Learning to Plan via Supervised Contrastive Learning and Strategic Interpolation: A Chess Case Study

Andrew Hamara
andrew_hamara1@baylor.edu
Baylor University
Waco, Texas, USA

Greg Hamerly
greg_hamerly@baylor.edu
Baylor University
Waco, Texas, USA

Pablo Rivas
pablo_rivas@baylor.edu
Baylor University
Waco, Texas, USA

Andrew C. Freeman
andrew_freeman@baylor.edu
Baylor University
Waco, Texas, USA

Abstract

Modern chess engines achieve superhuman performance through deep tree search and regressive evaluation, while human players rely on intuition to select candidate moves followed by a shallow search to validate them. To model this intuition-driven planning process, we train a transformer encoder using supervised contrastive learning to embed board states into a latent space structured by positional evaluation. In this space, distance reflects evaluative similarity, and visualized trajectories display interpretable transitions between game states. We demonstrate that move selection can occur entirely within this embedding space by advancing toward favorable regions, without relying on deep search. Despite using only a 6-ply beam search, our model achieves an estimated Elo rating of 2593. Performance improves with both model size and embedding dimensionality, suggesting that latent planning may offer a viable alternative to traditional search. Although we focus on chess, the proposed embedding-based planning method can be generalized to other perfect-information games where state evaluations are learnable. All source code is available at https://github.com/andrewhamara/SOLIS.

CCS Concepts

• **Computing methodologies** → **Knowledge representation and reasoning**; **Game tree search**; *Planning under uncertainty*.

Keywords

Contrastive Learning, Representation Learning, Chess AI, Latent Planning, Transformer Models, Evaluation-Based Search

ACM Reference Format:
Andrew Hamara, Greg Hamerly, Pablo Rivas, and Andrew C. Freeman. 2025. Learning to Plan via Supervised Contrastive Learning and Strategic Interpolation: A Chess Case Study. In *Proceedings of Undergraduate and Master's Consortium at KDD 2025 (KDD-UMC '25).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

The ability to plan and reason through complex decisions is a defining characteristic of intelligence. Chess has long served as a landmark of both human and artificial intelligence, as mastery of the game requires a combination of memory, strategic planning, and calculation. Its strict rules and combinatorial complexity make it an ideal domain for studying structured decision making in an environment where exhaustive search is impractical. State-of-the-art chess engines such as AlphaZero [23] and Stockfish [20] achieve superhuman performance by pairing neural network regressors with powerful tree search. Their mastery, broadly speaking, arises from the capacity to search deeply and efficiently through vast branches of possible continuations.

In contrast, expert human players seldom depend on deep search [14]. Instead, they identify strong moves via intuition developed over years of analysis and validate their choices with shallow lookahead. As a result, while engines are invaluable for tasks such as opening preparation and post-game analysis, they fail to replicate the reasoning patterns of elite players [14, 15, 25]. This mismatch limits their utility as training tools and has prompted growing interest in engines that better reflect human opponents. Models such as Maia and Maia2 [15, 25] pursue this goal by framing human alignment as a classification task, selecting moves that humans are most likely to play. Rather than emulating human decisions, we seek to explore whether a learned latent space can support a more efficient, human-like search process that reflects the selectivity of human intuition.

To this end, we introduce a chess engine built on a transformer encoder that replaces regression and deep search with a directional planning process. We train our model using supervised contrastive learning to embed board positions into a continuous latent space aligned with state-of-the-art engine evaluations. Within this space, distance reflects evaluative similarity, and linear interpolation enables interpretable trajectories between game states. Crucially, move selection occurs entirely within the embedding space by selecting actions that advance toward known favorable regions.

Despite using only a shallow search of six half-moves, our model achieves an estimated Elo rating of 2593. We show that performance scales with model size and embedding dimensionality, and that the learned space supports meaningful embedding arithmetic. These results suggest that contrastive training may offer a viable path toward structured representations that support efficient planning, without the need for deep search.

Although our experiments focus on chess, the underlying approach of learning evaluation-aligned embeddings and selecting

KDD-UMC '25, Toronto, Canada
2025. ACM ISBN 978-1-4503-XXXX-X/25/08
https://doi.org/10.1145/nnnnnnn.nnnnnnn

actions by advancing through latent space is more general. It extends to other zero-sum, perfect-information games such as Go and Shogi. In these settings, where position evaluations are available or are learnable, directional planning provides an efficient alternative to deep search.

2 Background

2.1 Chess Programming

In 1928, von Neumann [26] introduced game theory to chess, describing it as a two-player, zero-sum, perfect information game to which his proposed fixed-depth minimax algorithm could apply. Several decades later, Shannon [22] adapted the minimax algorithm for chess, guided by static evaluation functions of game states. In parallel, Turing [2] introduced his chess-playing "Machine," the most advanced engine of its time, though too complex for the hardware available to him.

In 1997, IBM successfully scaled chess computing with their Deep Blue engine to defeat the reigning world chess champion, Garry Kasparov, in a head-to-head match. Notably, it is estimated that Deep Blue evaluated roughly 200 million positions per second [11, 12], compared to Kasparov's 5 [14].

Early versions of Stockfish [20] followed a similar approach to Deep Blue: handcrafted evaluation functions paired with deep alpha-beta tree search. The methodology was compelling and achieved superhuman performance, though it became clear that the limiting factor was the combined chess mastery of those designing the static evaluation function.

The rating bottlenecks associated with static evaluation inspired a vein of research independent of human guidance. AlphaZero [23] and its open-source counterpart Leela Chess Zero [1] made a significant leap to engines that learned to evaluate positions entirely through self-play reinforcement learning, with no prior knowledge beyond the rules of chess. Similarly, modern versions of Stockfish adopted an efficiently updatable neural network evaluation module (NNUE) [17] to accelerate position evaluation while maintaining traditional alpha-beta search.

While these advances have produced engines that far surpass human players in strength, the underlying search algorithms remain largely unchanged. They still follow principles first proposed by von Neumann and Shannon and later integrated into Deep Blue, relying on deep alpha-beta search through millions of states [14]. Yet it remains remarkable that human players, searching only a few continuations per second, display an understanding that engines achieve only by evaluating millions [14]. In this work, we explore whether contrastive learning can enable a more efficient search process by treating planning as traversal through an embedding space.

2.2 Contrastive Learning

Contrastive learning of representations encourages a model to embed similar inputs closer together and dissimilar inputs farther apart in a latent space [3]. More concretely, given a batch I containing an anchor input x_i, an encoder $f(\cdot)$ maps the anchor to a normalized embedding $z_i = f(x_i)$. Positive pairs (z_i, z_j) correspond to semantically similar inputs, while all other pairs of samples in the batch are considered negatives.

A common contrastive objective is InfoNCE [27], defined as:

$$\ell_{i,j} = -\log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k \in A(i)} \exp(\text{sim}(z_i, z_k)/\tau)},$$

where $A(i) = I \setminus \{i\}$ is the set of all samples in the batch excluding the anchor i, $\text{sim}(\cdot, \cdot)$ denotes a similarity metric, and τ is a scalar temperature parameter. In the case of self-supervised contrastive learning, the positive sample z_j is an augmentation of the anchor z_i, and labels are not required [3, 13].

When labels are available, the supervised contrastive loss (SupCon) [13] extends InfoNCE by allowing multiple positives per anchor. The SupCon loss is defined as:

$$\mathcal{L}_{\text{sup}} = \sum_{i \in I} \frac{-1}{|P(i)|} \sum_{p \in P(i)} \log \frac{\exp(\text{sim}(z_i, z_p)/\tau)}{\sum_{a \in A(i)} \exp(\text{sim}(z_i, z_a)/\tau)},$$

where $P(i)$ denotes the set of positive examples corresponding to anchor i, and $A(i)$ is defined as above. Increasing the number of positives and negatives has been shown to improve representation quality [4, 13].

3 Methods

We now describe the components of our approach, including tokenization, model architecture, training procedure, and action selection during inference.

3.1 Input Representation and Tokenization

Each chess position is represented using its Forsyth-Edwards Notation (FEN) string, which compactly encodes the piece placement, side to move, castling rights, en passant target, half-move clock, and full-move counter. Following the tokenization scheme proposed in Ruoss et al. [21], we tokenize each FEN into a fixed-length sequence of 77 tokens by expanding run-length encodings.

3.2 Encoder Architecture

Our encoder is a multi-layer transformer [28] adapted for tokenized chess positions. We denote the number of transformer layers as L, the hidden dimension as H, the number of self-attention heads as N, and the output embedding dimension as D. We summarize the model configurations used in our experiments in Table 1.

Table 1: Transformer model configurations used in our experiments.

Model	L	H	D	N	MLP Size	Params
Small	6	512	512	16	512	8M
Base	6	1024	1024	16	1024	41M

Each input sequence is embedded into H-dimensional vectors through a learned token embedding matrix. A special classification token (CLS) [6, 7] is prepended to the embedded sequence to aggregate information across the input. Because the input sequences are fixed length, we add a learned positional encoding to each token embedding [21]. The resulting sequence is processed by L stacked transformer encoder layers with GELU [10] activations

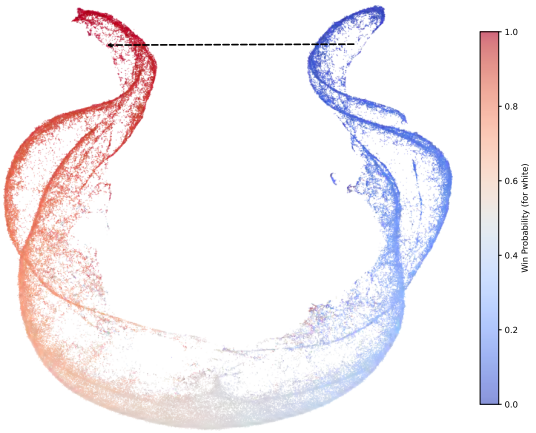


Figure 1: UMAP projection of the learned embedding space of our Base model, colored by win probability (red = White favored, blue = Black favored). The dashed black arrow represents the advantage axis $\vec{a}$.

and a dropout [24] rate of 0.1. The final hidden state corresponding to the CLS token is extracted, passed through a linear projection, and ℓ_2 normalized.

3.3 Supervised Contrastive Training

We train our encoder using 5 million randomly sampled positions from the ChessBench dataset [21], each annotated with a Stockfish-evaluated win probability for the player to move. We normalize all evaluations to represent White's perspective, such that values near 1.0 indicate a decisive advantage for White and values near 0.0 indicate an advantage for Black.

To define positive samples for training, we set $\delta = 0.05$ as the evaluation margin for identifying similar positions. For each anchor, we precompute all positions whose win probabilities differ by less than δ and randomly sample five as positives during training. To build the batch-level mask, we compare the win probabilities of all samples and mark pairs as positive if their evaluation difference is below δ.

We apply supervised contrastive learning (SupCon; see Section 2.2) using the constructed batch masks to define positive and negative pairs. Cosine similarity is used to compute pairwise scores, and we set $\tau = 0.07$ [8, 19] to control the sharpness of the softmax distribution. We train both models using stochastic gradient descent with a momentum parameter of 0.9 [3, 7, 9] for 400,000 steps with a batch size of 128. Training is conducted on four NVIDIA L40S GPUs (48 GB memory each) with data parallelism [18].

We visualize the learned representation space of our Base model in Figure 1 using UMAP [16] for 2D projection. Each point represents an encoded chess position from a test dataset, colored by its Stockfish-evaluated win probability (for White). The black dashed arrow indicates the advantage axis $\vec{a}$, though its appearance in 2D does not reflect its true geometric trajectory.

3.4 Embedding-Guided Beam Search

During inference, candidate moves are evaluated based on their alignment with an advantage axis. We first compute element-wise mean embeddings μ_{white} and μ_{black} over positions evaluated as $p = 1.0$ (white has forced checkmate) and $p = 0.0$ (black has forced checkmate), respectively, and define the advantage vector $\vec{a} = \mu_{\text{white}} - \mu_{\text{black}}$.

As illustrated in Figure 2, the search begins from a starting position (leftmost *Starting board state*). All legal continuations are enumerated (*Extract possible moves*) and tokenized. Each resulting child position is passed through our *Transformer encoder* (purple blocks) in parallel, producing H-dimensional embeddings. Each candidate embedding z' is linearly projected into the latent space and scored by its cosine similarity with the advantage vector $\vec{a}$: $\text{score}(z') = \cos(z', \vec{a})$ (*Cosine similarity* block). The resulting scalar values reflect how promising each move is in terms of directional alignment with winning positions.

If the current search depth is equal to a predefined maximum S (as checked in the green *S levels deep?* decision node), the search terminates, and the initial move leading to the best final similarity score is selected (*Select first move...*). Otherwise, the top-k highest scoring candidate positions (*Top-k* module) are selected and the process continues recursively, as shown by the loop labeled *Recursive search.*

4 Results

4.1 Experimental Setup

Each model was evaluated against Stockfish 16 with a fixed per-move time limit of 50 ms [21]. We configured our embedding beam search with a width (k) of 3 and ablated over search depths. At each depth, our model played at least 600 games against Stockfish configured with different Elo caps using its internal `UCI_LimitStrength` setting. Caps were selected to ensure that each model configuration yielded at least one matchup with a positive win rate and one with a negative win rate. Elo ratings were computed using Bayesian logistic regression via the BayesElo [5] program with the default confidence parameter of 0.5. Summarized Elo estimates are presented in Table 2.

Table 2: Estimated Elo ratings for our models by search depth and size.

Model Size	Depth 2	Depth 3	Depth 4	Depth 5	Depth 6
Small	2067	2282	2388	2487	2548
Base	2115	2318	2433	2538	2593

4.2 Performance Across Search Depths

Our results show that Elo ratings increase consistently with search depth. The base model climbs from 2115 at depth 2 to 2593 at depth 6, roughly matching the strength of Stockfish configured to 2600 Elo. The small model follows a similar pattern with a consistent gap of 30–50 Elo relative to the base model.

Gains are nearly linear up to depth 5, but level off at depth 6. This tapering is likely due to the limitations of greedy beam search,

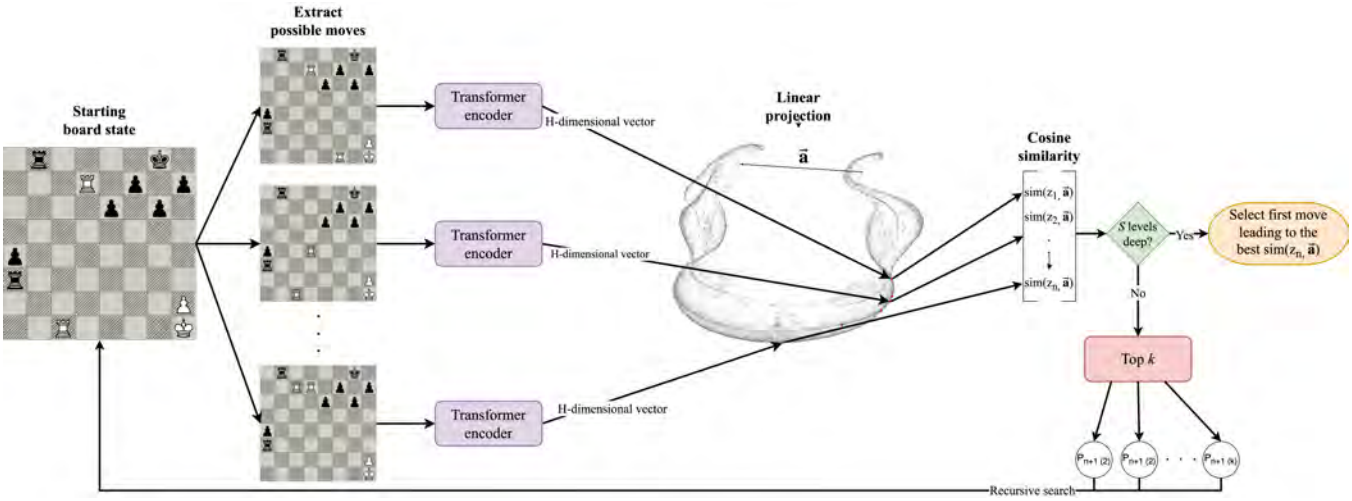
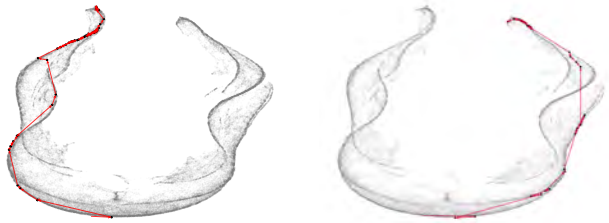


Figure 2: System overview of the embedding-guided beam search. Candidate moves are embedded, scored via similarity to an advantage vector, and selected recursively based on top-k alignment.

which constructs plans incrementally. Because each step depends on the top-k projections from the previous state, strong continuations outside this trajectory will be excluded. As a result, our model can miss advantageous lines that would only be considered via wider exploration.

4.3 Qualitative Game Trajectories

We visualize latent trajectories of real games to better understand the interpretability of the embedding space. Figure 3 shows three representative examples: a game won by White, a game that remains balanced throughout, and a game won by Black. Each trajectory is plotted over the same 2D projection from a sample dataset. Positions are embedded independently and connected with arrows to indicate move progression. Games that are decisively won by one player trace smooth paths through the space, while closely contested games fluctuate around the center.



(a) White steadily gains advantage. **(b) Black steadily gains advantage.**

(c) Game remains balanced throughout.

Figure 3: Latent trajectory visualizations of three games embedded in the shared representation space. Red arrows indicate the progression of positions as the game unfolds.

5 Discussion

We have presented a chess engine that selects moves by advancing through a learned embedding space without relying on deep search. Despite using only a 6-ply search, the system achieves an Elo rating of 2593. Visualizations of real games further show that latent trajectories follow smooth evaluative trends.

While these results are promising, the system has notable limitations. Firstly, the greedy beam search cannot revise early commitments, restricting its ability to recover from errors. Broader planning strategies, such as policy networks, non-greedy beam exploration, and search memoization may improve robustness. Secondly, the current training pipeline assumes a fixed evaluation target derived from Stockfish, which may not reflect human intuition. Future work could explore reinforcement learning fine-tuning, larger models, and a broader sampling of positives to further shape the structure of the embedding space.

Overall, these results suggest that latent-space reasoning may offer a viable alternative to conventional search. We hope this work contributes to a broader rethinking of planning in games and decision-making domains, not as brute-force optimization but as strategic interpolation through learned representations.

References

[1] The LCZero Authors. 2018. LeelaChessZero. https://lczero.org Accessed: 2024-04-25.
[2] Cyril Burt. 1955. Faster than Thought: A Symposium on Digital Computing Machines. Edited by BV Bowden.

[3] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A Simple Framework for Contrastive Learning of Visual Representations. arXiv:2002.05709 [cs.LG] https://arxiv.org/abs/2002.05709

[4] Shaif Chowdhury, Greg Hamerly, and Monica McGarrity. 2024. Active Learning Strategy Using Contrastive Learning and K-means for Aquatic Invasive Species Recognition. In *2024 IEEE/CVF Winter Conference on Applications of Computer Vision Workshops (WACVW)*. 848–858. doi:10.1109/WACVW60836.2024.00097

[5] Rémi Coulom. 2008. Whole-history rating: A Bayesian rating system for players of time-varying strength. In *Computers and Games (LNCS, Vol. 5131)*. Springer, 113–124. doi:10.1007/978-3-540-87608-3_10

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805 [cs.CL] https://arxiv.org/abs/1810.04805

[7] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv:2010.11929 [cs.CV] https://arxiv.org/abs/2010.11929

[8] Rohit Girdhar, Alaaeldin El-Nouby, Zhuang Liu, Mannat Singh, Kalyan Vasudev Alwala, Armand Joulin, and Ishan Misra. 2023. ImageBind: One Embedding Space To Bind Them All. arXiv:2305.05665 [cs.CV] https://arxiv.org/abs/2305.05665

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385 [cs.CV] https://arxiv.org/abs/1512.03385

[10] Dan Hendrycks and Kevin Gimpel. 2023. Gaussian Error Linear Units (GELUs). arXiv:1606.08415 [cs.LG] https://arxiv.org/abs/1606.08415

[11] Feng-Hsiung Hsu. 2022. *Behind Deep Blue: Building the computer that defeated the world chess champion.* Princeton University Press.

[12] Feng-hsiung Hsu, Murray S Campbell, and A Joseph Hoane Jr. 1995. Deep Blue system overview. In *Proceedings of the 9th international conference on Supercomputing*. 240–244.

[13] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. 2021. Supervised Contrastive Learning. arXiv:2004.11362 [cs.LG] https://arxiv.org/abs/2004.11362

[14] Matthew Lai. 2015. Giraffe: Using Deep Reinforcement Learning to Play Chess. arXiv:1509.01549 [cs.AI] https://arxiv.org/abs/1509.01549

[15] Reid McIlroy-Young, Siddhartha Sen, Jon Kleinberg, and Ashton Anderson. 2020. Aligning Superhuman AI with Human Behavior: Chess as a Model System. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '20)*. ACM, 1677–1687. doi:10.1145/3394486.3403219

[16] Leland McInnes, John Healy, and James Melville. 2020. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. arXiv:1802.03426 [stat.ML] https://arxiv.org/abs/1802.03426

[17] Yu Nasu. 2018. Efficiently Updatable Neural-Network-Based Evaluation Functions for Computer Shogi. Accessed: 2024-04-25.

[18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. arXiv:1912.01703 [cs.LG] https://arxiv.org/abs/1912.01703

[19] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. arXiv:2103.00020 [cs.CV] https://arxiv.org/abs/2103.00020

[20] Tord Romstad, Marco Costalba, Joona Kiiski, Gary Linscott, Yu Nasu, Motohiro Isozaki, and Hisayori Noda. 2008. Stockfish. https://stockfishchess.org Accessed: 2025-04-22.

[21] Anian Ruoss, Grégoire Delétang, Sourabh Medapati, Jordi Grau-Moya, Li Kevin Wenliang, Elliot Catt, John Reid, Cannada A. Lewis, Joel Veness, and Tim Genewein. 2024. Amortized Planning with Large-Scale Transformers: A Case Study on Chess. arXiv:2402.04494 [cs.LG] https://arxiv.org/abs/2402.04494

[22] Claude E Shannon. 1950. XXII. Programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 41, 314 (1950), 256–275.

[23] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. 2017. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. arXiv:1712.01815 [cs.AI] https://arxiv.org/abs/1712.01815

[24] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15, 1 (Jan. 2014), 1929–1958.

[25] Zhenwei Tang, Difan Jiao, Reid McIlroy-Young, Jon Kleinberg, Siddhartha Sen, and Ashton Anderson. 2024. Maia-2: A Unified Model for Human-AI Alignment in Chess. arXiv:2409.20553 [cs.AI] https://arxiv.org/abs/2409.20553

[26] J v. Neumann. 1928. Zur theorie der gesellschaftsspiele. *Mathematische annalen* 100, 1 (1928), 295–320.

[27] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2019. Representation Learning with Contrastive Predictive Coding. arXiv:1807.03748 [cs.LG] https://arxiv.org/abs/1807.03748

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2023. Attention Is All You Need. arXiv:1706.03762 [cs.CL] https://arxiv.org/abs/1706.03762

Received 15 May 2025