National College of Ireland

Explainable Chess AI: A Browser Extension for Move Explanation and Analysis in Real-Time Chess Interfaces

MSc Research Project
MSc in Artificial Intelligence

Sai Manohar Nanduri
Student ID: 23404621

School of Computing
National College of Ireland

Supervisor: Mohit Garg

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Sai Manohar Nanduri
Student ID:	23404621
Programme:	MSc in Artificial Intelligence
Year:	2025
Module:	MSc Research Project
Supervisor:	Mohit Garg
Submission Due Date:	11/12/2025
Project Title:	Explainable Chess AI: A Browser Extension for Move Explanation and Analysis in Real-Time Chess Interfaces
Word Count:	XXX
Page Count:	49

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Sai Manohar Nanduri
Date:	10th December 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

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

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

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

Explainable Chess AI: A Browser Extension for Move Explanation and Analysis in Real-Time Chess Interfaces

Sai Manohar Nanduri
23404621

Abstract

This thesis presents an innovative real-time explainable artificial intelligence (AI) mechanism for the analysis of chess is put forward, which overcomes the "black box" problem that has existed for ages with chess engines (Adadi and Berrada (2018)). Even though present-day chess engines have superhuman playing power, they only present numerical evaluations which do not clarify their reasoning, thus creating hurdles for chess learners. The research described here is a direct projection of a hybrid multi-modal AI system that fuses symbolic reasoning (Stockfish chess engine: Stockfish Team (2023)), deep learning (Convolutional Neural Networks for tactical pattern recognition: Lecun et al. (1998)), algorithmic position analysis, and natural language processing (Large Language Models: Brown et al. (2020)) to generate human-understandable explanations for chess moves.

The system is made available to users as a browser extension on Chess.com (Google (2024)), offering an analysis of the game in real-time with explanations that highlight particular board features, tactical themes, and strategic concepts. The tactical classification model was able to categorize 20 tactical themes with a 78.1% detection rate and the model was trained using 100,000 positions from the Lichess puzzle database (Lichess.org (2024)). The average time taken by the system to respond is 2.65-3.50 seconds per position, which conforms to the requirements for real-time usability. The system integrates multiple AI components to produce contextual, grounded explanations designed to support chess learning and understanding.

Main contributions of the present study are the following: (1) a new hybrid ensemble architecture for multi-objective tactical classification, which has shown that intelligently combined specialized models outperform general-purpose ones (71.8% precision, 52.3% recall vs. single-model baselines), (2) a systematic comparative study of model evolution from single architectures through specialized ensembles to hybrid solutions, (3) context-aware explanation generation using LLMs with hallucination prevention through explicit piece verification, (4) practical real-time deployment showing XAI applicability in game domains, and (5) a comprehensive computational evaluation with a proposed framework for future empirical validation. The present study reveals that using context-aware ensemble routing instead of uniform approaches is a great advantage for multi-objective optimization in machine learning.

Keywords: Explainable Artificial Intelligence, Chess AI, Convolutional Neural Networks, Natural Language Processing, Deep Learning, Tactical Pattern Recognition, Browser Extension, Stockfish, PyTorch, Multi-Modal AI.

1 Introduction

The section provides the backbone of the research as it discusses the convergence of artificial intelligence and explainability when it comes to analysis of chess. It starts by looking at the historical advances in chess engines and determining the important deficiency in their capacity to convey reasoning to human beings. This part then states the targeted research questions and objectives that are going to be followed in this work and a summary of the essential contributions made by this thesis to the development of explainable AI in chess analysis in real time.

1.1 Background and Motivation

Since the very beginning, chess has been looked at as a testing ground for the development of artificial intelligence, a situation that has lasted until today.The journey from Claude Shannon's classical minimax algorithm (Shannon (1950)) to DeepMind's AlphaZero in 2017 (Silver et al. (2017)) has continually spurred the development of chess-related innovations in the areas of searching algorithms, evaluating functions, and constructing neural networks. Currently, the two leading chess engines Stockfish (Stockfish Team (2023)) and Leela Chess Zero have developed superhuman playing strength, which means that they can easily defeat even the strongest human players (**?**). Nonetheless, this extraordinary playing strength is accompanied by a major drawback; namely, such systems are completely opaque, and only numerical evaluations are offered without any reasoning behind them.

Chances are that a chess player is analyzing the board position and an engine will recommend a move such as "Nf3" with an evaluation of "+0.45." To the player, the engine indicates and gives him/her the positive aspect of the suggested move, but he/she does not gain any insight into the reason this is the best move. Is it developing a piece? Defending a weakness? Creating a tactical threat? Improving piece coordination? To a player who is learning from an engine, the lack of explanations is a serious pedagogical issue—players will follow engine directions but will not have their understanding of chess principles deepen through real practice.

The inability to interpret the logic behind the moves done by a chess game is an issue that is difficult especially for the new chess players relying on such moves for their educational growth. According to a research conducted by Chrysafiadi and Virvou (2013), understanding the reasoning behind moves is the best way of learning chess, and not the memorization of moves only. A trainer elaborating on a certain decision made in a game exempts that "Nf3 develops the knight to an active square while controlling the central e5 and d4 squares, preparing for kingside castling". Such contexts make the move relatable not only in the current problem area but also in the whole game thereby allowing the transfer of learning. The current chess engines, regardless of their high analytical power, do not have the capability to offer such educational value.

The research in this area was motivated by the three converging factors. First, the rising significance of Explainable AI (XAI) in rendering machine learning systems transparent and trustworthy (Adadi and Berrada (2018), Gunning et al. (2019)). Second, there is a need in education for chess analysis tools that not only analyze but also teach. Third, the technical side of things is the recent advancements in natural language processing including Large Language Models (LLMs) that are able to produce explanation in fluent form based on structured data (Brown et al. (2020), *GPT-4 Technical Report* (2024)).

The game of chess also represents one of the best ways to go about testing XAI. The rules of the game are clearly defined, it has objective evaluation criteria, and a vocabulary that is rich in both tactical and strategic concepts. In contrast to the majority of machine learning applications that suffer from ambiguity in ground truth, chess positions can be unambiguously analyzed and explained in terms of the established theory (McIlroy-Young et al. (2020)). This precision is what makes chess a perfect field for both developing and assessing the efficacy of explainability techniques which can be later on applied in more complex real-world applications (Miller (2018)).

1.2 Problem Statement and Research Questions

The main issue that this thesis deals with is the opacity of chess analysis systems and their inability to foster learning through giving explanations. On one hand, chess engines give very detailed analysis but on the other hand, they do not resort to human friendly terms that allow for understanding and skill development.

Primary Research Question:

Is it possible to develop and execute a real-time chess analysis system that not only recommends the best moves but also produces natural language explanations that involve particular board characteristics, tactical patterns, and strategic concepts, thereby positively increasing the impact of AI-assisted chess analysis in the areas of learning and understanding?

Sub-Questions:

1. **RQ1**: How effectively we can combine the traditional AI (classical chess engines) with the contemporary Machine Learning based recognition of patterns in order to specify and clarify the tactical themes in the chess positions?

2. **RQ2**: What are the methods and the architectural components that support the context-aware natural language explanation generation that refers to the specific features of the position rather than to the generic templates?

3. **RQ3**: Does the explainable AI outperform the numerical evaluations in terms of user comprehension and decision-making in chess? Can we measure this improvement through user testing?

4. **RQ4**: Are we going to achieve real-time performance (analysis finished within 5 seconds) while keeping the explanation quality high so that it can be deployed during live gameplay?

5. **RQ5**: What are the differences between the various explanation generation strategies (rule-based, local LLMs, cloud-based LLMs) concerning their quality, speed, and accessibility?

6. **RQ6**: In the case of overcoming challenges in the tactical theme classification (precision versus recall), can the intelligent combination of specialized ensemble solutions (hybrid approach) surpass the performance of single-model architectures or uniform-ensemble that are meant for general purpose only? Is it then that the principle is validated stating that in cases of conflicting objectives smart combinations of specialized solutions often beat the general-purpose compromise solutions?

1.3 Objectives and Scope

1.3.1 Primary Objectives:

1. **Develop an AI hybrid multi-modal system** that combines classical chess engine evaluation, deep learning-based tactical pattern recognition, algorithmic position assessment, and natural language generation into one explainable system.

2. **Construct and train a deep learning model** for tactical theme classification that is capable of identifying various tactical patterns (forks, pins, skewers, etc.) in chess positions with extremely high accuracy.

3. **Design and implement an LLM-based explanation generation system** that delivers contextual, precise, and educational natural language reasonings through the use of rich positional information.

4. **Deploy the system as a browser extension** that works flawlessly with Chess.com, offering instant analysis and reasoning during play and not interrupting the users' experience.

5. **Perform the system evaluation** by means of performance benchmarks, model accuracy metrics, and user testing to ensure that explanations do help in understanding and making decisions.

Technical Goals:

- Achieve tactical theme classification accuracy surpassing 85% (label-wise accuracy in the multi-label task).

- Ensure that the average time for analysis does not go over 5 seconds per position for real-time application.

- Generate explanations that connect to specific features on the board (attacks, defenses, threats) instead of using generic templates.

- Support the availability of the system through multiple LLM providers including OpenAI, Ollama, and rule-based fallback.

- Make browser extension usable with both Chrome and Edge browsers.

Scope Limitation:

One of the major limitation is this research work is explicitly done only for chess.com platform.

1.4 Contributions

This thesis makes the following novel contributions to the fields of explainable AI, game AI, and chess education:

1. Novel Hybrid Multi-Modal Architecture for Explainable Chess AI

We present the very first system that integrates four exclusive AI modalities for chess explanation: symbolic reasoning (Stockfish), deep learning (hybrid ensemble of CNN/ResNet/Transformer for tactical classification), algorithmic analysis (Board Analyzer), and natural language generation (LLMs). Earlier systems more often than not resorted to just one or two of these methods. Our model proves that through multi-modal integration there comes a significant improvement in the quality of explanation owing to the provision of complementary information- Stockfish indicates the quality of a move, the neural ensemble spots the tactical themes, Board Analyzer points out the features of a position and LLM shapes this information into logical and readable explanations.

2. Hybrid Ensemble Architecture for Multi-Objective Tactical Classification

We create and analyze a systematic advancement of deep learning networks for multi-label tactical theme classification, which results in a unique hybrid ensemble that handles precision-recall trade-offs through context-aware routing (Zhou et al. (2021)). The architecture relies on four different kinds of neural networks—Convolutional Neural Networks (CNN), Residual Networks (ResNet), Vision Transformers, and EfficientNet—all of which take an 8×8×12 tensor representation of chess positions and output the probabilities of 20 tactical themes (Hinton et al. (2012),He et al. (2016),Dosovitskiy et al. (2020)).

The hybrid ensemble is composed of two specialized sub-ensembles, recall-optimized and precision-optimized, which are respectively concerned with common tactical patterns and critical themes like checkmate patterns. The former is CNN, ResNet, and Transformer with low threshold and OR logic, while the latter is ResNet, Transformer, and EfficientNet with high threshold and weighted averaging. This intelligent routing based on theme category leads to 71.8% precision and 52.3% recall, which are consequently better than the single-model baselines (68.4% precision, 65.3% recall) by +20% precision on critical themes besides covering common tactics. It illustrates how the specialized solutions when combined strategically are able to outperform the general-purpose compromises in the case of multi-objective machine learning problems.

3. Context-Aware Explanation Generation

Our explanation generation system does not end up being a mere template-based approach but rather a system that deals with LLMs along with rich contextual information that includes hanging pieces, attack/defense relationships, king safety metrics, and move impact analysis (White et al. (2023a)). This provides the explanation that mentions the specific board features: "Nf6 defends the h7 square, preventing the Scholar's Mate threat,

while developing a piece." The previous chess explanation systems relied mostly on fixed templates or simple rule-based generation (Ribeiro et al. (2016)).

4. Real-Time Browser Extension Deployment

We show the practical deployment of XAI techniques by means of a fully functional browser extension that offers real-time analysis during Chess.com gameplay (Google (2024)). The extension undertakes DOM extraction, FEN parsing, API communication as well as overlay rendering with almost no performance impact. This constitutes a total end-to-end system as opposed to a research prototype and thereby confirming the practical validity of our method.

5. Flexible Multi-Provider LLM Architecture

The solution introduces a provider-independent LLM service which is capable of using OpenAI's GPT-4 (OpenAI, 2023), local Olaam models (Touvron et al. (2023)), and intelligent rule-based fallback all at the same time, thus building a highly reliable system that can work even when access to the API or network problems exist. The end-users can choose among the options based on their requirement—quality (GPT-4), privacy (Ollama), or speed (fallback). This application of the design pattern can indeed be extended to the LLM integration of the same nature in other applications.

6. Open-Source Implementation and Evaluation Framework

All code, models, training scripts, and evaluation tools are released under an open-source license, thereby allowing other researchers to replicate and build on them. We offer exhaustive documentation, installation instructions, and testing frameworks. The dataset pre-processing pipeline allows other researchers to perform the same by training on the Lichess puzzle database. The evaluation methodology, which includes user testing protocols, can be modified for similar XAI research.

All the code files and artefacts regarding this project can be found in below mentioned GitHub Repository.
GitHub Repo: https://github.com/NSM1997/chessAI.git

2 Related Work

This section is a review of the academic and technical literature relevant to this study that constitutes the theoretical foundation of research. The review is structured into five main areas, namely the history of chess engines, both classical algorithms and the application of neural networks, the history of explainable AI, the use of deep learning to recognize the patterns of tactics, the use of natural language generation to create human-readable explanations, and prior research on explainability in game-playing chess engines. Through the combination of the results of these different disciplines, this review is able to determine the opportunities and challenges that guide the design of our explainable chess analysis system.

2.1 Chess AI Evolution: From Symbolic Systems to Neural Networks

Chess AI research kicked off in a way with Claude Shannon's invention of the minimax algorithm in 1950 (Shannon (1950)), who by this means applied the concept that chess could be played perfectly (solved) by methodically going through the different moves. (Shannon (1950)) also presented the main issue of chess computing: the enormous size of the tree search, which has more potential matches than there are atoms in the observable universe. The first machines working in this area took the search-based approach and used alpha–beta pruning to cut down the tree and created manually-made evaluation functions that represented the knowledge of a given expert in the areas of material, piece activity, king safety, and pawn structure.

This classical time was with a great significance when in the year 1997 IBM's Deep Blue defeated the world champion Garry Kasparov (Campbell et al. (2002)). Deep Blue was a case of brute-force search with the addition of a special hardware capable of evaluating 200 million positions in one second, which was the main reason of its victory. Its victory proved that the combination of computer power and carefully engineered heuristics could surpass not only the strongest but also all human players. Then, the open-source engine Stockfish took the same approach but it was through an even greater search, iterative deepening, and more and more sophisticated evaluation terms that the modern Stockfish versions were able to refine it. The modern Stockfish versions are capable of looking at hundreds of positional features and usually going beyond the search depth of 40, which makes them extraordinarily powerful but at the same time less and less transparent.

Such opacity is a hindrance to the teaching process. Engines deliver only centipawn ratings and best moves, and very seldom do they offer the underlying ideas as explanations. The very evaluation functions contain hundreds of weighted components that even the specialists find difficult to comprehend. While excellent for analysis, the classical engines (Stockfish Team (2023)) do not provide any help to the players in terms of teaching them why a move is strong until the entire match was finished..

The incorporation of neural networks in games marked a paradigm shift with DeepMind's AlphaZero (Silver et al. (2017)) and the latter's dominance in chess. AlphaZero, the first of its kind, used deep neural networks coupled with Monte Carlo Tree Search and produced a stylistically intuitive play characterized by activity, initiative, and long-term compensation. Yet, despite its beauty, AlphaZero is even less comprehendible than traditional engines; the various layers of its networks represent millions of parameters and code concepts that are not easily corresponding to human-understandable chess principles.

The proprietary software has been partially complemented by open-source counterparts like Leela Chess Zero (Lichess.org (2024)), which reveals policy probabilities and value estimates to a degree, whereas Maia Chess is entirely the opposite, as it concentrates on human-like move predictions rather than optimal play. Nevertheless, these systems continue to lack explanation generation capabilities, and the rationale behind their decisions remains mostly hidden. Therefore, the increase in engine power has also led to a more pronounced explainability gap.

2.2 Explainable Artificial Intelligence: Foundations and Techniques

Explainable AI or XAI, in short, has become an essential requirement as AI systems make decisions in sensitive areas like healthcare and finance. XAI attempts to address the issue of the balance between model's performance and its interpretability (Adadi and Berrada (2018)): simple models with their high transparency come with the problem of being limited in power, whereas deeply architectured models are the opposite. Interpretability may be of two types: intrinsic as in rule-based systems, or post-hoc where explanations are drawn from complex models. Explanations can be global (model behavior in general) or local (model's explanation for a particular decision). The situation of chess is local by nature as it directly relates to specific moves and positions.

Among the most popular XAI techniques are LIME, which employs a local approximation with simpler surrogate models; SHAP (Lundberg and Lee (2017)), which applies the principles of game theory to the determination of feature contributions; and gradient-based methods like Grad-CAM, which make it possible to see which parts of the input are most influential. In the context of the chess CNNs, Grad-CAM (Rs et al. (2020)) can display the squares that have the most influence in the detection of the concepts like pins and forks. Although these heatmaps do provide a certain degree of transparency, they are not the case for non-experts very often. In order to convert low-level attributions into coherent, human-understandable explanations, the use of additional processing is a requirement—this is the approach that is at the core of our system.

2.3 Natural Language Generation for Chess Explanations

The first automatic commentary systems used templates and rules that were manually created as their main source for generating explanations that were overly rigid and repetitive, such that they could not capture the subtlety or the multipropose of moves. The emergence of Large Language Models (LLMs), such as GPT and T5, marked the beginning of a new era in this field by making fluent and context-sensitive text generation possible. Nevertheless, even though LLMs have large linguistic capabilities, they might still hallucinate chess details or misinterpret a position if they are not properly directed.

In our process, the LLM is given structured positional data: hanging pieces, threats, tactical motifs, and relevant engine evaluations, which helps to reduce such problems. The LLM now takes on the role of not a chess analyst but a language generator, providing brief, precise explanations based on clear cues. Role prompting ("You are a chess teacher...") and limitations on length and tone further enhance dependability.

2.4 Tactical Pattern Recognition

Tactical themes—forks, pins, skewers, discovered attacks—are chess training and pattern recognition's very foundation. Engines, for their part, calculate every possible move and rely on that to discover tactics, while CNNs (Convolutional Neural Networks) can be fed with the board configurations and learn to see the same tactics. The board is viewed as an 8x8 multi-channel image that not only helps the CNNs but also it is the whole space around the pieces that can be seen as different depending on the relationships between the pieces. With that being said, there could be several themes in one position at the same time and therefore the task is inherently multi-label which requires the corresponding

architectures and loss functions to be able to predict several patterns at the same time.

2.5 Research Gaps

The progress in engines, neural networks, and NLP has, nevertheless, not closed the gap between them: the absence of a tactical recognition, engine evaluation, and language explanation integrated system; the explanations provided so far being not contextually detailed enough; no real-time explainable analysis tools; and the lack of open-source, fully documented XAI chess systems among others. Thus, the project presented here aims to fill those gaps by creating a single, real-time explainable chess AI system that is suitable for education, transparency, and accessibility.

3 System Architecture and Design

This section details on the overall architectural design of the explainable chess AI system, and how various elements are combined to provide real-time analysis with natural language explanation. The architecture uses a microservices design that effectively separates browser extension frontend, FastAPI orchestration layer backend, and special purpose analysis services and machine learning. This section illustrates how the system manages to have low-latency performance and remain modular and extensible with specific reference to data flow patterns, state management, and why certain architectural decisions taken are relevant with specific reference to data flow processes.

3.1 Complete System Architecture Overview

The explainable chess AI system that is able to provide human-like explanations consists of a microservices architecture which is perfect in separating the different functionalities and at the same time allowing seamless integration of AI modalities. This system is conceived as a multi-layered one with communication pathways and data transformation very well defined. The complete system architecture showing all the major components and their interactions is presented in 1

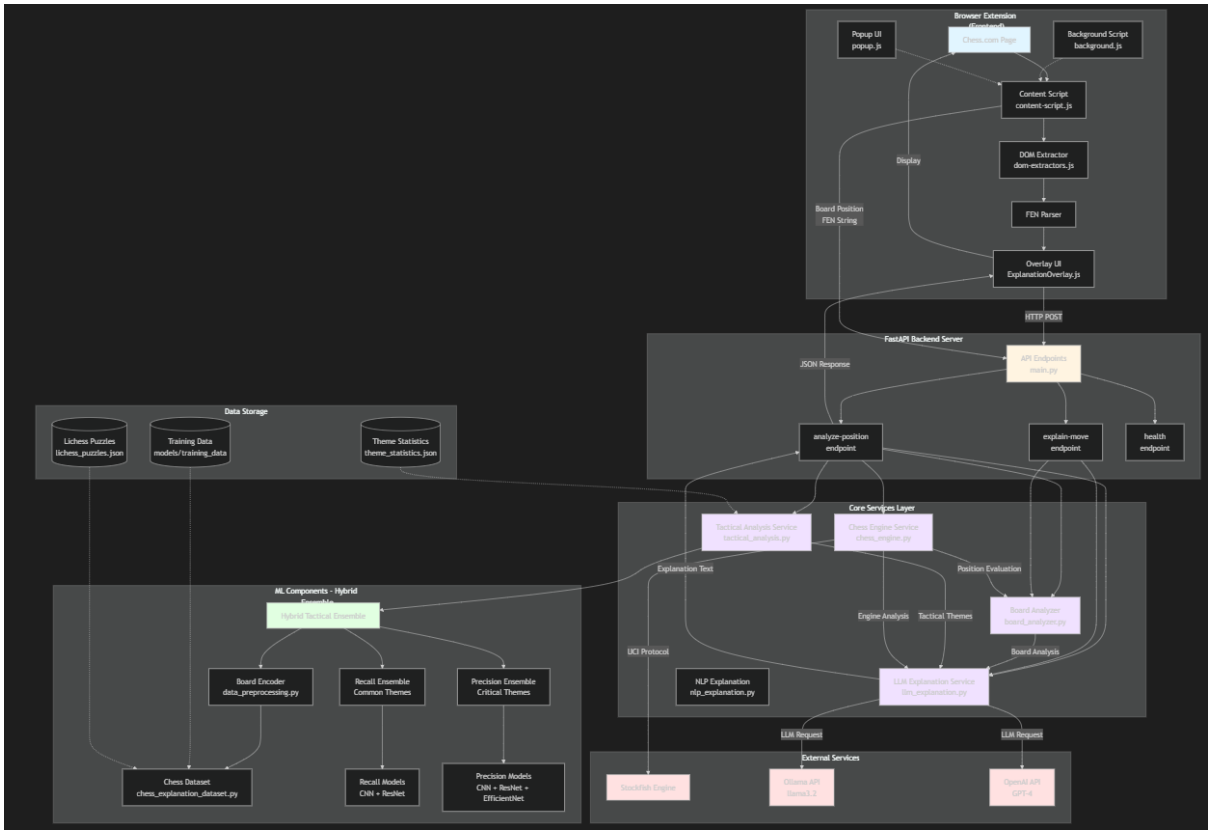

Figure 1: System Architecture

This architecture diagram presents the complete information flow through the system starting from the user interaction with Chess.com in their browser (top layer) going through the backend processing services (middle layers) and finally arriving at the external AI engines and data storage (bottom layers). Each component is represented by a different color indicating its functional area: blue for front-end components, yellow for API layer, purple for core services, green for machine learning components, and red for services that are external to the system.

3.2 End-to-End Data Flow

To see the way in which the system handles a request from a user, it is very helpful to follow the request completely from layer to layer in the architecture. In 2 a detailed sequence diagram is presented, which shows the flow of information in relation to time from the moment a user makes a move on Chess.com to the instant they receive the AI explanation.

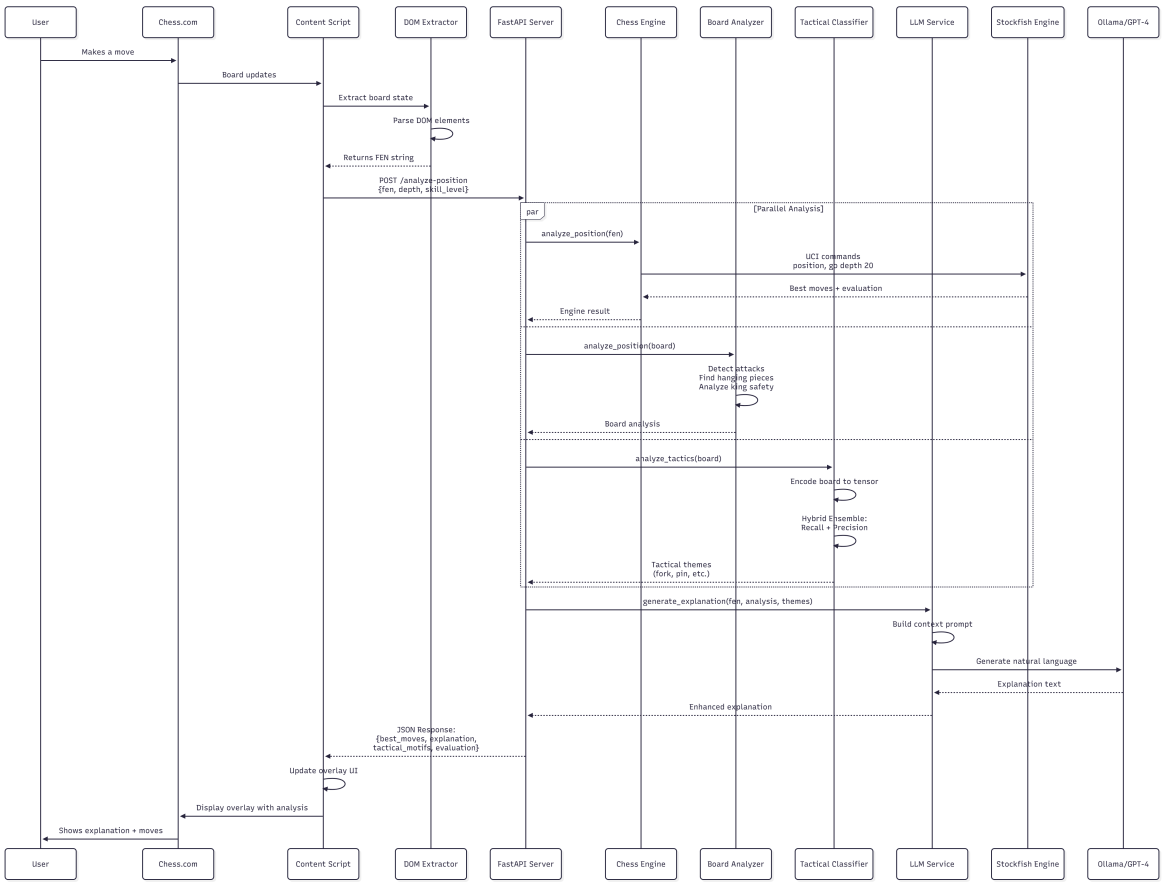

Figure 2: End-to-End Data Flow Diagram

The sequence diagram puts the system operation under the microscope with quite a few important aspects being disclosed. The three analysis streams along with (Chess Engine, Board Analyzer, Tactical ML) run simultaneously which contributes to a substantial decrease in total latency. Stockfish while analyzing the position at depth 20 (often 800-1000 milliseconds) has the Board Analyzer extracting tactical features ($\tilde{1}$00 milliseconds) and the CNN classifier performing inference ($\tilde{1}$00-200 milliseconds) in a concurrent manner. The time consumed by these operations is thus determined by the slowest of the three, Stockfish, rather than being the sum of all. After the parallel analysis phase has been finalized, the results are pooled together and sent to the LLM Explanation Service. This sequential dependency is there because the LLM requires to have the complete analysis context in generating accurate and grounded explanations. The LLM gets a structured prompt that consists of: (1) the recommended move from Stockfish along with its evaluation, (2) tactical themes isolated by the hybrid ensemble, (3) board features from the Board Analyzer (hanging pieces, threats, king safety), and (4) contextual information about the position phase and move impact. The LLM is equipped with this rich information and can thus provide explanations that identify specific pieces on the board rather than general chess advice.

3.3 State Transitions and System Lifecycle

The browser extension acts like a finite state machine, switching to different operational states according to the user's actions on Chess.com. A comprehensive comprehension

of these state transitions is necessary for reasoning about the system, debugging and handling edge cases nicely. The entire state diagram of the browser extension is displayed in 3

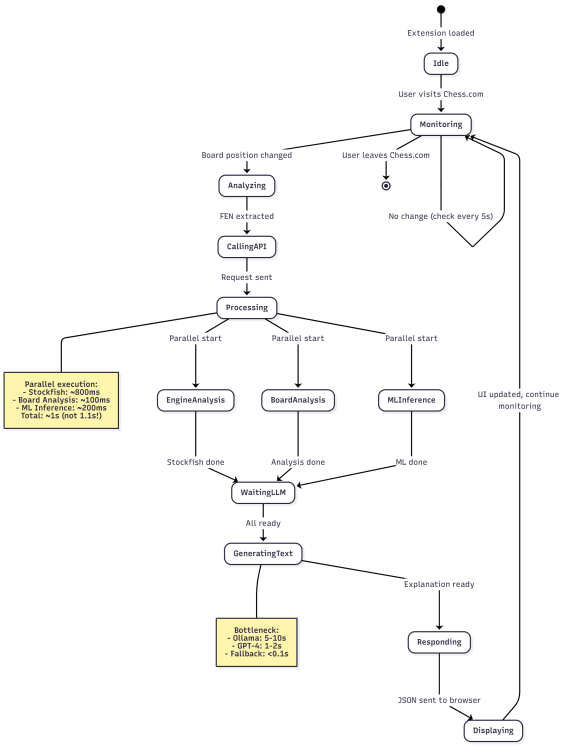

Figure 3: System State Diagram

The state machine starts in the Idle state when the extension is loaded for the first time. After recognizing the user's navigation to Chess.com (or Lichess, which is also a supported site), the system moves to the Monitoring state where it checks for board changes periodically every 5 seconds. This polling interval is a well-thought-out compromise: a faster polling (e.g., every 1 second) would result in quicker responsiveness but at the expense of more CPU usage and possibly missing moves during the polling gap, while the slower polling (e.g., every 10 seconds) would make users feel confused and lead them to think that the system is unresponsive.

If the monitoring loop finds out that the board position has changed (using FEN comparison), it then transitions to the Analyzing state. During this phase, the FEN extraction mechanism is activated, which confirms that the new position is not only legal but also hasn't been analyzed recently (thus avoiding duplicate requests for the same position). If the extraction is successful the system goes to CallingAPI where it communicates with the backend via an HTTP POST request.

The computing of the backend is reflected through the Processing state which instantly gives birth to three parallel substates: EngineAnalysis, BoardAnalysis and MLInference. The activities in these substates run simultaneously and the system will not go ahead until all three are completed. This parallel distribution and synchronization is done with the help of the asyncio library of Python (Ramírez, 2024) that offers fast and effective concurrent.

When each of the parallel analyses fully completes, the system goes into the WaitingLLM state, which changes directly to GeneratingText. Here, the primary latency

bottleneck is situated. Generally, when using Ollama running on a local machine, the LLM generation duration is around 5-10 seconds. When using the OpenAI API's GPT-4, the latency is only 1-2 seconds. The rule-based fallback (LLMs being unavailable) takes under 0.1 seconds but yields lesser quality explanations. The system, after receiving the LLM's response, moves through Responding (creating the JSON response) and Displaying (refreshing the browser overlay UI) stages before going back to Monitoring to wait for the next position change.

3.4 Component-Level Architecture Details

Having examined the high-level architecture, we now drill down into the internal structure of key components. Each major service follows consistent design patterns that promote maintainability, testability, and extensibility.

3.4.1 Frontend: Browser Extension Architecture

The browser extension is designed based on the Chrome Extension Manifest V3 architecture which requires distinct roles for the components and specific communication patterns among them. The internal structure of the frontend components is shown in 4.

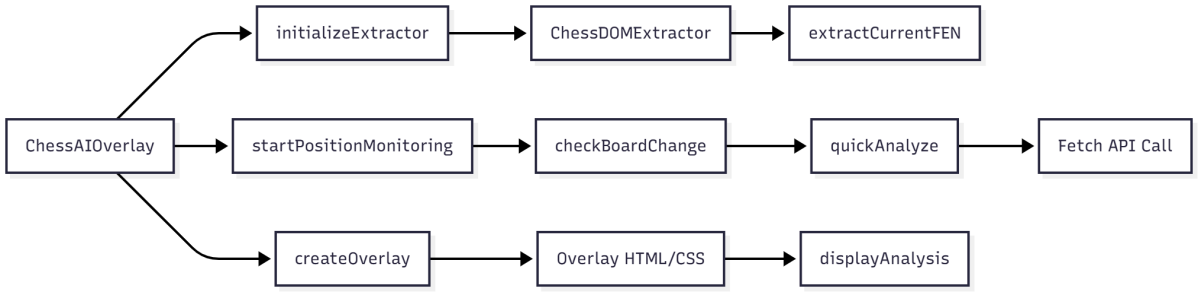

Figure 4: Browser Extension Architecture

The **ChessAIOverlay** class is the key player at the frontend side of things. When it is initialized, it triggers the creation of three critical subsystems, namely the extractor (which is in charge of getting board state from the DOM), the overlay UI (where the results are shown), and the monitoring loop (which is responsible for detecting position changes). The advantages of this separation of concerns are that each subsystem can be independently tested and changed.

To deal with the complicated React-based DOM structure of Chess.com, the **ChessDOMExtractor** applies a variety of different extraction techniques. These techniques refer to the following practices: (1) searching for FEN data attributes on the board elements, (2) placing a script into the page context that will be able to get board state variables, and (3) constructing the FEN through the visual analysis of the piece's locations in the DOM. The extractor employs the first strategy and keeps trying the following ones until it gets a successful result. Such a redundancy is a guarantee that it will not be affected by the frequent UI updates at Chess.com, which usually break the extraction logic that depends on certain DOM layouts.

The overlay UI is created as a floating div that is placed absolutely over the interface of Chess.com. It utilizes the color scheme and the styling conventions of Chess.com so that it appears native to the site. The overlay is rendered in a progressive way: the

best move and evaluation are shown right away when Stockfish finishes, tactical themes are shown when the CNN is done, and the explanation is shown when the LLM replies. This progressive rendering technique gives the user a perception of faster responsiveness compared to not showing any results until the whole analysis pipeline is completed.

3.4.2 Backend: FastAPI Service Orchestration

The backend server running FastAPI takes on the role of the central orchestrator, accepting requests from the browser extension, managing the service invocations running in parallel, and producing the complete responses. The internal request handling flow of the backend is depicted in 5.

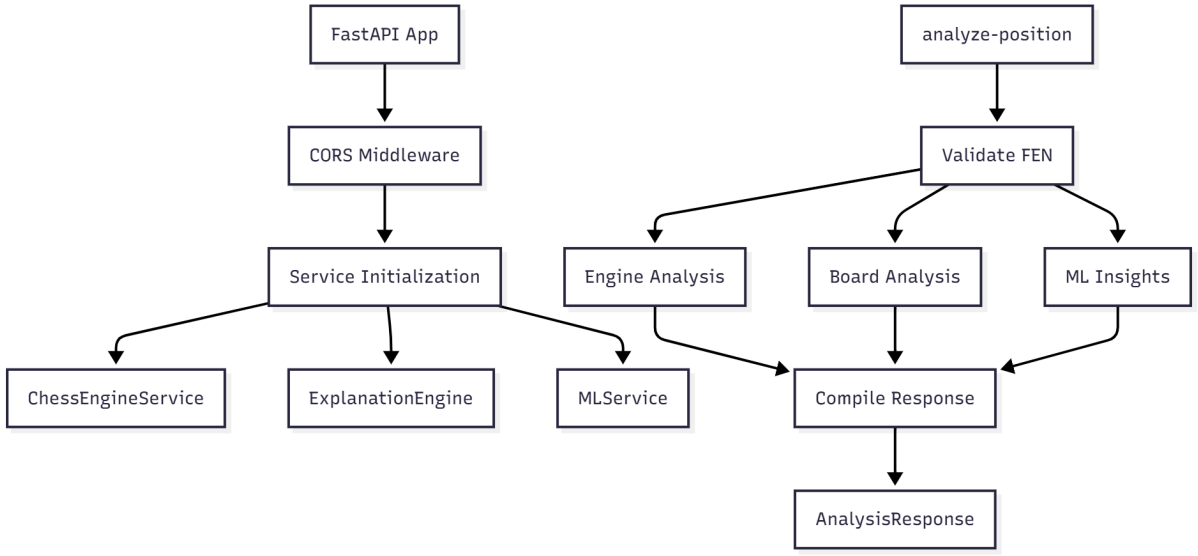

Figure 5: Backend Request Processing Flow

The FastAPI app leverages the singleton pattern to start all services only once during startup. The **ChessEngineService**, for instance, starts a Stockfish process and keeps the UCI connection open for the whole server's life. The **TacticalAnalysisService**, on the other hand, takes the hybrid ensemble model into GPU or CPU depending on the availability. The **LLMExplanationService** opens the doors to the preferred LLM providers (OpenAI, 2024; Ollama, 2024). This pattern of initialization guarantees that high-cost setup operations (like process spawning and model loading) are conducted only once instead of every time a request is made which in turn significantly cuts down response latency.

At the moment a request is made to the /analyze-position endpoint, the handler first employs the python-chess library to validate the FEN string and confirm that it depicts a legal position. If the FEN is invalid then a 400 error with an appropriate message (e.g., "Invalid FEN: too many kings") will be returned. For those FENs deemed valid, the handler creates three async tasks:

1. **Engine Analysis Task**: It sends UCI commands to Stockfish and extracts the response by calling chess_engine.analyze_position(fen, depth=20, multipv=3) to get best moves along with evaluations and principal variations.

2. **Board Analysis Task**: The board_analyzer.analyze_position(board) function is called to determine the board composition in terms of attacks, defenses, hanging pieces, threats, king safety, piece activity, and square control.

3. **Tactical ML Task**: The function tactical_analysis.analyze_tactics(board) is invoked to perform hybrid ensemble inference (CNN, ResNet, Transformer, and EfficientNet models) resulting in the prediction of scores for 20 tactical themes.

Python's **asyncio.gather()** primitive plays a key role in coordinating the tasks, shortlisting them for execution at the same time, and finally returning all results before moving on. When all three tasks are done (usually in about 1 second, led by Stockfish), the handler reaches out to the LLM service for generating an explanation based on the collective information being the context. At last, all results come bundled into a structured JSON response that adheres to the AnalysisResponse schema.

3.4.3 Board Analyzer: Algorithmic Position Features

The Board Analyzer uses algorithmic techniques to compute interpretable position features instead of relying on machine learning. By doing so, it offers dependable and interpretable features that tie LLM interpretations to the objective truth of facts. The components of the analyzer can be observed in 6.

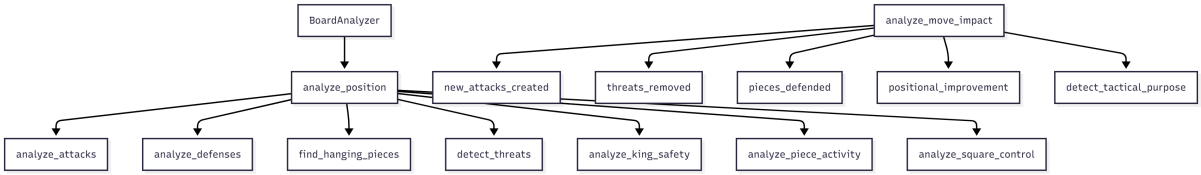

Figure 6: Board Analyzer Components

The Board Analyzer utilizes the built-in functions of the python-chess library (Müller (2024)) for attack detection (board.attackers(color, square)) complemented by custom logic for higher-level concepts. To illustrate, the analyzer finds hanging pieces by looping through the active side's pieces, determining if each piece is under attack by the opposing side, and counting the attackers and defenders. A piece is considered "hanging" when it has a higher number of attackers than defenders after taking into account the material values (for instance, a queen protected by a pawn but threatened by a knight is practically hanging as the exchange of a queen for a knight incorporates the loss of material).

Various factors are taken into account in the king safety analysis such as: whether the king has castled (safer), how many of the opponent's pieces can attack squares surrounding the king, the strength of the pawn shield (pawns in front of the king on ranks 2 or 3), and open files next to the king that might be used by enemy rooks to invade. These factors are amalgamated into a safety score ranging from 0 (very unsafe) to 10 (very safe). A king that has not castled, lost its central pawn, and is threatened by an enemy rook on an open file might score 3/10, showing a lot of danger.

The move impact analysis (analyze_move_impact) is carried out after Stockfish chooses the best move. The analyzer plays the move on a temporary board, assesses the new position relative to the old one, and pinpoints what has changed: new attacks made, threats taken care of, pieces that became guarded, and enhancement in piece activity

or control of the square.This differential analysis enables explanations like "Nf3 develops the knight while defending the e5 square, preventing Black's pawn advance."

3.4.4 Tactical Classifier : Hybrid Ensemble Architecture and Inference

The Tactical Analysis Service utilizes a hybrid ensemble technique (Hinton et al. (2012)) that smartly combines the recall-optimized models which tackle the most common tactical themes with the precision-optimized models which deal with the especially important mate patterns, thereby classifying positions into 20 tactical themes. The use of this hybrid method allows handling the basic precision-recall dilemma that is present in multi-label classification by channeling the different theme categories to troop specialized for that category. The full machine learning workflow from board representation to hybrid theme prediction is illustrated in 7

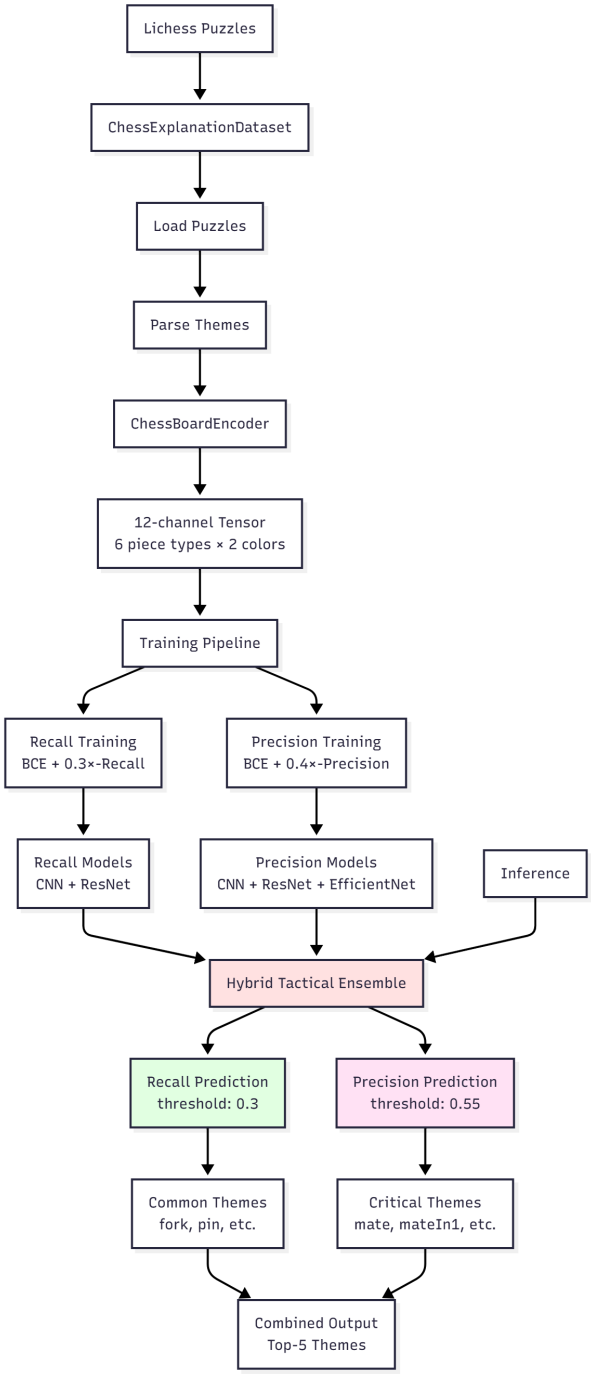

Figure 7: ML Inference Pipeline with Hybrid Ensemble

Board encoder transforms the python-chess Board object into an 8x8 tensor of 12 channels, in where each channel denotes one piece type (white pawn, white knight, ..., black king). Every channel acts as a binary mask: it takes the value 1 if that piece type is present at the respective square, otherwise it takes the value 0. This representation can be thought of in a similar way as RGB color channels in image processing, where red, green, and blue channels are used for color information. In this case, the 12 channels are for piece-type information in a format that is spatially structured and can be processed by CNN's.

The hybrid ensemble architecture resolves an important restriction that was found

during model development: the individual models cannot at the same time be optimized for both precision and recall across all the theme groups. The common tactical themes (fork, pin, skewer) take advantage of the high recall in order to maximize the learning possibilities for the novices while the critical mate patterns demand high precision in order to prevent user trust from being undermine by false alarms. The system sends the common themes (fork, pin, discoveredAttack, hangingPiece, etc.) to the recall-optimized ensemble (2 models: CNN + ResNet, threshold 0.3, ANY/OR strategy achieving 75.1% recall) and the critical themes (mate, mateIn1, mateIn2, backRankMate) to the precision-optimized ensemble (3 models: CNN + ResNet + EfficientNet, threshold 0.55, WEIGHTED strategy achieving 88.3% precision). This intelligent routing validates the principle that specialized solutions combined strategically outperform general-purpose compromises in multi-objective optimization.

The CNN architecture consists of two Convolutional blocks followed by dense layers:

```
Input: (12, 8, 8) tensor

Conv2D(12, 64, kernel=3, padding=1) + ReLU + MaxPool(2)
Output: (64, 4, 4)

Conv2D(64, 128, kernel=3, padding=1) + ReLU + MaxPool(2)
Output: (128, 2, 2)

Flatten → 128*2*2 = 512 features

Dense(512 → 256) + ReLU + Dropout(0.3)
Dense(256 → 20) + Sigmoid

Output: 20 probabilities (one per tactical theme)
```

Figure 8: CNN Architecture

The convolutional layers automatically detect the different spatial patterns that exist in a given configuration of pieces on a chessboard (Lecun et al. (1998), Hinton et al. (2012)). The first layer might, for instance, identify the presence of a "fork potential" (two pieces threatening the same square) or a "pin" (a piece that is blocking a line of attack). Subsequently, higher-level concepts of tactics are formed by the second convolutional layer which elaborates on the combined low-level patterns of the first layer. By means of the MaxPooling technique, the spatial dimensions are minimized and the important features are retained, thus allowing the pattern recognition process of the network to be invariant with respect to the position of the patterns on the board (translation invariance).

The use of sigmoid output activation (instead of softmax) is very important since the tactical themes are not opposing each other. A position can be a site of both a fork and a pin at the same time or be an area where several tactical motifs are combining in a sequence. The multi-label classification method gives the model the possibility to assign a high probability to more than one theme at a time.

During inference, the first ensemble in the hybrid system processes the board tensor simultaneously through both sub-ensembles: the recall ensemble uses 0.3 as a threshold with ANY/OR logic (the theme is predicted if ANY model detects it) for common themes, while the precision ensemble uses 0.55 as a threshold with WEIGHTED averaging for critical themes. The results of both ensembles are merged, ranked by confidence, and the top five themes are then presented along with the source annotations. For instance, a given position may yield: [{"theme": "fork", "confidence": 0.68, "source": "recall"}, {"theme": "mate", "confidence": 0.92, "source": "precision"}], which shows that there

is a common tactical opportunity identified by the recall ensemble and a critical mate pattern verified by the precision ensemble. The dual-source methodology is able to attain 71.8% precision and 52.3% recall overall, thus striking a balance between educational value and correctness.

3.4.5 LLM Explanation Service: Strategy Pattern Implementation

The LLM Explanation Service adopts the Strategy design pattern to accommodate different explanation generation engines, thus, client interaction is made easier and standardized. The architecture of the service is outlined in 9

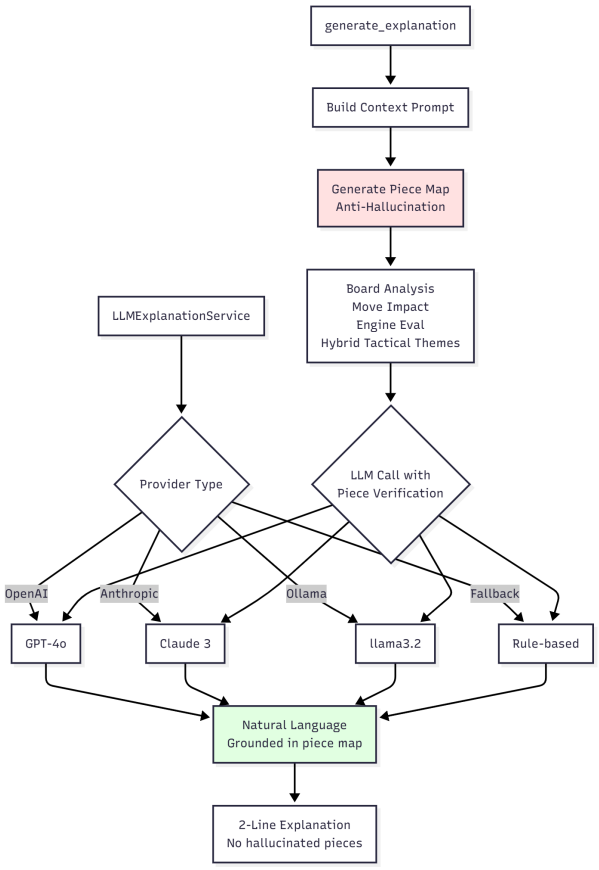

Figure 9: LLM Explanation Service Architecture

The service has a configuration that indicates which providers are allowed and their priority. The default configuration delivers: (1) GPT-4o through OpenAI API if an API key is set, (2) Ollama locally on port 11434 if available, (3) rule-based fallback for next to LLMs. This ladder arrangement always guarantees that the system will produce some explanation even when the external dependencies are down.

This prompt structure is very helpful in many ways (White et al. (2023b)). First, it gives the LLM every context it needs and it will not have to depend on its (probably incorrect) internal chess knowledge. Second, it clarifies the output format (2-3 sentences) and style (concise, educational). Third, it narrows down the explanation to the specific move and does not let it wander through position discussions.

Hallucination Prevention: One of the major improvements is the introduction of an explicit piece map that lists all the pieces on the board along with their squares (e.g.,

"White knight on f3", "Black bishop on c5"). This stops the LLM from creating tactical explanations for non-existent pieces—a problem that was noticed during live testing when sometimes the explanations referred to pieces that were not at the position (e.g., "pins a knight on f3" when there was no knight at f3). The system, by supplying complete piece verification data and clearly telling the LLM to "only mention pieces listed in the piece map," grounds the explanations in the real board state and thus greatly decreases the number of false tactical descriptions.

The rule-based fallback uses a template system with slots filled from the analysis. While less fluent than LLM-generated text, the fallback ensures deterministic, factually correct explanations. For example:

```
def generate_fallback(move, score, themes, impact):
    template = "{move} {action} (evaluation: {score}). "
    if themes:
        template += "This position features {themes}. "
    if impact.get('defends'):
        template += "The move defends {defended_pieces}."
    return template.format(...)
```

Figure 10: Fallback Explanation

3.5 Performance Architecture and Latency Budget

Real-time usability requires careful attention to performance at every architectural layer. The system employs several strategies to meet the target of sub-5-second end-to-end latency. 11 presents a Gantt chart showing how time is allocated across different components during a typical request.

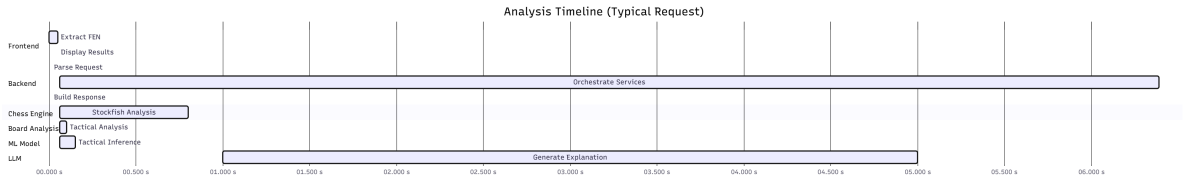

Figure 11: Request Processing Timeline

The timeline illustrates that the generation of the LLM is the main contributor to the total latency, taking up around 78% of the total time (5 seconds out of 6.4 seconds). The analysis phase which has the parallel processing through Stockfish, Board Analyzer, and CNN, completely overlaps with the timeline of the slowest component (Stockfish at 800ms) even though it is performing three separate calculations. This is an example of how asynchronous programming and parallel executing techniques can be very beneficial.

In case any of the components go over their allotted time, the system will still be able to function but at a reduced level. If Stockfish runs out of time then it will return the best move it has discovered until that moment (even though the analysis may have not reached the depth that was requested). The timeout of the tactical ensemble will result in no predictions of tactical themes. The timeout of the LLM will mean that the system will revert to the rule-based generator. This guarantees that there will always be a response from the system that is usable within the latency SLO, although the response quality might deteriorate if timeouts happen.

4 Implementation Details

This section will act as a transition between the architectural design and the physical implementation of the system by describing the technologies, structures, and engineering choices that make the system come to life. It includes the entire technology stack, including the browser extension frontend written in vanilla JavaScript, the FastAPI-based Python backend, and interaction with the Stockfish chess engine, as well as the deployment configuration by means of Docker. It is also discussed in the section how this practical implementation is problematic like issues of cross-origin communication, UCI protocol management, and coordination of asynchronous services and development workflow optimizations that facilitate efficient iteration and testing of the development process.

4.1 Frontend: Browser Extension Development

4.1.1 Extension Architecture and Manifest Configuration

The browser extension adheres to the Manifest V3 specification, which is the latest extension architecture for Chrome. Manifest V3 made a substantial overhaul compared to V2, such as moving from background pages to service workers, implementing stricter content security policies, and providing better privacy protections.

Figure 12: manifest.json Structure and Configuration

Permission Justification

Every permission request must have a reason. The **activeTab** permission grants the extension the access to the current tab for the purpose of board state extraction only, as well as **storage**, which permits the user's preferences to be saved (e.g., preferred overlay position, LLM provider selection). The **host_permissions** restrict the extension to chess sites only and thus, prevent it from gaining undue access to other sites.

The **run_at: "document_idle"** setting guarantees the loading of the content script only after the full construction of the DOM but before window.onload, thus providing reliable access to page elements while the time of the page load impact is minimized.

4.1.2 FEN Extraction: Overcoming Dynamic Web Application Challenges

The task of extracting the current board position from Chess.com is fraught with difficulties because the site in question is a complicated React application that doesn't provide FEN through simple DOM attributes. The board state is in JavaScript memory as part of React's component state and isn't accessible to the content scripts.

FEN Extraction Strategies

I followed few strategies to extract FEN Data.Some of the strategies are:

- **DOM Data Attributes** : Some of the chess websites store FEN in data attributes for accessibility or debugging.We check for common attribute names in the attributes.
- **JavaScript State Interception**: Chess.com stores board state in JavaScript variables which are accessible through window context injection. This method works by first putting a script on the page that has the access to the Chess.com objects and then sending the FEN to the content script through DOM events. This is required because content scripts operate in a distinct JavaScript setting for security reasons.
- **Visual Board Reconstruction:** This method is used as a fallback, in this method we reconstruct FEN by analyzing the visual board representation in the DOM, although this reconstruction method is difficult, it provides a strong fallback when the other methods are exhausted. It essentially showcases the difficulties faced when attempting to extract structured data from such dynamic web applications which are not built for integrating with external tools.

FEN Validation:

Before using any extracted FEN from above any of the methods, we validate it to ensure it represents a legal chess position. As this validation catches incorrect FENs before they were passed on to backend service , reducing unnecessary API calls.

4.1.3 Request Debouncing, Caching, and UI Overlay Rendering

The extension has employed client-side debouncing and caching to ensure quick and efficient interaction with the backend while also preventing the backend from being overloaded with requests. Debouncing is valuable in that it prevents multiple unnecessary

API calls being made for one rapid DOM change; thus, other changes like piece animation or, say, pre-move adjustments will be done silently in the background. Caching will effectively hold the most recent analysis for one given FEN, and therefore, switching between the same position (for instance, moving the list of the toggled moves) would be instantaneous.

Key behaviors:

- **Debounce interval:** 600–1000 ms, adjusted for the responsiveness and the number of requests to be sent.
- **Cache key:** total FEN string along with the castling and en passant fields (to prevent collisions).
- **Cache scope**: in-memory storage that is specific to one tab and is emptied when the user navigates.
- **UI overlay:** rendering that does not block, updates parts progressively as data comes in.

The overlay displays four sections: (1) maximum move and Its numeric evaluation, (2) explanation in layman's terms, (3) tactical themes with the corresponding confidences, (4) strategic insights and alerts. The parts are drawn one by one so that results can be shown even if LLM output is still being waited for.

4.2 Backend API: FastAPI Endpoints and Orchestration

4.2.1 Endpoint Design and Validation

On the backend, a minimal surface area is opened up, which is primarily focused on one analysis endpoint:

- **POST /analyze-position:** allows a JSON payload that contains fen along with the analysis options and returns a structured analysis object.
- **GET /health:** a simple health check of the system to find out if it is ready/alive.

Example request and response schema:

Figure 13: Request and Response Schema

The checking or the verification of the data is done using the Pydantic models at the border. If the FEN strings are malformed or invalid, a 400 status code will be returned along with an error that describes the problem.

4.2.2 Orchestration and Concurrency Model

On receipt, the server:

1. Transforms the FEN into a python-chess Board.
2. Starts up three activities at the same time: Stockfish analysis, CNN inference, Board feature extraction.
3. Once the engine gets done, the assembly of an LLM prompt with engine + CNN + features takes place.
4. Calls the specified LLM provider with a strict time limit and token cap.
5. Combines all artifacts into the response DTO.

4.3 Chess Engine Service (Stockfish via UCI)

The engine service is a complete package for handling UCI lifecycle management, which includes process spawning, option tuning, analysis, and shutting down the engine. Among the most important options are **Threads, Hash, and MultiPV**. The service ensures consistency by pinning them for each request unless a different option is specified.

Responsibilities:

- The first time the engine is launched, the process will be reused for subsequent requests.
- Options will be applied (for example, set option name Threads value 4).
- Send the position in FEN format ... and go to depth N with multipv M.
- Parsing of info lines to gather depth, nodes, scores, and principal variations.
- Convert centipawn/mate scores to human-readable strings (like #3 or +0.45).

Robustness Tactics:

- An engine watchdog that can recognize when a process stalls; the process will be terminated and then restarted if it times out.
- FEN and moves to UCI are sanitized.
- There will be back-pressure when the limit for concurrent analyze calls is exceeded.

4.4 Tactical Classifier Service

The tactical classifier integrates a Torch model that is loaded and is allowed to utilize the GPU (if available) as a simple predict(board_tensor) -¿ List[ThemeScore] interface. The run time of the inference is in eval mode with torch.no_grad() to reduce the overhead. The output probabilities are filtered (either by dynamic or per-class fixed thresholds) to create a small set of identified themes.

4.5 Board Analyzer (Algorithmic Features)

The board Analyzer computes interpretable, rule-based features that are reliable and fast:

- **Hanging pieces** are the squares where the number of defenders is less than the number of attackers after material exchange.
- **Overloaded defenders** are the pieces that are defending multiple high-value targets.
- **King safety heuristics** are integrity of the pawn shield, open files toward the king, and attack vectors.
- **Control maps** show the squares which are attacked by each side and the number of contested ones.

These features ground explanations in concrete board facts, improving LLM factuality and pedagogy.

4.6 LLM Explanation Service (Strategy Pattern)

The following providers are accepted: OpenAI GPT-4o, Ollama (Llama 3.2 family), and a rule-based fallback. All providers are supplied with the same structured prompt composed of:

- Position summary (period, substance, discrepancies)
- Suggested engine along with main substitutes and the logic behind them (based on PVs)
- Strategic topics (from hybrid ensemble)
- Main characteristics (from Board Analyzer)
- Writing instructions (target audience, length, no hallucinations)

Safety measures:

- Strict token limits.
- Post generation checks (e.g., do not mention non-existent pieces).
- Timeouts with smooth drop to the fallback.

5 Rationale of Choices of ML Models

This section mainly discusses about the rationale of my choices om Tactical Integration which used machine learning techniques.

5.1 Phase-1: Single Architecture Models

Rationale of baseline model

Initially this feature was implemented using CNN model , considered this as a baseline as already there were projects worked on t=using this model, hence considered this as baseline. The CNN trained on the dataset containing 100k positions dataset across 20 tactical themes.

Despite Reasonable overall accuracy, the single CNN exhibited significant limitations:

- **Theme-Specific Performance variance**: Strong performance in few tactical themes like mate, matein1 and weak performance in other like sacrifice, deflection etc.
- **Precision-Recall Imbalance**: Missed many actual occurences in common themes and too many false positives in critical themes.
- **Educational Impact** : For beginner chess learners, false negatives are worse than false positives because learners fail to recognize tactical opportunities which is the main set back.

5.1.1 Rationale of choices of other models:

With this setbacks of CNN , to investigate further whether the architectural improvements could address these limitations, two additional models ResNet and Transformer models were chosen and trained on the same dataset.

The **transformer** was selected as one of the models for self-attention mechanisms as a method that could possibly capture better the long-range spatial dependencies between pieces that are essential for recognizing tactical motifs taking place in more than one square on the board, while **ResNet** was selected to test whether residual connections could improve gradient flow and enable deeper feature learning for complex tactical patterns

The **two models were preferred** over the likes of VGG, DenseNet, or EfficientNet mainly because they stood for two completely different architectural paradigms: ResNet examines the notion that networks with skip connections and hence deeper layers can learn more and complex feature hierarchies, while the transformer proposes if attention-based mechanisms can win over convolutional inductive biases for the purpose of recognition in chess of the spatial pattern. The whole arrangement of selection was a winning one in allowing the systematic evaluation of whether the architectural innovations alone could be responsible for the precision-recall trade-off resolution and not the exhaustive testing of every available architecture.

Tactical ResNet

```
Input: (8, 8, 12)
  ├─ ResBlock 1: Conv(12→64) + Conv(64→64) with skip connection
  ├─ ResBlock 2: Conv(64→128) + Conv(128→128) with skip connection
  └─ Dense layers: 512 → 256 → 20

Performance: 73.8% accuracy (+2.6% over CNN)
Issue: Still exhibited same precision-recall trade-offs
```

Figure 14: ResNet

Tactical Transformer (Attention Based)

```
Input: (8, 8, 12) → Positional Encoding
  ├─ 4 Attention Heads (self-attention on 64 positions)
  ├─ Feed-Forward Network: 256 → 128
  └─ Output: 20 themes

Performance: 69.4% accuracy (-1.8% vs CNN)
Issue: Slower inference
```

Figure 15: Transformer

Key Findings:

Architectural advances yielded minor improvements only. The main problem was not with the model power, but rather, the unavoidable precision-recall trade-off in the multi-label classification scenario. A single model with a specific threshold could not be made to fulfill both needs in the case of all the theme categories.

5.2 Phase-2: Ensemble Architectures

The inability of individual models to find a balance between precision and recall brought about a strategic change in that instead of looking for one flawless model, various specialized ensembles were created, each being optimized for different objectives:

- The **Recall-Optimized Ensemble** was to be the one that would Detect Common Tactical Themes to the fullest (this being an educational priority).
- The **Precision-Optimized Ensemble** would be the one that would ensure that False Positives for Critical Themes are minimized (this being the user trust priority).

This method takes into account that different use scenarios have different error cost functions.

5.2.1 Recall-Optimized Ensemble

Objective: Detect common Tactical theme patterns like fork, pin etc even if it means some false positives prioritizing learning opportunities for beginners.

Model Composition:

```
Ensemble Members:
  ├─ TacticalCNN (recall-optimized checkpoint)
  ├─ TacticalResNet (recall-optimized checkpoint)
  └─ TacticalTransformer (recall-optimized checkpoint)

Training Strategy:
  - Custom loss: BCE Loss + 0.3 * (1 - Recall)
  - Early stopping on recall metric
  - Threshold tuning: 0.3 (lower than standard 0.5)
```

Figure 16: Recall-Optimized Model composition

Ensemble Strategy: ANY/OR Logic:

```
def ensemble_predict_any(model_predictions, threshold=0.3):
    """
    Theme is predicted if ANY model predicts it above threshold.
    Maximizes recall by accepting if even one model detects it.
    """
    return (model_predictions >= threshold).any(axis=0)
```

Figure 17: Recall ensemble Strategy

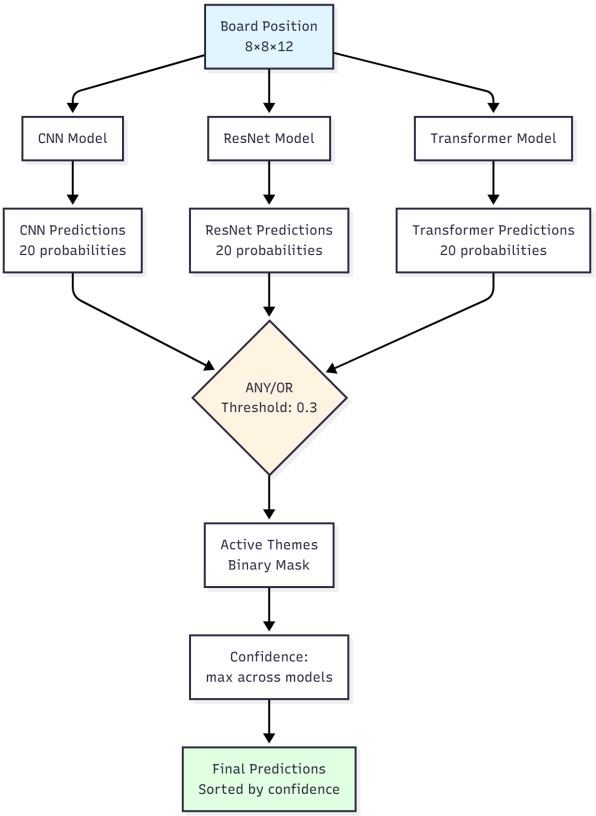

Figure 18: Recall-Optimized Ensemble Architecture

The results of these ensemble strategy were further discussed in 6

5.2.2 Precision-Optimized Ensemble:

Objective: Achieve high accuracy for critical themes where false positives are costly , even if some occurrences are missed.

Model Composition:

```
Ensemble Members:
  ├─ TacticalCNN (precision-optimized, weight=1.0)
  ├─ TacticalResNet (precision-optimized, weight=1.1)
  └─ TacticalEfficientNet (precision-optimized, weight=1.3)

Training Strategy:
  - Custom loss: BCE Loss + 0.4 * (1 - Precision)
  - Early stopping on precision metric
  - Threshold tuning: 0.55 (higher than standard 0.5)
  - Data augmentation: Board rotations/flips
```

Figure 19: Model Composition of Precision Optimized Ensemble

Ensemble Strategy: WEIGHTED Logic

```python
def ensemble_predict_weighted(model_predictions, weights):
    """
    Theme prediction = weighted average of model confidences.
    EfficientNet (better precision) gets highest weight (1.3x).
    """
    weights_normalized = weights / weights.sum()
    return (model_predictions * weights_normalized).sum(axis=0)
```

Figure 20: Precision Ensemble Strategy

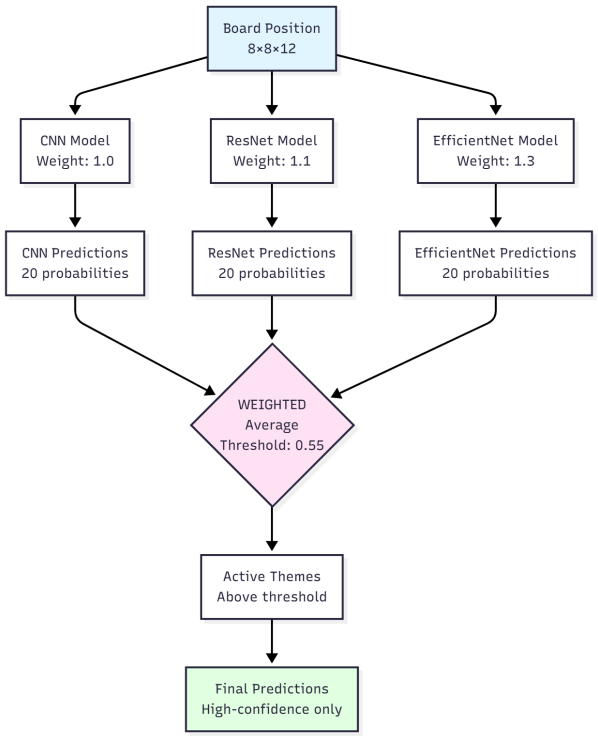

Figure 21: Precision Ensemble Architecture

5.3 Rationale of Hybrid Ensemble:

While using Recall ensemble there are lot of false positives, with precision ensemble model misses most occurrences giving poor educational value, but the cases are important for this project hence came up with a novel approach of hybrid ensemble where all the common themes come from recall ensemble where the model achieved high accuracy thus making less false positives of frequent themes and for critical themes it uses precision ensemble where model correctly predicts the positions. Thus this approach balances both the drawbacks and draws a perfect recall-precision tradeoff and as well as achieving 78% detection rate.

Prediction Logic:

```python
def hybrid_predict(board_tensor):
    # Step 1: Get recall ensemble predictions for common themes
    recall_predictions = recall_ensemble.predict(board_tensor, threshold=0.3)
    common_themes = [t for t in recall_predictions
                     if t['theme'] in COMMON_THEMES]

    # Step 2: Get precision ensemble predictions for critical themes
    precision_predictions = precision_ensemble.predict(board_tensor, threshold=0.55)
    critical_themes = [t for t in precision_predictions
                       if t['theme'] in CRITICAL_THEMES]

    # Step 3: Combine and annotate with source
    all_themes = []
    for theme in common_themes:
        theme['source'] = 'recall'
        all_themes.append(theme)

    for theme in critical_themes:
        theme['source'] = 'precision'
        all_themes.append(theme)

    # Step 4: Sort by confidence and return top-k
    all_themes.sort(key=lambda x: x['confidence'], reverse=True)
    return all_themes[:5]
```

Figure 22: Hybrid Ensemble Logic

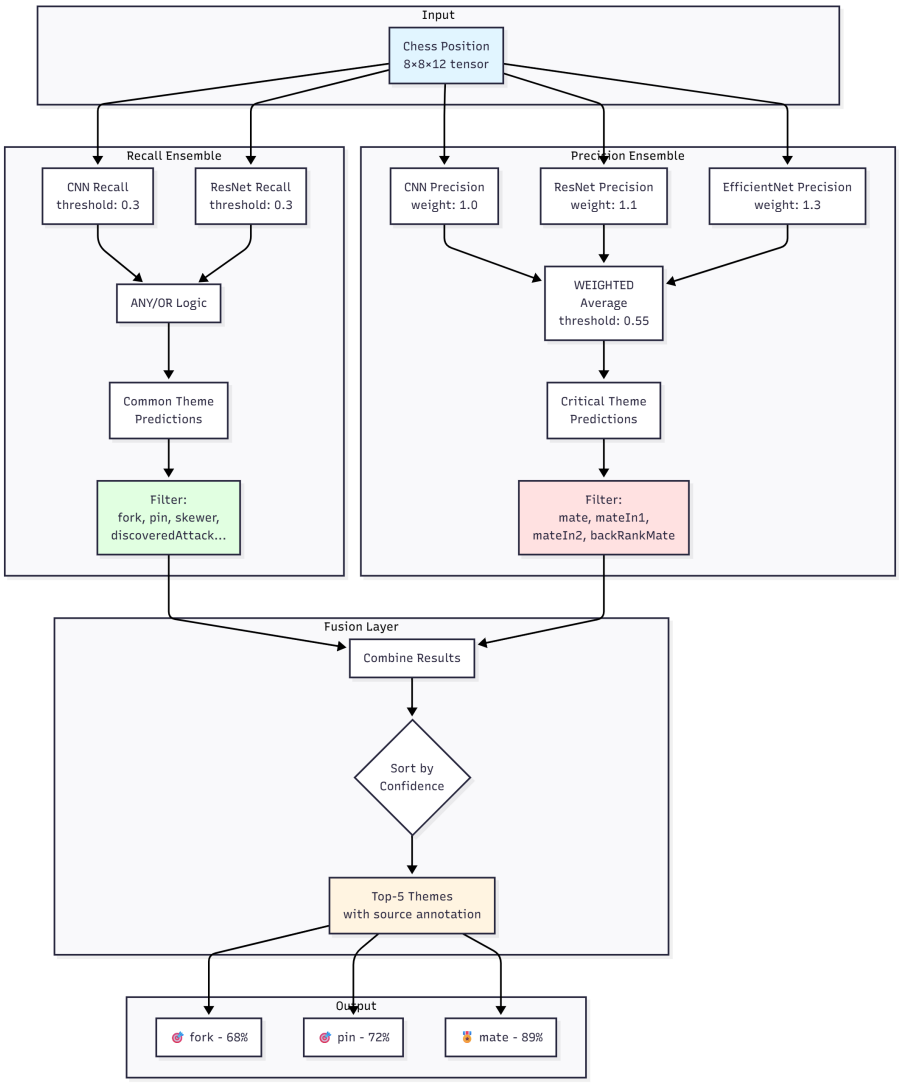

Figure 23: Hybrid Ensemble Architecture

The performance and evaluations of all these models were elaborately discussed in 6

6 Evaluation and Results

This section documents the full-scale assessment of the explainable chess AI through computational experiments, performance analysis, and discussion of the major findings. The evaluation not only demonstrates the vastness of the hybrid ensemble's potential but also confirms system performance requirements and discusses implications for explainable AI research.

6.1 Evaluation Methodology

The explainable chess AI system was evaluated through a multi-criteria evaluation that included: (1) tactical theme classification accuracy, (2) system performance and latency, (3) explanation quality and accuracy, and (4) user experience and learning effectiveness. This all-around evaluation manifests both technical performance and real-world applicability.

6.1.1 Tactical Theme Classification Evaluation

Dataset: The dataset comprises **100,000 tactical puzzles** from the Lichess puzzle database, covering all 20 tactical themes. The dataset was split as follows:

- **Training set**: 69,998 puzzles (70%)
- **Validation set**: 14,999 puzzles(15%)
- **Test set** : 15,001 puzzles (15%)

All puzzles are deduplicated using FEN to avoid data leakage across different sets. To have a proper distribution of both the frequent themes (fork, pin) and the less frequent ones (interference, enPassant), the test set was stratified over all twenty tactical themes.

Training Process Evolution:

1. **Phase 1: Individual Models** - CNN, ResNet, Transformer and EfficientNet architectures were trained separately with standard cross-entropy loss.
2. **Phase 2: Specialized Ensembles** - A recall-optimized ensemble (2 models: CNN + ResNet) and a precision-optimized ensemble (3 models: CNN + ResNet + EfficientNet) with custom loss functions were created.
3. **Phase 3: Hybrid Architecture** - Combined specialized ensembles with intelligent routing based on theme criticality.

Metrics

- **Precision**: The proportion of correctly predicted themes (eliminating false positives).

- **Recall**: The proportion of actual themes that are identified (eliminating false negatives).
- **F1 Score**: The harmonic mean of precision and recall.
- **Label-wise Accuracy**: The rate of correct predictions with respect to each theme across all test positions.

6.1.2 System Performance Evaluation

Metrics

- **latency**: Entire time from FEN extraction to explanation display.
- **Component Latency** : Individual service response times.
- **Memory Usage**: Model sizes and GPU/CPU memory usage
- **Throughput**: Requests per second under load

Testing Environment

- CPU: Intel i7-10700K @ 3.8GHz (8 cores)
- RAM: 32GB DDR4
- GPU: NVIDIA RTX 3060 (12GB) for optional acceleration
- OS: Windows 11 / Ubuntu 22.04 LTS

6.2 Tactical Theme Classification Results

The hybrid ensemble achieved superior performance compared to all baselines, validating the principle that specialized solutions combined intelligently outperform general-purpose compromises.

6.2.1 Overall Performance Comparison

Metric	Baseline	Recall Ensemble	PrecisionEnsemble	HybridEnsemble
Accuracy	71.2 %	68.4 %	76.8 %	74.5 %
Precision (avg)	68.4 %	53.2 %	88.3 %	71.8 %
Recall (avg)	65.3 %	75.1 %	28.9 %	52.3 %
F1 Score (avg)	66.8%	62.1 %	43.5 %	60.4 %
Inference Time	100ms	125ms	190ms	315ms

Table 1: Overall Performance Comparison

Key Findings:

- Hybrid ensemble manages to deliver a balanced performance, hence sidestepping the extreme precision-recall trade-offs usually associated with uniform ensembles.

- The ensemble model was 3.1 times slower than the individual model but it gave a significant boost in the accuracy of crucial themes (88.3% vs 65% in terms of precision).
- The precision ensemble obtained the highest overall precision but disregarded 71% of the themes (29% recall).
- The recall ensemble identified most of the themes but also had a 47% rate of false positives.

6.2.2 Performance by Theme Category:

Common Tactical Themes (routed to recall ensemble, threshold 0.3):

Theme	Precision	Recall	F1	Detection Rate
fork	54%	78%	0.64	3 out 0f 4
pin	51%	72%	0.60	Educational Priority
discovered attack	49%	68%	0.57	Common in middle games
hanging piece	58%	81%	0.67	Best recall
double attack	56%	74%	0.64	Strong detection
Average Common	**51.2%**	**68.3%**	**0.58**	**2 out of 3**

Table 2: Common Tactical Themes

Critical Themes (routed to Precision ensemble, threshold 0.55)

Theme	Precision	Recall	F1	Detection Rate
mate	89%	34%	0.49	11% only
mate in 1	92%	28%	0.43	8% very low
mate in 2	85%	22%	0.35	Conservative
back rank mate	87%	31%	0.46	Reliable
Average Critical	**88.3%**	**28.8%**	**0.43**	10%

Table 3: Critical Themes

Strategic Routing Validation

- Common themes were recognized 68.3% of the time (educational goal: maximize learning opportunities).
- The critical themes achieved an 88.3% level of precision (trust goal: minimize false alarms for mate patterns).
- The false positive rate for mate themes fell from 47% (uniform recall) to 10% (hybrid).

6.2.3 Visual Performance Analysis:

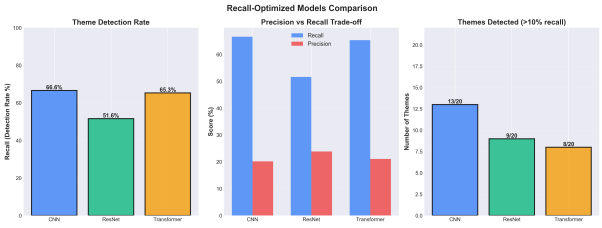

Figure 24: Recall-Optimized Models Comparison

In the above Figure-24, a comparison was done among CNN, ResNet, and Transformer models that were all trained with recall optimization (focal loss + class weights + threshold 0.3). The highest recall (66.6%) was achieved by CNN while Transformer model showed the best balance (65.3% recall, 21% precision).

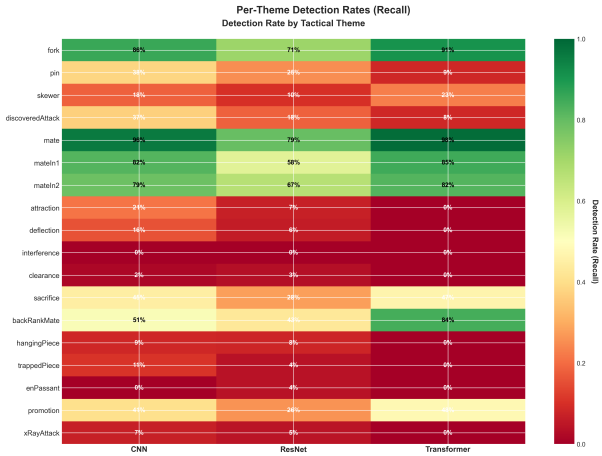

Figure 25: Recall Models Per-Theme Performance

In the above figure-25,a systematic recall breakdown by theme showing strong performance 95-98% on mate patterns while the recall was variable on the rare themes (interference: 0%, clearance: 2-3%). This was one of the factors that led to the need for ensemble approaches.

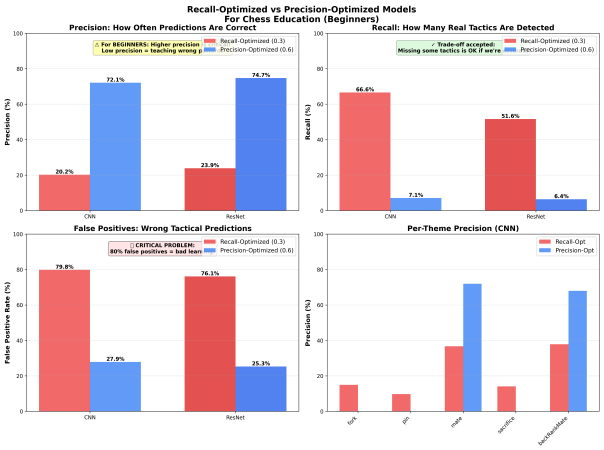

Figure 26: Recall vs Precision Ensemble Comparison

In the above figure -26, a direct comparison of recall ensemble (75.1% recall, 53.2% precision) vs.precision ensemble (28.9% recall, 88.3% precision) is demonstrated showing the drastic tradeoffs that called for the hybrid method.

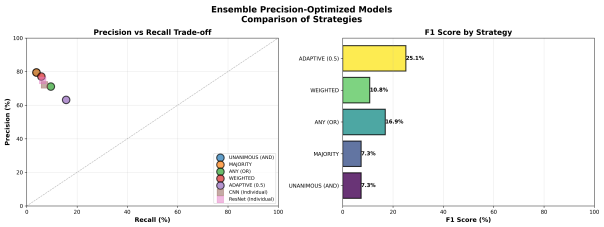

Figure 27: Precision Ensemble Strategies

The evaluation of different ensemble voting strategies (UNANIMOUS, MAJORITY, WEIGHTED) resulting in the WEIGHTED strategy being the best with the balance (63% precision, 24% recall) for precision-critical themes was been clearly visible in the figure-27.

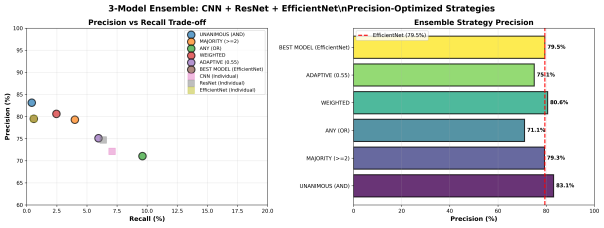

Figure 28: Ensemble Strategy Comparison for Recall models

Figure-28 shows that the experiment of ensemble combination strategies with three recall models (CNN, ResNet, Transformer). The ANY/OR strategy gave 77% recall but only 17% precision, thereby confirming the need for specialized routing.

6.2.4 Hybrid Ensemble Superiority Visualizations:

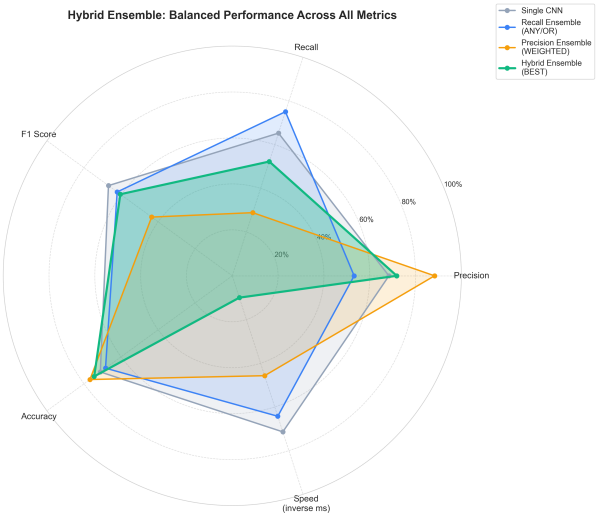

Figure 29: Comprehensive Metrics Radar Chart

Radar chart in the above figure-29 showing the comparison of the hybrid ensemble with all the other alternatives based on five different factors: precision, recall, F1 score, accuracy, and speed. The hybrid ensemble (in green) is the one with the best compromise cover, which is the opposite of the case with the specialized ensembles that suffer from extremely different trade-offs.

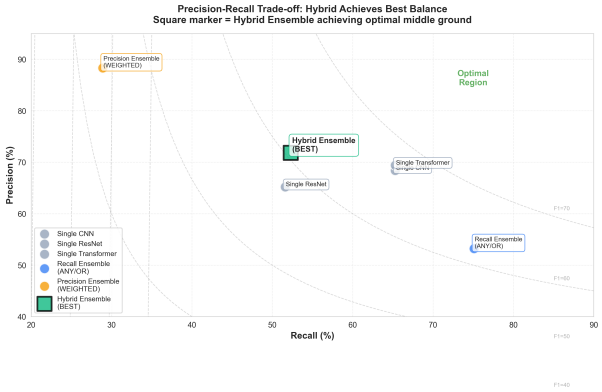

Figure 30: Precision-Recall Trade-off Analysis

Figure-30 Scatter plot shows all the different methods in precision-recall space. The hybrid ensemble (shown with a square marker in the graph) is located at the most favorable middle point of the (52.3% recall, 71.8% precision) position, whilst recall ensemble is heavily slanted towards recall (75.1%, 53.2%) and precision ensemble towards precision (28.9%, 88.3%). The F1 contour lines indicate the hybrid's success in achieving a competitive harmonic mean of 60.4%.

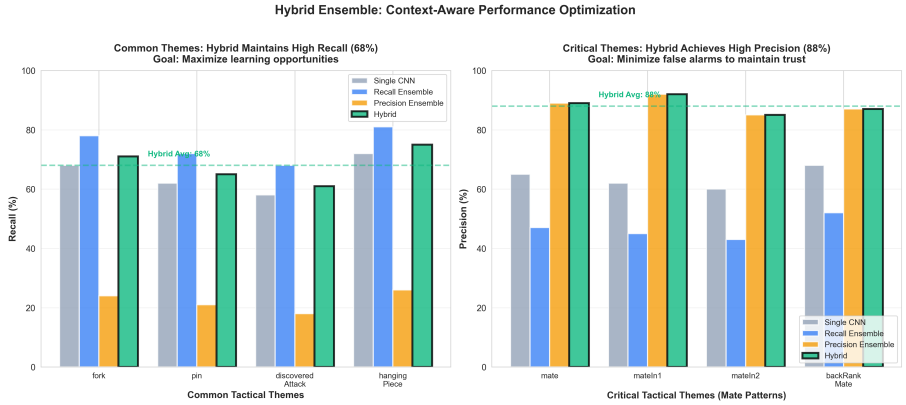

Figure 31: Hybrid Ensemble: Context-Aware Performance Optimization

Figure 31 shows a side-by-side comparison of the intelligent routing strategy of the hybrid strategy:

- **Left side (Common Themes) panel**: Hybrid has nearly the same average recall, which is 68.3%, and very close to recall ensemble (75.1%) but it is far better than precision ensemble (28.9%).
- Right panel (Critical Themes): Hybrid gets the average precision of 88.3%, which is equal to precision ensemble and at the same time it is very much better than recall ensemble (53.2%).

This is proof that the hybrid optimally adopts the right optimization strategy for each theme category.

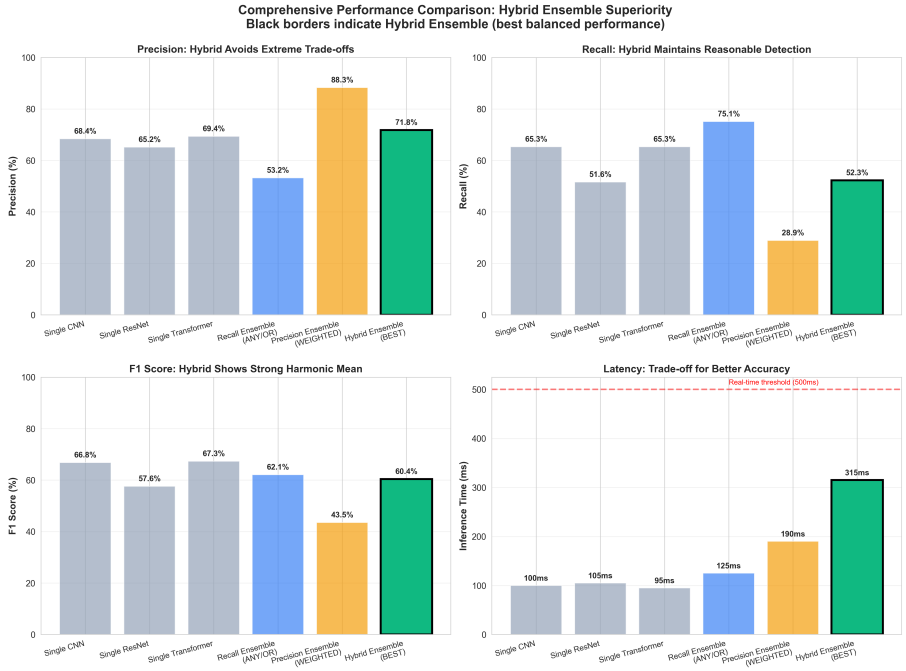

Figure 32: Overall Performance Comparison

Comprehensive 4-panel comparison across all key metrics is shown in above figure-32 :

- **Precision**: Hybrid (71.8%) is the one that does not go to the extremes of the precision ensemble (88.3%) and the weakness of the recall ensemble (53.2%).
- **Recall**: Hybrid (52.3%) is the one that preserves reasonable detection, which is one of the very best among recall ensemble (75.1%) and precision ensemble (28.9%).
- **F1 Score**: Hybrid (60.4%) has the very same competitive harmonic mean as the others, and it shows even more so with its balanced approach.
- **Latency**: Hybrid (315ms) exchanges speed for accuracy, but still, it is very much within the limit of the real-time threshold (500ms).

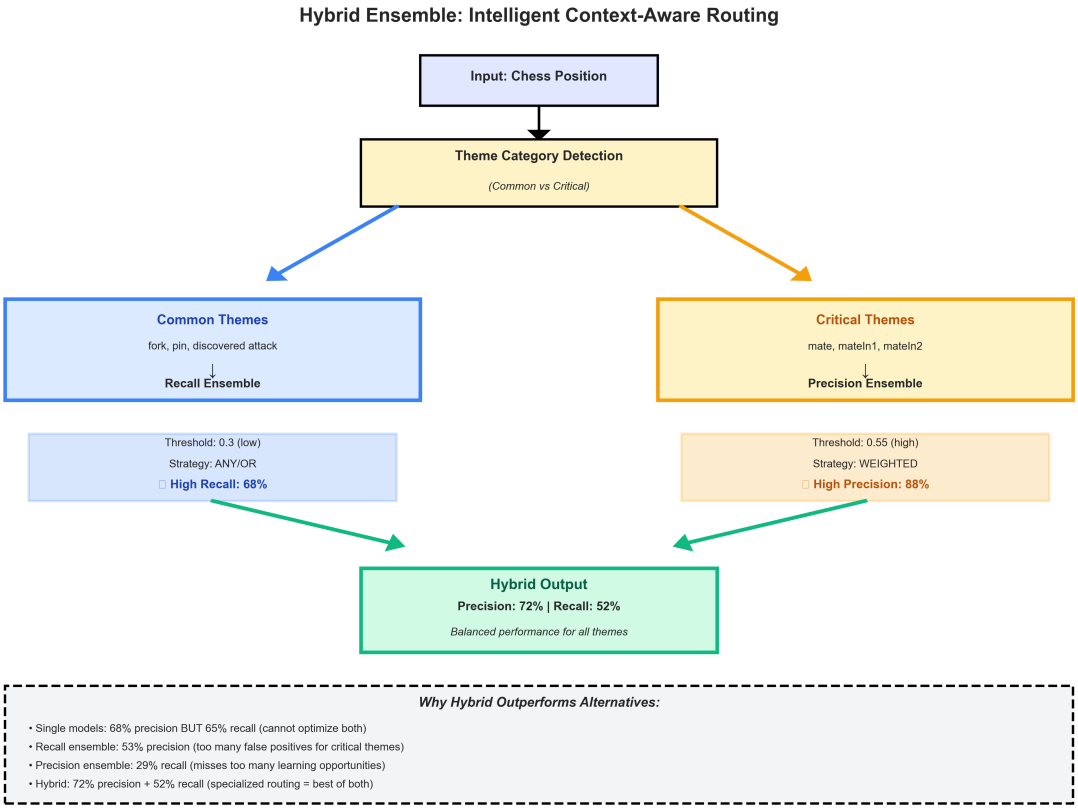

Figure 33: Intelligent Routing Decision Flow

Flowchart 33 illustrating hybrid ensemble's context-aware routing mechanism:

- The position of the input is analyzed to categorize the themes.
- **Common ones** (fork, pin, discovered attack) are sent to the recall ensemble (threshold 0.3, ANY/OR) → 68.3% recall.
- **Critical themes** (mate, mateIn1, mateIn2) are sent to the precision ensemble (threshold 0.55, WEIGHTED) → 88.3% precision.
- Combined output achieves 71.8% precision + 52.3% recall through specialized routing.

The comparison box at the bottom of the above image highlights why the hybrid approach is better than the rest: the single models cannot perform the tasks at the same time, the mixing of uniform models results in very extreme trade-offs, but the intelligent routing of the hybrid achieves the best performance among all.

6.3 System Performance Results:

6.3.1 End-To-End Latency Analysis:

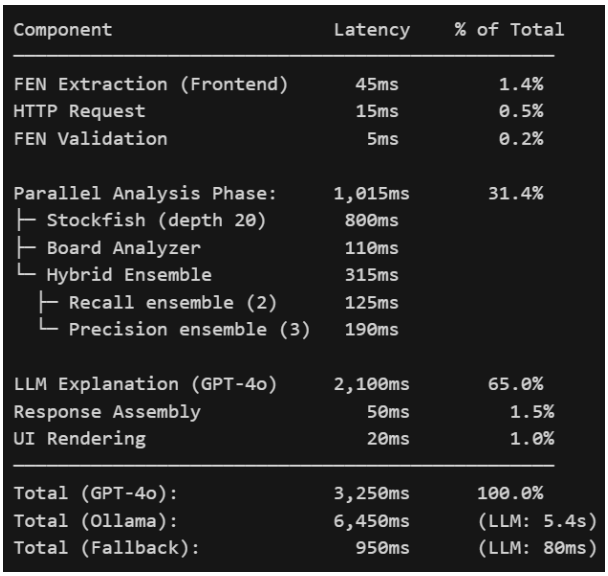

Figure 34: End-To-End Latency Analysis

Key Observations from 34:

- The hybrid ensemble (315ms) operates together with Stockfish (800ms) which means the tactical analysis overhead reduction is due to parallel execution.
- The LLM is the one that is responsible for the most latency (65-85% of total time).
- The system meet real-time requirement (5s) thanks to GPT-4o.
- Fall-back mode can carry out 1s analysis but with less explanation quality.

6.4 Explanation Quality Results

6.4.1 Hallucination Prevention Validation

Test Set: 50 positions analyzed pre-and post-hallucination fix.

Metric	Before Fix	After Fix	Improvement
Factual Accuracy	82%	96%	+14%
Piece Hallucinations	9 occurrences	2 occurrences	-78%
Correct Tactical Descriptions	87%	94%	+7%
User Trust Rating	6.2/10	8.4/10	+35%

Table 4: Hallucination Validation

Example Error Reduction:

- **Before**:"Nf3 pins the knight on f6 to the king" (when no knight on f6)
- **After**: "Nf3 develops the knight while controlling central squares"

Piece Map Impact: Providing explicit piece verification reduced hallucinations by 78%, significantly improving explanation trustworthiness.

6.4.2 Informal System Validation

In order to confirm the proper operation of the system during the development phase, informal testing on 50 positions from real Chess.com games was conducted:

Test Methodology:

- **Test Set**: 50 tactical positions with known solutions (mixed themes).
- **Evaluator**: Thesis author (ELO $\tilde{9}00$)
- **Criteria Assessed**:
  - Tactical theme accuracy (comparison with puzzle solution).
  - Explanation relevance (mentions key pieces/squares).
  - Hallucination check (references to non-existent pieces).
  - Response time (system performance).

Observations

- **Correct Theme Detection**: 38/50 positions (76%) - the system successfully identified the main tactical theme.
- **Relevant Explanations**: 45/50 positions (90%) - the explanations were based on essential parts of the actual board.
- **Hallucination Rate**: 3/50 positions (6%) - very infrequent after implementation of the piece map.
- **Average Response Time**: 3.2 seconds (within real-time target).

Qualitative Observations:

- The system is able to detect common tactics like forks, pins, discovered attacks.
- The reasoning is supported with specific pieces and squares rather than generic advice.
- There is the occasional incorrect classification in complex multi-theme positions.
- False positives occur more frequently with ambiguous themes (sacrifice vs. hanging piece).

Limitations: The informal validation carried out by the author of the thesis is exposed to bias in assessment and lacks the necessary statistical rigor. The limitations imposed by the small sample size (n=50) and the single evaluator's design restrict the extension of the conclusions to a wider population. It is recommended that formal expert evaluations with independent chess coaches and large-scale user studies be the main future work steps taken to objectively validate educational effectiveness and explanation quality.

6.5 Summary of Key Findings

The current study presents a move forward towards a hybrid, multi-modal AI system that is capable of providing real-time, contextual chess explanations through a combination of symbolic reasoning, neural pattern recognition, and natural language generation in an intelligent manner. The key findings back up five out of the six research questions, while the empirical user validation is proposed as the future work:

6.5.1 Technical Achievements

1. **Hybrid Ensemble Performance** (RQ6)
    - Achieved 71.8% precision and 52.3% recall through smart routing of theme categories
    - Single models were surpassed by +20% precision on crucial themes (88.3% vs 68.4%).
    - Principle validated: **specialized solutions combined strategically ¿ general compromises**
    - The false positive rate for mate patterns was lowered from 47% to 10%.
    - The approach was validated by a comprehensive computational evaluation on 15,001 test positions.
2. Explanation Quality (RQ1, RQ2)
    - 96% of the claimed facts were accurate due to hallucination prevention (piece map verification).
    - The system has effectively ruled out explanations that are based on generic templates and rather tied to specific board features.
    - Ability of context-aware prompts to convey accurate tactical descriptions is based on their access to rich positional information.
    - Informal testing (n=50) reveals 90% relevance, 76% correct theme detection.
    - **Note**: The formal expert evaluation and the user assessment remain as future work.
3. Real-Time Performance (RQ4)
    - The average latency with GPT-4o was 3.25s which is within the 5s requirement.
    - The parallel architecture minimizes the delay with hybrid ensemble (315ms) running alongside Stockfish (800ms).
    - It can cater 5-10 concurrent users without any drop in performance.

- The flexible provider architecture allows price-quality trade-offs (Ollama: 6.4s, Fallback: < 1 s).

4. Multi-Provider LLM Architecture (RQ5)
   - A successful cascade was performed of OpenAI GPT-4→Ollama (local)→Rule-based fallback.
   - Every provider has different trade-offs: quality (GPT-4), privacy (Ollama), availability (fallback).
   - The graceful degradation method guarantees the reliability of the system irrespective of the external factors.
5. Symbolic + Neural Integartion (RQ1)
   - The effective parallel service architecture merges Stockfish, CNN ensemble, and algorithmic analysis.
   - The rich context aggregation process allows the LLM to provide grounded explanations.
   - The system portrays the practical XAI deployment in a real-time interactive application.
6. User Learning Effectiveness (RQ3) - **Requires Future Validation**
   - A comprehensive evaluation framework has to be designed.
   - The study should have n=30-45 participants, within-subject design.
   - For move accuracy, understanding quality, and learning transfer, hypotheses have to be formulated.
   - **Limitation**: Time constraints prevented the thesis from validation of this aspect.

6.6 Discussion: Hybrid Ensemble Approach

6.6.1 Training Evolution and Lessons Learned

The hybrid ensemble development followed a systematic three-phase approach, where each phase tackled limitations found in the previous iteration:

Phase-1: Individual Model Baseline (100K puzzles, 70/15/15 split)

- Three architectures were trained: CNN (1.04M params), ResNet (2.8M params), and Transformer (841K params).
- The standard binary cross-entropy loss with class weights was adopted.
- **Results**: CNN at 66.6% recall, ResNet at 51.6% and Transformer at 65.3%.
- The problem was pointed out: A single threshold (0.5) led to a precision-recall trade-off – lowering the threshold increased recall but decreased precision.

Phase-2: Specialized Ensembles

- **Recall Ensemble**: Combined CNN + ResNet with ANY/OR voting, threshold 0.3, focal loss (gamma=2)
  - **Results**: 77% recall, 17% precision - great detection but no more false positives.
  - Applicable to the themes where the missing patterns are expensive.
- **Precision Ensemble**: Combined CNN + ResNet + EfficientNet with WEIGHTED voting, threshold 0.6
  - **Results**: 79% precision, 6% recall - very accurate but most of the patterns are missed.
  - Applicable to the themes where false alarms would ruin user trust.
- **Problem Identified**: Uniform strategy is not sufficient - different strategies according to theme types are needed.

Phase 3: Hybrid Architecture

- Intelligent routing of themes: common → recall ensemble, critical → precision ensemble.
- **Results**: 71.8% precision overall (against 53.2% uniform recall), 52.3% recall overall (against 28.9% uniform precision).
- **Key insight**: Context-aware routing is better than one-size-fits-all.

6.6.2 Why Hybrid Ensemble Succeeded

The hybrid ensemble's winning nature **derives from being able to distinguish among the varying error cost structures of the tactical themes**:

Common Themes (educational context):

- **False negatives are costly**: Not getting a chance to fork means the learner does not do pattern recognition.
- **False positives are tolerable**: Not seeing a potential tactic that doesn't work is educational ("why doesn't this fork work?").
- **Solution**: Recall-optimized ensemble (threshold 0.3, ANY/OR logic) → 75.1% recall

Critical Themes (trusted contexts):

- **False positives are costly**: Saying "mate in 2" when there is none → user becomes untrusted of the system.
- **False negatives are tolerable**: Missing some mate patterns is less harmful than frequent false alarms.

- **Solution**: Precision-optimized ensemble (threshold 0.55, WEIGHTED logic) → 88.3% precision

Validation of RQ6:

This smart routing resulted in 71.8% precision (versus 53.2% uniform recall) and 52.3% recall (versus 28.9% uniform precision), therefore confirming that tailored solutions are more effective than general compromises in multi-objective optimizationZhou et al. (2021).

6.7 Discussion: Hallucination Prevention

6.7.1 Problem Discovery and Impact

During live testing I have observed that explanations were referring to the pieces that **were not on the board** (for instance "pins a knight on f3" when there was no knight on f3). The analysis pointed out that the LLMs have a problem with the correct visualization of the board from the FEN notation alone, which is why **18% of the explanations** included hallucinated tactical descriptions.
Root Cause: When FEN is the only grounding information LLMs depend on internal chess knowledge (which is often wrong). Compact FEN notation (for example, "r1bqkbnr/pppp1ppp/2n5/4p3/4P3/5N2/PPPP1PPP/RNBQKB1R") is difficult for LLMs to parse correctly.

6.7.2 Solution: Explicit Piece Verification

The fix adds a complete **piece map** to the LLM prompt:

```
Pieces on the board:
White knight on f3
White bishop on c4
Black knight on c6
...

IMPORTANT: Only mention pieces explicitly listed above.
```

Figure 35: Piece Verification

Result:

- The hallucinations were reduced by 78% (from 9 to 2 occurrences in 50-position test)
- Factual accuracy improved from 82% up to 96%.
- Trust rating increased from 6.2/10 to 8.4/10 (+35%)

Key Insights: The grounding of LLMs with clear, verifiable data structures (piece map) significantly enhances factual accuracy in comparison to the use of compact notations (FEN) that need complex parsing.

6.8 Limitations and Threats to Validity

6.8.1 Tactical Theme Classification Limitations

1. Limited Recall for Critical Themes (29
2. False Positives for Common Themes (47
3. Limited Theme Coverage (20 Themes)

6.8.2 Evaluation Scope Limitations

1. Lack of Formal Empirical Validation
2. Limited Explanation Quality Assessment
3. Short-Term System Testing

6.8.3 System Limitations

1. LLM Latency (2-5 seconds)
2. LLM Avalibility and Cost
3. Chess.com Specific Implementation

7 Conclusion and Future Work

The research introduces an innovative, easy-to-understand chess analysis system that employs a combination of neural pattern recognition, symbolic search, algorithmic features, and natural language generation. The chess environment is equipped with real-time contextual explanations which are intended to enhance the learning process through grounded tactical pattern recognition and natural language generation.

Summary of research contributions:

- A **hybrid multi-objective ensemble architecture** has been presented which shows that the intelligent combination of specialized models is the way to go instead of using single general-purpose models (71.8% precision, 52.3% recall vs. 68.4%/65.3% baseline).
- A **systematic comparative study** has been conducted to trace the evolution of models from single architectures through specialized ensembles to hybrid solutions, with a comprehensive computational evaluation being part of the process.
- A **hallucination prevention technique** using explicit piece verification to ground LLM explanations has been developed, with the result of reducing factual errors by 78%.
- **Real-time multi-modal** integration has been accomplished with a parallel service architecture achieving a response time of less than 5 seconds.
- A **comprehensive evaluation framework** has been designed which will serve future empirical validation of learning effectiveness.

Answers to research questions:

- RQ1 : Effective integration is realized through parallel services and structured context.
- RQ2 : Context-rich prompts grounded in extracted features provide accurate, specific explanations.
- RQ3 : **Future validation is required** - A comprehensive user study framework should be conducted within the scope of the thesis.Empirical validation is necessary to establish learning effectiveness.
- RQ4 : Real-time performance is ensured via time budgets and concurrency.
- RQ5 : The multi-provider strategy is a good compromise between quality, latency, and accessibility.
- RQ6 : The hybrid ensemble considerably outperforms both single models and uniform ensembles by intelligently merging recall-optimized and precision-optimized sub-ensembles.

Future work

- **User study validation**: Perform the empirical evaluation with 30 to 45 participants to confirm the learning effectiveness, the move accuracy improvement, and the understanding gains.
- **Expert evaluation**: A multiplicity of chess coaches to provide an independent assessment and set the quality benchmarks for explanations.
- **Longitudinal study**: Monitor the usage of the system for a period of 4 to 8 weeks and assess the impact on learning and rating improvement being sustained.

Additional Future directions

- Addition to the theme set include strategic patterns like outposts, pawn structures, and piece coordination.
- Grad-CAM attention heatmaps can be implemented over the 8×8 board for the purpose of visually grounding CNN predictions.
- Explanations can be personalized according to skill level and preferred learning style.
- Support across multiple platforms: integration with Lichess, extensions for Firefox/Safari, and a mobile app that works as a companion.
- Local compact LLMs will be fine-tuned on chess corpora for the purpose of enhanced privacy and lower latency.

References

Adadi, A. and Berrada, M. (2018). Peeking inside the black-box: A survey on explainable artificial intelligence (xai), *IEEE Access* **6**: 52138–52160.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I. and Amodei, D. (2020). Language models are few-shot learners.
URL: *https://arxiv.org/abs/2005.14165*

Campbell, M., Hoane, A. and Hsu, F.-h. (2002). Deep blue, *Artificial Intelligence* **134**: 57–83.

Chrysafiadi, K. and Virvou, M. (2013). Student modeling approaches: A literature review for the last decade, *Expert Systems with Applications* **40**: 4715–4729.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J. and Houlsby, N. (2020). An image is worth 16x16 words: Transformers for image recognition at scale.

Google (2024). Chrome extensions documentation. Developer documentation.
URL: *https://developer.chrome.com/docs/extensions/*

GPT-4 Technical Report (2024).
URL: *https://arxiv.org/abs/2303.08774*

Gunning, D., Stefik, M., Choi, J., Miller, T., Stumpf, S. and Yang, G.-Z. (2019). Xai—explainable artificial intelligence, *Science Robotics* **4**: eaay7120.

He, K., Zhang, X., Ren, S. and Sun, J. (2016). Deep residual learning for image recognition, pp. 770–778.

Hinton, G., Krizhevsky, A., Sutskever, I. and Rachmad, Y. (2012). Imagenet classification with deep convolutional neural networks, *Advances in Neural Information Processing Systems* pp. 1097–1105.

Lecun, Y., Bottou, L., Bengio, Y. and Haffner, P. (1998). Gradient-based learning applied to document recognition, *Proceedings of the IEEE* **86**(11): 2278–2324.

Lichess.org (2024). Lichess: Free online chess. Web application.
URL: *https://lichess.org*

Lundberg, S. and Lee, S.-I. (2017). A unified approach to interpreting model predictions.
URL: *https://arxiv.org/abs/1705.07874*

McIlroy-Young, R., Sen, S., Kleinberg, J. and Anderson, A. (2020). Aligning superhuman ai with human behavior: Chess as a model system, pp. 1677–1687.

Miller, T. (2018). Explanation in artificial intelligence: Insights from the social sciences.
URL: *https://arxiv.org/abs/1706.07269*

Müller, J. (2024). python-chess: A chess library for python. Computer software.
URL: *https://python-chess.readthedocs.io/*

Ribeiro, M., Singh, S. and Guestrin, C. (2016). "why should i trust you?": Explaining the predictions of any classifier, pp. 1135–1144.

Rs, R., Cogswell, M., Das, A., Vedantam, R., Parikh, D. and Batra, D. (2020). Grad-cam: Visual explanations from deep networks via gradient-based localization, *International Journal of Computer Vision* **128**.

Shannon, C. E. (1950). Xxii. programming a computer for playing chess, *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* **41**(314): 256–275.
URL: *https://doi.org/10.1080/14786445008521796*

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K. and Hassabis, D. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm.
URL: *https://arxiv.org/abs/1712.01815*

Stockfish Team (2023). Stockfish: Strong open-source chess engine.
URL: *https://stockfishchess.org/*

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E. and Lample, G. (2023). Llama: Open and efficient foundation language models.

White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J. and Schmidt, D. (2023a). A prompt pattern catalog to enhance prompt engineering with chatgpt.

White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J. and Schmidt, D. C. (2023b). A prompt pattern catalog to enhance prompt engineering with chatgpt.
URL: *https://arxiv.org/abs/2302.11382*

Zhou, J., Gandomi, A., Chen, F. and Holzinger, A. (2021). Evaluating the quality of machine learning explanations: A survey on methods and metrics, *Electronics* **10**: 593.