# Premove Prediction in Online Human Chess

Sitan Huang
huans16@stanford.edu

## 1 Introduction

A *premove* in online chess is a move committed before the opponent's turn ends. The decision to premove stems from a human optimization process that trades off the risk of misanticipating the opponent's reply against the reward of saving clock time. As there are no standard heuristics for this judgment, predicting premoves requires modeling personalized player behavior alongside the rule-based context of the game.

Traditional chess engines search move trees to identify optimal continuations (Silver et al. 2017; Sadmine et al. 2024). To better align with human play, machine learning models predict the next move by training on large datasets of human games (Oshri et al. 2015; McIlroy-Young et al. 2020). Convolutional neural networks (CNNs) have been explored for board-based pattern recognition under this context (Oshri et al. 2015). Later systems such as AlphaZero extended this idea with deep residual towers (ResNets) operating on multi-plane board encodings (Silver et al. 2017; *Neural network topology - Leela Chess Zero* 2025). Further, sequential architectures have been explored using Long Short-Term Memory (LSTM) networks (Pawar n.d.) to predict next moves.

While these models capture the board's spatial context, other works account for the temporal dynamics of human play. Recently, a hybrid CNN-LSTM model that process player move times has shown to improve human behavioral predictions such as ELO estimation (Omori et al. 2025). However, this insight has not yet been applied to premove prediction. Zhang et al. (2024) introduced *ALLIE*, a GPT-2 architecture model trained on human game logs that predicts human-like next moves and move times. Premoves appear as 0-second moves in game logs, yet existing approaches, including ALLIE, do not model premoves as an explicit binary action, and time predictions do not include sub-0.5s behavior (Zhang et al. 2024). Further, we argue that the decision to premove is made during the preceding turn, making the thinking time for the prior move a potentially predictive feature. While works like ALLIE condition on board histories, they omit a player's move-time history, which may contain key information about individual timing strategies.

To our knowledge, no prior works have explicitly modeled premove behavior in online human chess. Our objective is to predict whether the next move will be a premove using spatial board features with temporal timing features. We benchmark three architectures: (i) a CNN in which board histories are encoded as separate input channels, following the multi-plane approach by AlphaZero and others (Silver et al. 2017; Oshri et al. 2015) and allowing convolutional blocks to cross reference present positions with recent history; (ii) a CNN-LSTM that feeds CNN-extracted spatial features and move times in explicit sequences into a LSTM; and (iii) a ResNet-CNN (Liu et al. 2025) with history-stacked planes to test whether deeper residual stacks further improve spatial feature extraction for this task. We **hypothesize** that player move-time history improves premove prediction. We will perform **ablation analysis** to benchmark model performance with and without player move-time history. Modeling of premove mechanics could enhance fair-play enforcement by distinguishing authentic human play from algorithmic execution in low-time scenarios. Moreover, incorporating these modeling into training tools could help new players better calibrate their risk-reward thresholds when managing critical clock deficits.

## 2 Methods

### 2.1 Data curation

We randomly scraped 50,000 public games from *chess.com* archives across 1000 to 3000 player ELOs. Games were restricted to *blitz* chess (3-5 minutes) as this variant concentrates premove behavior while keeping sequence lengths tractable and eliminating the high behavioral variability in *bullet* chess. Parsing used the open-source `python-chess` package (*python-chess 1.11.2* 2025). Model training used `pytorch` (*PyTorch* 2025). Each move in time is an example data point.

Splits were made at the *game* level: a held-out test set of 100 games, and the remaining games split 90%/10% into train/validation. Due to the dataset's scale, the validation split was sufficiently large and did not use the 70%/30% split typical for smaller datasets. Games were shuffled using fixed seeding. A summary of the dataset is outlined below:

- **Train**: 3,521,012 move examples, of which 133,544 were positive (3.79%).
- **Validation**: 359,985 examples, 13,532 positive (3.76%).
- **Test**: 7,154 examples, 316 positive (4.42%).

### 2.2 Premove/label definition

Let `prev_clk` be the mover's clock before a move and `curr_clk` after the move; let `inc` be the per-move increment from `[TimeControl]`. The measured thinking time is

$$\text{spent} \;=\; \max\{0,\; \text{prev\_clk} - \text{curr\_clk} + \text{inc}\} \quad (1)$$

A premove is defined as spent = 0.1 s (per *chess.com* specification), producing a binary label $y \in 0, 1$ (1 = premove).

### 2.3 Feature construction

Examples are built up to each move once sufficient history is present. Two modalities are used:

**Board and board history** We encode a board position with 12 one-hot planes (WP, WN, WB, WR, WQ, WK, BP, BN, BB, BR, BQ, BK), each of size $8 \times 8$. We stack the last $H = 4$ positions preceding the current decision, yielding $12H$ planes. This decision of $H = 4$ was based on pilot experiments, which found that higher values did not yield meaningful gains in validation accuracy. A side-to-move plane (all ones if White-to-move, else zeros) is concatenated at the end, giving $C_b = 12H + 1 = 49$ channels:

$$\text{boards} \in \{0, 1\}^{C_b \times 8 \times 8} \tag{2}$$

The history buffer is updated with the post-move board so that at decision time the last stored position corresponds to the current pre-move board.

In the CNN-LSTM model (as detailed in later sections), the board positions are fed as sequences into the model ($(C_b = 12 + 1) \times 8 \times 8$), with the last channel being the side-to-move plane. The board CNN network is then fed into a layer of LSTM.

**Move-time history and game-context** For the CNN and ResNet-CNN models (as detailed in later sections), we encode the most recent $H = 4$ move times for the mover and opponent; move times are accompanied with an 8-d game context feature vector:

$$\underbrace{(\text{self}_{-1}, \ldots, \text{self}_{-H})}_{H} \parallel \underbrace{(\text{opp}_{-1}, \ldots, \text{opp}_{-H})}_{H} \parallel \underbrace{(\text{base}, \text{inc}, \text{fullmove}, \text{remain\_frac}, \text{phase}, \text{tc}, \text{elo}_{\text{self}}, \text{elo}_{\text{opp}})}_{\textbf{game context features: } 8} \tag{3}$$

where remain_frac = prev_clk/base, phase = $\min(1, \text{fullmove}/40)$, and tc = base/300 (since 300s is the upper bound of a blitz game). In total, non-board features yield $C_t + 8 = 2H + 8 = 16$ scalars per example. Timing features are z-scored per-feature using train-set statistics and reused for all splits.

In the CNN-LSTM model, the mover and opponent move times are fed as pairs of $[\text{self}_H, \text{opp}_H]$ into an MLP before connecting to a layer of LSTM.

### 2.4 Models

Model hyperparameter search was performed to arrive at the following architectures:

**CNN** Convolutional Neural Networks process grid-like data by learning spatial hierarchies of features. Through the use of learnable filters, the CNNs detect local patterns in pieces arrangements that are translation-invariant. The board pathway is a CNN of two $3 \times 3$ convolutions with 256 channels, interleaved with ReLU and BatchNorm, and followed by AdaptiveMaxPool to $1 \times 1$ and a linear layer to a 128-d board embedding. The timing and game context pathway is a 3-layer MLP (dims $C_t + 8 \to 64 \to 32 \to 32$ with ReLU). The concatenated $(128 + 32)$-d vector feeds a 2-layer MLP (32 hidden units) that outputs a single logit for premove prediction. Total trainable parameters: 746,561.

**ResNet-CNN** The ResNet architecture helps to overcome the vanishing gradient problem prevalent in CNNs by allowing gradients to flow from output to the earlier layers via bypass connections (Liu et al. 2025). Specifically, residual blocks utilize skip connections to add the input of a block directly to its output, effectively learning a residual function $F(x) + x$; this allows the model to train significantly deeper networks without vanishing gradients. The board input is a stack of feature planes representing $H$ historical positions, plus an additional plane for the side-to-move, totaling $C_{\text{board}} = 12H + 1$ channels. When the `ignore_history` flag is set, only the most recent position is used ($H = 1$). This input is processed by a residual tower comprising a 3×3 convolutional stem, a stack of 8 residual blocks (2 convs per block, 32 filters), a global max-pooling layer, and a linear layer to produce a 128-d board embedding. Two MLPs map game context features ($8 \to 32$) and move-time histories ($C_t \to 64 \to 32$). The latter is omitted when `ignore_times` is set. A final prediction head concatenates the available embeddings $(128 + 32 + \mathbf{1}_{\text{time}}32) \to 32 \to 1$; if time is ignored it reduces to $(128 + 32) \to 32 \to 1$. Total trainable parameters: 176,577 (full model), 172,897 (sans time history), 166,209 (sans board history).

**CNN-LSTM** LSTM networks are capable of learning order dependence in sequence prediction problems. LSTMs maintain an internal cell state regulated by gating mechanisms, allowing them to capture long-term temporal dependencies across a sequence of moves. We process a window of board positions by sharing a small CNN per step (two 3×3 convs with 256 channels, ReLU, BatchNorm, global 1×1 pool, linear) to obtain 128-d board embeddings, which are fed to a 1-layer LSTM (hidden 128). Simutaneously, at each step we form a 2-vector of *(self, opp)* move times, map it with a step MLP ($2 \to 32$) and pass the resulting sequence to a 1-layer LSTM (hidden 64). The 8-d game context features go through an MLP ($8 \to 32$). We concatenate the embeddings and use a head of $224 \to 64 \to 1$ to produce the output logit. Total trainable parameters: 823,937.

**MLP-only variant** This variant uses only the game context features and excludes all board features and move-time histories. The inputs are processed by a single hidden layer (32 units, ReLU) to form a 32-d context embedding. A two-layer prediction head (32 hidden units, ReLU) maps this embedding to the output logit for premove prediction. This model serves as the minimum benchmark reference. Total trainable parameters: 1,377.

### 2.5 Training

Our data is strongly class-imbalanced. For this reason, we used a weighted random sampler to obtain 50/50 positive-negative mix per batch. We used Focal Loss (Lin et al. 2018) to increase contribution of misclassified examples in the gradient:

$$\begin{aligned} \mathcal{L} \;&=\; \alpha(1-p_t)^{\gamma}\,\mathrm{BinaryCrossEntropy}(\hat{z}, y), \\ \alpha = 0.75,\ \gamma &= 2.0 \quad \text{(Lin et al. 2018)} \end{aligned} \tag{4}$$

where $p_t = \sigma(\hat{y})$ if $y = 1$ and $1 - \sigma(\hat{y})$ otherwise.

We used Adam with lr $= 10^{-4}$, weight decay $= 10^{-4}$, batch size 256, and automatic mixed precision on CUDA. We fit with early stopping on validation balanced accuracy with patience=5.

### 2.6 Evaluation

We evaluate on validation and test sets using precision, recall, false positive rate, and balanced accuracy:

$$\mathrm{BalAcc} \;=\; \tfrac{1}{2}\Big(\tfrac{\mathrm{TP}}{\mathrm{TP+FN}} + \tfrac{\mathrm{TN}}{\mathrm{TN+FP}}\Big) \tag{5}$$

Focal Loss is our training objective; using it for benchmarking would be misleading for several reasons: (i) it is a parameterized objective subject to hyperparameter optimization $(\alpha, \gamma)$; (ii) the magnitude of Focal Loss is sensitive to the class prior and the per-example difficulty mix, so it confounds model quality with dataset composition; (iii) for $\gamma > 0$ it is not a proper scoring rule and can encourage overconfident probabilities, making loss reductions poorly aligned with calibration or downstream decision quality; (iv) it emphasizes "hard" examples by design, which biases comparisons toward architectures that best exploit this reweighting rather than those that yield the best operating characteristics at a fixed threshold. For these reasons, we report the interpretable metrics that most directly reflect real-world use cases — i.e., precision, recall, false positive rate, accuracy, balanced accuracy — on the held-out test dataset, which should be stable across changes to the training objective.

## 3 Results

We benchmarked the three models (i.e., CNN, ResNet-CNN and CNN-LSTM) using the held-out test dataset, which are shown in Table 1. All three models achieved $> 95\%$ balanced accuracy.

Table 1: Final Test Dataset Performance for CNN, ResNet-CNN and CNN-LSTM models. All models had access to past $H = 4$ board and move-time histories.

| Final Test Dataset Performance | | | |
|---|---|---|---|
| | CNN | **ResNet-CNN** | CNN-LSTM |
| **Balanced Accuracy** | 95.9% | **98.2%** | 96.5% |
| Recall | 95.9% | **99.7%** | 96.2% |
| Precision | 52.0% | 57.9% | 58.6% |
| False Positive Rate | 4.1% | 3.3% | 3.1% |
| Focal Loss | 0.0361 | **0.0237** | 0.0257 |
| F1 Score | 0.67 | **0.73** | 0.73 |

We performed ablation analysis on the ResNet-CNN model by excluding board and/or move-time history inputs. We also include an MLP-only network (Section 2.4) that processes the game context features only (Equation 3) for comparison. The results are shown in Table 2.

Table 2: Ablation Analysis. Final test dataset performance are performed on the unmodified ResNet-CNN with access to four (4) past board and move-time histories, ResNet-CNN sans time history, and sans board history.

| **Input Features** | | | | |
|---|---|---|---|---|
| Game context | X | X | X | X |
| Board | X | X | X | |
| Board history | X | X | | |
| Time History | X | | | |
| **Test Set Performance** | | | | |
| | ResNet-CNN | | | MLP |
| **Balanced Accuracy** | **98.2%** | **81.7%** | **76.8%** | **72.1%** |
| Recall | 99.7% | 80.4% | 78.5% | 67.1% |
| Precision | 57.9% | 17.9% | 12.8% | 12.0% |
| False Positive Rate | 3.3% | 17.0% | 24.8% | 22.8% |
| Focal Loss | 0.0237 | 0.0676 | 0.0937 | 0.1109 |
| F1 Score | 0.73 | 0.29 | 0.22 | 0.20 |

To assess the bias/variance of the models, we performed additional ablation analysis by varying the training data available to each model and using balanced accuracy as the evaluation metric (Figure 1). When working on problems with heavily imbalanced datasets, the balanced accuracy score is preferred over the F1-score to ensure that our evaluation does not reward classifying positives correctly more than the negatives.

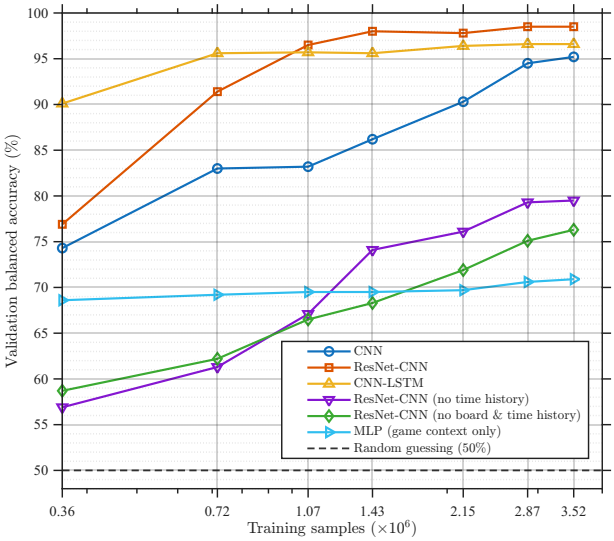


Figure 1: Ablation Analysis on Training Data Size for CNN, ResNet-CNN, CNN-LSTM and MLP models. For ResNet-CNN, three variations were benchmarked: ResNet-CNN with access to past $H = 4$ board and move-time histories, sans time history, and sans board history. Maximum training data size is RAM-limited.

The Precision-Recall Curve of the full ResNet-CNN model is presented in Figure 2.

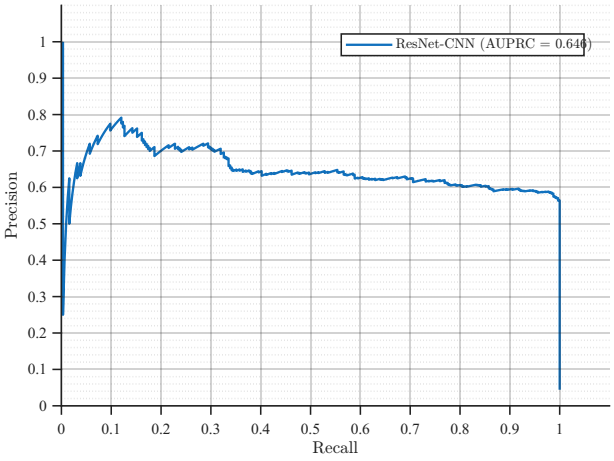


Figure 2: Precision-Recall Curve of the ResNet-CNN model.

### 3.1 Discussion

This work studies whether premoves in online blitz chess can be predicted from recent board and move-time history. Across three architectures, the ResNet-CNN achieved the strongest overall performance on the held-out test set (balanced accuracy 98.2%, recall 99.7%, precision 57.9%; Table 1, p. 2). Balanced accuracy is the primary selection metric as premoves are rare (≈4-5% positives), and evaluating using F1 or raw accuracy alone would risk overvaluing the majority negative class. While all three CNN-based models achieved similar performance, the ResNET's performance and rapid convergence time (Figure 1) may be attributed to the residual towers' ability to overcome the vanishing gradient descent problem in typical CNN networks (Liu et al. 2025).

While the balanced accuracy of 98.2% (Table 1) is sufficiently high, the precisions were consistently ≈ 60% across all three architectures. This is likely due to nondeterministic factors and noise present in human decision-making under the low time constraints that motivate premoves, and individual heterogeneity in play styles. Given an identical move history and game context, the same player is not guaranteed to premove consistently across trials, and different players may likewise disagree in their premove decisions.

The Precision-Recall curve (Figure 2; AUPRC=0.646) demonstrates that the ResNet's performance is significantly above the baseline precision of ≈0.04 (4.42% positives). By tuning the model output threshold, precision may be increased at the cost of the near-perfect recall achieved in this work (99.7%; Table 1). For example, for fair play enforcement applications, picking a threshold with high precision may be more relevant than optimizing recall or overall balanced accuracy.

Inclusion of the current board position meaningfully increased prediction performance to 76.8% balanced accuracy from the 72.1% of the baseline MLP model that did not process any board or move-time information at all. The base MLP model achieved substantial predictive performance over random guessing; this is likely due to the MLP model learning a correlation between whether a game is in the end stages (as signaled by the game context features in Equation 3) and premove probability.

We hypothesized that move-time history may improve premove prediction. In support of this hypothesis, removing move-time history most substantially impacted model performance relative to spatial features, decreasing the balanced accuracy from 98.2% in the full ResNet-CNN to 81.7% (Table 2). Removing board history decreased performance by a lesser amount (balanced accuracy decreased from 81.7% to 76.8%; Table 2). Taken together, these results imply that (i) who is to move now and what the current position is suffice for most spatial/contextual needs of the premove classifier, and (ii) recent timing dynamics (i.e., the player's and opponent's last few movz'e times) carry the dominant information about whether a premove will be used next. This supports the notion that premove is a time-management strategy more than a board-tactic phenomenon: players appear to premove when their recent cadence, time remaining, and phase of the game make the risk-reward tradeoff favorable.

## 4 Conclusion

This work demonstrates premove behavior in online blitz chess can be predicted by using spatial board contexts and player move-time history. Among the benchmarked architectures, the ResNet-CNN achieved the highest test-set performance (98.2% balanced accuracy), outperforming standard CNN and CNN-LSTM variants. Our ablation analysis supports that recent move-time history is the dominant predictor over board history. Future works may benefit from exploring other network architectures such as Transformer-based models to better account for long-range sequential dependencies. Player-specific embeddings may be explored to account for individual play style heterogeneity and develop chess AIs with human-like premove behavior.

## 5 Team Contributions

- **Sitan Huang**: Conceptualization, Writing, Methods, Software, Visualization, Data Curation, Project Management.

## References


Lin, Tsung-Yi et al. (Feb. 7, 2018). *Focal Loss for Dense Object Detection*. DOI: 10.48550/arXiv.1708.02002. arXiv: 1708.02002[cs]. URL: http://arxiv.org/abs/1708.02002 (visited on 11/10/2025).

Liu, Xingyu and Kun Ming Goh (Oct. 28, 2025). *ResNet: Enabling Deep Convolutional Neural Networks through Residual Learning*. DOI: 10.48550/arXiv.2510.24036. arXiv: 2510.24036[cs]. URL: http://arxiv.org/abs/2510.24036 (visited on 11/10/2025).

McIlroy-Young, Reid et al. (Aug. 23, 2020). "Aligning Superhuman AI with Human Behavior: Chess as a Model System". In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1677–1687. DOI: 10.1145/3394486.3403219. arXiv: 2006.01855[cs]. URL: http://arxiv.org/abs/2006.01855 (visited on 10/04/2025).

*Neural network topology - Leela Chess Zero* (2025). URL: https://lczero.org/dev/backend/nn/ (visited on 11/10/2025).

Omori, Michael and Prasad Tadepalli (2025). "Chess Rating Estimation from Moves and Clock Times Using a CNN-LSTM". In: vol. 15550, pp. 3–13. DOI: 10.1007/978-3-031-86585-5_1. arXiv: 2409.11506[cs]. URL: http://arxiv.org/abs/2409.11506 (visited on 10/04/2025).

Oshri, Barak and Nishith Khandwala (2015). "Predicting Moves in Chess using Convolutional Neural Networks". In.

Pawar, Ambar Balkrishna (n.d.). "An Empirical Study of AttLSTM Neural Networks for Chess Move Prediction". In: *School of Computing National College of Ireland* ().

*python-chess 1.11.2* (2025). URL: https://python-chess.readthedocs.io/en/latest/ (visited on 12/03/2025).

*PyTorch* (2025). PyTorch. URL: https://pytorch.org/ (visited on 12/03/2025).

Sadmine, Quazi Asif, Asmaul Husna, and Martin Müller (2024). "Stockfish or Leela Chess Zero? A Comparison Against Endgame Tablebases". In: *Advances in Computer Games*. Ed. by Michael Hartisch, Chu-Hsuan Hsueh, and Jonathan Schaeffer. Vol. 14528. Series Title: Lecture Notes in Computer Science. Cham: Springer Nature Switzerland, pp. 26–35. ISBN: 978-3-031-54967-0 978-3-031-54968-7. DOI: 10.1007/978-3-031-54968-7_3. URL: https://link.springer.com/10.1007/978-3-031-54968-7_3 (visited on 11/10/2025).

Silver, David et al. (2017). *Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*. Version Number: 1. DOI: 10.48550/ARXIV.1712.01815. URL: https://arxiv.org/abs/1712.01815 (visited on 11/10/2025).

Zhang, Yiming et al. (Oct. 4, 2024). *Human-aligned Chess with a Bit of Search*. DOI: 10.48550/arXiv.2410.03893. arXiv: 2410.03893[cs]. URL: http://arxiv.org/abs/2410.03893 (visited on 10/04/2025).