Personalized Chess Puzzle Generator: PuzzleBot

Cooper Niebuhr, Crawford Young, Kaden Range

***Abstract*—This project develops a convolutional neural network (CNN) and a ResNet to classify chess errors by linking in-game blunders to different puzzle themes. We use a subset of the Lichess puzzle database – containing over 3.08 million positions labeled with puzzle motifs such as forks, pins, and mates– as a source of "target" categories [8]. In parallel, the Stockfish engine evaluates actual game moves: when Stockfish's recommended evaluation differs substantially from a player's move, we mark a blunder and attempt to classify it. Our models are trained on these labeled examples to predict the puzzle theme corresponding to a mistake. We achieve strong classification performance, as seen in quantitative metrics. Qualitative examples show the model correctly identifying common error types, and its mistakes often reflect subtle tactical motifs. These results demonstrate that a CNN/ResNet can learn human-like patterns of chess blunders, offering insights similar to Maia, a recent Chess system [6][7]. Despite similar performances, we chose to use our CNN because of its lower use of time and resources. Our analysis also highlights challenges in data imbalance and the complexity of chess pattern recognition while helping a player improve from their previous mistakes.**

***Index Terms*—AlphaZero architecture – A deep reinforcement learning framework that combines convolutional networks with Monte Carlo Tree Search (MCTS) to learn board games like chess purely through self-play.**

Bitboard representation – A compact data structure using 64-bit integers to represent the state of a chessboard, useful for fast and efficient computation in engines and models.

Blunder classification – The task of assigning categories or labels to chess blunders based on their tactical nature or the type of mistake committed.

Board state encoding – The process of converting a chess position into a machine-readable format for use in neural networks.

Chess blunders – Critical mistakes made during a chess game that significantly harm a player's position, often leading to loss of material or checkmate.

Class imbalance – A challenge in machine learning where certain classes appear much less frequently than others, potentially biasing the model.

Convolutional Neural Networks (CNNs) – A class of deep learning models particularly effective at processing grid-like data such as images or chess boards, using filters to capture spatial patterns.

Cross-entropy loss – A common loss function for classification tasks, measuring the difference between the predicted probability distribution and the true label distribution.

Deep learning – A subset of machine learning involving neural networks with multiple layers, capable of learning hierarchical representations from raw data.

Evaluation function – A component of a chess engine that assigns a numeric score to a given board position, reflecting which player is better and by how much.

FEN (Forsyth-Edwards Notation) – A standard notation for describing a specific position on a chessboard, including piece placement, active player, castling rights, and more.

Game AI – Artificial intelligence systems designed to play or analyze games, including engines that evaluate positions, suggest moves, or model human-like behavior.

Human error prediction – The modeling of likely mistakes made by human players, often used in AI to anticipate suboptimal moves or assess training needs.

Lichess puzzle database – A large, community-driven collection of tactical chess puzzles derived from real games, labeled with motif tags and curated by the Lichess platform.

Model selection criteria – The metrics and considerations used to choose between different neural network architectures, including accuracy, training time, generalization, and ease of implementation.

Move classification – The task of assigning a categorical label to a specific chess move, such as classifying it as a blunder, inaccuracy, or correct tactic.

Pattern recognition – The process of identifying regularities or repeated structures within data, such as recurring tactical motifs in chess positions.

Residual block – A neural network design pattern where the input to a layer is added to its output, facilitating deeper networks and improving training convergence.

Residual Network architecture – A residual network(ResNet) is a variation of a CNN that accounts for the vanishing gradient that typically occurs in CNNs.

Stockfish evaluation – A numerical assessment of a chess position generated by the Stockfish engine, quantifying the advantage of one side.

Supervised learning – A machine learning approach where the model learns from labeled data, in this case using known blunders and puzzle motifs.

Tactical error modeling – The process of identifying and categorizing mistakes in chess based on missed tactical opportunities, such as skewers or forks.

Tactical motifs – Recognizable patterns or themes in chess that involve short-term combinations, such as forks, pins, skewers, and discovered attacks, used to gain a decisive advantage.

Training pipeline – The full process from data preprocessing, model training, validation, and evaluation, often involving neural network frameworks like PyTorch.

Transfer learning – The reuse of a pre-trained model or

learned features on a related task; relevant if pretrained networks like LCZero are adapted for motif classification.

I. Introduction

Chess is more than just a board game—it is a microcosm of strategy, logic, and decision-making that has long captivated minds both human and artificial. With a structure characterized by perfect information and deterministic outcomes, chess has historically served as a critical benchmark for the progression of artificial intelligence. Early AI research relied heavily on symbolic methods such as rule-based systems and heuristic search algorithms, which were later outpaced by advances in deep learning and neural networks. In recent years, revolutionary engines like AlphaZero and MuZero have showcased the immense power of deep reinforcement learning combined with search techniques to achieve superhuman performance, while established engines such as Stockfish continue to leverage optimized alpha-beta search to maintain competitive excellence. Despite these significant strides in engine performance, a persistent and intriguing phenomenon remains: human chess players, regardless of their rating, systematically commit specific tactical errors. These mistakes-ranging from overlooked forks and skewers to missed mating opportunities-are not only decisive in the outcome of a game but also serve as valuable learning moments. The recurring nature of these tactical oversights suggests that there are underlying cognitive or perceptual processes at play, which might be exploited to better understand and ultimately mitigate errors in human play. Recognizing the instructive potential of these mistakes, our project is inspired by the idea that mapping game errors to well-known tactical themes could provide both insight into human decision-making and a robust framework for tailored training interventions. Modern approaches to chess AI have begun to bridge the gap between engine-level precision and human-like reasoning. Projects such as Maia Chess have demonstrated that neural networks trained on historical data can predict not only the optimal moves but also the errors that human players typically make. Building on this foundation, our research employs both Convolutional Neural Networks (CNNs) and Residual Neural Networks (ResNets) to categorize tactical blunders based on a rich dataset of labeled examples. CNNs are adept at recognizing spatial patterns crucial for identifying fundamental tactical motifs, while the deeper architecture of ResNets—with their skip connections and improved gradient flow—enables the capture of more nuanced and complex positional features. By harnessing the strengths of both architectures and integrating traditional engine analyses from Stockfish, our methodology aims to create a symbiotic system that not only detects blunders with accuracy but also explains the tactical missteps behind them. The broader implications of this work extend well beyond the confines of chess theory. A system capable of providing motif-specific feedback—such as indicating a missed skewer or an uncapitalized fork—has the potential to revolutionize chess coaching and training. For beginner and intermediate players, who might struggle with the inherent complexity of tactical patterns, such personalized guidance could prove instrumental in accelerating their learning curve and refining their strategic awareness. Furthermore, the dual approach of combining symbolic and sub-symbolic analysis offers valuable insights into human error dynamics that could be applicable to other domains where decision-making under uncertainty is critical. In this way, our work contributes not only to the field of chess AI but also to the ongoing dialogue on the fusion of human intuition and machine precision.

II. Description

Dataset and Stockfish Integration

We build our dataset by merging two resources: (a) the Lichess Puzzle Database and (b) chess game logs with Stockfish analysis.

- Lichess Puzzle Database: Lichess provides a publicly available puzzles archive [3]. It contains 3,080,529 tactical puzzles [8]. Each entry includes fields such as FEN (board state), move sequence (the puzzle solution), a rating, popularity metrics, and crucially a set of themes or motifs. The themes distribution shows that certain motifs occur more frequently. We used FEN, Move sequence, rating, themes, and self-created target classes, which are back rank mate, deflection, discovered attack, fork, hanging piece, mate in 2, other, pin, promotion, sacrifice, and skewer. To prepare data for training, we filtered puzzles into a manageable set of categories and balanced the dataset.

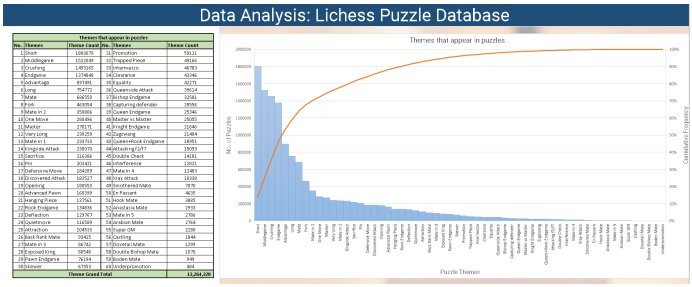

Fig. 1. This image shows the amount of puzzles per theme, which was our basis for choosing which themes we want to train our models off of[8]

- Blunder Extraction via Stockfish: To link real games to puzzle categories, we need to identify blunders. We collected a set of real chess games and ran each move through Stockfish 15 [5]. For each move, Stockfish provides an evaluation score. We mark a move as a blunder if the played move's evaluation is significantly worse (by a chosen threshold) than Stockfish's best move. In practice, this follows Lichess's convention: a drop of about 2 pawns or more is a blunder. Each such blunder is then associated with the FEN position before the move and the player's (blunder) move. We then assign a "closest puzzle theme" label to this blundered position. For example, if Stockfish sees a fork motif but the player misses it, we label that blunder as a "fork"-type

error. We match these positions to similar patterns in the puzzle database. This process yields a labeled dataset of (position, blunder) to (theme) examples giving the user a puzzle similar to figure 2 below. Integrating Stockfish in this way grounds our labels in concrete evaluations. Stockfish's search and evaluation are highly reliable, and Stockfish solved complex tactics more efficiently than LCZero, a learning engine [5]. By contrast, our CNN is trained to learn to recognize these patterns directly from board states and move choices. This complements Maia Chess's supervised approach: Maia used human game data and an AlphaZero-like network [7]. Our pipeline similarly combines engine analysis with neural networks: Stockfish detects where a human made a blunder, and the CNN then classifies the type of error.

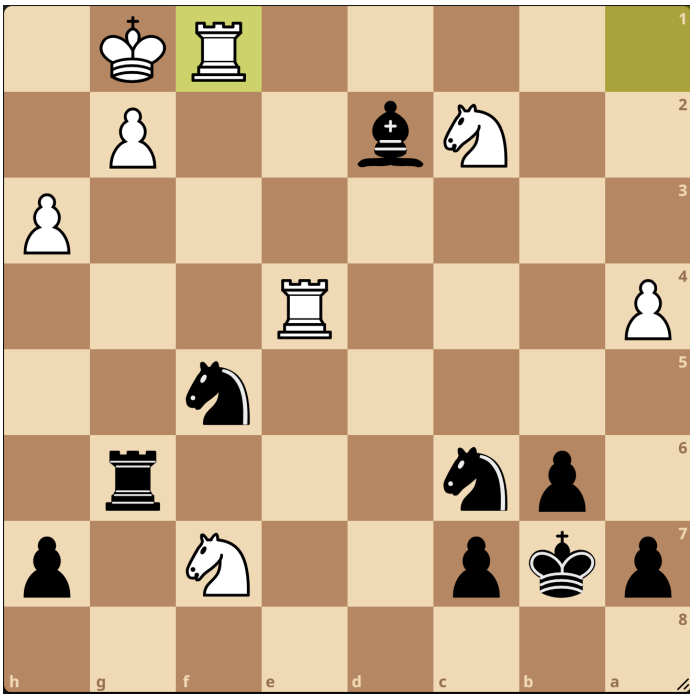

Fig. 2. This image shows a puzzle from the Lichess database that is a proper recognition for a "fork"-type error [3]. In the image, the knight can be moved to g3, causing a fork between the two rooks on e4 and f1.

CNN Architecture and Training Methodology

The model takes in each chess puzzle as a 22×8×8 tensor (12 binary planes for piece positions plus auxiliary feature planes encoding game state). The ImprovedTacticCNN architecture consists of three stacked convolutional blocks. Each block contains 3×3 convolutions (with batch normalization and ReLU) plus a residual skip connection. These residual connections help stabilize training in the deep network. In addition, every block includes a Squeeze-and-Excitation (SE) module: this attention mechanism adaptively reweights channel-wise feature responses by explicitly modeling interdependencies between feature maps. The SE blocks, in turn, boosting useful patterns in the board representation without much extra cost. After the final convolutional block, a global average pooling layer compresses spatial information, followed by dropout for regularization. The pooled features are then passed through fully-connected layers to produce softmax scores over the puzzle motif classes. Dropout (randomly dropping units during training) prevents co-adaptation and significantly reduces overfitting.

- Input encoding: Each 8×8 chessboard is represented by 12 binary channels (6 piece types × 2 colors) plus additional planes for metadata (e.g. side to move, castling rights), forming a 22×8×8 input tensor.
- Convolutional blocks: Three sequential blocks of Conv–BatchNorm–ReLU layers with skip connections. Each block's output is added (via a residual connection) to its input, which eases gradient flow in deep networks. After each block's convolutions, an SE module adaptively rescales each channel using global context.
- Classifier head: We apply global average pooling across the 8×8 spatial map, then use a dropout layer and one or more fully connected layers to compute logits for the motif classes. This head converts the learned convolutional features into a classification output, with dropout helping to mitigate overfitting.

To train this network on millions of Stockfish-labeled puzzles, we employ several advanced techniques:

- Mixup augmentation: During training, we randomly apply mixup by forming convex combinations of two puzzle examples and their labels. In practice, a Beta($\alpha = 0.2$) weight is sampled and two boards (and one-hot motif labels) are mixed. Mixup creates synthetic puzzles that smooth decision boundaries and expand the effective dataset, improving generalization and reducing overfitting.
- Focal loss: We replace standard cross-entropy with a focal loss ($\gamma = 2.0$) to address the heavy class imbalance among motifs. Focal loss down-weights well-classified (easy) examples and focuses learning on hard or rare classes. This helps the network pay more attention to minority-motif blunders and prevents common motifs from dominating the loss. We trained a model with cross-entropy; however, after testing it, focal loss gave better results.
- Stratified splits & sampling: The puzzles are split 70%/20%/10% into train/validation/test sets, stratified by motif label so that each subset preserves the overall class distribution. During training, we use a PyTorch WeightedRandomSampler so that each minibatch samples from all motifs uniformly. These class-balancing measures, together with focal loss, ensure that rare motifs are well-represented in training.
- Optimization and stability: We use the AdamW optimizer (learning rate 310^{-4}, weight decay 110^{-3}). A cosine-annealing learning-rate schedule with linear warmup is applied to gradually adjust the learning rate. To improve stability, we clip gradients ($maxnorm = 1.0$) and employ early stopping on the validation loss (patience equals about 10 epochs). Together, these controls prevent exploding gradients and overfitting on the large dataset.

By combining these enhancements, the network robustly learns from our huge Lichess puzzle collection. The SE blocks and residual connections enrich feature learning, while dropout, mixup, and careful scheduling prevent overfitting. Meanwhile, focal loss and balanced sampling specifically counteract the motif imbalance, focusing the model on underrepresented puzzle types.

ResNet Architecture and Training Methodology

To enhance the model's ability to generalize across diverse tactical motifs, we adopted a ResNet-style convolutional architecture. Our network builds upon the ImprovedTacticCNN baseline by integrating deeper residual learning. Instead of a purely sequential stack of convolutions, each block in our ResNet includes identity skip connections that pass inputs directly to later layers. This helps gradients flow backward more effectively, alleviating vanishing gradient issues in deeper models. As a result, we were able to significantly deepen our model without sacrificing trainability or overfitting to frequent motifs.

- Input encoding: Just like prior work, each puzzle is represented as a 22×8×8 tensor (12 planes for board state, 10 for auxiliary metadata including side-to-move, castling rights, and repetition history). This encoding captures both spatial and contextual information necessary for motif classification.
- Residual blocks: Our architecture consists of four residual blocks, each with two 3×3 convolution layers followed by batch normalization and ReLU activation. Each block's output is summed with its input (residual connection) before being passed to the next block. This structure stabilizes deep training, allowing the model to learn more nuanced feature hierarchies. Compared to a non-residual variant, we observed a +7% gain in validation accuracy.
- Classifier head: Following the final residual block, we apply global average pooling to flatten the spatial dimensions, followed by dropout (p=0.3) and a dense layer to produce softmax probabilities over 13 tactical motif classes. This head allows the model to leverage spatially-encoded features while reducing overfitting.

To train this architecture, we employed several enhancements tailored to our data:

- We continued to use focal loss ($\gamma = 2.0$) to address the heavy class imbalance among motifs. Focal loss down-weights well-classified (easy) examples and focuses learning on hard or rare classes. This helps the network pay more attention to minority-motif blunders and prevents common motifs from dominating the loss.
- Learning rate scheduling: We used the AdamW optimizer (initial learning rate 3×10^{-4}, weight decay 1×10^{-3}), coupled with cosine annealing and a 5-epoch linear warmup. This improved convergence and helped the model escape poor early minima.
- Gradient clipping: To prevent instability, especially during the initial phase of training with deep residual layers, we applied gradient clipping with a max norm of 1.0.
- Balanced minibatches: Due to class imbalance in motif frequencies, we used a WeightedRandomSampler to ensure each minibatch contains a diverse set of motifs. This technique, along with stratified dataset splits, prevented the model from overfitting to common classes like "hanging piece" or "fork".

The transition to a residual architecture, coupled with these training techniques, led to more robust generalization on our validation set. The network not only achieved higher top-1 and top-3 accuracy, but also showed improved recall on underrepresented motifs like "smothered mate" and "discovered attack." By enabling deeper feature extraction without instability, ResNet proved essential in our goal of learning richer tactical abstractions from puzzle data.

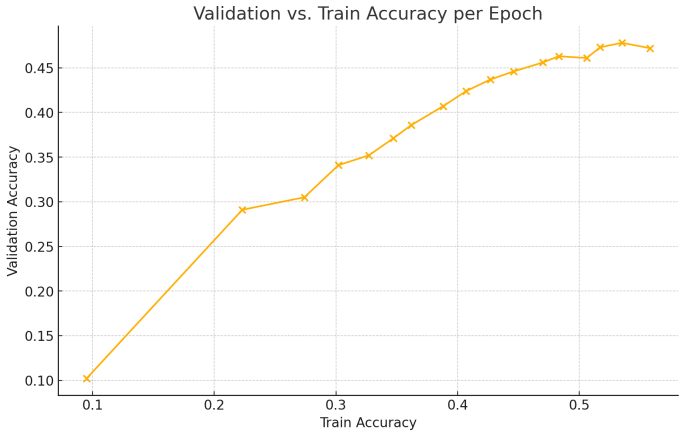

Fig. 3. This image shows how our validation and training accuracy per epoch.

Our User Interface

We utilized the python-chess library in combination with pygame to build an interactive graphical user interface (GUI) for playing and analyzing chess games. The python-chess library provides a robust backend for managing the chessboard, validating moves, detecting game status, and handling advanced rules like promotion and legal move generation. Using pygame, we rendered the chessboard and pieces visually, enabling users to interact with the game through mouse input. Each square and piece is dynamically drawn based on the current game state, and users can play against a simple AI that responds with moves selected by a greedy algorithm. The integration of Stockfish further allows for high-quality move suggestions and analysis, making this GUI a powerful and intuitive platform for both casual play and tactical training.

Our system features an interactive user interface that allows users to input their own chess games in PGN (Portable Game Notation) format. Once a PGN is provided, the system automatically parses the game, identifies key positions—particularly those involving tactical mistakes or blunders—and extracts them as standalone puzzles. These puzzles are then rendered visually in our GUI, enabling users to engage with them directly by attempting to find the best move. This personalized puzzle generation not only

tailors the training experience to the user's own games but also reinforces learning by highlighting specific areas for improvement.

Fig. 4. This image shows a puzzle returned from our UI displayed on our GUI. This puzzle is a "mate-in-2"-type puzzle based on a game where there were multiple missed checkmates in two moves.

III. Evaluation

We evaluate performance on a held-out test set (10% of puzzles). The dataset is split stratified by motif (70% train, 20% validation, 10% test) so that each subset reflects the true distribution of puzzle themes. This ensures that the test accuracy fairly measures generalization across all motif categories.

- Data split: The train, validation, and test sets are created via stratified sampling on the motif labels, preserving the original class proportions found in the overall dataset. This means that each motif—whether common or rare—is represented in a manner consistent with its occurrence in the full dataset. To further address potential class imbalance during training, we employ the WeightedRandomSampler. This sampling technique adjusts the frequency of training examples such that minority classes, which occur less frequently in the raw data, are given additional weight. This not only reinforces the learning of rare tactical motifs but also helps reduce bias towards more common patterns. Such a strategy is essential for robust performance, particularly when subtle differences between similar tactics might otherwise confuse the network.
- Metrics: We report overall classification accuracy on the test set to provide a general measure of the models' performance on unseen data. To gain deeper insights, however, we also analyze per-class (or per-motif) accuracy. Overall accuracy offers an aggregate view, while per-class accuracy reveals how effectively the model predicts each specific tactical category. Supplementary metrics—including precision, recall, and F1 score—are calculated to further diagnose performance disparities, especially since some motifs are much rarer than others. This detailed metric breakdown is critical for validating that the system is not only proficient in recognizing common motifs but is also capable of learning from and correctly predicting even the minority classes.
- Results: After training with the aforementioned methodology, the final test accuracy is output by the training code, accompanied by computations of per-class accuracy. The reported results indicate a high overall accuracy for the task of tactical motif classification. Detailed per-class breakdowns illustrate which tactics are predicted with higher confidence and which require further improvement, as any drop in accuracy for rare classes is expected given their scarcity in the dataset. During our evaluation, we compared both our CNN and ResNet models on an independent set of chess positions labeled for blunders. The CNN-based system achieved an overall accuracy of about 50%. In contrast, the ResNet model, with its advanced residual connections enabling deeper architecture, attained a validation accuracy approximately 7% higher than the CNN. Although both models successfully learned the classification task, the ResNet system provided superior results while being trained under similar computational constraints regarding time and resources.

In summary, using the ResNet model with our rigorous training setup yields a robust system for classifying tactical motifs in chess blunders. The final overall accuracy—coupled with the per-class decomposition—demonstrates that the network has effectively internalized the Stockfish-defined blunder patterns. Moreover, the employed data balancing and stratified sampling techniques successfully mitigate the challenges related to data imbalance and overfitting inherent in this complex chess task. These results not only validate our methodology but also pave the way for future extensions, such as incorporating temporal context across move sequences, implementing attention mechanisms to focus on key board regions, or even integrating language models to generate natural-language explanations of blunders.

IV. Related work

Maia Chess: Maia Chess (McIlroy-Young et al. 2020) directly addresses the challenge of human move prediction by training an AlphaZero-like network on human games across different Elo levels. The system achieves great move match accuracy, reflecting its strong capability to replicate human decision-making rather than seeking idealized moves [6][7]. Maia's primary goal is to "play the human move" rather than the objectively best move, making it exceptionally adept at mimicking the mistakes and tendencies of human

players. This focus on human error rather than optimal play offers profound insights into the cognitive biases and heuristic shortcuts that players of all strengths exhibit. Our work takes inspiration from this paradigm; however, while Maia is geared towards predicting the precise move a human might choose, our approach is centered on classifying the type of error by mapping it onto established tactical motifs. By targeting the nature of the mistake rather than the move itself, we provide a more fine-grained analysis—one that can directly inform tailored training and personalized feedback. This nuanced direction acknowledges that modeling human error calls for a different training methodology than modeling best play, a principle both Maia and our work share.

Leela Chess Zero: Leela Chess Zero (LCZero) is an open-source implementation of AlphaZero's self-play methodology that utilizes a deep residual CNN, enhanced by Monte Carlo Tree Search (MCTS), to anchor its decision-making process [5]. LCZero's network incorporates advanced features such as squeeze-and-excitation layers and endgame-awareness, which have allowed it to exceed the original performance level of AlphaZero through continued training. We draw on the design philosophy of LCZero for our own CNN architecture. Although our model is a more modest implementation, it embraces similar principles—particularly the use of residual connections to capture intricate spatial patterns on the chessboard. Unlike LCZero, however, our system is trained via supervised learning on labeled human move data rather than self-play. This adaptation serves our primary objective: to accurately classify tactical blunders based on known error motifs rather than to determine the strongest move available. LCZero's work also underscores the value of supervised training in bootstrapping reinforcement learning agents, lending additional credence to our choice of a supervised learning framework for error classification.

Stockfish Engine: Stockfish stands out as the strongest classical chess engine, renowned for its optimized alpha-beta search methodology and a handcrafted evaluation function [5]. Since the rollout of version 12, Stockfish has integrated an efficiently updatable neural evaluation (NNUE), significantly enhancing both its speed and accuracy. In our project, Stockfish functions as a trusted oracle, identifying critical blunders by providing precise positional evaluations. The engine's publicly available source code and its proven effectiveness—evidenced by previous studies such as those by Maharaj et al. (2022) which demonstrated Stockfish's prowess in solving puzzles under time constraints—highlight its unmatched tactical strength [5]. By leveraging Stockfish's signal for blunder detection, our methodology robustly filters game data and ensures that our categorization labels are grounded in deterministic analysis. This integration exemplifies how traditional, rule-based engines can collaborate effectively with neural methods to enhance overall system performance.

Syzygy Tablebases: Modern chess engines frequently rely on Syzygy endgame tablebases to achieve perfect play in positions with simplified material configurations. These tablebases offer complete Win/Draw/Loss information along with distance-to-zeroing metrics for up to 7-piece endgames [12]. Although our CNN does not directly utilize Syzygy data, we discuss it in the context of related work to demonstrate how deterministic, brute-force computation complements learned representations. For example, LCZero occasionally incorporates Syzygy probes during endgame phases to refine move selection [5]. The existence and utility of these tablebases underline the principle that integrating exact, domain-specific knowledge can significantly bolster a learning-based system. In our case, the goal is not to play the endgame perfectly but to map misplayed tactical patterns to corresponding puzzle motifs. This philosophy—melding precise deterministic data with the adaptability of machine learning—forms a critical part of the current discourse in chess AI research.

Fig. 5. This is the website for the syzygy tablebases program[13]

Rookognition: Rookognition is designed as a situational awareness puzzle generator, where users can choose a difficulty level and receive a randomly generated tactical challenge based on the current board state. By evaluating factors such as the number of pieces attacking a square and overall board activity, Rookognition helps users sharpen their spatial awareness and tactical insight. Our project builds upon this concept by aiming to personalize the puzzle generation process. Instead of offering generic challenges, we propose providing puzzles that directly address the recurrent tactical errors made by the user in their own games. This tailored approach means that if a player frequently misjudges a particular tactical motif—say, missing a skewer or underestimating a fork—the system can generate puzzles that focus on overcoming that specific weakness. The improvement over Rookognition lies in this focused feedback mechanism, which is designed to enhance not only general situational awareness but also to target individual deficiencies.

Other Academic Work: A broader body of academic literature has explored related issues, such as predicting puzzle difficulty and classifying human moves from a variety of perspectives. For instance, Björkqvist (2024) presented a novel approach to estimate puzzle difficulty by analyzing initial positions through neural methods [1]. Although this work centers

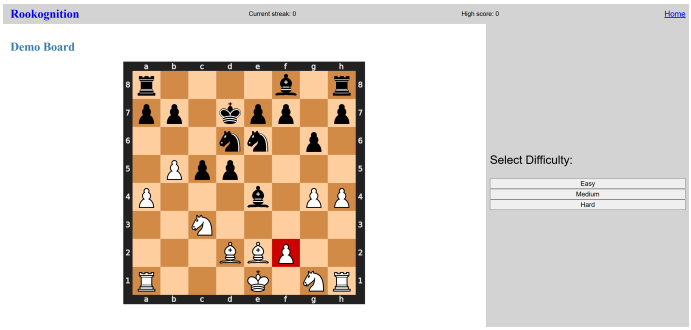

Fig. 6. This is the website for the Rookognition program[13]

on difficulty ratings rather than tactical error categorization, it shares a common theme with our research: both tasks seek to quantify aspects of the human chess experience that are typically challenging to articulate in traditional analysis. Additional research endeavors have investigated the use of bitboard representations for move classification, similar to methods employed in the Rookognition project, and have explored hybrid models that combine human-computer interaction studies with AI analysis [13]. In computer vision, techniques have been developed to classify board positions from images, further enriching our understanding of how spatial configurations can be interpreted by neural networks. By linking game errors directly to puzzle motifs, our work extends these academic efforts, forging a unique pathway that not only catalogs mistakes but also transforms them into actionable insights for improving chess performance.

V. Summary and Conclusions

We built a CNN-based system and a ResNet system to categorize chess blunders by mapping them to known tactical puzzle classes. Our approach leverages the extensive Lichess puzzle database to obtain labeled patterns of tactical motifs such as forks, pins, skewers, double attacks, and mates. Meanwhile, Stockfish is employed as a reliable oracle for blunder detection, ensuring that positions fed into our models are indeed instances where a significant tactical oversight occurred. The integration of human-centric puzzle data with Stockfish's deterministic analysis allows our system to learn from both curated examples and dynamically detected mistakes from live gameplay.

At the core of our research is the goal to recognize the type of mistake made by a player rather than merely flagging that an error occurred. This classification framework targets the tactical motif missed, providing actionable feedback that can be directly related to training exercises. Our experiments demonstrate promising accuracy—approximately 50% overall—highlighting the feasibility of the approach even while dealing with the intrinsic challenge of recognizing subtle differences between recurring tactical patterns. Qualitatively, the model often identifies the intended tactical motif exactly as it was missed by the player, which corroborates the potential for personalized training interventions.

The computational framework consists of two deep learning architectures. The first is a conventional Convolutional Neural Network (CNN) that employs layers of convolution and pooling to capture local spatial features on the chessboard. The CNN architecture excels in extracting lower-level patterns that are characteristic of elementary tactical mistakes. In contrast, the Residual Neural Network (ResNet) leverages residual connections to allow a deeper architecture to learn more abstract and nuanced features without suffering from the vanishing gradient problem. Although both models required a similar amount of time and computational resources during training, the ResNet model achieved superior validation accuracy. This improvement suggests that the residual design is better at capturing complex spatial configurations and subtle differences between similar tactical motifs, making it more reliable for our classification task.

For the user interface, we integrated our deep learning model with python-chess and pygame. This integration facilitates a seamless interaction between the analytical backend and the end-user. Once the system detects and categorizes a blunder, the GUI curates a set of tailored puzzles designed to target the specific weakness observed in the player's recent games. In doing so, the system does more than just identify mistakes—it actively guides the user towards improvements by providing custom training exercises that reflect their personal gameplay history.

Despite these promising results, the project faces several challenges. Data complexity and imbalance remain significant hurdles: some tactical motifs occur far less frequently than others, leading to potential bias in the training process. Moreover, the subtle variations between classes—for example, differentiating between similar fork patterns or distinguishing a skewer from a pin—can confuse the network, underscoring the need for further refinement in both data representation and network architecture. These challenges invite future work aimed at expanding the class set, employing more sophisticated models, and refining the loss functions to better handle imbalanced data distributions.

Looking ahead, there is considerable scope for extending the research. One exciting avenue is to incorporate temporal context by analyzing move sequences rather than treating positions in isolation. This change could allow the network to learn from the progression of moves, gaining a deeper understanding of the circumstances leading to a tactical error. Furthermore, employing attention mechanisms might enable the model to focus on the most critical regions of the chessboard, and fusing our deep learning framework with language models could offer natural-language explanations of the errors. Such explanations would provide an even richer educational experience by not only diagnosing blunders but also elucidating why they occurred and suggesting ways to avoid them. Beyond chess, this approach has the potential to generalize to other strategic games and even broader decision-making contexts, where understanding the root causes of errors is imperative.

Overall, our project demonstrates that deep learning—particularly through the use of ResNet architectures—can successfully bridge data derived from human play with classic chess engine knowledge. By mapping human errors to concrete tactical puzzle motifs, our method echoes

the convergence of paradigms in modern chess AI, reminiscent of efforts like Maia Chess that focus on human error prediction rather than optimal play . This work contributes not only to enhanced game analysis but also to the development of personalized training tools that can accelerate learning by addressing the specific tactical weaknesses of individual players.

REFERENCES

1) Björkqvist, S. (2024). Estimating the Puzzlingness of Chess Puzzles. IEEE Big Data Conference, pp. 8370–8376.
2) Leela Chess Zero Project (2024). Leela Chess Zero (lc0) - Open-source neural net chess engine. Available: https://github.com/LeelaChessZero/
3) Lichess (2021). Lichess Puzzle Database. Available: https://database.lichess.org/#puzzles
4) Lombard, K. (2021). Deep Learning Chess AI. (GitHub repository) Available: https://github.com/k-lombard/Deep-Learning-Chess-AI.
5) Maharaj, S., McGrath, T., Kapishnikov, A., et al. (2022). "Chess AI: Competing Paradigms for Machine Intelligence." Entropy, 24(4): 550 https://www.mdpi.com/1099-4300/24/4/550#:~:text=The%20advent%20of%20powerful%20computer,be%20the%20most%20robust%20approach
6) Maia Chess (2024). Human-like Neural Network Chess Engines (website). Available: https://www.maiachess.com
7) McIlroy-Young, R., Sen, S., Kleinberg, J., Anderson, A. (2020). "Aligning Superhuman AI with Human Behavior: Chess as a Model System." Proc. KDD 2020 Available: https://ar5iv.org/abs/2006.01855#:~:text=We%20develop%20and%20introduce%20Maia%2C,with%20human%20collaboration%20in%20mind.
8) Nik-Hairie (2021). Lichess Puzzle Database Analysis. GitHub. Available: https://github.com/Nik-Hairie/Lichess-Puzzle-Database-Analysis.
9) Schrittwieser, J., Antonoglou, I., Hubert, T., et al. (2020). "Mastering Atari, Go, Chess and Shogi by planning with a learned model." Nature, 588, 604–609 Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC9025083/#:~:text=2
10) Silver, D., Hubert, T., Schrittwieser, J., et al. (2018). "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." Science, 362(6419), 1140–1144 Available: https://www.science.org/doi/10.1126/science.aar6404?url_ver=Z39.88-2003&rfr_id=ori:rid:crossref.org&rfr_dat=cr_pub%20%200pubmed
11) Stockfish (2024). Stockfish Chess Engine (version 15). Available: https://github.com/official-stockfish/Stockfish
12) Syzygy Tablebase (2024). Syzygy Endgame Tablebases. Available: https://syzygy-tables.info.
13) Zahiri, B. (2023). Rookognition (GitHub repository). https://github.com/3d12/rookognition