AlphaZero in Sparsely Rewarded Games: Limits and Auxiliary Supervision

author names withheld

Under Review for NExT-Game 2026

Abstract

AlphaZero has demonstrated that a neural-guided Monte Carlo Tree Search can achieve superhuman performance, but strong play does not necessarily imply perfect play. We study this gap in two oracle-evaluable domains with contrasting structure: Connect Four, a solved partisan game with exact game-theoretic values, and Chomp, an impartial game whose optimal play is governed by Grundy-number structure. Under a unified self-play + MCTS pipeline, we compare vanilla AlphaZero, a multi-frame variant, and an AlphaZero Auxiliary Loss (AZAL) that adds oracle-derived policy supervision. We find that vanilla AlphaZero achieves strong play across both domains but cannot preserve the exact trajectories required for optimal play: in Connect Four, it fails to maintain the optimal line of play, while in Chomp, it fails to consistently restore the $g = 0$ invariant. Multi-frame inputs alone do not remove this gap. Nevertheless, AZAL substantially improves optimality recovery, reaching perfect play in Chomp and near-perfect play in Connect Four. These results suggest that the main bottleneck lies less in the weakness of the standard AlphaZero search-learning signal.

Keywords: AlphaZero, Monte Carlo Tree Search, self-play reinforcement learning, game-theoretic learning, oracle evaluation, perfect play

1. Introduction

AlphaZero showed that neural-guided Monte Carlo Tree Search (MCTS) can achieve superhuman performance from rules alone [3, 9, 10]. Yet superhuman play is not identical to perfect play. An AlphaZero-style system may achieve excellent results while still fail to choose the optimal move. This distinction has become especially important in games that depend on sparse global invariants rather than dense local tactics. In such settings, an early mistake can redirect self-play onto off-oracle branches, causing search targets to reinforce suboptimal lines of play. This weakens the positive feedback loop on which AlphaZero depends [2, 8, 11].

This paper studies the gap between strong play and perfect play through two contrasting domains: Connect Four and Chomp. More broadly, this setting provides a controlled testbed for understanding learning dynamics in strategic environments. Connect Four is a solved partisan game with exact game-theoretic values, allowing direct comparison to winning trajectories [1]. By contrast, Chomp is an impartial combinatorial game whose winning structure is naturally studied through Grundy numbers. Despite classical theoretical results, the explicit optimal strategies of Chomp remain difficult to characterize [4, 6]. Together, these games proffer a useful test to determine if AlphaZero fails across games with different strategic structure.

We provide an empirical study of AlphaZero-style learning on both domains under a standard self-play + MCTS pipeline. We evaluate vanilla AlphaZero against exact oracles, compare it with a

multi-frame variant inspired by Riis, and introduce an AlphaZero Auxiliary Loss (AZAL) variant designed to provide a stronger learning signal. Across both domains, we find that vanilla AlphaZero cannot reliably recover perfect play. We further find that the multi-frame modification does not bridge this gap. However, AZAL substantially improves performance, reaching perfect play on Chomp and near-perfect play on Connect Four.

Our contributions are as follows:

1. Vanilla AlphaZero achieves strong play without reliably recovering perfect play in Connect Four and Chomp.
2. We compare two remedies under a unified self-play + MCTS framework: multi-frame inputs and AZAL.
3. AZAL substantially improves oracle alignment, suggesting that the main bottleneck lies in the standard AlphaZero search-learning signal.

2. Problem Setting and Oracle Evaluation

We study the gap between *strong* and *perfect* play in two oracle-evaluable domains with contrasting structure: Connect Four and Chomp. Connect Four is a solved partisan game with exact game-theoretic values, while Chomp is an impartial game whose optimal play is governed by Grundy-number structure. These domains let us test whether AlphaZero-style agents preserve the trajectories required for exact play rather than merely achieving strong empirical performance.

For Connect Four, we use the perfect solver of Pascal Pons [7] to evaluate moves and positions against exact game-theoretic values. For Chomp, we use a recursive Grundy-number solver, where a winning position satisfies $g(s) \neq 0$ and an optimal move reaches a successor with $g(s') = 0$ [5]. In both domains, oracle annotations allow us to measure whether a learned policy remains on the optimal trajectory throughout play. Additional oracle and encoding details are deferred to A.1 and A.2.

3. Methods

We compare three closely related AlphaZero-style agents: vanilla AlphaZero, a multi-frame variant, and AlphaZero-Auxiliary Loss (AZAL). In all cases, self-play games are generated with MCTS guided by a policy–value network, replay tuples are collected from those games, and the network is updated from search-improved policy targets and game outcomes. The methods therefore differ only in the input provided to the network and in the strength of the supervision signal.

Vanilla AlphaZero. Our baseline follows the standard AlphaZero recipe [10]: a policy–value network is trained from self-play data, where MCTS produces improved policy targets and terminal outcomes provide value targets. The policy is learned only through search-improved self-play. Full architecture, search, and optimization details are deferred to Section B.1.

Multi-frame AlphaZero. To test if the main bottleneck comes from state representation, we also evaluate a multi-frame variant inspired by Riis [8]. The training loop, search procedure, and optimization objective are identical to vanilla AlphaZero; the only difference is that the network receives a short stack of recent states instead of a single board snapshot. We therefore treat multi-frame AlphaZero as a representation ablation rather than a distinct learning algorithm. See Section B.2.

AlphaZero-Auxiliary Loss (AZAL). To strengthen the training signal, AZAL augments the standard AlphaZero objective with an oracle-derived auxiliary policy loss:

$$\mathcal{L} = \mathcal{L}_{\text{policy}} + \mathcal{L}_{\text{value}} + \lambda_{\text{aux}}\mathcal{L}_{\text{aux}}.$$

This auxiliary term favors oracle-consistent actions during training, but leaves self-play, MCTS, and value targets unchanged. Thus, any improvement under AZAL is evidence that the standard learning signal—not merely the search procedure or network family—is part of the bottleneck. For labeled states, the auxiliary term is a cross-entropy loss against a softened oracle target distribution

$$q(a \mid s) \propto \big(\mathbf{1}[a \in \mathcal{B}(s)] + \varepsilon\big)\mathbf{1}[a \in \mathcal{A}(s)],$$

where $\mathcal{B}(s)$ is the oracle-optimal moveset and $\mathcal{A}(s)$ is the legal action set. The per-state auxiliary loss is then

$$\mathcal{L}_{\text{aux}}(s) = -\sum_{a} q(a \mid s) \log p_\theta(a \mid s),$$

where $p_\theta(a \mid s)$ is the policy distribution predicted by the network for state s, obtained by applying a softmax to the policy-head logits. This loss is averaged over labeled states only. Thus, AZAL biases the policy head toward oracle-consistent actions during training without changing the search procedure itself. Domain-specific label construction in Connect Four and Chomp is deferred to Section B.3.

Trace-level metrics. To summarize oracle alignment compactly, we report two shared trace-level metrics: *oracle-match rate*, the fraction of oracle-labeled moves that lie in the oracle-optimal action set, and the *longest oracle-consistent chain*, defined as the longest contiguous run of oracle-consistent moves within the rollout. These quantities are reported separately for the first and second player. The results of trace-level metrics appear in Table 1, while trace panels are shown in Figure 1.

For each trained policy, all reported trace-level metrics are computed from a single deterministic greedy rollout from the standard initial state.

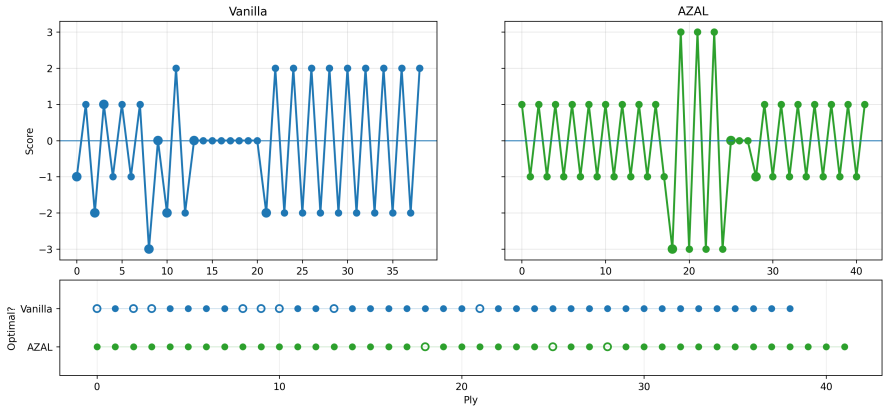
4. Vanilla AlphaZero: Strong Play Without Perfect Play

4.1. Connect Four

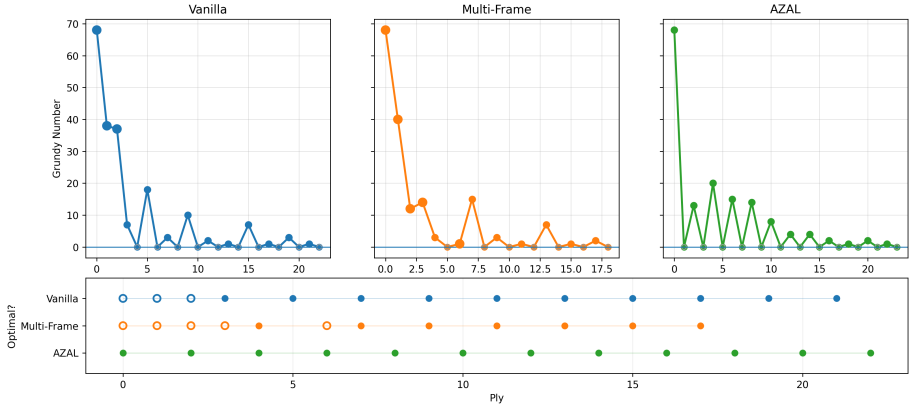
Vanilla AlphaZero learns a strong Connect Four policy, but it does not recover perfect play. The main failure is an inability to *preserve* the optimal moveset. This distinction is clearest at the trace level.

Trace evidence. The self-play rollout in Table 4(a), together with the compact metrics in Table 1 and the score trajectory in Figure 1(a), shows that vanilla AlphaZero does not reliably preserve the winning continuation. The first-player oracle-match rate is only 0.800, while the second-player rate is 0.789, and the longest oracle-consistent chains remain limited to 14 and 8 moves, respectively. Visually, Figure 1(a) shows repeated oscillation across positive, zero, and negative score regions.

The key failure is therefore not an inability to play locally strong moves. Rather, vanilla AlphaZero repeatedly leaves the optimal continuation and relinquishes its winning edge. Furthermore, the extended sequence of neutral moves in midgame suggests the model struggles to identify decisive moves. This is precisely the gap between strong play and perfect play in Connect Four: the model produces tactically plausible play, but it does not reliably preserve the small set of value-preserving decisions required for exact optimality.



(a) Connect Four trace panel.



(b) Chomp 10×11 trace panel.

Figure 1: Deterministic greedy-rollout trace panels across games. Filled markers denote oracle-consistent moves; hollow markers denote non-oracle moves.

4.2. Chomp

4.2.1. RECTANGULAR-BOARD FAILURE

For Chomp, the picture remains similar. Vanilla AlphaZero does not preserve the winning invariant, and the gap between strong and perfect play is immediate and structural. Once the agent fails to return the game to g=0, the characteristics of optimal play has already been lost.

Trace evidence first. The self-play traces on 9×10 and 10×11 in Table 2(b) and Table 2(d), together with the metrics in Table 1 and the trajectories in Figure 1(b), show that vanilla AlphaZero does *not* reliably preserve the $g = 0$ manifold once winning opportunities arise. On 9×10, the first-player oracle-match rate is only 0.600 and the longest oracle-consistent chain is 3; on 10×11, those quantities collapse to 0.000 and 0, respectively. By contrast, second-player match rates remain

Table 1: Trace-level metrics across model variants under deterministic greedy rollout from the standard initial state. Match_i denotes oracle-match rate for player i, and Chain_i denotes the longest oracle-consistent chain for player i.

Game	Model	Match_1	Chain_1	Match_2	Chain_2
Chomp 9×10	Vanilla	0.600	3	0.875	7
	Multi-Frame	0.000	0	0.667	4
	AZAL	1.000	19	0.000	0
Chomp 10×11	Vanilla	0.000	0	0.909	10
	Multi-Frame	0.250	1	0.750	6
	AZAL	1.000	12	0.000	0
Connect Four	Vanilla	0.800	14	0.789	8
	AZAL	0.905	9	0.952	12

higher in both cases, indicating that later local continuations can still look reasonable, even after the first player has already lost the invariant.

Figure 1(b) make this failure visually explicit. The early plies contain non-oracle moves from winning positions, after which the rollout no longer alternates cleanly between nonzero and zero Grundy states. A characteristic pattern emerges: AlphaZero sometimes makes correct moves in the early game, but only near the late game—after the board has been reduced substantially—does play become consistently oracle-aligned.

5. Strengthening the Signal: Multi-frame and AZAL

The vanilla results suggest the main bottleneck lies in the weakness of the standard search-learning signal. We therefore examine two possible remedies. The first is a multi-frame representation motivated by prior work on impartial games [8]; the second is AlphaZero Auxiliary Loss (AZAL), which injects sparse oracle-derived supervision directly into training.

5.1. Multi-frame AlphaZero

The multi-frame variant fails to resolve the main limitations of vanilla AlphaZero. On harder Chomp boards, its training curves in Figures 2 and 3 plateau at levels that are comparable to or worse than vanilla AlphaZero, especially in policy and total loss. The self-play traces in Table 3(a) and Table 3(c), the compact metrics in Table 1, and the trajectories in Figure 1(b) all show that multi-frame AlphaZero still fails to preserve the winning invariant reliably. On 10×11, for example, the first-player oracle-match rate rises only to 0.250 and the longest oracle-consistent chain remains just 1, indicating that additional temporal context alone is not sufficient to recover exact play.

5.2. AZAL on Connect Four

AZAL improves Connect Four substantially to near-perfection, reducing and delaying the errors that remain decisive under vanilla AlphaZero.

Trace evidence. In self-play, AZAL is markedly more stable than vanilla AlphaZero; see Table 4(b), Table 1, and Figure 1(a). Relative to vanilla AlphaZero, the first-player oracle-match rate rises from

0.800 to 0.905, while the second-player rate rises from 0.789 to 0.952. The corresponding score trajectory in Figure 1(a) also shows a longer value-preserving prefix before the rollout eventually leaves the optimal self-play branch.

The key difference from vanilla AlphaZero is that the number of decisive branch errors is substantially reduced and pushed later into the game. Even after the first major deviation, later play often remains locally strong within the new tactical regime. This is precisely what defines near-perfect play in Connect Four: AZAL preserves the correct moveset much longer than vanilla AlphaZero, but still does not eliminate all decisive errors.

5.3. AZAL on Chomp

AZAL changes the Chomp results more dramatically, materially improving both training dynamics and gameplay traces and recovering perfect play on the larger rectangular board.

Trace evidence. The self-play traces in Table 3(b) and Table 3(d), together with the metrics in Table 1 and the trajectories in Figure 1(b), show that AZAL preserves the winning invariant throughout play on both 9×10 and 10×11. In Table 1, the first-player oracle-match rate reaches 1.000 on both boards, with longest oracle-consistent chains of 19 and 12, respectively. Visually, the Chomp panels in Figure 1 exhibit the alternating nonzero/zero pattern characteristic of exact play rather than the leakage seen in vanilla and multi-frame AlphaZero.

6. Discussion and Conclusion

The central result of this paper is that strong play is not the same as perfect play. Across both Connect Four and Chomp, vanilla AlphaZero achieves strong empirical behavior, yet trace-level oracle evaluation shows that it fails to preserve the sparse long-horizon structure required for exact play.

The results suggest a specific failure mode: standard AlphaZero self-play produces moving targets generated by the agent's own evolving approximation, so early branch errors can distort the future training distribution. The negative multi-frame result indicates that additional temporal context alone is not enough in our setting, while AZAL shows that stronger supervision can recover substantially more of the relevant structure.

The two domains expose different versions of this same issue. In Connect Four, exact play is fragile as small early deviations can change the game-theoretic trajectory, so AZAL remains near-perfect rather than exact. In Chomp, optimal play is governed by the invariant $g(s) \neq 0 \rightarrow g(s') = 0$, and AZAL recovers this structure cleanly on the larger rectangular boards.

Overall, the main bottleneck appears to be the weakness of the standard AlphaZero search-learning signal rather than representation alone. More broadly, our results suggest that high win rates can conceal systematic failures on the small set of decisions that determine exact optimality. Better supervision closes much of the gap to perfect play, while the remaining challenge is to recover exact play without relying on external oracles.

References

[1] L. V. Allis. *A Knowledge-Based Approach of Connect-Four*. PhD thesis, Vrije Universiteit Amsterdam, 1988.

[2] Anonymous. Weak neural networks and the limits of alphazero-style learning. *Symmetry*, 2026.

[3] T. Anthony, Z. Tian, and D. Barber. Thinking fast and slow with deep learning and tree search. *arXiv preprint arXiv:1705.08439*, 2017.

[4] A. E. Brouwer, G. Horváth, I. Molnár-Sáska, and C. Szabó. On three-rowed chomp. *Integers: Electronic Journal of Combinatorial Number Theory*, 5(G07), 2005.

[5] D. Gale. The sprague–grundy theorem. *Combinatorial Game Theory Notes*, 2015.

[6] David Gale. A curious nim-type game. *The American Mathematical Monthly*, 81(8):876–879, 1974. doi: 10.2307/2319446.

[7] Pascal Pons. connect4: Connect 4 solver. https://github.com/PascalPons/connect4, 2015. Accessed 2026.

[8] S. Riis. Mastering NIM and impartial games with weak neural networks: An AlphaZero-inspired multi-frame approach. *arXiv preprint arXiv:2411.06403*, 2024.

[9] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 2017.

[10] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, and D. Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[11] B. Zhou and S. Riis. Impartial games: A challenge for reinforcement learning. *arXiv preprint arXiv:2205.12787*, 2022.

Appendix A. Oracle Implementation Details

A.1. Connect Four oracle details

Our Connect Four oracle is the perfect solver of Pascal Pons [7], which evaluates positions via exact negamax alpha–beta search with transposition reuse. Actions are indexed by column with illegal moves masked; internally, the solver uses a compact bitboard representation. The solver returns an exact *game-theoretic score*. Let $\sigma(s, a)$ denote the score of taking action a in state s, and let

$$m(s) := \mathrm{nbMoves}(s). \tag{1}$$

Then

$$\sigma(s,a) = \begin{cases} \dfrac{WH + 1 - m(s)}{2}, & \text{if } a \text{ wins immediately,} \\ -v\big(T(s,a)\big), & \text{otherwise,} \end{cases} \tag{2}$$

and the value of a state is

$$v(s) = \max_{a \in \mathcal{A}(s)} \sigma(s,a). \tag{3}$$

Positive values indicate forced wins, negative values forced losses, and zero draws; magnitude reflects distance to resolution. Although standard 6×7 Connect Four is solved [1], exact evaluation remains computationally expensive in tactically delayed positions.

A.2. Chomp oracle details

Our Chomp oracle is a recursive *Grundy-number solver*. Each state s has a Grundy number $g(s) \in \mathbb{N}$, where $g(s) = 0$ iff the position is losing [5]. The oracle is defined as

$$g(s) = \mathrm{mex}\big(\{g(s') : s' \in \mathcal{N}(s)\}\big), \tag{4}$$

where $\mathcal{N}(s)$ is the set of legal successors. From any winning state ($g(s) \neq 0$), an optimal move reaches a successor with $g(s') = 0$.

States are represented by non-increasing row-length profiles $\mathbf{s} = (s_0, \ldots, s_{H-1})$. A move selecting (r, c) truncates all rows at and below r:

$$s'_{r'} = \min(s_{r'}, c) \quad \text{for } r' \geq r.$$

Actions are indexed by $n = rW + c$ with masking of illegal moves. The solver recursively evaluates successors with memoization. The number of reachable states grows exponentially with board size, making large boards computationally challenging.

Appendix B. Additional Method Details

B.1. Vanilla AlphaZero implementation

The vanilla baseline uses a convolutional residual policy–value network of the standard AlphaZero. Given an encoded board state s, the network outputs (i) policy logits over the fixed action space and (ii) a scalar value estimate:

$$(\mathbf{p}_\theta(s), v_\theta(s)) = f_\theta(s). \tag{5}$$

In our implementation, the input is a single board frame. The network begins with a 3×3 convolution, batch normalization, and ReLU activation, then passes through a stack of residual blocks with identity skip connections. The shared trunk is followed by separate policy and value heads. The policy head ends in a linear projection to the action space, while the value head ends in a scalar output with $\tanh$ activation so predictions lie in $[-1, 1]$.

More concretely, if $\mathbf{h}_0$ denotes the output of the initial convolutional block, then the residual trunk computes

$$\mathbf{h}_{k+1} = \mathbf{h}_k + F_k(\mathbf{h}_k), \tag{6}$$

where each F_k consists of two 3×3 convolutions with batch normalization and ReLU nonlinearity. The policy head maps the final shared representation to logits over legal actions, and the value head maps it to a single scalar:

$$\mathbf{p}_\theta(s) \in \mathbb{R}^{|\mathcal{A}|}, \qquad v_\theta(s) \in [-1, 1]. \tag{7}$$

Search uses a PUCT-style selection rule,

$$a^* = \arg\max_a \left[Q(s,a) + C\, P(a \mid s) \frac{\sqrt{\sum_b N(s,b)}}{1 + N(s,a)} \right], \tag{8}$$

where $Q(s,a)$ is the running value estimate, $N(s,a)$ is the visit count, and $P(a \mid s)$ is the policy prior returned by the network. Root priors are perturbed with Dirichlet noise during self-play, legal moves are masked before normalization, and the final root visit counts define the improved policy target.

Training data are generated by parallel self-play. For each visited state, the replay buffer stores the encoded state, the normalized MCTS visit-count distribution, and the final game outcome from the perspective of the player to move at that state. During optimization, the network is trained on minibatches sampled from this replay memory using the standard policy-plus-value loss

$$\mathcal{L}_{\mathrm{AZ}} = \mathcal{L}_{\mathrm{policy}} + \mathcal{L}_{\mathrm{value}}, \tag{9}$$

where the policy term is cross-entropy against the MCTS-improved target distribution and the value term is mean-squared error against the terminal game outcome.

B.2. Multi-frame AlphaZero implementation

The multi-frame AlphaZero model is included as a representation ablation. All search, replay, and optimization procedures are kept fixed so that the comparison isolates the effect of input representation alone.

The only substantive change relative to vanilla AlphaZero is the input encoding. Instead of supplying a single board snapshot s_t, the model receives a short history of recent board states,

$$(s_t, s_{t-1}, \ldots, s_{t-K+1}),$$

stacked along the channel dimension. In our implementation, the replay buffer stores encoded histories constructed from the three most recent frames of game state, rather than a single-frame board tensor.

The multi-frame network uses the same residual architecture as the vanilla baseline, with one change at the input layer: the first convolution accepts multiple input channels instead of one. In the

implementation, the start block maps `num_frames` input channels into the shared hidden dimension, after which the same residual backbone, policy head, and value head are used as in the single-frame model. Thus, the multi-frame experiment should be interpreted as a controlled representation change, not as a different search procedure or optimization pipeline.

B.3. AZAL-specific implementation

AZAL shares the same network architecture, self-play loop, MCTS procedure, and value targets as vanilla AlphaZero. The only implementation change is the addition of an auxiliary policy term:

$$\mathcal{L} = \mathcal{L}_{\text{policy}} + \mathcal{L}_{\text{value}} + \lambda_{\text{aux}}\mathcal{L}_{\text{aux}}, \tag{10}$$

where λ_{aux} corresponds to the implementation parameter `p_move_lambda`. Domain-specific oracle-label construction for Connect Four and Chomp is described below.

Connect Four auxiliary labels. In Connect Four, oracle supervision comes from the perfect solver of Pascal Pons. During self-play, each replay element stores not only the encoded state, MCTS policy target, and final outcome, but also the move sequence leading to that state. After each self-play iteration, duplicated sequences are removed and the remaining unique sequences are queried in batches through the solver. The returned exact scores are then converted into oracle-best movesets and cached as replay-side labels before gradient updates begin. This means that oracle labels are attached once per iteration rather than recomputed inside every training batch.

Chomp auxiliary labels. In Chomp, oracle labels are derived from the Grundy solver. Positions that are terminal or fail oracle evaluation are masked out of the auxiliary term. To avoid repeated work, the set of best oracle moves is cached by serialized board state. As a result, the fraction of batch elements that contribute to the auxiliary loss can vary across training.

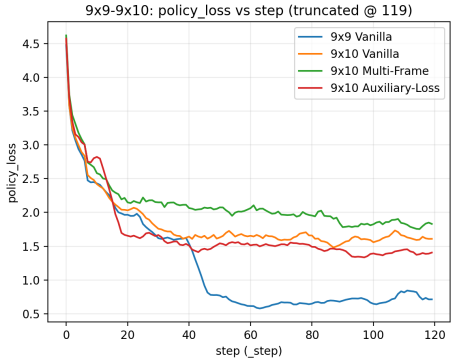
B.4. Protocol and logging details

All experiments use the same high-level training loop. At each iteration, the current network is placed in evaluation mode and used to guide MCTS-based self-play across multiple games in parallel. For each root state, MCTS returns visit counts over legal actions, these counts are normalized to produce an improved policy target, and an action is sampled from a temperature-adjusted distribution for rollout generation. Once a game terminates, every stored state in that trajectory is assigned the final outcome from the perspective of the player to move at that state and added to replay memory.

We log policy, value, and total losses; for AZAL, we additionally log the auxiliary loss and labeled fraction. In Connect Four, we track oracle-query statistics per iteration. We also record self-play traces and evaluate each move with oracle annotations in both domains.

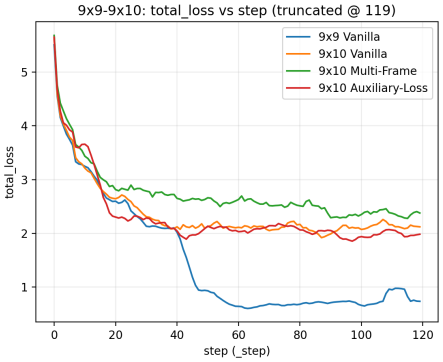
We evaluate each method against exact or oracle-derived notions of optimality rather than relying only on empirical win rate. In Connect Four, we compare play against the exact game-theoretic scores returned by the perfect solver. In Chomp, we use the Grundy oracle to classify positions as winning or losing and to determine whether a move preserves the invariant $g(s) \neq 0 \rightarrow g(s') = 0$ that characterizes optimal first-player control.

Figure 2: Training curves for Chomp on the 9×9 and 9×10 boards. Curves are truncated to a common training horizon within this board-size group to enable direct comparison of convergence trends. The panels report policy loss, value loss, total loss, and, for the 9×10 board, the auxiliary-policy loss p_{move}.

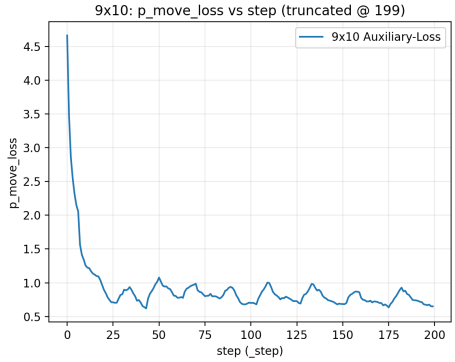


(a) 9×9 and 9×10 policy loss

(b) 9×9 and 9×10 value loss

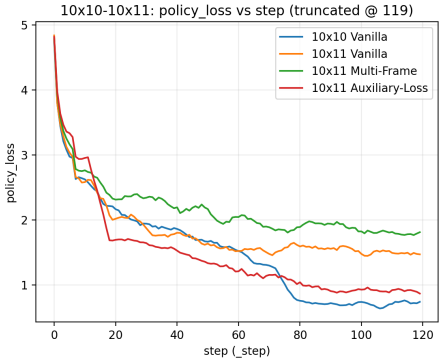


(c) 9×9 and 9×10 total loss

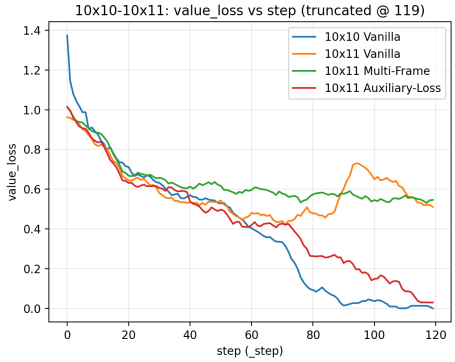


(d) 9×10 p_{move} loss

Figure 3: Training curves for Chomp on the 10×10 and 10×11 boards. Curves are truncated to a common training horizon within this board-size group to enable direct comparison of convergence trends. The panels report policy loss, value loss, total loss, and, for the 10×11 board, the auxiliary-policy loss p_{move}.

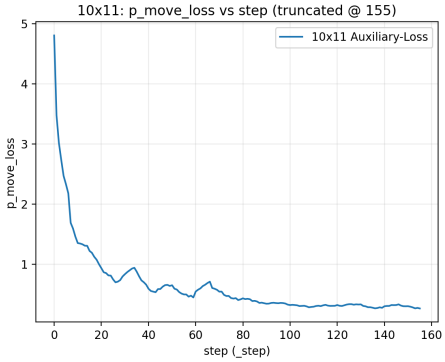


(a) 10×10 and 10×11 policy loss



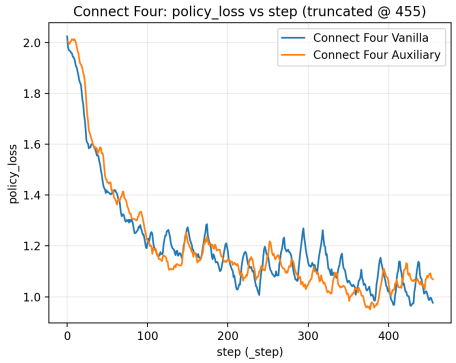
(b) 10×10 and 10×11 value loss

(c) 10×10 and 10×11 total loss

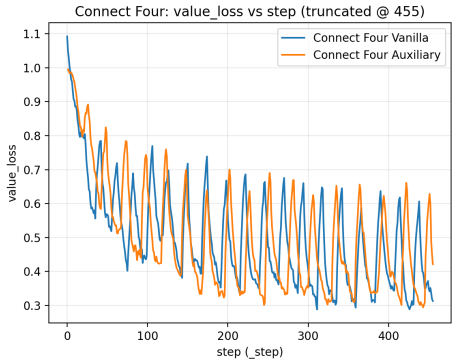


(d) 10×11 p_{move} loss

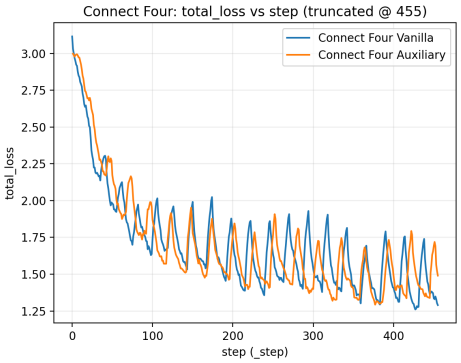
Figure 4: Training curves for Connect Four across model variants. Curves are truncated to a common training horizon to enable direct comparison of convergence trends. The panels report policy loss, value loss, total loss, and the auxiliary-policy loss p_{move}.



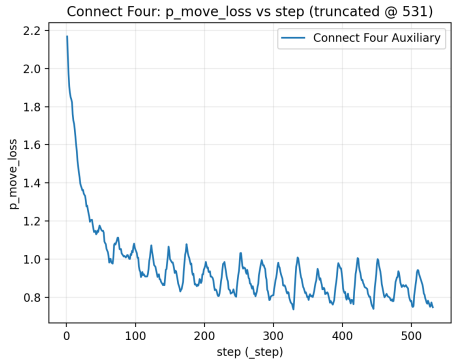
(a) Policy loss



(b) Value loss



(c) Total loss



(d) p_{move} loss

Table 2: Move-level trace tables for vanilla AlphaZero on Chomp in self-play across board sizes. Here $g(s_t)$ denotes the Grundy number of the position before the move at ply t. When $g(s_t) = 0$, the position is losing, so the oracle does not define a "best" move; these entries are shown as –.

Ply t	Player	$g(s_t)$	AZ move n	Oracle best n	Optimal?
0	AZ-1	20	10	10	yes
1	AZ-2	0	72	–	–
2	AZ-1	15	8	8	yes
3	AZ-2	0	63	–	–
4	AZ-1	1	7	7	yes
5	AZ-2	0	54	–	–
6	AZ-1	3	6	6	yes
7	AZ-2	0	5	–	–
8	AZ-1	1	45	45	yes
9	AZ-2	0	36	–	–
10	AZ-1	7	4	4	yes
11	AZ-2	0	3	–	–
12	AZ-1	1	27	27	yes
13	AZ-2	0	18	–	–
14	AZ-1	3	2	2	yes
15	AZ-2	0	1	–	–
16	AZ-1	1	9	9	yes
17	AZ-2	0	0	–	–

(a) Chomp Vanilla AZ (9×9)

Ply t	Player	$g(s_t)$	AZ move n	Oracle best n	Optimal?
0	AZ-1	15	72	67, 84	no
1	AZ-2	48	24	46	no
2	AZ-1	12	52	33, 52, 60	yes
3	AZ-2	0	60	–	–
4	AZ-1	13	33	8, 17, 33	yes
5	AZ-2	0	18	–	–
6	AZ-1	15	23	23	yes
7	AZ-2	0	8	–	–
8	AZ-1	4	42	16	no
9	AZ-2	18	14	14	yes
10	AZ-1	0	32	–	–
11	AZ-2	3	13	13	yes
12	AZ-1	0	22	–	–
13	AZ-2	11	50	50	yes
14	AZ-1	0	5	–	–
15	AZ-2	3	30	11, 30	yes
16	AZ-1	0	12	–	–
17	AZ-2	1	4	4	yes
18	AZ-1	0	11	–	–
19	AZ-2	1	3	3	yes
20	AZ-1	0	10	–	–
21	AZ-2	2	1	1	yes
22	AZ-1	0	0	–	–

(b) Chomp Vanilla AZ (9×10)

Ply t	Player	$g(s_t)$	AZ move n	Oracle best n	Optimal?
0	AZ-1	19	11	11	yes
1	AZ-2	0	90	–	–
2	AZ-1	1	9	9	yes
3	AZ-2	0	80	–	–
4	AZ-1	15	8	8	yes
5	AZ-2	0	70	–	–
6	AZ-1	1	7	7	yes
7	AZ-2	0	6	–	–
8	AZ-1	3	60	60	yes
9	AZ-2	0	5	–	–
10	AZ-1	1	50	50	yes
11	AZ-2	0	4	–	–
12	AZ-1	7	40	40	yes
13	AZ-2	0	3	–	–
14	AZ-1	1	30	30	yes
15	AZ-2	0	2	–	–
16	AZ-1	3	20	20	yes
17	AZ-2	0	1	–	–
18	AZ-1	1	10	10	yes
19	AZ-2	0	0	–	–

(c) Chomp Vanilla AZ (10×10)

Ply t	Player	$g(s_t)$	AZ move n	Oracle best n	Optimal?
0	AZ-1	68	6	24	no
1	AZ-2	38	93	46	no
2	AZ-1	37	35	46	no
3	AZ-2	7	88	88	yes
4	AZ-1	0	26	–	–
5	AZ-2	18	45	45	yes
6	AZ-1	0	25	–	–
7	AZ-2	3	34	34	yes
8	AZ-1	0	15	–	–
9	AZ-2	10	23	23, 77	yes
10	AZ-1	0	12	–	–
11	AZ-2	2	66	66	yes
12	AZ-1	0	55	–	–
13	AZ-2	1	5	5	yes
14	AZ-1	0	4	–	–
15	AZ-2	7	44	44	yes
16	AZ-1	0	33	–	–
17	AZ-2	1	3	3	yes
18	AZ-1	0	22	–	–
19	AZ-2	3	2	2	yes
20	AZ-1	0	1	–	–
21	AZ-2	1	11	11	yes
22	AZ-1	0	0	–	–

(d) Chomp Vanilla AZ (10×11)

Table 3: Move-level trace tables for Multi-Frame AlphaZero and AlphaZero-Auxiliary Loss on Chomp in self-play across board sizes. Here $g(s_t)$ denotes the Grundy number of the position before the move at ply t. When $g(s_t) = 0$, the position is losing, so the oracle does not define a "best" move; these entries are shown as –.

Ply t	Player	$g(s_t)$	AZ move n	Oracle best n	Optimal?
0	AZ-1	15	48	67, 84	no
1	AZ-2	52	46	66	no
2	AZ-1	25	30	54, 72	no
3	AZ-2	21	4	15	no
4	AZ-1	9	22	12	no
5	AZ-2	7	12	12	yes
6	AZ-1	0	11	–	–
7	AZ-2	1	3	3	yes
8	AZ-1	0	20	–	–
9	AZ-2	3	2	2	yes
10	AZ-1	0	1	–	–
11	AZ-2	1	10	10	yes
12	AZ-1	0	0	–	–

(a) Chomp Multi-Frame AZ (9×10)

Ply t	Player	$g(s_t)$	AZ move n	Oracle best n	Optimal?
0	AZ-1	15	84	67, 84	yes
1	AZ-2	0	55	–	–
2	AZ-1	25	74	18, 36, 74	yes
3	AZ-2	0	49	–	–
4	AZ-1	10	83	83	yes
5	AZ-2	0	46	–	–
6	AZ-1	7	18	18	yes
7	AZ-2	0	54	–	–
8	AZ-1	5	82	35, 82	yes
9	AZ-2	0	73	–	–
10	AZ-1	10	35	35	yes
11	AZ-2	0	60	–	–
12	AZ-1	24	7	7	yes
13	AZ-2	0	44	–	–
14	AZ-1	17	53	53	yes
15	AZ-2	0	52	–	–
16	AZ-1	20	34	34	yes
17	AZ-2	0	43	–	–
18	AZ-1	18	26	26	yes
19	AZ-2	0	25	–	–
20	AZ-1	4	41	41	yes
21	AZ-2	0	5	–	–
22	AZ-1	2	33	33	yes
23	AZ-2	0	32	–	–
24	AZ-1	13	24	24	yes
25	AZ-2	0	31	–	–
26	AZ-1	9	3	3, 12	yes
27	AZ-2	0	22	–	–
28	AZ-1	9	50	50	yes
29	AZ-2	0	2	–	–
30	AZ-1	6	40	40	yes
31	AZ-2	0	21	–	–
32	AZ-1	4	30	30	yes
33	AZ-2	0	20	–	–
34	AZ-1	2	11	11	yes
35	AZ-2	0	10	–	–
36	AZ-1	1	1	1	yes
37	AZ-2	0	0	–	–

(b) Chomp Auxiliary Loss AZ (9×10)

Ply t	Player	$g(s_t)$	AZ move n	Oracle best n	Optimal?
0	AZ-1	68	70	24	no
1	AZ-2	40	100	24	no
2	AZ-1	12	25	2	no
3	AZ-2	14	12	2	no
4	AZ-1	3	10	10	yes
5	AZ-2	0	9	–	–
6	AZ-1	1	7	99	no
7	AZ-2	15	77	77	yes
8	AZ-1	0	66	–	–
9	AZ-2	3	6	6	yes
10	AZ-1	0	55	–	–
11	AZ-2	1	5	5	yes
12	AZ-1	0	4	–	–
13	AZ-2	7	44	44	yes
14	AZ-1	0	33	–	–
15	AZ-2	1	3	3	yes
16	AZ-1	0	11	–	–
17	AZ-2	2	1	1	yes
18	AZ-1	0	0	–	–

(c) Chomp Multi-Frame AZ (10×11)

Ply t	Player	$g(s_t)$	AZ move n	Oracle best n	Optimal?
0	AZ-1	68	24	24	yes
1	AZ-2	0	19	–	–
2	AZ-1	13	99	99	yes
3	AZ-2	0	18	–	–
4	AZ-1	20	67	10, 67	yes
5	AZ-2	0	10	–	–
6	AZ-1	15	17	17	yes
7	AZ-2	0	56	–	–
8	AZ-1	14	15	15	yes
9	AZ-2	0	88	–	–
10	AZ-1	8	7	7, 14, 34	yes
11	AZ-2	0	34	–	–
12	AZ-1	4	13	13	yes
13	AZ-2	0	66	–	–
14	AZ-1	4	23	5, 23	yes
15	AZ-2	0	4	–	–
16	AZ-1	2	55	55	yes
17	AZ-2	0	44	–	–
18	AZ-1	1	12	12	yes
19	AZ-2	0	2	–	–
20	AZ-1	2	22	22	yes
21	AZ-2	0	11	–	–
22	AZ-1	1	1	1	yes
23	AZ-2	0	0	–	–

(d) Chomp Auxiliary Loss AZ (10×11)

Table 4: Move-level trace tables for vanilla AlphaZero and AlphaZero-Auxiliary Loss on Connect Four in self-play. Here $\sigma(s_t)$ denotes the oracle score associated with the action chosen at ply t from state s_t. Positive values indicate winning continuations, negative values indicate losing continuations, and zero indicates drawing continuations.

Ply t	Player	$\sigma(s_t)$	AZ move n	Oracle best n	Optimal?
0	AZ-1	-1	5	3	no
1	AZ-2	1	4	4	yes
2	AZ-1	-2	2	4, 5	no
3	AZ-2	1	2	4	no
4	AZ-1	-1	4	4	yes
5	AZ-2	1	4	4	yes
6	AZ-1	-1	4	4	yes
7	AZ-2	1	4	4, 5	yes
8	AZ-1	-3	4	3, 2, 5, 0, 6	no
9	AZ-2	0	0	3	no
10	AZ-1	-2	2	0	no
11	AZ-2	2	2	2	yes
12	AZ-1	-2	2	3, 2, 0, 6	yes
13	AZ-2	0	2	5	no
14	AZ-1	0	0	3, 0	yes
15	AZ-2	0	6	3, 1, 5, 0, 6	yes
16	AZ-1	0	0	3, 5, 0	yes
17	AZ-2	0	0	3, 1, 5, 0, 6	yes
18	AZ-1	0	0	3, 1, 5, 0, 6	yes
19	AZ-2	0	0	3, 1, 5, 0, 6	yes
20	AZ-1	0	3	3, 1, 5, 6	yes
21	AZ-2	-2	3	1, 5, 6	no
22	AZ-1	2	3	3	yes
23	AZ-2	-2	3	3	yes
24	AZ-1	2	5	5	yes
25	AZ-2	-2	5	5	yes
26	AZ-1	2	3	3, 5	yes
27	AZ-2	-2	3	3, 5, 6	yes
28	AZ-1	2	5	5	yes
29	AZ-2	-2	5	5	yes
30	AZ-1	2	5	1, 5, 6	yes
31	AZ-2	-2	6	1, 6	yes
32	AZ-1	2	1	1, 6	yes
33	AZ-2	-2	6	6	yes
34	AZ-1	2	6	6	yes
35	AZ-2	-2	6	6	yes
36	AZ-1	2	6	6	yes
37	AZ-2	-2	1	1	yes
38	AZ-1	2	1	1	yes

(a) Connect Four Vanilla AZ

Ply t	Player	$\sigma(s_t)$	AZ move n	Oracle best n	Optimal?
0	AZ-1	1	3	3	yes
1	AZ-2	-1	3	3	yes
2	AZ-1	1	3	3	yes
3	AZ-2	-1	3	3	yes
4	AZ-1	1	3	3	yes
5	AZ-2	-1	2	3, 2, 4, 1, 5, 0, 6	yes
6	AZ-1	1	2	2, 5	yes
7	AZ-2	-1	2	2	yes
8	AZ-1	1	2	2	yes
9	AZ-2	-1	0	2, 5, 0	yes
10	AZ-1	1	5	3, 2, 5	yes
11	AZ-2	-1	6	6	yes
12	AZ-1	1	4	3, 2, 4, 5, 0, 6	yes
13	AZ-2	-1	4	4	yes
14	AZ-1	1	4	4	yes
15	AZ-2	-1	4	4, 5, 6	yes
16	AZ-1	1	4	4	yes
17	AZ-2	-1	2	3, 2, 4, 1, 5, 0, 6	yes
18	AZ-1	-3	6	5	no
19	AZ-2	3	6	6	yes
20	AZ-1	-3	6	6	yes
21	AZ-2	3	6	6	yes
22	AZ-1	-3	4	3, 2, 4, 5, 0, 6	yes
23	AZ-2	3	3	3, 2, 5, 0, 6	yes
24	AZ-1	-3	6	2, 5, 0, 6	yes
25	AZ-2	0	1	2, 5, 0	no
26	AZ-1	0	1	1	yes
27	AZ-2	0	1	1	yes
28	AZ-1	-1	1	2	no
29	AZ-2	1	2	2	yes
30	AZ-1	-1	0	1, 5, 0	yes
31	AZ-2	1	5	5, 0	yes
32	AZ-1	-1	5	1, 5, 0	yes
33	AZ-2	1	5	5, 0	yes
34	AZ-1	-1	0	1, 5, 0	yes
35	AZ-2	1	0	0	yes
36	AZ-1	-1	1	1, 5, 0	yes
37	AZ-2	1	1	1	yes
38	AZ-1	-1	5	5	yes
39	AZ-2	1	5	5	yes
40	AZ-1	-1	0	0	yes
41	AZ-2	1	0	0	yes

(b) Connect Four Auxiliary Loss AZ