Bondarenko Stefaniia, Lecturer of the Department of Software Engineering, Petro Mohyla Black Sea National University, Mykolaiv, Ukraine, ORCID ID: https://orcid.org/0009-0007-8767-1032

ARCHITECTURE OF A WEB-BASED SYSTEM FOR THE ANALYSIS OF TACTICAL PATTERNS AND THE AUTOMATED DETECTION OF ANOMALOUS DEVIATIONS IN CHESS GAMES

Abstract

The article discusses approaches to building a distributed web-based information system for the intelligent analysis of sports data that combines tactical pattern categorization and anomalous play detection within a single software solution. The aim of the study is to design a conceptual microservice architecture, develop the mathematical framework, and implement a scalable web application capable of automating the full cycle of processing, retrospective analysis, and statistical evaluation of digital chess records in PGN format.

The methodological foundation consists of software engineering patterns, distributed systems theory, and mathematical statistics, in particular the calculation of average centipawn loss, the Z-score statistical deviation index based on the three-sigma rule, and the match rate metric relative to top chess engine evaluation lines.

A modular approach to design was used: the user management business logic, the analytical core, relational data storage, and inter-service communication mechanisms were separated. As a result of the research, a system topology was proposed that ensures the integrated execution of two key tasks-formalizing an individual playing style and ensuring academic or sporting integrity against computer assistance (cheating). The paper describes the core relational system entities, the mechanism for asynchronous non-blocking request dispatching via

HTTP protocol, and the approach to service deployment, in which the base API is implemented in C# (.NET Core), and the resource-intensive analytical processing module is implemented as a separate service on FastAPI (Python) interacting with the Stockfish engine via the UCI protocol.

The practical significance of the results lies in the ability to use the proposed solution by chess referees, coaches, and analysts for monitoring fair play, conducting deep performance evaluation, and automated generation of analytics reports without the need for local infrastructure deployment. The developed system can serve as a foundation for further improvement through the integration of time-series analysis for time management, adaptation of models to shortened time controls (blitz and bullet), and the implementation of deep learning methods for advanced tactical style clustering.

Keywords: distributed systems, microservice architecture, chess analytics, centipawn loss, Z-score, FastAPI, .NET Core, anti-cheating.

Introduction. The rapid digitalisation of the sports industry and the integration of cutting-edge information technologies into traditional analytical processes have significantly transformed the methodology of training and performance evaluation in modern competitive chess. Thanks to the emergence and scaling of global gaming platforms (such as Chess.com, Lichess, etc.), chess is increasingly evolving towards Big Data, where millions of games are generated daily in digital formats. This vast array of information contains unique hidden regularities, behavioural markers and strategic patterns, the expert analysis of which is practically impossible to carry out manually due to limited human resources and the highly time-consuming nature of the process.

In this context, there is an urgent scientific and practical need to create automated intelligent systems capable of performing comprehensive retrospective and real-time analysis of digital records of chess games. The importance of this

task stems from two key aspects. On the one hand, there is a need to formalise and categorise a chess player's individual playing style (identifying favourite tactical manoeuvres, opening repertoire, typical mistakes and areas of risk), which forms the fundamental basis for constructing an effective training process. On the other hand, the issue of ensuring academic and sporting integrity in the context of remote competitions has become critically important. The availability of extremely powerful local and cloud-based chess engines has led to an unprecedented proliferation of computer assistance (the unauthorised use of third-party software during a game – so-called cheating).

Solving this problem requires combining modern software engineering methods, distributed systems theory, statistical analysis and machine learning within a single architectural solution. The creation of a scalable web-based platform that allows referees, coaches and analysts to seamlessly upload large volumes of match data, automatically calculate metrics of deviations from the initial lines of computer analysis, and classify playing patterns is an important scientific and technical challenge. It directly influences the development of sports analytics, the cybersecurity of gaming platforms, and the methodology of data mining.

Analysis of recent research and publications. The issue of data mining in sports disciplines, particularly in chess, has been the focus of many researchers in the field of applied computer science and artificial intelligence over the past decade. General approaches to constructing evaluation functions and search algorithms in the game are laid out in the fundamental works of the developers of classical chess engines, which are based on alpha -beta pruning, as well as in modern research into deep neural network architectures such as NNUE (Efficiently Updatable Neural Networks) and reinforcement learning algorithms, implemented in projects such as AlphaZero and Leela Chess Zero [1-3].

Kenneth W. Regan has made a significant contribution to the mathematical modelling of a chess player's behaviour and the identification of deviations from natural human play [4, 5]. His mathematical models of game thinking are based on the analysis of probability distributions of human move selection depending on the complexity of the position and the engine's evaluation, which allows statistical deviations and suspicion indices to be calculated using probability theory and the Chi-square goodness-of-fit test. Scientific journals and works of industry experts have also extensively covered the use of classical machine learning methods, in particular support vector machines, random forests and logistic regression, for classifying playing styles based on features such as average game duration, frequency of tactical combinations and patterns of piece manoeuvring [6, 7].

However, despite the availability of powerful mathematical tools and local analysis scripts, the engineering and technological aspects of creating open, integrated systems have not been sufficiently covered in the literature. Most existing solutions fall into two distinct categories:

1. closed commercial anti-cheat systems of major gaming platforms (e.g., Chess.com algorithms), whose internal architecture, data models and metrics are trade secrets and inaccessible to external researchers or local federations;

2. local academic scripts (most often written in Python), which effectively solve narrow tasks of analysing individual PGN files, but have low scalability, do not support multi-user data processing and require a high level of user expertise to configure the infrastructure (engines, databases, etc.).

The previously unresolved aspects of the overall problem lie in the absence of flexible architectural solutions that combine the analytical potential of chess microservices and classical machine learning models with reliable enterprise-grade backend platforms within a single web-oriented solution. This article is dedicated

to addressing this technological gap by designing and justifying the architecture of a distributed web system that separates the resource-intensive computations of chess analysis from the business logic and provides the end user (referee or coach) with a convenient toolkit without the need for local deployment of heavy software.

Aim of the article (statement of the problem). The aim of this study is to design a conceptual microservice architecture, develop the mathematical framework, and implement a scalable web-based information system that automates the full cycle of processing, retrospective analysis, and categorisation of digital records of chess games in PGN format.

To achieve this objective, the following tasks must be addressed:

1. propose and justify the topology of a distributed web system that separates the business logic of user management and report storage from the resource-intensive computations of chess analysis;

2. develop a relational data model and database schema that ensures the storage of structured information about the course of games, accuracy metrics and detected anomalies;

3. formalise the mathematical framework for evaluating game metrics (in particular, average centipawn loss and the Z-score statistical deviation) to construct an algorithm for the initial detection of anomalous play;

4. demonstrate the software implementation of the core analytical microservice based on the Python programming language, utilising specialised chess libraries and an interface for interacting with chess engines via FastAPI.

Main part. The development of high-load information systems for the intelligent analysis of sports data requires the use of architectural patterns that ensure high scalability, fault tolerance and the ability to flexibly allocate computational resources. The process of processing and analysing chess games

using modern engines (such as Stockfish) is characterised by high utilisation of the central processing unit (CPU) or graphics processing unit (GPU), as the search for variations and the calculation of position evaluations for each move require a significant number of computational operations per second.

With this in mind, a distributed microservice architecture for the system was designed and implemented as part of this research. Conceptually, it consists of two key isolated components:

1. Core backend service implemented on the enterprise-grade .NET Core platform (C# programming language). This microservice is responsible for user authentication and authorisation, maintaining profiles of chess players, coaches and arbiters, managing queues of requests for analysis, as well as storing final reports and providing access to them via a RESTful API.

2. Analytics service implemented using the Python programming language and the FastAPI web framework. Python was chosen due to the availability of a well-developed ecosystem of libraries for working with chess data (python-chess) and tools for machine learning and statistical analysis (scikit-learn, numpy). This service runs in an isolated container, interacts directly with the compiled binary files of the Stockfish engine via the UCI (universal chess interface) and performs complex mathematical calculations on demand.

Interaction between microservices is organised using an asynchronous HTTP protocol (or message queues), which prevents the main user interface from being blocked during lengthy game calculations. A conceptual diagram of service interaction and load distribution is shown in Fig. 1.

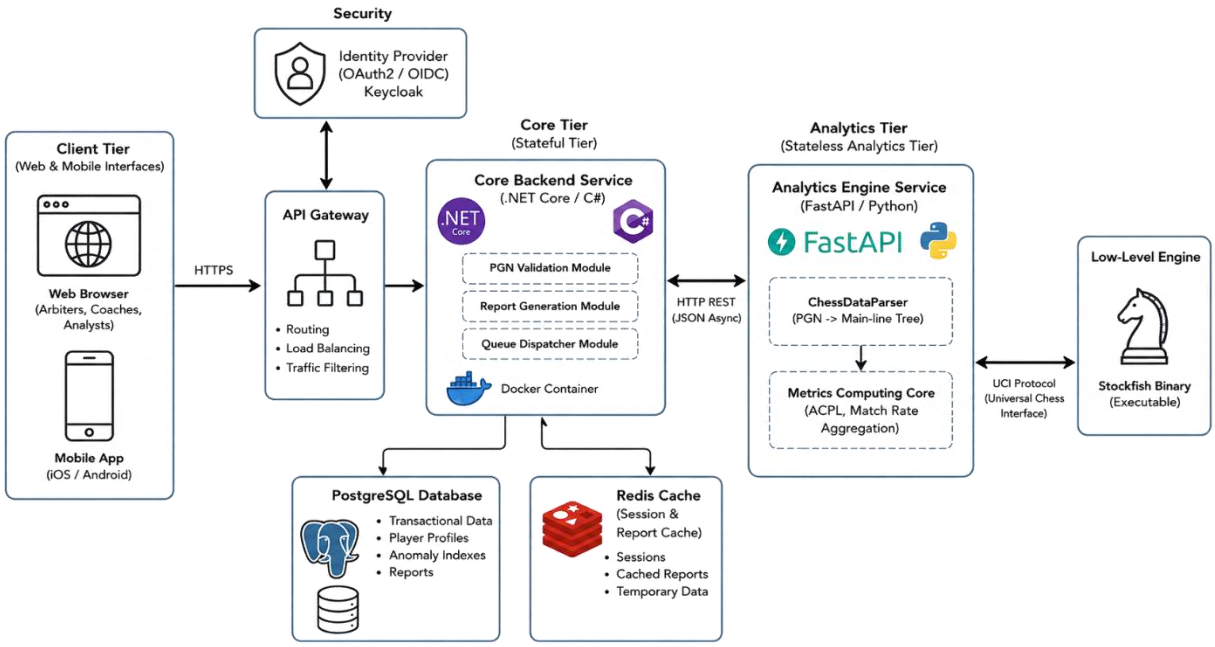


Figure 1 – Conceptual architecture of a distributed system

An important stage in system design is the development of a relational data model, which must ensure the efficient storage of structured information and rapid retrieval for the creation of analytical dashboards. Based on an analysis of the subject area, a database schema was developed (Fig. 2), which contains the following key entities:

– user – stores account details (ID, password hash, role: referee, coach, player, email);

– chess player – an entity representing the profile of a specific player whose games are to be monitored (surname, current Elo rating, national federation);

– chess game describes the general metadata of the uploaded game (date, tournament, result, player's piece colour, link to the PGN file);

– game move – a weak entity associated with a specific game. It stores the move number, notation in SAN format, timestamp (number of seconds spent per move), and the engine's position evaluation before and after the move;

– the analytical report links the user who made the request to the game being analysed. It contains the resulting integrated metrics: average centipawn loss

(ACPL), Z-score, identified stylistic patterns (e.g., a tendency towards tactical complications) and the system's final verdict on the presence of anomalies.

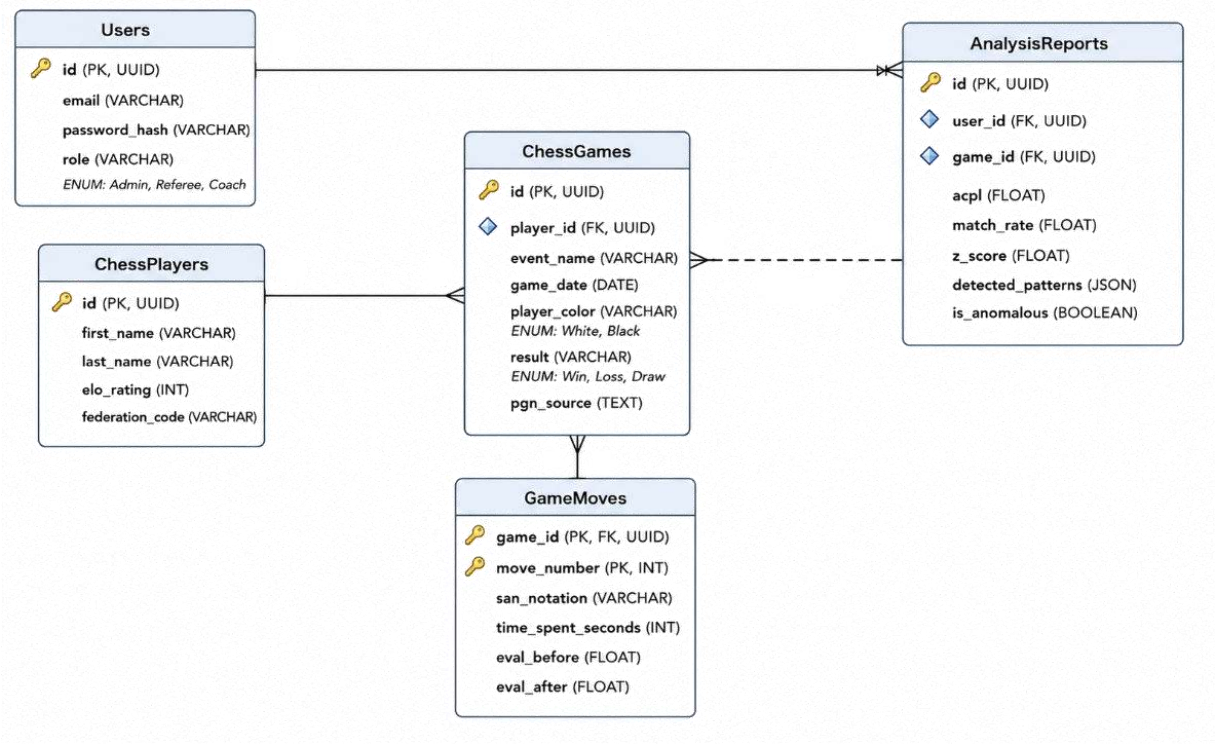


Figure 2 – ER diagram

For the mathematical formalism of evaluating the quality of moves and identifying potentially anomalous deviations in a chess player's actions, Centipawn Loss (CPL) [8] was chosen as the base metric. A centipawn is one-hundredth of a pawn and serves as the standard unit of measurement for advantage in modern chess engines.

Let a game consist of a sequence of moves by player $T = \{t_1, t_2, \dots, t_n\}$, where *n* is the total number of moves made by the player under study. For each move t_i the chess engine calculates, at a given fixed search depth (e.g., depth = 20), a vector of evaluations for the best possible continuations in the position.

Let $E_{max}(i)$ denote the engine's evaluation of the absolutely best (optimal) move in the position prior to move t_i, and $E_{act}(i)$ denote the engine's evaluation of the move t_i actually made by the player. Then the loss of centipawns on the i-th

move (ΔCPL_i) is defined as the difference between the mathematically optimal value and the value of the actual move:

$$\Delta CPL_i = E_{max}(i) - E_{act}(i) \quad (1)$$

By definition, if a player makes the best move from the engine's perspective (the first line), then $\Delta CPL_i = 0$. In the event of a mistake or the selection of a weaker continuation, the value $\Delta CPL_i > 0$.

To obtain an overall assessment of the entire batch, the average centipawn loss (ACPL) is calculated using the following formula:

$$ACPL = \frac{1}{n}\sum_{i=1}^{n} \Delta CPL_i \quad (2)$$

A low ACPL value (0-15 centipawns) indicates exceptionally high playing accuracy, which is unusual for human-level chess players (particularly in complex tactical positions) and is typically only seen in supercomputers.

However, relying solely on the average value is insufficient, as even a beginner can achieve a zero centipawn loss in simple 'forced' positions or endgames with few pieces. Therefore, to identify artificial interference, the system calculates a standardised deviation score (Z-score) for each game relative to the player's historical profile.

Suppose that, for a particular chess player, based on their previous *M* fair games, the mathematical expectation (historical mean) of centipawn loss, μ, and the standard deviation, σ, are known; the latter is calculated using the formula:

$$\sigma = \sqrt{\frac{1}{M}\sum_{j=1}^{M}\left(ACPL_j - \mu\right)^2} \quad (3)$$

The statistical anomaly index (Z-score) for the new batch analysed, denoted by $ACPL_{new}$, is then calculated as:

$$Z = \frac{ACPL_{new} - \mu}{\sigma} \quad (4)$$

In accordance with the principles of mathematical statistics and the law of the normal (Gaussian) distribution:

– If $|Z| \leq 2$, the game falls within the natural variance of the player's ability;

– If $-3 \leq Z < -2$, a significant and suspicious improvement in the accuracy of the game is observed.

– If $Z < -3$, the accuracy of the game exceeds three standard deviations (the three-sigma rule), which, with a probability of $P > 99.7\%$, indicates the presence of external computer assistance (an anomaly), as a human is physically incapable of consistently demonstrating such performance.

In addition, the analytical microservice calculates the Match Rate (MR), which reflects the percentage of a player's moves that fully match the first or second lines of the Stockfish engine in critical (unforced) positions:

$$MR = \frac{k_{match}}{n_{critical}} \times 100\% \quad (5)$$

where k_{match} is the number of moves made by the computer's first lines, and $n_{critical}$ is the number of moves in positions where the player had a choice between several alternative options with different ratings. The combination of the ACPL, Z-score and MR metrics provides a reliable mathematical basis for decision-making by the system's anti-cheating module.

To ensure compliance with enterprise development standards, high modularity and maximum isolation of computational threads, the core of the analytical microservice has been broken down into two logical service components. The first component is responsible for parsing, validating and processing incoming native data in PGN format [9], whilst the second component encapsulates the asynchronous logic for interacting with the chess engine via the UCI transport protocol and the calculation of differential metrics.

Below is the software implementation of the first module (written in Python with chess library [10]), which performs the import, stream-based parsing of the move tree, and the transformation of the text array of information into a structured game state object.

class ChessDataParser:

```
    @staticmethod
    def parse_pgn_stream(pgn_data: str) -> chess.pgn.Game:
        if not pgn_data or not pgn_data.strip():
            raise ValueError("The input PGN stream must not be empty")
        pgn_io = io.StringIO(pgn_data)
        game = chess.pgn.read_game(pgn_io)
        if game is None:
            raise ValueError("Parsing failed: the PGN structure is corrupted or invalid")
        return game
```

The ChessDataParser class described here implements the object factory pattern, transforming raw text into a main-line move tree. The next component, which is the most critical in terms of CPU utilisation, is the computational service that directly manages the Stockfish chess engine subprocess and implements mathematical algorithms for evaluating positions.

To ensure the scalability of computational processes and the implementation of the Separation of Concerns pattern, the architecture of the ChessAnalysisEngine analytical component is based on the principle of hierarchical decomposition. The top level of abstraction is responsible for initialising the interaction session with the chess engine via the UCI transport protocol, iterative management of input data streams, and the final statistical aggregation of calculated metrics. The software implementation of the base control interface class, which handles query dispatching and the lifecycle management of system resources, is shown in the listing below (written in Python).

```
class ChessAnalysisEngine:
    def __init__(self, engine_path: str, default_depth: int = 20):
        self.engine_path = engine_path
        self.default_depth = default_depth
```

```
async def compute_game_metrics(self, game: chess.pgn.Game, custom_depth: Optional[int] = None) -> dict:
    depth = custom_depth or self.default_depth
    transport, engine = await chess.engine.popen_uci(self.engine_path)
    board = game.board()
    total_cpl: float = 0.0
    moves_count: int = 0
    match_count: int = 0
    try:
       for move in game.mainline_moves():
          cpl, is_match = await self._evaluate_single_move(engine, board, move, depth)
          if cpl is None:
             continue
          total_cpl += cpl
          moves_count += 1
          if is_match:
             match_count += 1
    finally: await engine.quit()
    acpl = round(total_cpl / moves_count, 2) if moves_count > 0 else 0.0
    match_rate = round((match_count / moves_count) * 100, 2) if moves_count > 0 else 0.0
    return {"acpl": acpl, "match_rate": match_rate, "evaluated_moves": moves_count}
```

To achieve high cohesion and isolate the computational context, the logic for the atomic evaluation of each game state transition has been encapsulated within a protected asynchronous method. This low-level service component interacts with an instance of the UCI process, performs a double scan of the position tree (before

and after a player's move is recorded) and calculates differential deviations in centipawns. The source code for the secure method of evaluating individual game situations and detecting the first computer line is presented below.

```
async def _evaluate_single_move(self, engine: chess.engine.UciProcess, board: chess.Board, move: chess.Move, depth: int) -> Tuple[Optional[float], bool]:
    analysis_before = await engine.analyse(board, chess.engine.Limit(depth=depth))
    score_before = analysis_before["score"].white() if board.turn == chess.WHITE else analysis_before["score"].black()
    if score_before.is_mate():
        board.push(move)
        return None, False
    best_eval = score_before.score(default=0)
    top_engine_move = analysis_before["pv"][0] if "pv" in analysis_before else None
    is_match = (move == top_engine_move)
    board.push(move)
    analysis_after = await engine.analyse(board, chess.engine.Limit(depth=depth))
    score_after = analysis_after["score"].black() if board.turn == chess.WHITE else analysis_after["score"].white()
    actual_eval = -score_after.score(default=0)
    cpl = max(0.0, float(best_eval - actual_eval))
    return cpl, is_match
```

To ensure the integrity of a distributed information system, it's critical to design a robust protocol for inter-service communication. As the FastAPI-based analytics microservice is stateless and focuses exclusively on mathematical

calculations, the coordination of the request lifecycle is handled by the main backend service (.NET Core).

Interaction between the application gateway, the PostgreSQL relational database, and the compute nodes is implemented using an asynchronous HttpClient within the REST architectural style. Upon receipt of a new request from the user (a .pgn file), the business logic service performs initial string sanitisation, registers the transaction in the database with a 'Processing' status, and forwards the deserialised JSON payload to the secure endpoint of the analytics module. The asynchronous non-blocking waiting mode allows the web server to process other client connections in parallel, minimising infrastructure downtime.

To confirm the functionality and validate the mathematical models of the designed architecture, comprehensive system testing was carried out using real sets of game data. The test sample consisted of groups of chess players of varying skill levels (from amateurs to grandmasters), as well as move sequences generated by commercial chess programs.

It has been established that, when analysing games played by players in the mass ranks (Elo rating 1200-1600), the ACPL varies between 45 and 80 units, whilst the MR doesn't exceed 35-42%. For qualified masters and grandmasters (rating >2500 Elo) in complex strategic games, the ACPL metric drops to between 18 and 30 centipawns, whilst the MR rises to 55–68%.

Figure 3 shows a graph of the distribution of differential accuracy indicators, generated by the information system's analytical microservice based on the results of a comparative analysis of a test sample of games. The visualisation illustrates the mathematical and stylistic difference between natural human play (with its inherent error dispersion) and artificially simulated computer intervention.

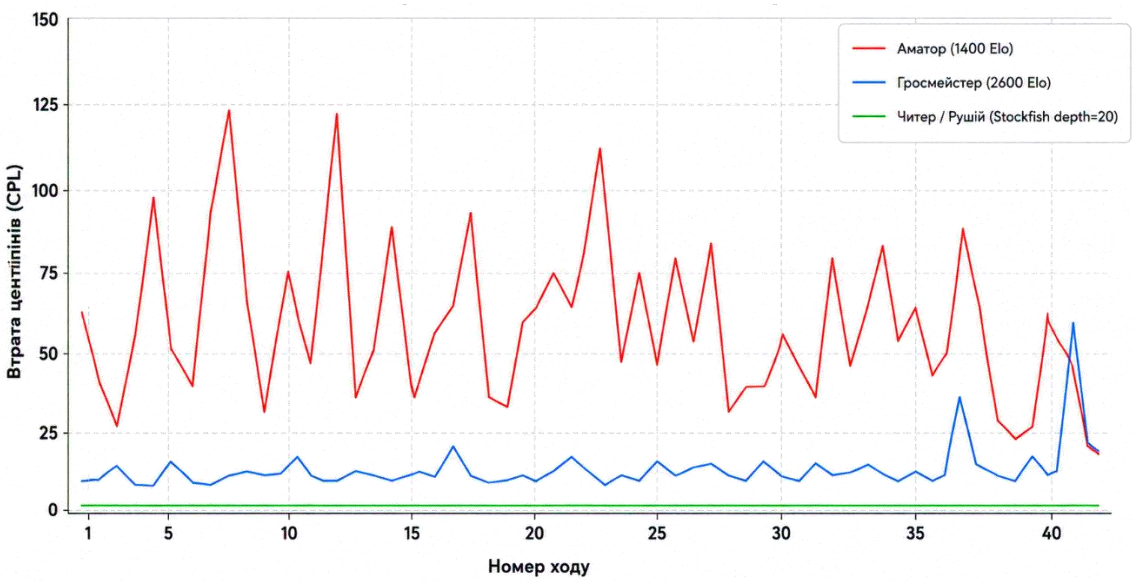


Figure 3 – Graph showing the distribution of centipawn losses across different categories of players

Graphical analysis confirms that the anomalous profile of computer hints is devoid of any signs of human factors (fatigue, emotional pressure, miscalculations). This visualised gap in the values provides empirical confirmation of the effectiveness of the system's mathematical software.

When simulating artificial intervention (or cheating), where a third-party engine with a search depth of depth=20 was used to select moves, the developed microservice recorded a drop in the ACPL metric to an anomalous level of 2.4–6.1 centipawns, and the match rate with Stockfish's first line reached 92.5%. The standardised deviation index, calculated based on the player's historical profile, was Z=-3.84, which, in accordance with the three-sigma rule, allowed the system to automatically generate an analytical report with a verdict on the presence of a critical anomaly and a high probability of the use of computer hints.

Conclusions from this study and prospects for further research in this area. Within the scope of this study, a pressing scientific and practical challenge has been successfully addressed: the design and justification of a conceptual architecture for a distributed, web-based information system for the automated

analysis of tactical patterns and the detection of anomalous deviations in chess games.

The main scientific and practical results of the work:

1. A distributed system topology has been proposed with a clear separation of the business logic for managing user accounts and analytical reports (implemented on the enterprise-grade .NET Core platform) from a resource-intensive analytical microservice based on Python and FastAPI. This approach has ensured high scalability, fault tolerance and isolation of the computationally intensive chess analysis threads.

2. A relational data model has been developed that optimises the storage of structured information on the course of games, move-by-move position evaluations and integral anomaly indices, ensuring rapid report generation for the end user.

3. A mathematical framework has been formalised, based on the detection of the centipede loss differential, average centipede loss and the statistical deviation coefficient Z-score, using the player's historical profile. Experimental testing confirmed the model's ability to identify anomalous play with high accuracy (P>99.7%), indicating the use of third-party software.

4. A professional software implementation of an asynchronous analytical microservice core with strict data typing and guaranteed lifecycle control of the Stockfish chess engine processes has been demonstrated, which prevents RAM leaks on the server infrastructure.

Prospects for further research in this area are aimed at expanding the system's analytical capabilities. A critically important direction is the integration of time-series analysis modules to evaluate a player's time management (the ratio of time spent on a move to the complexity of the current position on the board). There're also plans to adapt mathematical models to the specifics of shortened time controls (blitz and bullet), where the variance of human errors is significantly

higher, and to introduce deep machine learning methods for a more detailed clustering of individual tactical styles among chess players.

References

1) Campbell M., Hoane A. J., Hsu F. H. Deep Blue. Artificial Intelligence. 2002. Vol. 134, Iss. 1-2. P. 57–83. DOI: 10.1016/S0004-3702(01)00129-1.

2) Silver D., Hubert T., Schrittwieser J. etc. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science. 2018. Vol. 362, Iss. 6419. P. 1140–1144. DOI: 10.1126/science.aar6404.

3) Romstad T., Costalba M., Kiiski J., Linscott G. Stockfish: Strong open-source chess engine. URL: https://stockfishchess.org (accessed: 25.04.2026).

4) Regan K. W., Haworth G. M. Intrinsic Chess Rating. AAAI Conference on Artificial Intelligence: proceedings of the Twenty-Fifth AAAI Conference, San Francisco, California, USA, August 7–11, 2011. Palo Alto: AAAI Press, 2011. P. 834–839. DOI: 10.1609/aaai.v25i1.7951 (accessed: 25.04.2026).

5) Biswas T., Regan K. Measuring Level-K Reasoning, Satisficing, and Human Error in Game-Play Data. 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), Miami, FL, USA, 9–11 December 2015. 2015. DOI: 10.1109/icmla.2015.233 (accessed: 26.04.2026).

6) Rosales Pulido H. A. Predicting the outcome of a chess game by statistical and machine learning techniques : Projecte Final de Màster Oficial / UPC, Facultat d'Informàtica de Barcelona, Departament d'Estadística i Investigació Operativa. Barcelona, 2016. URL: https://upcommons.upc.edu/server/api/core/bitstreams/0b6f26cb-0b1d-4c82-817e-a3f3440f466d/content (accessed: 26.04.2026).

7) Cheng Isaac, Camargo Chico. Machine Learning to Study Patterns in Chess Games. 2023. DOI: 10.13140/RG.2.2.30894.52807

8) Linville R. Understanding average centipawn loss in chess. chess.com. URL: https://www.chess.com/blog/raync910/average-centipawn-loss-chess-acpl (accessed: 30.04.2026).

9) Chess games. lichess.org. URL: https://database.lichess.org (accessed: 02.05.2026).

10) Fiekas N. Chess 1.11.2. PyPI. URL: https://pypi.org/project/chess/ (accessed: 02.05.2026).