for example through techniques like Alpha-Beta pruning, move ordering and null move pruning.
In 2017, however, Google DeepMind released AlphaGo Zero [9], a program capable of playing the game of Go without requiring evaluation functions created by humans, using convolutional networks to evaluate a given position. This methodology was adapted to chess the following year, again by Google DeepMind, under the name AlphaZero, immediately proving itself superior to the best engine at the time, Stockfish, defeating it 29% of the time (and drawing the remaining games) [10].
The dominant strength of this new technology was limited by costly hardware (i.e., the best GPUs available)[11], until a very specific kind of neural network, called "Efficiently Updatable Neural-Network" (NNUE) [12] was created for Shogi, and later ported for chess. NNUE could run on consumer-grade CPUs.

Stockfish introduced it in version 12 [13], gaining 80 elo points [11].

Today, the playing strength of chess engines is far superior to that of humans, in large part thanks to the advent of Neural Networks and NNUE. Engines are now used as tools to refine skills at every level, with even the strongest Grand-Masters (the highest chess title achievable) memorizing long sequences of engine-recommended moves in preparation for the World Chess Championships.

1.2 Chess Engine Fundamentals

1.2.1 Board Representation

The representation of chess positions and games in digital and analytical contexts relies on standardized notational systems, each serving a distinct purpose within computational and human-readable frameworks.
Forsyth-Edwards Notation (FEN) provides a precise snapshot of a single board position, encoding critical information such as piece placement, active color, castling rights, en passant targets, and move counters.
In contrast, Portable Game Notation (PGN) captures the sequential flow of entire games, employing Standard Algebraic Notation (SAN) to denote individual moves in a compact and widely interpretable format. Together, these systems form the foundational infrastructure for recording, analyzing, and transmitting chess data in both human and machine-readable formats.

1.2.2 Minimax

The Minimax algorithm is well-suited for evaluating and selecting optimal moves in two-player, zero-sum, perfect-information games such as chess, and is therefore widely used in chess engines. It works by recursively exploring possible future game states, assuming that one player (the maximizer) seeks to maximize their advantage while the opponent (the minimizer) aims to minimize it. At each level of the game tree, Minimax alternates between selecting the best move for the current player and anticipating the strongest countermove from the opponent. By propagating evaluations from terminal or depth-limited positions back up the tree, it identifies the move that leads to the most favorable outcome under the assumption of perfect play. However, the algorithm must contend with an enormous branching factor in chess: from the initial position, there are 20 legal moves, as eight pawns can move one or two squares ($8 \times 2 = 16$), and two knights can each jump to two different squares ($2 \times 2 = 4$), yielding $16 + 4 = 20$ possible options. More broadly, Claude Shannon famously estimated that the game-tree complexity of chess is on the order of 10^{120} possible games [2], a number so vast that it renders exhaustive search infeasible.

1.2.3 Alpha-Beta Pruning

To address the computational challenge posed by the vast search space in chess, the Minimax algorithm is commonly enhanced with alpha-beta pruning, a technique that significantly improves efficiency without affecting the final outcome. Alpha-beta pruning eliminates branches in the game tree that cannot possibly influence the final decision,

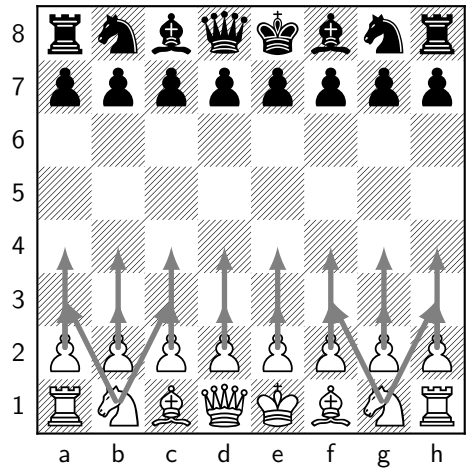


Figure 1.1: Every possible move from the starting position.

based on bounds: "alpha" representing the best already explored option for the maximizer, and "beta" for the minimizer. If a node's value is found to be worse than a previously examined alternative, further exploration of that branch is abandoned. This selective pruning reduces the number of nodes evaluated, allowing deeper lookahead within the same computational budget. In ideal conditions, alpha-beta pruning can reduce the time complexity of Minimax from $\mathcal{O}(b^d)$ to $\mathcal{O}(b^{d/2})$, where b is the branching factor and d the depth [14]. It also has no drawbacks in its worst-case scenario, as it simply reverts to Minimax. The use of Alpha-Beta Pruning makes it really important to order moves from best to worst, with heuristics, to help reach ideal conditions.

1.2.4 Quiescence Search

A quiescence search is a selective extension of the standard fixed-depth search in chess engines, designed to address the horizon effect, a phenomenon where the engine avoids inevitable tactical outcomes (such as material loss) by making superficially better but strategically weaker moves, simply because it cannot see beyond a fixed search depth [7]. To mitigate this, instead of evaluating positions directly at the leaves of the main search tree, a quiescence search continues only through tactically volatile moves, such as captures, checks, or threats, until a quiet position is reached, one where no such immediate tactical possibilities exist [15]. At each node in this search, the engine can choose to either explore a capture or return the static evaluation the current position if no advantageous tactical continuation is available.

1.2.5 Late Move Reduction

Late Move Reduction (LMR) is a heuristic optimization technique to improve search efficiency. The core idea behind LMR is that moves considered later in a move list (which are ordered by expected strength) are less likely to yield a better evaluation. Therefore, these "late" moves can be initially searched at a reduced depth, saving computation time. If such a move returns a promising score (i.e., causes a beta-cutoff or exceeds a threshold), it is then re-searched at the full depth. This approach allows the engine to

focus its resources on more promising branches of the game tree while still maintaining tactical robustness and overall accuracy.

1.2.6 Null Move Pruning

Null move pruning is a pruning technique used to efficiently eliminate unpromising branches. The method involves making a "null move", where the side to move skips its turn, and then performing a shallow search from the opponent's perspective. If the result of this reduced-depth search exceeds the beta threshold (i.e., suggests a strong response by the opponent), it is assumed that the original position is strong enough to not require further exploration, and the branch is pruned. This heuristic is based on the assumption that giving up a move should worsen the position. If it doesn't, the position is likely already advantageous. Null move pruning significantly reduces the search space but must be applied carefully to avoid tactical oversights, especially in zugzwang positions.

1.3 Large Language Models

Large Language Models (LLMs) represent a transformative class of deep learning models designed to process, generate, and understand human-like language. These models leverage vast amounts of training data to learn statistical patterns and semantic structures in natural language, enabling them to perform a wide range of linguistic tasks with minimal supervision. At the core of most LLMs lies the transformer architecture [16], which employs self-attention mechanisms to effectively model contextual dependencies between words across long sequences.
Contemporary LLMs (such as GPT, DeepSeek, and LLaMA) predominantly adopt a decoder-only transformer architecture. This design choice facilitates efficient autoregressive text generation and supports parallelized training across massive datasets. As a result, modern LLMs are scaled to hundreds of billions of parameters, demonstrating remarkable capabilities across zero-shot, few-shot, and in-context learning scenarios.
The development of LLMs typically follows a two-stage training process. During pretraining, models are exposed to large-scale corpora, using unsupervised learning objectives like masked language modeling or next-token prediction. This stage allows the model to acquire general-purpose linguistic and world knowledge. Subsequently, fine-tuning is employed to adapt these models to specific domains or tasks, often through supervised learning or techniques like Reinforcement Learning from Human Feedback (RLHF), which helps align model outputs with human expectations.
A particularly notable aspect of LLMs is the emergence of complex behaviors as model scale increases. These emergent abilities often arise without explicit programming and have been shown to correlate with model size and data diversity [17].
Hallucinations (particularly in the aforementioned decoder-only models), bias amplification, and high computational costs remain active research areas.

1.3.1 Temperature

Temperature is a scalar parameter, commonly chosen within the interval $[0, 1]$, that governs the randomness of the output generated by an LLM. Increasing the temperature amplifies the randomness of token selection, thus enhancing the diversity and creativity

of the generated text. However, higher temperatures may also increase the likelihood of incoherent or erroneous outputs. Conversely, lower temperatures reduce randomness, resulting in more deterministic yet often repetitive responses [18].

1.4 Related Works

LLMs have recently been explored in a variety of roles within the domain of games, including but not limited to gameplay, commentary, and game analysis [19]. This section outlines the key approaches and findings in the use of LLMs and other AI models in relation to chess, focusing on the aforementioned roles.

Players

Several recent efforts have investigated the use of LLMs as chess-playing agents.
One of the simplest approaches is explored in [20], which investigates the impact of prompting strategies on ChatGPT's chess-playing performance without any task-specific fine-tuning. The study evaluates a range of prompt variations such as including the full move history, reminders of previously committed illegal moves and repeated restatements of the rules at each turn. These strategies aim to mitigate issues such as attention decay and rule forgetfulness over long games.
The results show that prompts repeating general information (e.g., chess rules) tend to degrade chess-playing abilities and increase the frequency of illegal moves. In contrast, prompts that reinforce contextual details (e.g., move history) can reduce illegal move frequency and slightly improve consistency. Despite these improvements, most of the games ended prematurely, due to illegal moves.
Nevertheless, this work underscores the importance of prompt design as a viable and low-cost way to steer general-purpose models toward more competent play, even in complex formal domains like chess. These findings were instrumental for the creation of ShashGuru.

Another notable example is MATE[1] [21], a model fine-tuned from `LLaMA 3.1-8B`. MATE was trained using a purpose-built dataset[2] where the input consists of the FEN, two candidate moves each with a natural language explanation and a list of moves identifying a tactical pattern in the position. The correct move is given as the target and the model is asked to choose between the two. Notably, the move annotations were created by Hou Yifan, a three times Women's World Chess Champion, lending credibility and precision to the dataset.
This formulation aims to allow the model to learn not only move selection but also to use natural language reasoning to achieve it. MATE showed strong performance compared to other commercial LLMs, even if it is not designed to compete with traditional chess engines in pure strength.

Furthermore, [22], a study about LLMs as players of chess puzzles, shows that newer, larger language models, like GPT-4, can perform on par with domain-specific fine-tuned models from the previous generation, like GPT-3.5 Turbo. This supports the idea that

[1] `https://huggingface.co/OutFlankShu/MATE`
[2] `https://huggingface.co/datasets/OutFlankShu/MATE_DATASET`

scale alone can partially compensate for the lack of targeted training. Rather than contradicting MATE's approach, these results reinforce its value: MATE achieves competitive performance not just through sheer size, but by combining fine-tuning with expert-crafted data. Together, these works highlight that both scale and domain adaptation remain effective strategies for developing strong chess-playing agents.
The exploration of LLMs as chess-playing agents is particularly relevant for ShashGuru, as [23] claims a better-performing chess-playing model also tends to perform better in commentary tasks, suggesting a shared underlying competence in evaluating positions.

Commentators

The creation of programs able to generate game commentary predates the current wave of large-scale language models. Early work ([24], [25], [26]) approached commentary generation through rule-based systems, which produced natural language outputs based on handcrafted logic tied to positional features. While these systems showed promise, they were often limited in flexibility and expressiveness.

A more recent evolution in this space comes from [27], which introduced neural network methods for generating chess commentary. However, their system does not rely on transformers-based models, but rather on custom-trained neural networks specifically designed for chess commentary.

Contemporary LLM-based systems tend to separate chess understanding from language generation. For example [28] acknowledges the precision requirements of chess analysis, so instead of relying on the LLM to determine the best move, they delegate this task to a dedicated engine, which is also used to annotate move quality throughout the game (e.g., blunders, inaccuracies, good moves). The LLM is then used to narrate or explain these annotations, effectively acting as a linguistic layer over a high-accuracy analytical core.
This type of hybrid architecture has been chosen as the foundation of ShashGuru, combining engine-based analysis with LLM-generated narration.

A notable contribution to this line of research is ChessGPT [29], which proposes a unified framework that bridges policy learning and language modeling in chess. Unlike prior approaches that treat gameplay and language separately, ChessGPT leverages a rich and diversified dataset that includes millions of annotated games, puzzles, blog posts, forum discussions, YouTube transcripts, and other chess-related text.
This approach allows ChessGPT to learn not only how to play but also how to discuss and explain the game. By integrating decision-making and narrative capabilities into a single model, ChessGPT offers a strong foundation for future work in multimodal, policy-aligned LLMs for chess.

Coaches

A role not covered in [19] is the use of LLMs as coaches for chess players: a direction recently explored in [30] and [31]. These works show how LLMs can provide adaptable, interpretable feedback to support learning and development. While ShashGuru is primarily an analysis and commenting tool, it offers inspiration for future extensions that

could deliver more skill-appropriate, pedagogically meaningful analysis. In particular, the possibility offered by some engines to adapt their evaluations to the strength of a human player is especially promising to support a LLM in this role.

Chapter 2

Underlying Technologies

2.1 ShashChess

ShashChess[1] is a customized chess engine derived from Stockfish, one of the strongest and most popular open-source engines available. While Stockfish excels in competitive strength, its recommendations can often appear opaque or counter-intuitive to human players, due to its reliance on NNUE evaluations. Furthermore, Stockfish aggressively prunes the game tree with sophisticated techniques such as Late Move Reduction and Null Move Pruning, which make it incredibly strong in matches, especially in short time controls [32], but reduce its capacity of solving niche complex positions, called "chaotic" by ShashChess creator [33].
ShashChess addresses Stockfish's limitations by tailoring its evaluation and search mechanisms to align more closely with human strategic understanding, making it especially suitable for training and analysis.

2.1.1 Evaluation Function

ShashChess draws on the work of Russian physicist and chess trainer Alexander Shashin, who proposed a mathematical model for evaluating positions based on five factors [34]:

- **Material** (*mass*)
- **Mobility** (*kinetic energy*)
- **Safety**
- **Packaging density** (*density*)
- **Expansion factor** (*potential energy*)

These factors are used to classify positions into three strategic modalities, named after three well knowsn chess world champions:

- **Tal** (*Attack*)
- **Capablanca** (*Strategy*)

[1] `https://github.com/amchess/ShashChess`

- **Petrosian** (*Defense*)

ShashChess dynamically identifies the modality most suitable for a given position and adjusts its evaluation function and search parameters accordingly. For positions that fall between these categories, it also supports two intermediate "boundary" modalities, allowing the engine to adapt flexibly in ambiguous or chaotic situations. This results in an engine that not only plays strongly, like Stockfish, but can also suggest moves in a style that aligns with both the positional demands and human understanding.

2.1.2 Search

Unlike Stockfish, which applies aggressive pruning and selective search techniques uniformly throughout the game, ShashChess adopts a more adaptive approach. Techniques such as Late Move Reduction and Null Move Pruning are selectively activated only when the engine determines they are effective. This typically happens in quiet or stable positions, while in highly complex or "chaotic" situations, where such techniques might overlook critical continuations, ShashChess disables them entirely. This context-sensitive selectivity improves the engine's ability to handle intricate positions, complementing its evaluation framework [32].

2.2 LLaMA 3

LLaMA 3 (Large Language Model Meta AI) is a family of state-of-the-art large language models developed by Meta, first released in 2024. The initial release included models with 8 billion (8B) and 70 billion (70B) parameters, trained on over 15T tokens that were all collected from publicly available sources, a dataset seven times larger than the predecessor LLaMA 2. To ensure training data quality, Meta developed a series of data-filtering pipelines, including heuristic filter, NSFW filters, semantic deduplication approaches and text classifiers that predict data quality [35].
In a subsequent release, LLaMA 3.1, Meta expanded the lineup to include a 450 billion (450B) parameter model, pushing the limits of scale and performance in open-weight LLMs.

These models are provided with open weights, promoting transparency, reproducibility, and wide-scale academic and industry research. Thanks to its size, the 8B model is easily deployable on consumer-grade hardware (even if costly), both GPUs (such as NVIDIA RTX 3090/4090) and CPUs with integrated NPU (such as Intel Core Ultra processors), enabling individual developers and researchers to experiment with powerful models locally without the need for large-scale compute clusters.

Performance-wise, LLaMA 3 models demonstrate strong results across a broad set of benchmarks, rivaling or surpassing closed models of similar sizes. The 70B and 450B models are competitive with the most advanced proprietary systems in reasoning, coding, and multilingual tasks, while retaining the flexibility and auditability that open-weight access provides [35].

In this work we will use the `LLaMA 3.1-8B` model, even though it is deployed on server-scale hardware, prioritizing scalability. Furthermore, future work may include fine-tuning, which would prove hard on larger models.

Chapter 3

Implementation

ShashGuru is designed as a two-tier web application, combining a Flask-based Python back-end with a Vue3 front-end.
The user interface presents an interactive chessboard where users can paste a PGN or FEN string, or play moves directly on the board to reach a specific position. All positions are normalized to FEN behind the scenes.
When an analysis is requested, the front-end sends the FEN to the Flask backend via a defined API endpoint (/analysis). The backend then invokes ShashChess, though any UCI engine can be used, and captures its evaluation output, which consists of:

- the three best moves;
- their centipawn (cp) scores, or mate evaluations if applicable;
- win/draw/loss (wdl) probabilities;
- a series of follow-up moves for each of the best moves.

The server interprets the engine's top-line score as the position's evaluation. To prepare the input for the LLM, ShashGuru uses a custom script to convert the FEN into natural language descriptions , mitigating potential confusion [36]. This prompt is then augmented with the engine output and forwarded to the LLM API. Finally, the frontend renders the LLM's analysis, providing users with both precise and human-readable chess insights. Users can also ask follow-up questions, which are sent to the backend API endpoint (/response) for further processing.

3.1 Frontend

The frontend of ShashGuru is designed to provide an all-in-one, user-friendly interface. It is implemented as a Single Page Application (SPA) using Vue 3 and the Composition API[1], emphasizing reactivity and modularity. This section outlines the architectural decisions, component interactions, and user experience considerations that shape the frontend system.

[1] `https://vuejs.org/guide/introduction.html`

3.1.1 Design Philosophy and Component Structure

ShashGuru follows a component-based architecture, a widely adopted paradigm in modern web development. This design decomposes the interface into reusable, logically distinct units, improving maintainability, testing, and scalability. The core structure consists of four primary components:

- **App.vue:** Serves as the root component. It manages global navigation and styling. The navigation bar is built directly into this file, given its simplicity, thereby avoiding unnecessary componentization.
- **HomeView.vue:** Acts as the primary orchestrator, coordinating interactions between the chessboard and the AI chat modules.
- **ChessBoard.vue:** Uses the `vue3-chessboard` component[2] for board rendering and integrates `chess.js`[3] for move validation. The component supports both FEN and PGN input, enabling users to analyze individual positions or entire games. A key method, `handlePGN()`, parses metadata from PGN headers (e.g., player names, results), enriching the user interface with contextual information. These headers are shown in `HomeView.vue`
- **AIChat.vue:** Handles communication with the backend APIs. Since the backend is stateless, this component maintains session state. In order to enhance user experience, it processes streamed responses to mitigate latency issues [37] and uses `markdown-it`[4] to enhance readability, allowing the AI to emphasize moves through bolding, bullet lists, and other formatting.

3.1.2 Data Flow and Reactivity

The frontend adopts a unidirectional data flow model to promote clarity and predictability. When a user interacts with the board, the following sequence occurs:

1. The chessboard emits a move event.
2. `ChessBoard.vue` validates the move using `chess.js`.
3. The resulting FEN is updated and propagated to `HomeView.vue`.
4. `HomeView.vue` passes the FEN to `AIChat.vue`, which, at the user request, calls to the backend for an analysis.
5. The response is rendered incrementally, using delimiters `[START_STREAM]` and `[END_STREAM]` to organize content flow.

[2] `https://github.com/qwerty084/vue3-chessboard`
[3] `https://www.npmjs.com/package/chess.js/v/0.13.0`
[4] `https://github.com/markdown-it/markdown-it`

3.2 Backend

The backend of ShashGuru is developed in Python and is the central orchestrator of the application's core functionality: chess position evaluation, prompt generation, and natural language analysis. It acts as a middleware layer between the frontend, the chess engine (ShashChess), and the LLM (Llama3.1-8B) , exposing functionality via a set of REST APIs. At the heart of the backend is a Flask application, which provides HTTP endpoints, manages request routing, and handles asynchronous communication between the various processing components.

Each part of the backend has a clearly defined responsibility, promoting modularity and testability.

3.2.1 Core Modules

Each part of the backend has a clearly defined responsibility, promoting modularity and testability.

LLMHandler.py – Natural Language Generation Interface

This module is responsible for initializing, managing, and interacting with the hosted large language model instance.

- **LLM Initialization:** On server startup, various instances of the LLM are initialized via vLLM and made ready for interaction.
- **Prompt Creation Functions:** Contains utilities to create structured and context-aware prompts, including both initial prompts (built using FEN context and engine evaluation) and follow-up prompts (based on conversational context).
- **Chat Management:** Maintains stateless interaction by accepting serialized chat histories (e.g., a JSON list of questions and answers) and uses that to construct the full conversation context. This ensures a seamless interaction for the end user.

EngineCommunication.py – Chess Engine UCI Interface

This module manages the lifecycle and interaction with any UCI-compatible chess engine.

- **Engine Abstraction Layer:** Designed to be engine-agnostic, allowing plug-and-play compatibility with any UCI engine, although currently using ShashChess. This is achieved by spawning the engine process and communicating over stdin/stdout using the UCI protocol.
- **Position Evaluation Logic:** Receives a FEN string and a target depth and instructs the engine to evaluate the position. The engine returns:
  - Top 3 moves, with their given score (centipawn or mate)
  - 3 follow-up moves (the expected best line for each move, stripped of the opponent's moves)
  - WDL of the position

- **Structured Result Output:** Parses the raw UCI output and returns a clean Python object that includes move recommendations and scores. This output is then used downstream for LLM prompt construction.

FenManipulation.py – FEN to Natural-Language Transformation

This utility script provides crucial preprocessing logic for translating Forsyth–Edwards Notation (FEN) strings into natural language descriptions.

- **Purpose:** LLMs often struggle with understanding raw FEN strings, especially in zero-shot contexts. By translating FEN into a natural language description of the board state (e.g., "Black king can castle kingside. a8 has a black rook, g1 has a white king"), the prompts become more grounded and significantly reduce the risk of LLM hallucinations [36].
- **Implementation Details:**
  - Extracts components from FEN, which are:
    - piece positions
    - side to move
    - castling rights
    - En-passant target square
    - Halfmove clock (useful for the 50-move-rule)
    - Fullmove number: number of turns, incremented after black move
  - Expands the positions into human-readable summaries using piece-square mapping.

3.2.2 REST API Endpoints

The Flask server exposes two primary endpoints that define ShashGuru's backend capabilities.

/analysis – Position Analysis

This endpoint follows the **POST** method and is designed to handle new FEN inputs in order to produce a comprehensive natural language evaluation of the corresponding chess position. The pipeline that governs this process is composed of several steps.

First, the system accepts as input a JSON object containing a valid FEN string (without server-side validation). This string serves as the representation of the current board state and acts as the primary data source for the subsequent analysis.

Once the input is received, the pipeline initiates an **engine evaluation** phase. During this stage, the `EngineCommunication.py` module is invoked to analyze the position using the UCI chess engine. This component returns key information such as the best continuation lines, numerical evaluation scores, and the depth of the analysis. These outputs provide the necessary computational insight required for generating a meaningful description of the position.

Following the engine's evaluation, the process moves to the **prompt enrichment** stage. Here, the `FenManipulation.py` module is utilized to convert the FEN string into a preliminary natural language description, as described before. This textual description is then enriched with the information derived from the engine analysis. The resulting prompt is structured carefully to ensure clarity and informativeness, preparing it for interaction with the LLM.
In the next step, this prompt is forwarded to the `LLMHandler.py` module, which manages communication with the LLM. The model processes the input and returns a detailed natural language explanation of the chess position, combining both the board layout and the strategic insights into a coherent narrative.

Finally, the system streams the response in two parts: first, it sends the constructed prompt as a single complete chunk, terminated by a designated end tag; then, it streams the model-generated explanation incrementally, delivering it in successive chunks as the text is being generated. This approach enhances the perceived responsiveness of the system and provides a smoother user experience.

/response – Chat Continuation

This endpoint is implemented using the **POST** method and is designed to handle follow-up questions concerning a chess position that has already been analyzed. In contrast to the initial evaluation endpoint, this route does not trigger any additional engine computations. Instead, its primary objective is to preserve the conversational context and generate coherent, context-aware responses using the language model.

The process begins with a JSON payload containing the chat history. The history size is set to 10. From this, the newly submitted question is separated and treated distinctly. Both the prior conversation and the new question are then passed to the language model via `LLMHandler.py`. By leveraging the accumulated dialogue, the system is able to maintain continuity across exchanges, enabling more natural, consistent, and informed interactions.

The output of this process is once again streamed back, without the history, which is assumed to be maintained by the frontend.

3.2.3 Engineering Considerations

- **Stateless API:** The backend is designed to be stateless between calls, which simplifies horizontal scaling. All session state is expected to be maintained by the frontend.
- **LLM Model Management:** The LLM subprocess is kept alive between requests to avoid cold starts and expensive startup (both time-wise and resource-wise).
- **LLM Fault Tolerance:** Errors from the model server (timeouts, overloads) are caught and reported to the user gracefully.

- **LLM Temperature:** The temperature parameter for the LLM was set to 0, as both empirical testing and [22] indicated that the deterministic nature of chess favors low temperatures to ensure consistent results.

3.2.4 Infrastructure for LLM Inference

ShashGuru's primary objective is research, with a secondary aim of being showcased at a prestigious chess event hosted by the University of Bologna in September 2025. During this event, ShashGuru will provide game analysis immediately following each match. Given the potential need to handle numerous analyses simultaneously, the system demands exceptional computational power.

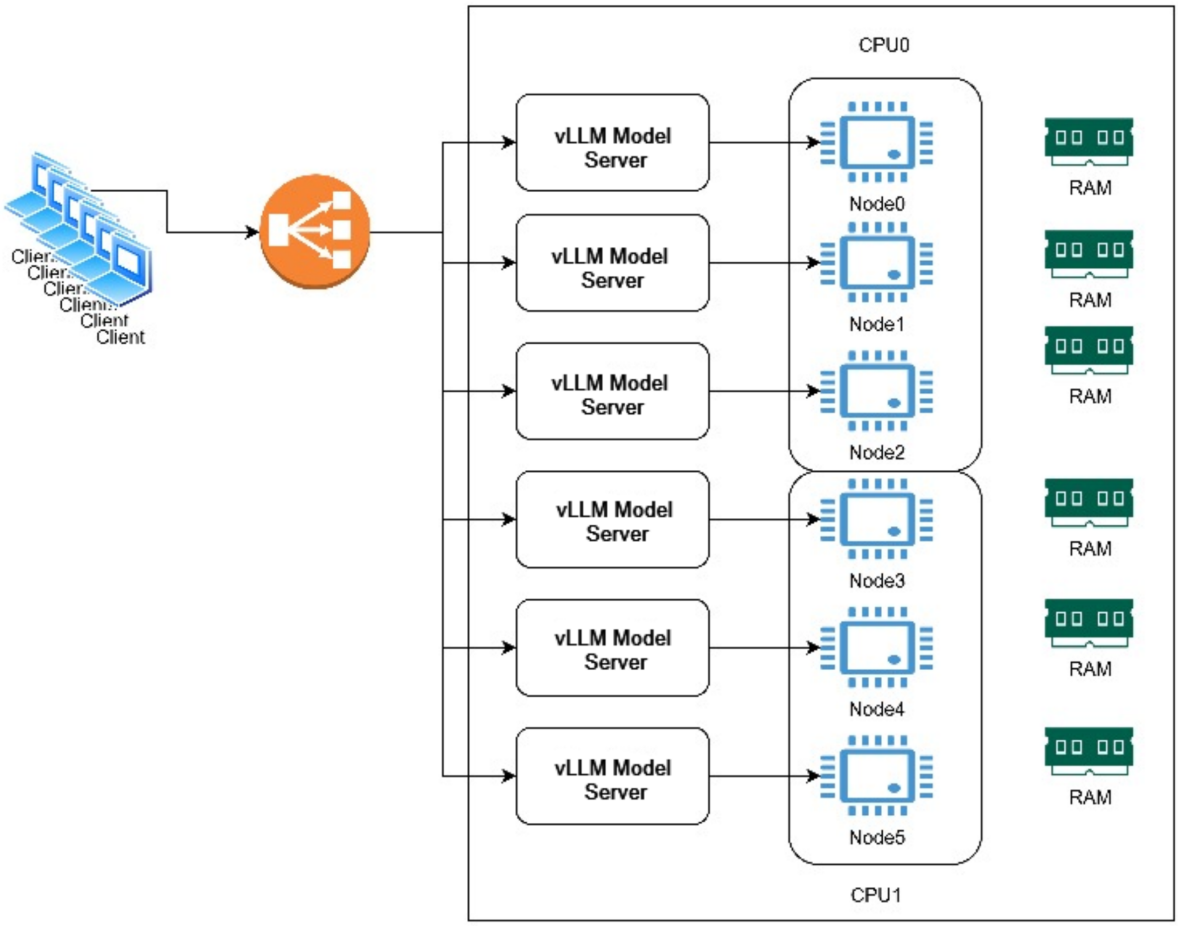


Figure 3.1: System architecture for load balancing. *Adapted under fair use for academic purposes from "Continuous batching and scaling" by Intel (OpenVINO Model Server 2025 documentation)*

Hardware Setup

To address these rigorous real-time processing requirements, ShashGuru leverages a high-performance server generously provided by Intel. Key components include:

- 2x Intel Xeon 6972P processors (delivering an impressive total of 192 cores and 384 threads)

- 1.25TB of DDR5 memory

Multi-NUMA Optimization

The system partitions workloads across Non-Uniform Memory Access (NUMA) nodes to minimize latency. Each NUMA node hosts a separate vLLM instance (see below), bound to dedicated CPU cores to avoid cross-node contention.

vLLM Runtime

The model is deployed using vLLM, an optimized inference engine that implements:

- PagedAttention: Efficient memory management for variable-length sequences.
- Continuous Batching: Increased throughput by processing multiple requests in parallel
- Compatibility: It is compatible with LLaMA.

These features make vLLM significantly faster than traditional inference libraries like Hugging Face Transformers, making it a more suitable choice for this project. Performance is further enhanced by quantizing the model to 4 bits, which reduces memory usage and accelerates inference speed.

Load Balancing

An NGINX reverse proxy distributes requests across multiple vLLM instances based on real-time load metrics.

Deployment

To deploy ShashGuru, a containerized architecture was implemented using Docker Compose. The various dockers contain the six vLLM model instances, the frontend and the backend. Each model instance has 64GB of RAM and 16GB of shared memory allocated.

3.2.5 Summary

ShashGuru's backend is an orchestration layer combining engine computation, language model reasoning, and prompt engineering into a cohesive analysis pipeline. It is modular by design, ensuring that the chess engines can be swapped, the LLM can be upgraded or changed, and user interactions and prompts are managed effectively.

3.3 Prompt Engineering

This section explains the structure of the prompt passed to the language model (LLM) for analyzing a chess position. The prompt is automatically generated from a given FEN string, enriched with engine evaluation data (from ShashChess), and formatted in natural language to guide the model in producing concise, high-quality analysis.

3.3.1 Failed attempts

Several initial strategies for prompt engineering proved ineffective or counterproductive. These are outlined below:

- **Direct Use of FEN Strings**: Early prompts supplied the FEN directly to the model without translating it into natural language via `fenManipulation.py`. This approach resulted in poor comprehension by LLaMA, leading to incoherent or irrelevant outputs. In particular it did not have a good comprehension of piece placement. It became clear that the model could not accurately interpret the raw FEN format.

- **Output Length Constraints**: Initial prompts emphasized brevity, requesting concise responses. While this limited verbosity, it also severely restricted the depth and quality of the answers, often yielding only a single sentence. Subsequent prompts encouraged more expressive responses, but this led to increased hallucinations and factual inaccuracies. In particular, the first half of the answer would be acceptable, but the second half consistently returned grave mistakes. Eventually, setting a limit of approximately 800 characters struck a workable balance between expressiveness and reliability.

- **Minimal Engine-derived Info in Initial Prompting**: An iteration of the system only presented a natural language description of the position (derived from the FEN) alongside the best move and its evaluation. While sufficient for generating a primary recommendation, this minimal context left the model ill-equipped to justify or discuss alternative moves in follow-up queries, leading to shallow or inconsistent reasoning.

- **Dual-Engine Cross-referencing**: One experimental setup involved querying two different engines (ShashChess and Alexander, with the latter being heuristic-based) and cross-referencing their evaluations in the prompt. The intention was to provide richer context and potentially more robust conclusions. In practice, however, this approach introduced ambiguity and inconsistency, confusing the language model rather than improving output quality. While the experiment did not yield the desired effect, the concept may still hold potential if refined further.

3.3.2 Input Format

The only input to the generation system is the FEN string. From this, all other prompt elements are computed or retrieved automatically.

3.3.3 Purpose and Benefits

This prompt structure ensures a clean separation of concerns:

- The natural language description enables the model to interpret the board without relying on raw FEN parsing capabilities [36].

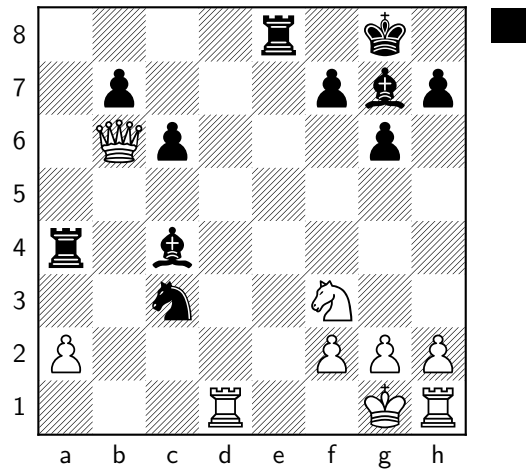
- The structured evaluation data focuses the model on accurate, relevant ideas rather than speculation.

- The consistent template allows the output to be concise and aligned with strategic themes important in chess (activity, coordination, weaknesses, etc.).

The system is robust to changes in board state and can generalize across a wide range of positions. By controlling the language and structure of the input, we improve both the interpretability and reliability of the model's analysis.

3.3.4 Example of prompt

The following is the automatically generated prompt for the position:



The prompt itself is shown in *italics* for clarity and separation from the surrounding document. Within the prompt, different elements are color-coded for interpretability:

- **Blue text**: Information calculated from the given FEN. This includes the board state description, side to move, move number, castling/en passant status, and a qualitative positional summary derived from simple heuristics.
- **Red text**: Data retrieved (without being elaborated further) from the chess engine (ShashChess), including the best move, evaluation score (in centipawns), and probabilities of win, draw, and loss.
- **Black text**: Template instructions, unchanged between prompts. These provide consistent formatting and guidance to the model on what to analyze and how to respond.

This formatting scheme ensures that the model has all necessary information presented in a natural, interpretable way, thus reducing ambiguity and increasing the reliability of its responses.

Chess Position Analysis Request
It's move number 25.
It's Black's turn to move
e8 has a black rook, g8 has a black king, b7 has a black pawn, f7 has a black pawn, g7 has a

black bishop, h7 has a black pawn, b6 has a white queen, c6 has a black pawn, g6 has a black pawn, a4 has a black rook, c4 has a black bishop, c3 has a black knight, f3 has a white knight, a2 has a white pawn, f2 has a white pawn, g2 has a white pawn, h2 has a white pawn, d1 has a white rook, g1 has a white king, h1 has a white rook, Every unmentioned square is empty.
There have been 0 move without captures.
There are no en-passant moves to be made.
Current situation: Black has a decisive advantage, with victory nearly assured.
Please provide concise analysis (<800 chars) covering:
1. ***Advantage Assessment*** *- Who stands better and the primary reason (material/ structure/ activity)*
2. ***Plan****- The best move is c3d1 (eval: 574 context points) and it considers that the principal variation (of only Black moves) is c4d5 h7h5 a4e4. Why is the move good and what is the plan? (max 1-2 ideas).*
3. ***Expected Outcome*** *- Give a brief summary of the winning chances, considering that the win probability for Black is 100.0%, draw probability is 0.0%, and loss probability is 0.0%.*

Focus on concrete factors like:
- Key weaknesses/squares
- Piece activity/coordination
- Pawn structure implications
- Immediate tactical motifs

3.3.5 Building the prompt

Natural Language FEN

As mentioned in 3.2.1, `FenManipulation.py` translates the FEN in natural language, to make it more understandable for the LLM.

Win Probability and Positional Classification

The "current situation" string is taken from a switch table based on win probability ranges, derived from the ShashChess framework. These ranges map to distinct positional classifications, each describing the nature of the advantage or balance in a chess position, as detailed in Table 3.1.
The win probability is computed using the formula:

$$WinProbability = w \cdot \frac{d}{2}$$

where w and d represent the win and draw probabilities, respectively, as estimated by the WDL (Win-Draw-Loss) model in ShashChess. This metric quantifies the expected winning chances, ranging from 0% (certain loss) to 100% (certain win).
The classification system spans a spectrum from Petrosian (defensive or losing positions) to Capablanca (balanced positions) and Tal (aggressive or winning positions), with intermediate categories reflecting varying degrees of advantage. Notably, the Chaos category captures dynamically unbalanced positions where neither side holds a clear edge. The complete mapping is provided in Table 3.1.
The original descriptions found in ShashChess documentation proved hard to understand for the LLM, probably because they lacked clear distinction of the side who is winning. They were then adapted to make it clearer, as shown in Table 3.2

Shashin Position's Type	**Win Probability Range**	**Description**
High Petrosian	[0, 5]	Winning: a decisive advantage, with the position clearly leading to victory.
High-Middle Petrosian	[6, 10]	Decisive advantage: dominant position and likely winning.
Middle Petrosian	[11, 15]	Clear advantage: a substantial positional advantage, but a win is not yet inevitable.
Middle-Low Petrosian	[16, 20]	Significant advantage: strong edge.
Low Petrosian	[21, 24]	Slight advantage with a positional edge, but no immediate threats.
Chaos: Capablanca-Petrosian	[25, 49]	Opponent pressure and initiative: defensive position.
Capablanca	[50, 50]	Equal position. Both sides are evenly matched, with no evident advantage.
Chaos: Capablanca-Tal	[51, 75]	Initiative: playing dictation with active moves and forcing ideas.
Low Tal	[76, 79]	Slight advantage: a minor positional edge, but it's not significant.
Middle-Low Tal	[80, 84]	Slightly better, tending toward a clear advantage. The advantage is growing, but the position is still not decisive.
Middle Tal	[85, 89]	Clear advantage: a significant edge, but still with defensive chances.
High-Middle Tal	[90, 94]	Dominant position, almost decisive, not quite winning yet, but trending toward victory.
High Tal	[95, 100]	Winning: a decisive advantage, with victory nearly assured.
Chaos: Capablanca-Petrosian-Tal	(Unclassified)	Total chaos: unclear position, dynamically balanced, with no clear advantage for either side and no clear positional trends.

Table 3.1: Mapping of win probabilities to Shashin positional types with descriptions, adapted from a table in the ShashChess GitHub repository, `https://github.com/amchess/ShashChess/blob/master/README.md`.

ShashChess Descriptions	**ShashGuru adaptations**
Winning: a decisive advantage, with the position clearly leading to victory.	*side* has a decisive disadvantage, with the position clearly leading to a loss.
Decisive advantage: dominant position and likely winning.	*side* has decisive disadvantage: the opponent has a dominant position and is likely winning.
Clear advantage: a substantial positional advantage, but a win is not yet inevitable.	*side* has clear disadvantage: a substantial positional disadvantage, but a win is not yet inevitable.
Significant advantage: strong edge.	*side* has a significant disadvantage: difficult to recover.
Slight advantage with a positional edge, but no immediate threats.	*side* has a slight disadvantage with a positional edge, but no immediate threats.
Opponent pressure and initiative: defensive position.	*side* is in a defensive position. The opponent has an initiative.
Equal position. Both sides are evenly matched, with no evident advantage.	The position is equal. Both sides are evenly matched, with no evident advantage.
Initiative: playing dictation with active moves and forcing ideas.	*side* has initiative: by applying pressure and it can achieve an edge with active moves and forcing ideas.
Slight advantage: a minor positional edge, but it's not significant.	*side* has a slight advantage: a minor positional edge, but it's not decisive.
Slightly better, tending toward a clear advantage. The advantage is growing, but the position is still not decisive.	*side* is slightly better, tending toward a clear advantage. The advantage is growing, but the position is still not decisive.
Clear advantage: a significant edge, but still with defensive chances.	*side* has a clear advantage: a significant edge, but still with defensive chances.
Dominant position, almost decisive, not quite winning yet, but trending toward victory.	*side* has a dominant position, almost decisive, not quite winning yet, but trending toward victory.
Winning: a decisive advantage, with victory nearly assured.	*side* has a decisive advantage, with victory nearly assured.
Total chaos: unclear position, dynamically balanced, with no clear advantage for either side and no clear positional trends.	Total chaos: unclear position, dynamically balanced, with no clear advantage for either side and no clear positional trends.

Table 3.2: Adaptation of position win probabilities from ShashChess to ShashGuru. Here *side* is either "White" or "Black", based on who is going to move next.

Chapter 4

Results

This chapter will focus on the results obtained by ShashGuru, both performance-wise and quality-wise.

4.1 User Evaluation

A questionnaire was sent to a group of chess players of varying skill levels. The goal is to assess ShashGuru's effectiveness in providing complete, accurate, and relevant analyses compared to traditional methods.
The questionnaire included 10 chess positions, analyzed by ShashGuru. Each position assessed:

- **Completeness**: Coverage of critical aspects.
- **Accuracy**: Logical consistency with chess principles.
- **Relevance**: Focus on the most important and instructive ideas.

Furthermore, to investigate the LLM's standalone capabilities, control analyses were generated by Deepseek with the original ShashGuru prompt structure, but removing:

- All engine-derived data (best move, evaluation score, etc)
- FEN-to-natural-language transformation

Users were then asked to express their preference between the ShashGuru system and base Deepseek, by voting from 1 to 5, where 1 is a strong preference for Deepseek and 5 a strong preference for ShashGuru.

Discussion of Results

In total, the form was answered by **16** people with varied chess experience, one of whom did not vote for preference between Deepseek and ShashGuru.
The user evaluation results of ShashGuru demonstrate an encouraging picture of its performance as a chess analysis tool. The collected data highlight both the strengths and limitations of the system in its current form.

From the scores in Table 4.1, it is clear that ShashGuru achieved moderate results in all three evaluated dimensions: completeness, accuracy, and relevance, each being between

Table 4.1: Average Ratings (1-5 scale) by Position

Position	Completeness	Accuracy	Relevance
1	2.87	2.94	2.81
2	3.12	3.06	3.12
3	3.31	3.50	3.31
4	3.62	3.75	3.50
5	3.37	3.37	3.37
6	3.62	3.19	3.37
7	3.81	3.87	3.93
8	3.87	4.00	3.93
9	3.50	3.56	3.81
10	3.06	3.00	3.06

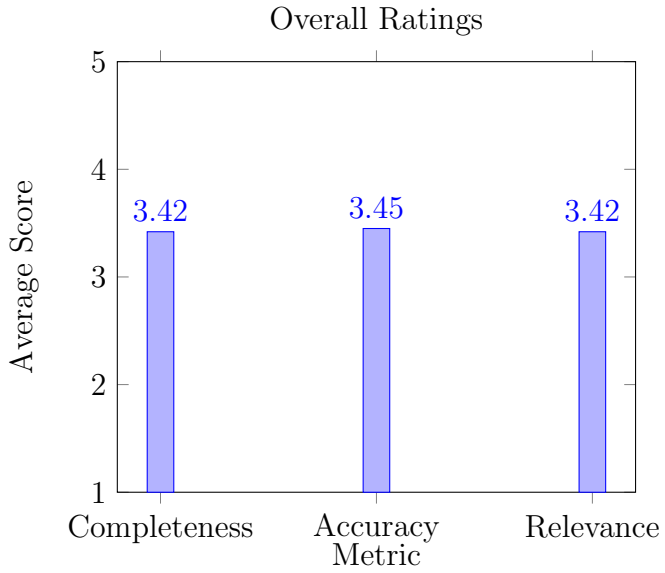


Figure 4.1: Performance across evaluation metrics

the 3.4 and the 3.5 mark on a scale of 1 to 5. This reflects a system that performs respectably, though not exceptionally, in its analytical capabilities. Although none of the metrics show poor performance, the ratings also indicate room for improvement, particularly in ensuring that the system consistently provides more in-depth, precise, and contextually meaningful evaluations.

However, when comparing ShashGuru to the control condition (Deepseek's outputs generated without engine guidance or structured natural language inputs) the preference scores (Figure 4.2) strongly favor ShashGuru. The average preference rating of 4.25 out of 5 underscores a marked user inclination towards ShashGuru's analyses. This contrast suggests that, even if ShashGuru is not yet an ideal analysis tool, its incorporation of engine-backed insights and structured prompts significantly elevates the perceived quality of its outputs. The overall takeaway is that while ShashGuru is not without shortcomings, the human evaluation clearly validates its methodology and positioning. The consistent advantage over the Deepseek baseline illustrates that ShashGuru is a promising step forward in combining large language models with chess engines. These results support further development and refinement, with an emphasis on increasing analytical

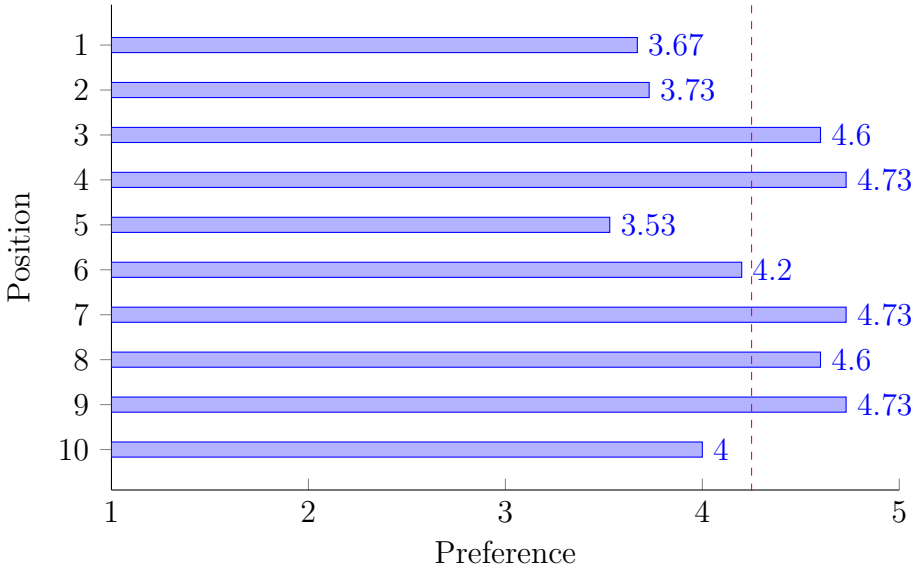


Figure 4.2: A score of 1 indicates a preference for Deepseek, while a score of of 5 a preference for ShashGuru. Red-dashed line is average

depth and aligning better with expert-level chess reasoning.

4.2 Hallucinations

To assess the factual accuracy of the method proposed for ShashGuru, we conducted a deeper analysis of the outputs used during the user evaluation. For each position, we counted the number of factual references made by the models. These include any statements that can be objectively classified as either true or false, excluding evaluations or recommendations about the "best play." For instance, a move is considered "correct" if it is legal, regardless of its strategic merit. Other examples of factual references include piece placement or explanations of rules. We then recorded the number of factual errors, such as illegal moves, incorrect descriptions of piece locations, or references to invented rules.
This evaluation enables us to quantify the reliability of the model. The results are summarized in Table 4.2, which presents both the absolute counts and the corresponding error rates for each model across the ten analyzed positions, along with cumulative totals and overall accuracy.

Examples of Errors

To better illustrate the types of factual mistakes observed, two representative examples are reported here (positions can be found in Appendix A:

- In Position 5, ShashGuru suggests a plan involving "promoting the rook to a stronger piece (queen)," which is, of course, not legal under standard chess rules.
- In Position 3, Deepseek recommends a pawn move while the king is in check. The suggested move does not block the check, making it illegal.

Table 4.2: Factual references and errors across the 10 analyses made by Deepseek and ShashGuru in the user evaluation.

Position	**ShashGuru**		**Deepseek**		**Error Rate (%)**	
	Errors	Facts	Errors	Facts	ShashGuru	Deepseek
1	0	8	3	13	0.00	23.08
2	4	8	4	12	50.00	33.33
3	0	1	9	16	0.00	56.25
4	1	5	6	14	20.00	42.86
5	0	5	4	15	0.00	26.67
6	1	4	6	13	25.00	46.15
7	0	4	2	7	0.00	28.57
8	0	2	4	11	0.00	36.36
9	0	4	6	12	0.00	50.00
10	3	7	3	9	42.86	33.33
Total	**9**	**48**	**47**	**122**	**18.75**	**38.52**

Discussion of results

These results, shown in Table 4.2, demonstrate a meaningful reduction in the overall error rate. While the drop from 38% to 18% indicates clear progress, the remaining error rate is still non-negligible, especially in chess. However, a more encouraging aspect of the results is the distribution of errors: all the mistakes are concentrated in four out of the ten positions, while the remaining six show no errors at all. This contrasts Deepseek's output, whose outputs all show some degree of mistakes.
This suggests that, rather than being uniformly unreliable, ShashGuru performs consistently well in most cases, with a few outliers contributing disproportionately to the overall error rate.
A paired two-tailed t-test confirms that the difference in error rates between ShashGuru and Deepseek is statistically significant, with a p-value of 0.009.

Additionally, it is interesting to note that ShashGuru consistently references fewer facts than Deepseek, perhaps a consequence of the low temperature it was set to. More precise investigation on this matter is left for future work.

4.3 Performance

We present a comprehensive evaluation of ShashGuru's infrastructure under three distinct request profiles, analyzing both latency distributions and quantitative metrics. The benchmarks compare High (3 req/s), Medium (1 req/s) and Low (0.2 req/s) request rates, with burstiness factors adjusted to simulate realistic traffic patterns. The request rates are shown in Figure 4.3.

4.3.1 Latency Distribution

The following metrics were considered:

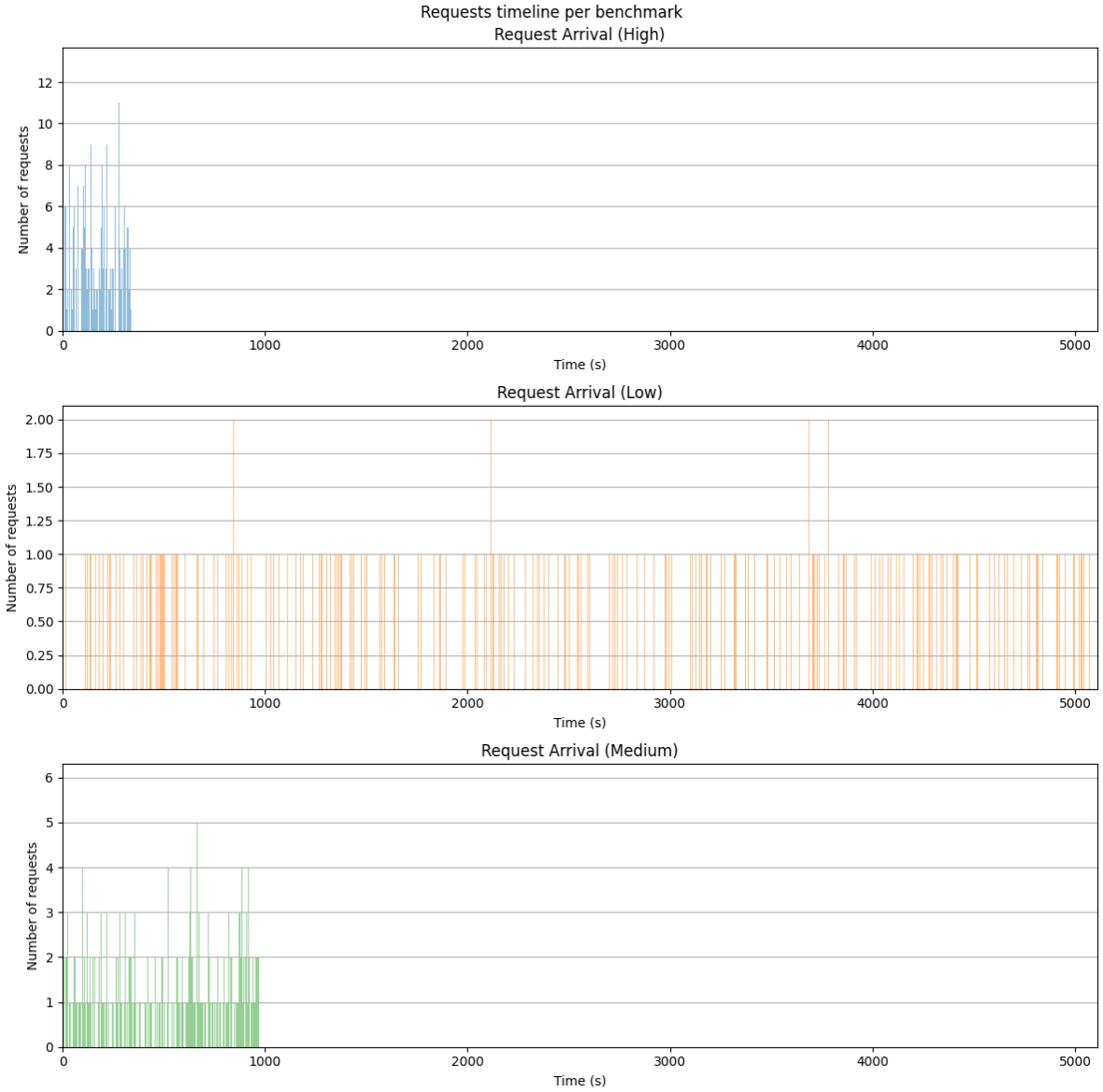


Figure 4.3: Request arrival patterns over time for High, Medium, and Low frequencies.

- TTFT (Time to First Token): The time (in seconds) from when a request is sent to when the first output token is received. Lower is better.
- ITL (Inter-Token Latency): The time (in seconds) between consecutive output tokens. Lower is better.
- Mean: The average value.
- Median: The middle value when sorted.
- P95/P99: The 95th/99th percentile (i.e., 95%/99% of values are below this).

Time to First Token (TTFT)

Figure 4.4 shows the TTFT distributions, while Table 4.3 provides their statistical breakdown.

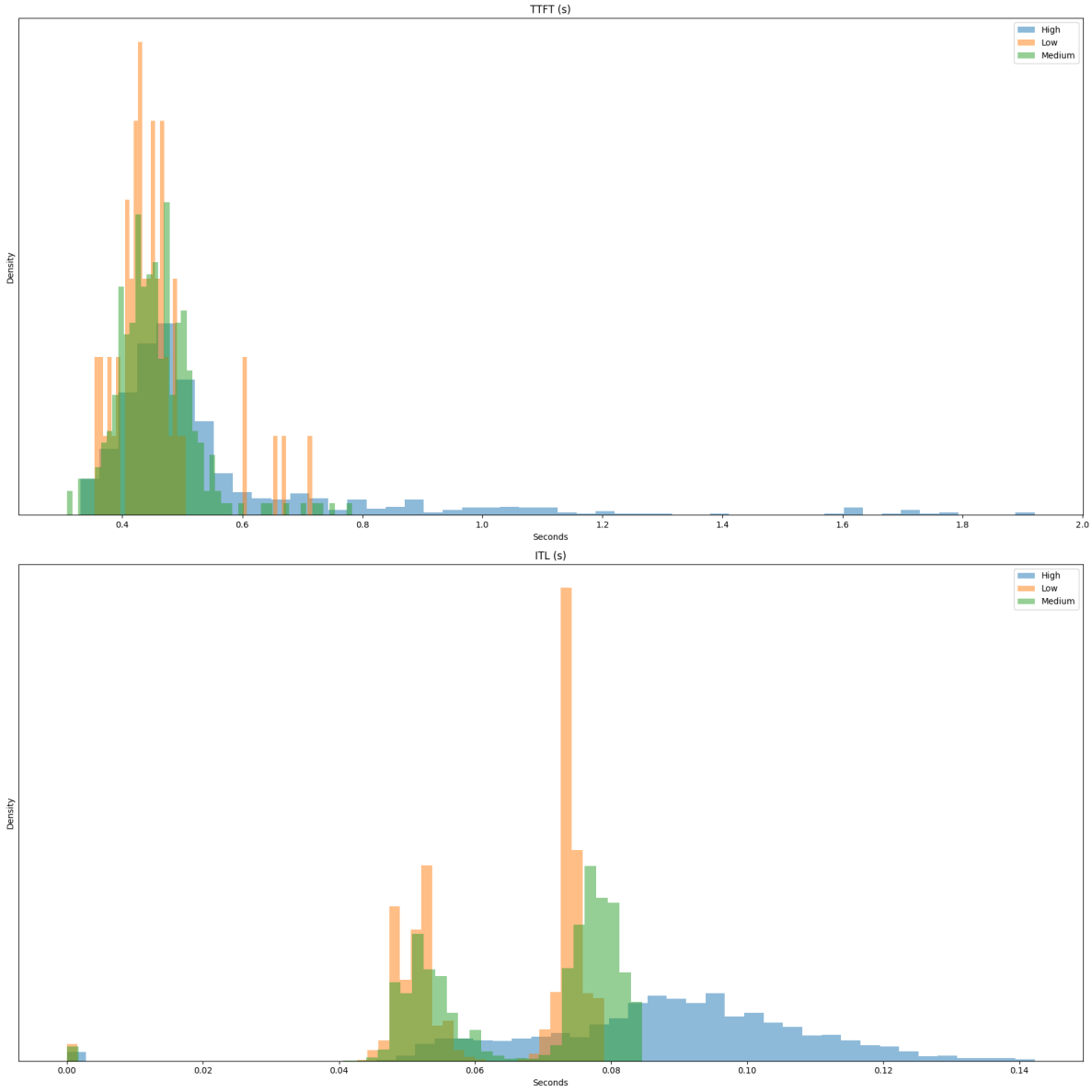


Figure 4.4: Latency distribution across different request frequencies. Top: TTFT. Bottom: ITL.

Key observations:

- **High-load degradation**: The 95th percentile TTFT (1.032s) exceeds the median by 114%, indicating severe tail latency under load. The 99th percentile spikes to 1.687s, which is 3.5 times more than the median latency.
- **Medium/Low consistency**: Both profiles show tight distributions, with P99 values remaining below 0.72s. Figure 4.4 reveals the advantage of low load in minimizing extreme outliers.

Inter-Token Latency (ITL)

Figure 4.4 shows the ITL distributions, while Table 4.4 provides their statistical breakdown.

Table 4.3: Time To First Token (TTFT) in seconds

Profile	**Mean**	**Median**	**P95**	**P99**
High	0.551	0.482	1.032	1.687
Medium	0.455	0.449	0.550	0.720
Low	0.451	0.436	0.604	0.688

Notable patterns:

- **High-load penalty**: Median ITL increases by 24% compared to low load, with the gap widening at P95. This confirms resource contention affects token generation more severely than initial response.
- **Tail resilience**: All profiles show compressed P95–P99 ranges ($\leq$0.012s difference), suggesting ITL distributions have shorter tails than TTFT.

Table 4.4: Inter-Token Latency (ITL) in seconds

Profile	**Mean**	**Median**	**P95**	**P99**
High	0.0884	0.0900	0.1192	0.1313
Medium	0.0676	0.0748	0.0825	0.0841
Low	0.0636	0.0726	0.0768	0.0785

Conclusions

This thesis introduced ShashGuru, a chess analysis tool that bridges the computational precision of modern chess engines with the natural language generation capabilities of Large Language Models. By combining ShashChess, a customized chess engine derived from Stockfish, with Meta's LLaMA 3.1-8B model, ShashGuru addresses the limitations of traditional chess engines, whose analyses often appear inexplicable to the average user, and LLMs, which struggle with accurate chess reasoning. This system provides a way to exploit the main strengths of both.
The implementation focused on delivering a responsive web application that transforms raw chess engine outputs into natural language explanations. The architecture, built on a Vue.js frontend a Flask backend, was designed to support stateless processing, modularity and future extensibility.
Central to this design was the concept of prompt engineering, which included the conversion of FEN notation into natural language descriptions to improve LLM comprehension and the careful choice of which engine-derived elements to feed as input, to avoid confusion.

A user study validated the effectiveness of this approach. Chess players of varying skill levels consistently preferred ShashGuru's output over that of unassisted LLMs. The system achieved moderate scores in completeness, accuracy and relevance, but showed significant superiority in perceived usefulness and clarity compared to the control condition. To complement these findings, a targeted evaluation of hallucinations was conducted by analyzing the factual correctness of the model's statements, revealing measurable error rates that help quantify the model's reliability.

Nevertheless, there are limitations. The analyses, while promising, still present hallucinations and leave room for improvement, as they are evidently still not at human-level analysis, at least not competent human level.
Future work could explore fine-tuning the LLM on annotated chess data and expanding the platform to support user-adaptive feedback (i.e., coach mode) to further boost its utility. Additionally, there is potential to generalize the approach to other strategic domains beyond chess.

In conclusion, ShashGuru represents a meaningful step toward interpretable and accessible chess AI, leveraging the complementary strengths of symbolic reasoning and natural language generation. It contributes both a working prototype and a modular framework that future researchers and developers can build upon in the evolving landscape of human-AI interaction.

Bibliography

[1] Y. Averbakh, *A history of chess: From chaturanga to the present day.* SCB Distributors, 2012.

[2] C. E. Shannon, "Xxii. programming a computer for playing chess," *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, vol. 41, no. 314, pp. 256–275, 1950.

[3] H. J. Berliner, "A chronology of computer chess and its literature," *Artificial Intelligence*, vol. 10, no. 2, pp. 201–214, 1978.

[4] M. Campbell, A. J. Hoane Jr, and F.-h. Hsu, "Deep blue," *Artificial intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.

[5] Wikipedia contributors, "Zero-sum game," https://en.wikipedia.org/wiki/Zero-sum_game.

[6] ——, "Perfect information," https://en.wikipedia.org/wiki/Perfect_information.

[7] M. E. Lombardo, A. Sansone, and D. F. Slezak, "Feature set analysis for chess nnue networks," Bachelor's Thesis, Universidad de Buenos Aires, 2024.

[8] C. G. Diderich and M. Gengler, "A survey on minimax trees and associated algorithms," *Minimax and Applications*, pp. 25–54, 1995.

[9] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[10] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[11] D. Klein, "Neural networks for chess," *arXiv preprint arXiv:2209.01506*, 2022.

[12] Y. Nasu, "Efficiently updatable neural-network-based evaluation functions for computer shogi," *The 28th World Computer Shogi Championship Appeal Document*, vol. 185, 2018.

[13] Stockfish Developers, "Introducing nnue evaluation," https://stockfishchess.org/blog/2020/introducing-nnue-evaluation/, 2020, accessed: 2025-06-20.

[14] Chess Programming Wiki contributors, "Alpha-beta," https://www.chessprogramming.org/Alpha-Beta, 2024.

[15] D. F. Beal, "A generalised quiescence search algorithm," *Artificial Intelligence*, vol. 43, no. 1, pp. 85–98, 1990.

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[17] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler *et al.*, "Emergent abilities of large language models," *arXiv preprint arXiv:2206.07682*, 2022.

[18] L. Salamone, "What is temperature?" https://blog.lukesalamone.com/posts/what-is-temperature/, 2023, accessed: 2025-07-01.

[19] R. Gallotta, G. Todd, M. Zammit, S. Earle, A. Liapis, J. Togelius, and G. N. Yannakakis, "Large language models and games: A survey and roadmap," *IEEE Transactions on Games*, 2024.

[20] M.-T. Kuo, C.-C. Hsueh, and R. T.-H. Tsai, "Large language models on the chessboard: A study on chatgpt's formal language comprehension and complex reasoning skills," *arXiv preprint arXiv:2308.15118*, 2023.

[21] S. Wang, L. Ji, R. Wang, W. Zhao, H. Liu, Y. Hou, and Y. N. Wu, "Explore the reasoning capability of llms in the chess testbed," *arXiv preprint arXiv:2411.06655*, 2024.

[22] B. Albert Gramaje, "Exploring gpt's capabilities in chess-puzzles," Master's thesis, Universitat Politècnica de València, 2023.

[23] H. Zang, Z. Yu, and X. Wan, "Automated chess commentator powered by neural chess engine," *arXiv preprint arXiv:1909.10413*, 2019.

[24] D. Michie, "A theory of evaluative comments in chess with a note on minimaxing," *The Computer Journal*, vol. 24, no. 3, pp. 278–286, 1981.

[25] J.-W. Liao and J. S. Chang, "Computer generation of chinese commentary on othello games," in *Proceedings of Rocling III Computational Linguistics Conference III*, 1990, pp. 393–415.

[26] A. Sadikov, M. Možina, M. Guid, J. Krivec, and I. Bratko, "Automated chess tutor," in *International Conference on Computers and Games*. Springer, 2006, pp. 13–25.

[27] H. Jhamtani, V. Gangal, E. Hovy, G. Neubig, and T. Berg-Kirkpatrick, "Learning to generate move-by-move commentary for chess games from large-scale social forum data," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018, pp. 1661–1671.

[28] A. Lee, D. Wu, E. Dinan, and M. Lewis, "Improving chess commentaries by combining language models with symbolic reasoning engines," *arXiv preprint arXiv:2212.08195*, 2022.

[29] X. Feng, Y. Luo, Z. Wang, H. Tang, M. Yang, K. Shao, D. Mguni, Y. Du, and J. Wang, "Chessgpt: Bridging policy learning and language modeling," *Advances in Neural Information Processing Systems*, vol. 36, pp. 7216–7262, 2023.

[30] Z. Tang, D. Jiao, R. McIlroy-Young, J. Kleinberg, S. Sen, and A. Anderson, "Maia-2: A unified model for human-ai alignment in chess," *arXiv preprint arXiv:2409.20553*, 2024.

[31] V. T. Markos, "Application of the machine coaching paradigm on chess coaching," Master's thesis, Ανοικτό Πανεπιστήμιο Κύπρου, 2021.

[32] A. Manzo and P. Ciancarini, "Enhancing Stockfish: a Chess Engine Tailored for Training Human Players," in *Proceedings 14th IFIP Int. Conf. on Entertainment Computing - ICEC*, ser. Lecture Notes in Computer Science, vol. 14455. Bologna, Italy: Springer, 2023, pp. 275–289.

[33] A. Manzo and A. Caruso, "The computer chess world-how to make the most of chess software," 2021.

[34] A. Shashin, *BEST PLAY: A New Method For Discovering The Strongest Move*. Mongoose Press, 2013.

[35] M. AI, "Meta llama 3: The next generation of open source llms," 2024. [Online]. Available: https://ai.meta.com/blog/meta-llama-3/

[36] A. A. Antonov, "Day 4: Don't use forsyth-edwards notation to play chess with llms," https://raku-advent.blog/2024/12/04/day-4-dont-use-forsyth-edwards-notation-to-play-chess-with-llms/, Dec. 2024, accessed: 2025-06-22.

[37] OpenAI, "Latency optimization: Make your users wait less," 2023, openAI API Documentation. [Online]. Available: https://platform.openai.com/docs/guides/latency-optimization#make-your-users-wait-less

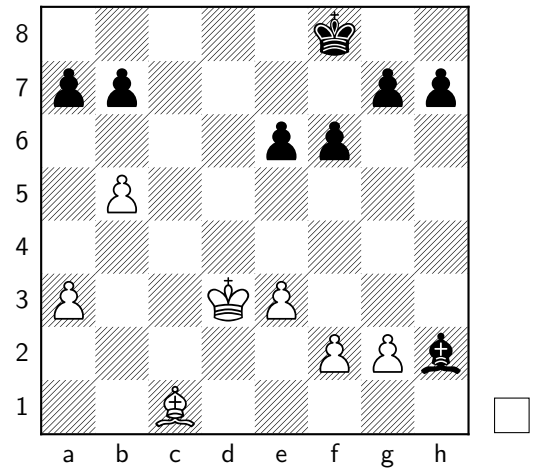
Appendix A

Raw Data Excerpts

Form Analysis

These are the 10 positions examined for the form, featuring analysis from both ShashGuru and DeepSeek.

Position 1



FEN: `5k2/pp4pp/4pp2/1P6/8/P2KP3/5PPb/2B5 w - - 0 30`

ShashGuru Analysis:

> Advantage Assessment: White stands better, primarily due to the initiative and active piece play. The pressure on Black's position, particularly the g7 pawn, can be exploited to create weaknesses.
>
> Plan: The best move is g2g3, which prepares to push the f-pawn, potentially targeting the weak g7 pawn. This move also supports the c1 bishop, enhancing its activity and coordination with other pieces. The principal variation c1d2 d3e2 e2f1 implies that this move is a necessary step in developing the bishop and preparing for further activity.
>
> Expected Outcome: With a 20.0% win probability, a 80.0% draw probability, and 0.0% loss probability, White's chances of winning are relatively low, but

still present. The current position is complex, and a draw is the most likely outcome, as Black can equalize with precise play. However, White's initiative and potential to create weaknesses on the kingside keep the win probability at 20.0

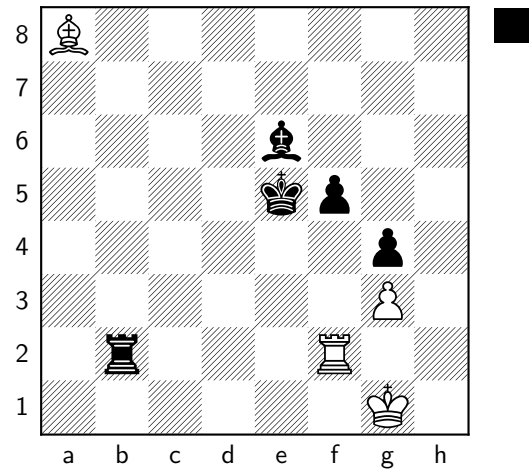
Deepseek Analysis:

1. Advantage Assessment: White is better due to material (extra pawn) and active pieces (bishop on c1 targets Black's weak kingside). Black's bishop on h2 is trapped, and the pawn structure favors White with a passed b5-pawn. Black's pawns are static and weak (a7, e6, f6).

2. Plan: Best move: Kd4, centralizing the king and preparing to push the b-pawn (b6-b7). The plan is to promote the b-pawn or tie Black down to stopping it, while the king supports the advance and White's bishop targets f4/g5 to restrict Black.

3. Expected Outcome: White should win with precise play. Black's counterplay is minimal (e.g., ...f5 or ...g5), but the b-pawn is too strong. Black's trapped bishop and weak pawns make defense nearly impossible. White's king activity seals the win.

Position 2



FEN: `B7/8/4b3/4kp2/6p1/6P1/1r3R2/6K1 b - - 17 55`

ShashGuru Analysis:

Advantage Assessment: Black stands significantly better, primarily due to the better pawn structure and the presence of a strong passed pawn on the d-file.

Plan: The best move for Black is b2f2 (eval: 522 context points). This move sets up the principal variation, e6d5 e5d5 d5e5, which aims to:

Create a passed pawn: Pushing the pawn to d5 (d6 is not a possibility due to f5 being a pawn) and then to d5-e5, with the pawn on e5 potentially becoming a powerful passed pawn that can be difficult for White to stop. Prepare to attack White's king: By pushing the pawn to e5, Black can potentially create a weakness on the e-file and set up a potential attack on White's king.

Expected Outcome: With Black's decisive advantage, the expected outcome is a win for Black, with a 100.0% win probability and 0.0% draw and loss probabilities.

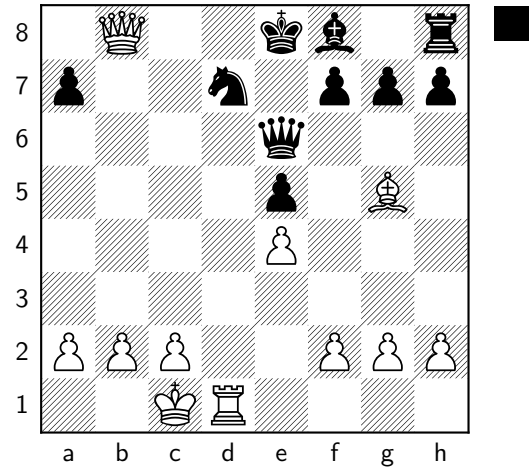
Deepseek Analysis:

1. Advantage Assessment: Black is better due to superior piece activity and pawn structure. The rook on b2 is very active, and the bishop on e6 controls key squares. White's pawns are fixed on the same color as their remaining bishop (g3/g4), limiting its scope.

2. Plan: Best move is ...Rb1+, forcing the white king to h2 (Kh2 avoids losing the rook). Then, push the passed pawn with ...g3+, opening lines for the rook and bishop. The idea is to create a mating net or win material by exploiting the pinned f2 rook.

3. Expected Outcome: Black has strong winning chances. The connected passed pawns (e5, g4) and active pieces should convert, though precise play is needed due to White's defensive resources (e.g., Rf3 to block). If Black plays accurately, the position is winning.

Position 3



FEN: `1Q2kb1r/p2n1ppp/4q3/4p1B1/4P3/8/PPP2PPP/2KR4 b k - 1 16`

ShashGuru Analysis:

Advantage Assessment: White stands better due to their significant material and positional advantage.

Plan: The best move is d7-b8, attacking the White queen. This move is good because it aims to eliminate a powerful attacking piece, but it's too late to change the outcome.

Expected Outcome: Black will lose. Key weaknesses include the exposed Black king and lack of piece coordination. White's pieces are actively placed, and their pawn structure supports a decisive attack. Immediate tactical motifs will lead to a forced checkmate.

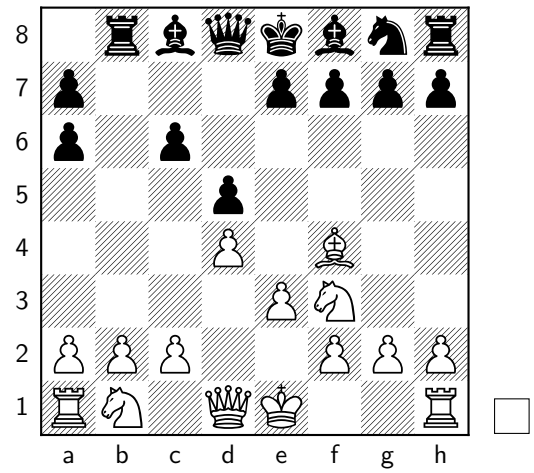
Deepseek Analysis:

1. Advantage Assessment: Black stands better due to superior piece activity and central control. The queen on e6 is well-placed, and the knight on d7 supports e5. White's bishop on g5 is active but lacks targets, and the queen on b8 is awkwardly placed. Material is equal, but Black's structure is more flexible.

2. Plan: Best move: 16...f6!, kicking the bishop while reinforcing e5. Idea: Open the f-file for the rook with ...f5 next, or play ...Qf7 to connect rooks and increase pressure. Alternatively, ...Nb6-c4 can target weak squares.

3. Expected Outcome: Black has good winning chances due to better coordination and potential kingside play. White's position is passive, and the misplaced queen on b8 is a long-term weakness. If Black consolidates with ...Qf7 or ...O-O-O, the initiative should grow.

Position 4



FEN: `1rbqkbnr/p3pppp/p1p5/3p4/3P1B2/4PN2/PPP2PPP/RN1QK2R w KQk - 2 6`

ShashGuru Analysis:

Advantage Assessment: White stands better due to their significant material and positional advantage. The primary reason is their control of key squares and better piece coordination.

Plan: The move f4-b8 is good because it aims to exploit Black's weaknesses on the queenside. The plan involves developing the rest of White's pieces and potentially creating a strong pawn center, as indicated by the principal variation b8-g3, h2-h3, and c2-c3, which suggests preparing to develop other pieces and control key central squares.

Expected Outcome: With a 100.0% win probability, White is almost certain to win. Their decisive advantage, combined with Black's passive position, makes it likely that White will successfully convert their advantage into a checkmate or force a resignation.

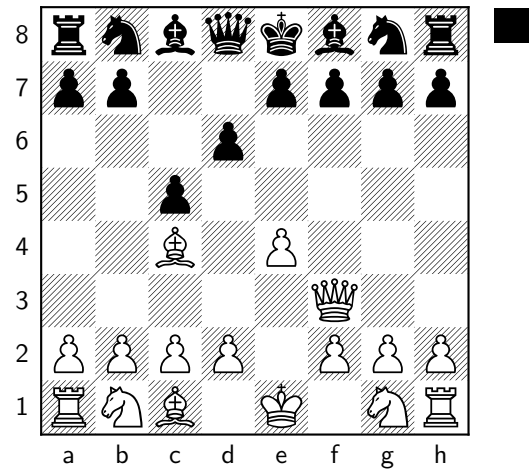
Deepseek Analysis:

Advantage Assessment: White stands slightly better due to superior piece activity and control of the center (e.g., Bf4 eyeing Black's weakened queenside, e3 knight supporting d4). Black's pawn structure is solid but passive, with the c6-pawn blocking the b7-bishop.

Plan: Best move: O-O, completing development and preparing to open the center with e4 or c4. Idea: Exploit Black's lagging development (queenside pieces inactive) by increasing pressure on d5 or targeting the c6-pawn (e.g., Qb3 or Rc1).

Expected Outcome: White has good chances to convert the space advantage into a lasting initiative, especially if Black fails to challenge the center. Black's position is defensible but requires precise play (e.g., ...Nf6, ...Be7). White's winning chances are 55-60% with accurate play.

Position 5



FEN: `rnbqkbnr/pp2pppp/3p4/2p5/2B1P3/5Q2/PPPP1PPP/RNB1K1NR b KQkq - 1 3`

ShashGuru Analysis:

Advantage Assessment: Black stands better due to their initiative and active pieces, putting pressure on White's position.

Plan: The move g8f6 is good as it develops the knight, prepares to control the center, and potentially attacks the White bishop on c4. The plan involves developing other pieces and applying pressure on the queenside with...b8c6,...a7a6, and...c6a5, targeting White's position.

Expected Outcome: With a win probability of 11.0% and a draw probability of 87.0%, the outcome is likely a draw, as Black's initiative doesn't translate into a significant material advantage, and White's position remains solid.

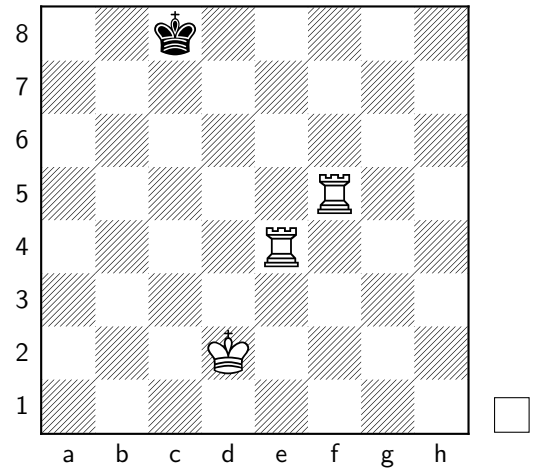
Deepseek Analysis:

Advantage Assessment: White stands slightly better due to superior piece activity (Bc4, Qf3 targeting f7) and central control (e4 pawn). Black's structure is solid but slightly passive, with the c5 pawn blocking the b7 bishop. No material imbalance.

Plan for Black: Best move is Nf6, developing the knight and preparing to challenge White's center (e.g., ...d5 or ...Bg4). Alternatively, ...Bg4 pins the queen, but Nf6 is more flexible. Idea: Complete development (O-O, Re8) and counter in the center.

Expected Outcome: Balanced but dynamic. White's initiative may lead to a small edge if Black fails to challenge the center. Black has solid chances with accurate play. Winning chances: 55% White (slight pressure), 45% Black (solid counterplay).

Position 6



FEN: `2k5/8/8/5R2/4R3/8/3K4/8 w - - 0 1`

ShashGuru Analysis:

Advantage Assessment: White stands better due to a significant material advantage and direct attacking possibilities against the Black king.

Plan: The best move, e4e7, is good because it directly attacks the Black king and prepares for a quick checkmate. The plan involves promoting the rook to a stronger piece (queen) via the principal variation f5f7, f7f8, putting immense pressure on the Black king.

Expected Outcome: With a 100.0% win probability, White is virtually guaranteed to win, given the decisive material advantage and the direct attacking lines available, leading to a swift checkmate.

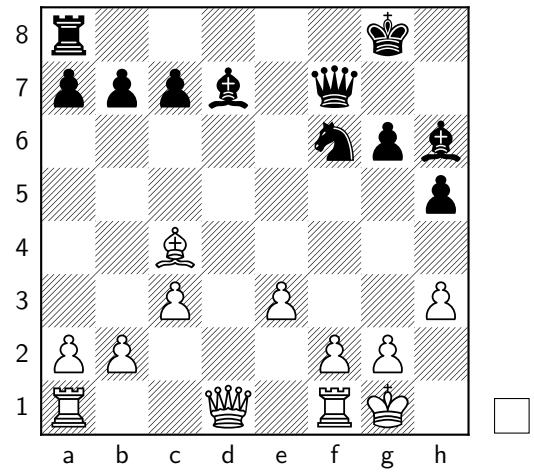
Deepseek Analysis:

1. Advantage Assessment: White is winning due to overwhelming activity (two rooks dominating the 4th/5th ranks) vs. Black's bare king and trapped knight on h8. Material is equal, but Black's pieces are paralyzed.

2. Plan: Best move: Re5! (triangulation to lose a tempo and force Black's king away, e.g., 1...Kg7 2 Rf7+ Kg8 3 Re8#). Alternatively, Rf8 (1. Rf8 Kxf8 2. Re8#) delivers immediate mate.

3. Expected Outcome: White forces mate in 2-3 moves. Black's knight never enters the game. A trivial technical win with precise rook coordination.

Position 7



FEN: `r5k1/pppb1q2/5npb/7p/2B5/2P1P2P/PP3PP1/R2Q1RK1 w - - 0 17`

ShashGuru Analysis:

Advantage Assessment: White stands better due to their significant material and positional advantage. The primary reason is their ability to launch a decisive attack on Black's position, exploiting weaknesses and lack of coordination.

Plan: The move c4-f7 is good because it directly attacks Black's queen and prepares to develop other pieces. The plan involves gaining control of key squares, specifically targeting Black's position with the principal variation d1-b3, b3-b7, and a1-d1, aiming to exploit Black's weaknesses and deliver a checkmate.

Expected Outcome: With a 100.0% win probability, White is expected to win the game, as their advantage is nearly insurmountable. The decisive attack and Black's lack of counterplay will likely lead to a swift victory for White.

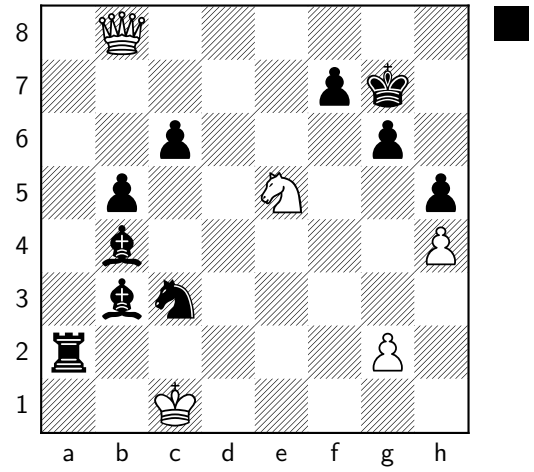
Deepseek Analysis:

Advantage Assessment: White stands better due to superior piece activity (centralized bishop on c4, rook on f1) and safer king. Black has a passive bishop on d7 and weak pawns (a7, d6).

Plan: Best move is Qd2, preparing to double rooks on the d-file to target d6. Alternatively, Rad1 directly increases pressure. The idea is to exploit Black's weak pawns and open lines for White's active pieces.

Expected Outcome: White has good winning chances by gradually increasing pressure on d6 and potentially breaking through on the queenside (e.g., b4-b5). Black's counterplay is limited due to passive pieces and structural weaknesses. White is slightly better, but accurate play is needed.

Position 8



FEN: `1Q6/5pk1/2p3p1/1p2N2p/1b5P/1bn5/r5P1/2K5 b - - 15 41`

ShashGuru Analysis:

Advantage Assessment: Black stands better due to a decisive material and positional advantage, with multiple pieces actively contributing to the attack.

Plan: The best move is b4a3, aiming for immediate mate. This move is good because it exploits White's weakened position, particularly the exposed king. The plan involves directly attacking the White king with the bishop.

Expected Outcome: With a 100.0% win probability, Black is assured of victory. The current position allows for a straightforward checkmate, leaving White with no viable defense.

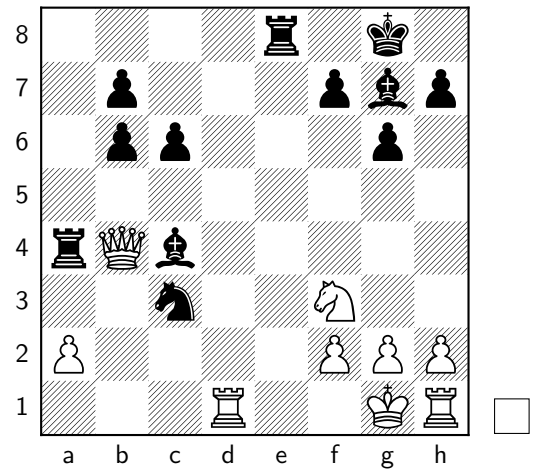
Deepseek Analysis:

Advantage Assessment: Black is better due to material (extra pawn) and superior piece activity. White's king is exposed, and Black's pieces (Rb2, Bb3, Nc3) dominate the position, controlling key squares and threatening further invasion.

Plan: Best move is Rb1, forcing the white king to c2 (1...Rb1 2.Kc2). Then, Na4 (or Nd5) to target weak pawns or prepare ...Bc4, increasing pressure. Idea: Tie White down to defense and gradually convert the material advantage.

Expected Outcome: Black should win with precise play. White's loose structure and passive pieces offer little counterplay. Black can slowly advance the kingside pawns (e.g., ...g5-g4) or coordinate pieces for a decisive attack.

Position 9



FEN: `4r1k1/1p3pbp/1pp3p1/8/rQb5/2n2N2/P4PPP/3R2KR w - - 2 25`

ShashGuru Analysis:

Advantage Assessment: Black stands better due to their strong pawn structure and piece activity, particularly the influence of their bishops and rooks.

Plan: The move b4d6 is good as it aims to challenge Black's control and prepares to potentially attack the black position. The plan involves pushing pawns (h2h4) to create weaknesses and using the queen to exert pressure.

Expected Outcome: With a 100% loss probability, White's chances are bleak. Black's superior position and coordination will likely lead to a decisive win, exploiting White's weaknesses and lack of counterplay.

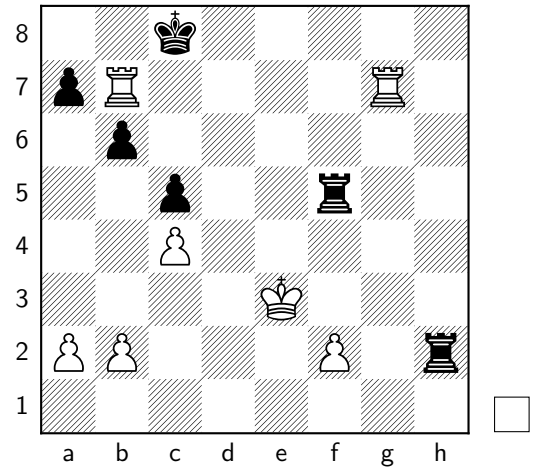
Deepseek Analysis:

Advantage Assessment: White stands better due to superior piece activity and Black's weak pawn structure (isolated a6 pawn, weak c6 pawn). The knight on c3 is loose, and White's queen and rook dominate the open files. Material is equal, but White's pieces are more coordinated.

Plan: Best move is Qxc4, winning the pawn due to the pin on the c-file (if ...bxc4, Rd8+ wins the bishop). Follow up with Rd6, increasing pressure on Black's weak pawns (c6, a6) and preparing to double rooks.

Expected Outcome: White has strong winning chances due to active pieces and Black's structural weaknesses. Black's king is exposed, and White can gradually convert the extra pawn or create a mating attack. Black's counterplay is limited.

Position 10



FEN: `2k5/pR4R1/1p6/2p2r2/2P5/4K3/PP3P1r/8 w - - 1 43`

ShashGuru Analysis:

Advantage Assessment: White stands decisively better due to their active rooks, control of key squares, and a strong pawn structure. The primary reason is White's rook activity and control of the f-file, which puts pressure on Black's position.

Plan: The best move is indeed b7c7 (eval: 466 context points). This move is good because it supports the potential rook exchange on the b-file, opens lines for the rook on g7 to potentially participate in the endgame, and strengthens White's pawn structure.

The plan involves:

Exchanging rooks on the b-file to reduce Black's piece activity and create a passed pawn. Utilizing the rook on g7 to potentially participate in the endgame, either by supporting a passed pawn or attacking Black's position.

Expected Outcome: White has a 100.0% winning probability, with 0.0% draw and loss probability. The decisive advantage is due to White's superior piece activity, pawn structure, and control of key squares, which will likely lead to a winning outcome.

Deepseek Analysis:

Advantage Assessment: White is better due to superior piece activity and material equality (both sides have a rook and pawns, but White's rooks dominate the 7th rank). Black's king is exposed, and the pawn on c5 is weak.

Plan: Best move is Rxb6, eliminating Black's pawn and opening the a-file for the other rook. Plan:

Use the active rooks to target Black's weak pawns (c5, b2) and coordinate with the king (e.g., Kd3-c4).

Push passed pawns (a3, c5) while keeping Black's king trapped.

Expected Outcome: White should win with precise play, as Black's pawns are vulnerable and the king cannot assist. Black's counterplay (e.g., Rxf2) is insufficient due to White's active rooks and king.

Appendix B

Chess Terminology

Centipawns

A unit of evaluation used by chess engines to measure the strength of a position. One centipawn equals one-hundredth of a pawn. For example, a score of +50 indicates a small advantage for White.

FEN (Forsyth–Edwards Notation)

A standard notation for describing a specific chess position in a single line of text. It encodes piece placement, active color, castling availability, en passant targets, halfmove clock, and fullmove number.

PGN (Portable Game Notation)

A plain-text format for recording entire chess games, including moves, player names, event information, and annotations.

SAN (Standard Algebraic Notation)

The common format for writing individual chess moves using abbreviated piece names and coordinates, such as Nf3 (knight to f3) or Qxe5 (queen captures on e5).

Zugzwang

A situation in which any legal move a player makes will worsen their position. Often seen in endgames, it forces the player to move when they would prefer to pass.