Supplementary Material for:

AlphaViT: A flexible game-playing AI for multiple games and variable board sizes

Kazuhisa Fujita

S1 Monte Carlo tree search

MCTS is a sampling-based tree-search algorithm that does not require a predefined evaluation function (Browne et al., 2012; Winands, 2017). It operates through four strategic steps: *Selection step*, *Expansion step*, *Playout step*, and *Backpropagation step* (Winands, 2017). In the *Selection step*, the tree is traversed from the root node to a leaf node. Child node c of parent node p is selected to maximize the score defined in Eq. S1:

$$\mathrm{UCT} = q_c/N_c + C\sqrt{\frac{2\ln(N_p+1)}{N_c+\varepsilon}}, \tag{S1}$$

where q_c is the cumulative value of c, N_p is the visit count of p, N_c is the visit count of c, and ε is a constant value to avoid division by zero. In the *Expansion step*, child nodes are added to the leaf node when the visit count of the leaf node reaches N_{open}. In the *Playout step*, a valid move is selected at random until the end of the game is reached. In the *Backpropagation step*, the result of the playout is propagated along the path from the leaf node to the root node. If the MCTS player itself wins, loses, and draws, $q_i \leftarrow q_i+1$, $q_i \leftarrow q_i-1$, and $q_i \leftarrow q_i$, respectively. q_i is the cumulative value of node i. The sequence of these four steps constitutes a single simulation. The simulation process is repeated N_{sim} times. After all simulations, MCTS selects the action corresponding to the most visited child node of the root. In this study, $C = 0.5$, $\varepsilon = 10^{-7}$, and $N_{\mathrm{open}} = 5$.

S2 Minimax algorithm

The minimax algorithm is a fundamental game-tree search technique that determines the optimal action by evaluating the best possible outcome for the current player. Each node in the tree contains a state, player, action, and value. The algorithm creates a game tree with a depth of $N_{\mathrm{depth}} = 3$. For the Othello variants, the algorithm expands the tree to the terminal nodes after the last six turns. The root node corresponds to the current state and minimax player. Next, the states corresponding to leaf nodes are evaluated. Then, the algorithm propagates the values from the leaf nodes to the root node. If the player corresponding to the node is the opponent, the value of the node is the minimum value of its child nodes. Otherwise, the value of the node is the maximum value of its child nodes. Finally, the algorithm selects the action corresponding to the root's child node with the maximum value. The evaluation of leaf nodes is tailored to each game.

For the Connect 4 variants, the values of the connections of two and three same-colored discs are $R \times c_{\mathrm{disc}}c_{\mathrm{minimax}}$ and $R^2 \times c_{\mathrm{disc}}c_{\mathrm{minimax}}$, respectively, where R is the base reward, and c_{disc} and c_{minimax} are the colors of the connecting discs and the minimax player's disc, respectively. The value of a node is the sum of the values of all connections on the corresponding board. The terminal nodes have a value of $R^3 c_{\mathrm{win}}c_{\mathrm{minimax}}$, where c_{win} is the color of the winner, and $R = 100$.

For the Gomoku variants, the values of the connections of two, three, and four same-colored discs are $R \times c_{\mathrm{disc}}c_{\mathrm{minimax}}$, $R^2 \times c_{\mathrm{disc}}c_{\mathrm{minimax}}$, and $R^3 \times c_{\mathrm{disc}}c_{\mathrm{minimax}}$, respectively. The value of a

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

node is the sum of the values of all connections on the corresponding board. The terminal nodes have values of $R^4 c_{\text{win}} c_{\text{minimax}}$ and $R = 100$.

For the Othello variants, the value of a node is calculated using Eq. S2.

$$E = \sum_x \sum_y v(x,y) o(x,y) c_{\text{minimax}}, \tag{S2}$$

where $v(x,y)$ is the value of cell (x,y) and $o(x,y)$ is the occupancy of cell (x,y). For Othello 6x6 and Othello, the minimax algorithm evaluates each cell using Eq. S3 and S4, respectively. $o(x,y)$ is 1, -1, and 0 if cell (x,y) is occupied by the first player's disc, the second player's disc, and empty, respectively. The terminal nodes have a value of $E_{\text{end}} = 1000 c_{\text{win}} c_{\text{minimax}}$.

$$v_{6\times6} = \begin{pmatrix} 30 & -5 & 2 & 2 & -5 & 30 \\ -5 & -15 & 3 & 3 & -15 & -5 \\ 2 & 3 & 0 & 0 & 3 & 2 \\ 2 & 3 & 0 & 0 & 3 & 2 \\ -5 & -15 & 3 & 3 & -15 & -5 \\ 30 & -5 & 2 & 2 & -5 & 30 \end{pmatrix}. \tag{S3}$$

$$v_{8\times8} = \begin{pmatrix} 120 & -20 & 20 & 5 & 5 & 20 & -20 & 120 \\ -20 & -40 & -5 & -5 & -5 & -5 & -40 & -20 \\ 20 & -5 & 15 & 3 & 3 & 15 & -5 & 20 \\ 5 & -5 & 3 & 3 & 3 & 3 & -5 & 5 \\ 5 & -5 & 3 & 3 & 3 & 3 & -5 & 5 \\ 20 & -5 & 15 & 3 & 3 & 15 & -5 & 20 \\ -20 & -40 & -5 & -5 & -5 & -5 & -40 & -20 \\ 120 & -20 & 20 & 5 & 5 & 20 & -20 & 120 \end{pmatrix}. \tag{S4}$$

S3 Additional MCTS experiment

In this study, an additional experiment was conducted to investigate the impact of varying simulation counts on the performance of MCTS in Connect 4, using the large board configuration. The MCTS algorithm was systematically evaluated using a fixed number of simulations ranging from 10 to 6400. Each MCTS variant was evaluated against the set of agents previously described (see Table 3), with the opponents' Elo ratings maintained at the values already established. Elo ratings for the MCTS variants were calculated using the method described in Subsection 5.2.

The results of these evaluations are presented in Table S1, indicating a significant improvement in Elo ratings as the simulation count increases from 10 to 1600. Beyond 1600, the incremental Elo gain becomes less pronounced; 6400 adds only ≈ 30 Elo, and 3200 shows no improvement, illustrating diminishing returns. This observed trend aligns closely with findings reported by Baier and Winands (2015), who demonstrated that enhancing the inference time, conceptualized as the number of simulations in MCTS, does not necessarily lead to continuous performance improvements in Connect 4. Furthermore, a substantial difference in performance between 100- and 400-simulation configurations is observed, which are selected as baseline configurations.

Table S1: Elo ratings of MCTS variants in Connect 4

Simulations	Elo rating	95% CI
10	1048.3	[1004.0, 1093.1]
100	1316.8	[1273.3, 1366.4]
200	1431.4	[1376.5, 1472.3]
400	1509.8	[1476.3, 1564.8]
800	1573.6	[1522.8, 1623.3]
1600	1676.7	[1611.0, 1715.0]
3200	1673.5	[1624.3, 1730.7]
6400	1708.4	[1674.1, 1767.1]

S4 Elo rating

S4.1 Calculation of Elo rating

Elo rating is a widely used metric for evaluating the relative performance of players in two-player games. This metric allows us to estimate the probability of one player defeating another based on their current ratings. Given two players A and B with Elo ratings $e(A)$ and $e(B)$, respectively, the probability that player A will defeat player B, denoted $p(A\text{defeats}B)$, is calculated using Eq. S5:

$$p(A \text{ defeats } B) = 1/(1 + 10^{(e(B)-e(A))/400}). \tag{S5}$$

After a series of N_G games between players A and B, player A's Elo rating is updated to a new value $e'(A)$ based on their performance. The update is performed using Eq. S6:

$$e'(A) = e(A) + K(N_{\text{win}} - N_G \times p(A \text{ defeats } B)), \tag{S6}$$

where N_{win} denotes the total score accumulated by player A across the N_G games (counting each win as 1, each draw as 0.5, and each loss as 0), and K is a factor that determines the maximum rating adjustment after a single game. In this study, $K = 8$.

S4.2 Calculation of 95% confidence intervals for Elo ratings

To quantify estimation uncertainty, 95% confidence intervals via percentile bootstrap (LMSYS Org, 2023; Tang et al., 2025) are computed as follows:

(1) Resample the full set of T game results with replacement from the original dataset (which contains T samples), yielding a new dataset of size T.

(2) Apply the same sequential-update Elo rule (initial rating = 1500, $K = 8$) to compute ratings for all players.

(3) Repeat steps 1 and 2 for $B = 1000$ replicates to obtain a distribution of Elo ratings.

(4) Define the 95% confidence interval for each player as the 2.5th and 97.5th percentiles of its bootstrap Elo distribution.

This yields point estimates from the original data and interval estimates from the bootstrap replicates, enhancing the statistical rigor of the evaluation.

S4.3 Calculation of 95% confidence intervals for ratio of Elo ratings between deeper and shallower models

An Elo rating is computed for each training iteration (t) by replaying the complete match log in chronological order. For player p, an iteration series $E_{p,t}(t$ = 1,2, ...,10,20,40, ...,100,200,300, ...,3000) is obtained. To summarize playing strength over the final one thousand iterations, the ten checkpoints in the interval $T = \{2100, 2200, \ldots, 3000\}$ are considered. The mean Elo in this window is $\bar{E}_{p,s} = \frac{1}{|T|}\sum_{t\in T} E_{p,t}$, where ($s \in \{\text{shallower}, \text{deeper}\}$) labels the shallower and deeper variants of the same architecture. From these averages, the relative performance ratio R is defined as $\bar{E}_{p,\text{deep}}/\bar{E}_{p,\text{shallow}}$. Additionally, the difference in average Elo ratings Δ is computed as $\bar{E}_{p,\text{deep}} - \bar{E}_{p,\text{shallow}}$. Sampling uncertainty for R is quantified using a non-parametric percentile bootstrap: the ten checkpoint rows are resampled $B = 1000$ times with replacement, the bootstrap statistics ($\{R^{*b}\}_{b=1}^{B}$) are recomputed, and a 95% confidence interval for R is derived from the 2.5th and 97.5th percentiles of the resulting empirical distribution.

S5 Additional AlphaZero experiments

These experiments systematically evaluated how architectural depth and optimization methods influenced the learning dynamics and performance of AlphaZero models. The baseline configuration for this study, denoted as Res6, utilized a neural network architecture comprising six residual blocks trained with the AdamW optimizer. The deeper variant, Res10, increased the architectural depth to ten residual blocks and similarly employed AdamW optimization. A further variant, Res10 fine-tuning, represented the Res10 model refined through fine-tuning by employing the same procedure

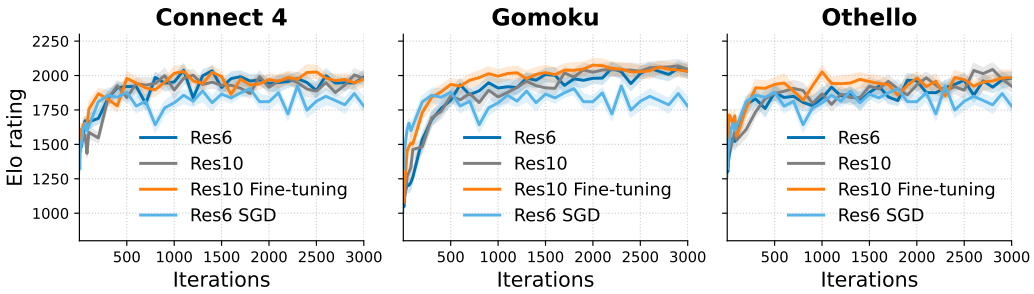

Figure S1: Elo rating progression over training iterations for AlphaZero variants on large board configurations in Connect 4, Gomoku, and Othello. Res6: AlphaZero with six residual blocks. Res10: AlphaZero with ten residual blocks. Res10 fine-tuning: Res10 with fine-tuning. Res6 SGD: Res6 trained using the SGD optimizer.

described in Subsection 5.4 of the main manuscript. Notably, before fine-tuning, only the heads were initialized with random weights because these heads for the small board configuration were not utilized and were replaced with new heads suitable for large board configurations. Finally, to specifically investigate the impact of optimizer selection, the Res6 SGD configuration maintained the six-block structure of Res6, but substituted AdamW with stochastic gradient descent (SGD). The SGD hyperparameters (the learning rate, weight decay, and momentum) were set to 0.05, 0.0001, and 0.9, respectively. The training process for Res6 SGD was identical to that denoted in Appendix B, except for the optimizer choice. Elo ratings were calculated according to the methodology detailed in Subsection 5.2 of the main manuscript.

Figure S1 illustrates the progression of Elo ratings over training iterations for the AlphaZero configurations across three games: Connect 4, Gomoku, and Othello, in large board scenarios. The results indicate that all AlphaZero variants achieve high Elo ratings with sufficient training. Notably, the Res10 fine-tuning configuration exhibits a rapid initial improvement in Gomoku, quickly attaining a high Elo rating relative to the non-fine-tuned Res10 configuration. Conversely, the Res6 SGD configuration consistently demonstrates inferior performance compared to its AdamW-optimized counterpart, plateauing prematurely and failing to achieve comparable Elo ratings across all games.

Collectively, these findings underscore the efficacy of Res6 architecture optimized with AdamW, affirming its suitability as a stable baseline for comparative studies. Moreover, AlphaZero demonstrates notable efficiency in terms of parameter count and computational resource requirements relative to transformer-based models, such as AlphaViT, AlphaViD, and AlphaVDA. In addition, deeper AlphaZero architectures benefit significantly from fine-tuning, rapidly achieving strong performance with reduced computational overhead.

S6 Computational performance

Tables S2 and S3 summarize the computational performance of each model for Othello, including the number of parameters, peak GPU memory consumption, and mean execution time during training and inference, respectively. All experiments were conducted on a custom-made computer equipped with an Intel Core i9-12900K CPU, 64 GiB of DDR5 RAM, and two NVIDIA RTX 4060 Ti 16 GiB GPUs. For training, the author employed a data-parallel approach utilizing both GPUs and ten parallel self-play games (five per GPU). During inference, a single process on a single GPU was used. To measure the training and inference speeds, two precision modes were considered: automatic mixed precision (FP16) and full precision (FP32). The experimental setup included PyTorch 2.3.1, NVIDIA Driver Version 560.35.03, and CUDA Version 12.6.

Peak GPU memory usage (in MiB) denotes the highest memory allocation recorded during training and inference. During training, peak memory usage represents the combined memory usage across both GPUs. Training speed was reported as seconds per iteration (s/iteration), where an iteration comprised one complete cycle of self-play, data augmentation, and network updates (as detailed in Appendix B). Inference speed was measured in seconds per decision (s/decision), reflecting the time

required to compute one move. Mean values for training and inference speeds were computed over ten training iterations and ten full games played by the agent against itself, respectively.

The results demonstrate that for training, FP16 generally provides faster performance and reduced memory usage compared to FP32 for AlphaViT, AlphaViD, and AlphaVDA, but not for AlphaZero. Conversely, during inference, FP32 is generally faster and more memory-efficient than FP16 for all models. Consequently, FP16 was used for training and FP32 was used for inference in this study.

Additionally, the author employed the number of model parameters and other hyperparameters (e.g., number of games per iteration and batch size) listed in Table A2 of the main manuscript for training and evaluating the agents because the author's computational resources were limited to a single computer with two GPUs, and the author is able to train and evaluate the agents with practical computation time for this research.

It is important to note that computation time is significantly influenced not only by model size but also by the game engine implementation, hardware specifications, and hyperparameters, such as the number of parallel processes for self-play and the number of simulations. Achieving a practical balance between these factors is crucial to ensure efficient training and inference.

Initially, for each proposed architecture, the author selected the shallowest encoder depth (4 encoder layers for AlphaViT and one encoder layer for AlphaViD/AlphaVDA) that resulted in approximately 11 million parameters. This depth was chosen to achieve a balanced trade-off between model size, computational efficiency, and expected performance. However, for large board configurations, initial results indicated that these shallower models (especially AlphaViD and AlphaVDA) did not exhibit competitive performance against the AlphaZero baseline. Consequently, the author introduced deeper models—AlphaViT with eight encoder layers (AlphaViT L8) and AlphaViD/AlphaVDA with five encoder layers (AlphaViD L5, AlphaVDA L5)—resulting in approximately 20 million parameters. These deeper configurations represented practical upper limits given the author's available computational resources, in particular the memory and performance constraints of the author's hardware setup consisting of two NVIDIA RTX 4060 Ti GPUs, each with 16 GiB of GPU memory.

Table S2: Computational performance during training

Model	Params (M)	Peak Mem (MiB)		Mean time (sec/iteration)	
		FP16	FP32	FP16	FP32
AlphaViT L4	11.2	6672	9582	131.2	132.5
AlphaViT L8	19.6	11260	17292	197.3	208.8
AlphaViD L1	11.5	4980	7866	113.7	114.1
AlphaViD L5	19.9	9390	14902	180.6	189.1
AlphaVDA L1	11.3	4540	7074	109.6	110.6
AlphaVDA L5	19.8	9232	14598	177.8	184.9
AlphaZero (6 ResBlocks)	7.1	3526	3107	112.2	102.9
AlphaZero (10 ResBlocks)	11.8	5262	4448	149.4	134.3

Peak memory is the maximum aggregated GPU usage observed over ten iterations. Mean iteration time (second/iteration) is averaged over the same interval.

References

Baier, H. and Winands, M. H. M. (2015). Mcts-minimax hybrids. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(2):167–179.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. (2012). A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43.

LMSYS Org (2023). Chatbot Arena: New Models & Elo System Update. `https://lmsys.org/blog/2023-12-07-leaderboard/`. Accessed 10 Jun 2025.

Tang, S., Wang, Y., and Jin, C. (2025). Is elo rating reliable? a study under model misspecification.

Winands, M. H. M. (2017). *Monte-Carlo Tree Search in Board Games*, pages 47–76. Springer Singapore, Singapore.

Table S3: Computational performance during inference

Model	Params (M)	Peak Mem (MiB)		Mean time (sec/decision)	
		FP16	FP32	FP16	FP32
AlphaViT L4	11.2	253	237	0.7541	0.6546
AlphaViT L8	19.6	307	261	1.038	0.6763
AlphaViD L1	11.5	251	239	0.7104	0.5802
AlphaViD L5	19.9	307	263	1.006	0.8018
AlphaVDA L1	11.3	249	237	0.7116	0.5943
AlphaVDA L5	19.8	305	261	0.9689	0.8199
AlphaZero (6 ResBlocks)	7.1	233	207	0.7864	0.7245
AlphaZero (10 ResBlocks)	11.8	273	227	0.9404	0.8293

Peak memory is the maximum single-GPU usage over ten self-play games. Mean decision time is averaged over the same games.