Extended Abstract

Motivation Gomoku (Five-in-a-Row) is a two-player board game where players alternate placing stones on a 15x15 grid, aiming to be the first to connect five in a line vertically, horizontally, or diagonally. On the surface, Gomoku is a simple game, but its large search space makes it a great challenge for developing game-playing AI. Success in Gomoku depends heavily on long-term planning and anticipating the opponent's strategy several moves ahead. This means reinforcement learning agents must learn to handle long action sequences and spatial patterns. Recently, transformer models with attention mechanisms have gained popularity for tasks like this, since they're able to process entire move histories and board states at once.

Method Inspired by AlphaGo Zero's success, we train agents via self-play reinforcement learning, with Monte Carlo Tree Search (MCTS) to explore game outcomes and guide learning. MCTS is paired with a learned policy-value network that evaluates positions and suggests promising moves. We experiment with three different model architectures for this policy-value model: (i) **ResNet** (a deep residual CNN), (ii) **HybridNet** (a CNN trunk + global self-attention), and (iii) **TemporalNet** (CNN trunk + temporal attention over the last K board configurations). Each network returns a move prior and a win-probability estimate that guide MCTS rollouts.

Implementation We adopted an existing Gomoku framework, incorporating our own architectures and modifying the environment accordingly. To make training tractable, we used a reduced board size of 6×6 and a win condition of 4 in a row. We trained our HybridNet model on 7×7 with a win condition of 5 in a row. Agents trained by playing against themselves, and MCTS guided move selection by expanding game state nodes and using the network's policy and value predictions.

Results Across our experiments, the **HybridNet** agent was the strongest performer, outperforming both the TemporalNet and the ResNet baselines in head-to-head matches. In a round-robin tournament between the agents, the HybridNet-based MCTS won the vast majority of games against the other two models. The TemporalNet performed second-best, while the ResNet performed worst. Our results indicate that the ResNet has difficulty capturing long-range dependencies and complex board states with purely convolutional layers. Qualitatively, we observed that the HybridNet's global attention enabled it to detect and respond to multi-directional threats (such as potential "forks" where a single move can create two winning lines) much more effectively than the other models. The TemporalNet-based agent showed strength in situations requiring remembrance of previous moves (for instance, noticing repeated setups or prolonged defenses). Overall, the HybridNet consistently demonstrated superior strategic play, confirming that introducing self-attention significantly improved the agent's playing strength.

Discussion As the HybridNet beat out the TemporalNet, we deduce that spatial reach matters more than temporal reach in Gomoku. Furthermore, as the HybridNet also outperformed the ResNet, we see that the attention layer, by identifying long-term spacial dependencies in a single operation instead of multiple convolutional hops, is better suited for Gomoku. Future extensions could blend both strengths: a dual-channel network that applies global self-attention within each frame and temporal attention across frames, or an adaptive attention scheme that allocates more heads to high-leverage regions such as board centers while sparsifying elsewhere. Such hybrids might further cut rollouts and raise sample efficiency while preserving interpretability.

Conclusion We found that the choice of model architecture has a profound impact on learning effective strategy. By experimenting with residual-CNN and spacial/temporal attention models, we learned that aligning the model design with the game's fundamental demands leads to the best performance. In Gomoku's case, a model that can capture global spatial patterns (the HybridNet) performed significantly better than those that couldn't. This suggests that transformer-like self-attention conduces powerful situational awareness and long-horizon planning. Architectural choices thus critically shape an agent's strategic development, and we recommend further research into hybrid architectures (both spatial and temporal) and adaptive attention techniques for game-playing AI. The insights from SigmaGo could generalize to other domains where long-term reasoning and global context are key, paving the way for more advanced and generalizable game-playing agents.

SigmaGo: Five Steps ~~Stones~~ Ahead

Emily Xia
Department of Computer Science
Stanford University
emxia18@stanford.edu

Eric Chen
Department of Computer Science
Stanford University
erichchen@stanford.edu

Robin Li
Department of Computer Science
Stanford University
lirobin@stanford.edu

Abstract

Gomoku presents a compelling challenge for reinforcement learning due to its simple rules but deep strategic complexity. In this project, we explore the effectiveness of attention-based neural network architectures in guiding Monte Carlo Tree Search (MCTS) for self-play Gomoku agents. We compare three models: a baseline ResNet, a HybridNet with global self-attention, and a TemporalNet with attention over prior board states. Using a reduced board setup for tractable training, we evaluate each model's performance through direct head-to-head matches and qualitative analysis. Our results show that the HybridNet outperforms both alternatives, excelling at detecting multi-directional threats and planning ahead. These findings highlight the value of self-attention in reinforcement learning for games requiring spatial awareness and long-term reasoning, and suggest promising directions for future hybrid architectures.

1 Introduction

Gomoku, or Five-in-a-Row, is a two-player board game where players alternate placing stones on a 15×15 grid, aiming to connect five stones in a line either horizontally, vertically, or diagonally. Despite its simple rules, Gomoku presents a rich strategic space with a vast number of possible game states, making it an ideal candidate for reinforcement learning (RL) research. Strategic depth, the importance of long-term planning, and the necessity of responding to opponent threats several moves in advance make Gomoku a meaningful domain for evaluating modern decision-making architectures.

Recent advances in game-playing AI, most notably AlphaGo, AlphaZero, and MuZero, have demonstrated the power of combining self-play reinforcement learning with Monte Carlo Tree Search (MCTS) and deep neural networks Silver et al. (2016, 2017a,b); Schrittwieser et al. (2020). These systems learn to play at a superhuman level entirely from scratch, relying only on the rules of the game and feedback from simulated play. Their success has inspired research into building similarly capable agents across a wide range of games and sequential decision problems.

Most of these agents rely on convolutional neural networks (CNNs) or recurrent architectures to encode game states. While effective, these models can struggle with capturing long-range dependencies or maintaining interpretability. In contrast, transformer-based architectures—originally designed for natural language processing—offer global self-attention mechanisms that allow them to attend to all positions in a sequence simultaneously Li et al. (2023). In RL, this capacity translates to a stronger ability to reason over entire board configurations and full histories of prior moves, which is particularly valuable in games like Gomoku where distant threats or tactical traps may not be locally

detectable. While several recent surveys explore transformers in RL Li et al. (2023); Glanois et al. (2022), empirical studies in competitive, combinatorial games remain limited.

In this project, we aim to bridge that gap. We evaluate the impact of transformer-based architectures in Gomoku by integrating them into the policy-value network of an MCTS-guided self-play reinforcement learning agent. Specifically, we compare three architectures:

1. **ResNet:** A baseline deep residual convolutional model that processes only the current board state.
2. **HybridNet:** A CNN trunk followed by a global self-attention layer over the spatial board representation, enabling it to capture long-range interactions between board positions.
3. **TemporalNet:** A model with temporal attention that processes a sequence of past board states, offering memory of historical context but limited spatial reach.

We utilize the Gomoku MCTS training framework from Song (2025), modernizing and adopting it for our complex model architectures. We train all agents through self-play using a reduced board size (6×6) and a win condition of 4 in a row for tractability. MCTS is used to explore game trajectories Kocsis and Szepesvári (2006), with the learned networks providing both the prior move distribution and state value estimates needed for efficient search. To assess performance, we conduct round-robin tournaments between agents, alternating which agent moves first and analyzing their gameplay qualitatively and quantitatively.

Our contributions are threefold:

- We adapt and evaluate transformer-style attention mechanisms in a classic two-player board game setting where both spatial and temporal reasoning are critical.
- We introduce and compare two novel attention-augmented architectures—HybridNet and TemporalNet—each tailored to different strategic demands of Gomoku.
- We demonstrate that HybridNet, by enabling global spatial reasoning, significantly outperforms other architectures, offering insights into how architectural choices shape strategic learning and sample efficiency.

These findings suggest that attention mechanisms, particularly spatial self-attention, are a valuable addition to reinforcement learning agents in structured, adversarial environments. Our results also motivate future research into hybrid and adaptive attention architectures that can dynamically allocate resources based on board context.

2 Related Work

Game-playing AI have been some of the most notable and successful applications of machine learning technology for several decades. Game-playing models are useful for several reasons, including benchmarking current algorithmic methods, ensuring AI safety, and even pioneering new approaches to training models. Some of the earliest examples of game-playing AI include TD-Gammon, which achieved expert-level performance in BackGammon with a simple three-layer feed-forward neural network (Tesauro et al., 1995). Another example came in 1997, when IBM's Deep Blue defeated the reigning world chess champion, Garry Kasparov.

In the past decade, Google DeepMind created AlphaGo, which attained superhuman performance using a deep neural networks and Monte-Carlo tree search algorithms (Silver et al., 2016). DeepMind further advanced their progress using a new paradigm: *tabula rasa* reinforcement learning. *Tabula rasa* RL trains a model by matching it against itself billions to trillions of times to learn optimal strategies without any prior human knowledge input. This resulted in two even more advanced models, AlphaGo Zero in Go (Silver et al., 2017b), and AlphaZero in chess (Silver et al., 2017a). To this day, these remain some of the most formidable engines in their respective games.

Subsequent work has extended these breakthroughs and showed the promise of further abstraction away from game-specific environments. For instance, MuZero learns the rules and dynamics of a game from scratch using a game-agnostic model, and is able to achieve even better performance than AlphaZero in Go, chess, and shogi (Schrittwieser et al., 2020). OpenAI Five demonstrated

the capability of multi-agent coordination and decision-making, able to defeat the Dota 2 World Champions (OpenAI et al., 2019).

Despite these breakthroughs, most agents still rely on convolutional or recurrent architectures that can struggle with long-horizon credit assignment and offer limited transparency into their decision processes. Transformers alleviate these constraints through global self-attention, but systematic evaluations of their impact in combinatorial games remain sparse (Glanois et al., 2022). In particular, surveys on transformer-based RL have only recently begun to map out the design choices and challenges of applying attention-based sequence models to decision-making tasks, but systematic evaluations in game-playing domains are scarce (Li et al., 2023).

Building on this lineage, we experiment with various architectures (including transformers) to build an agent to play Gomoku. With transformers, by attending to complete move histories, encoding board relations, and unifying offline match datasets with self-play, our approach aims to deliver richer strategic insight, clearer decision rationales, and efficient learning dynamics in the classic "five-in-a-row" setting.

3 Method

3.1 Environment

We built our game environment on the environment in Song (2025), modifying it to run on a **6 × 6** board with a win length of **4** to keep search and training tractable on limited compute and training time for this project. Each state is encoded as a 4-plane tensor: current-player stones, opponent stones, last move location, and player-to-move indicator. For architectures that require history, we maintain a *history stack* holding the past K board tensors (default $K{=}4$).

3.2 Model Architectures

Though we tried a diverse array of different architectures, we exhibit four notable architectures in this paper. We innovate three architectures and using the existing baseline CNN from Song (2025). Our architectures each outputting a policy prior $\pi_\theta(s) \in \mathbb{R}^{36}$, each corresponding to placing a stone in one of the 36 squares in the 6×6 board, and a scalar value $v_\theta(s) \in [-1, 1]$ corresponding to the value of the current state.

The model architectures are displayed below. Each consists of a backbone followed by a policy head outputting into $\mathbb{R}^{36}$ and a value head outputting into $[-1, 1]$.

1. **Vanilla CNN:** The backbone has three convolutional layers with 3×3 kernels and 32, 64, and 128 feature channels, each followed by a ReLU.

   Policy head: a 1×1 convolution reduces channels to 4, ReLU, flatten, and a fully-connected map $\mathbb{R}^{4\times6\times6} \to \mathbb{R}^{36}$, followed by `log-softmax`.

   Value head: a 1×1 convolution to 2 channels, ReLU, flatten, FC $2 \times 6 \times 6 \to 64$, ReLU, FC $64 \to 1$, and a $\tanh$ squashing to $[-1, 1]$.

2. **ResNet:** Inspired by AlphaGo Zero's tower, the network begins with a 3×3 stem convolution that maps 4 input planes to 256 channels, followed by batch-norm and ReLU. The stem feeds a **20-block residual tower**; each block comprises BN–ReLU–3×3 conv–BN–ReLU–3×3 conv with an identity skip connection.

   Policy head: a 1×1 conv to 2 channels, BN, ReLU, flatten, FC $2 \times 6 \times 6 \to 36$, `log-softmax`.

   Value head: a 1×1 conv to 1 channel, BN, ReLU, flatten, FC $6 \times 6 \to 256$, ReLU, FC $256 \to 1$, $\tanh$.

3. **HybridNet:** same 3-layer CNN trunk as baseline, then a single **8-head global self-attention** over the 6×6 flattened patches. Fixed 2-D sinusoidal positional embeddings ($E = 128$) are added pre-layernorm; a post-layernorm residual connection stabilises training. Policy/value heads mirror the baseline. Furthermore, due to the permutation-invariance of self-attention, we added a fixed cosine position encoding to every board state before passing it through the network.

4. **TemporalNet:** There is a tunable parameter K which we typically set to be $K = 4$. Each of the K + 1 latest boards is passed through a shared 4-layer CNN trunk ($4 \to 32 \to 64 \to 128 \to 128, 3 \times 3$ + BN + ReLU, global avg-pool) to get 128-d vectors.

3.3 Monte Carlo Tree Search

During play, we run **500** simulations per move. Each node stores visit count N, mean action value Q, and prior P from the network. Dirichlet noise (α=0.3, weight = 0.25) at the root encourages exploration as in AlphaZero.

Each simulation consists of four phases—*selection*, *expansion*, *evaluation*, and *backpropagation*—as follows:

Selection Starting from the root node s_0, recursively descend the tree. At each internal node s_t, pick the action

$$a_t \;=\; \arg\max_a \Big[\, Q(s_t, a) \;+\; c_{\text{puct}}\, P(s_t, a)\, \frac{\sqrt{\sum_b N(s_t, b)}}{1 + N(s_t, a)} \Big]$$

and move to the corresponding child node s_{t+1}. Repeat until reaching a leaf node s_L (either unexpanded or terminal).

Expansion If s_L is non-terminal and has never been expanded, query the policy–value network

$$\big(P(s_L, \cdot),\, v(s_L)\big) \;=\; f_\theta(s_L),$$

which returns a set of priors $\{P(s_L, a)\}$ over legal actions and a value estimate $v(s_L) \in [-1, 1]$. Create one child for each legal action a, initializing

$$N(s_L, a) = 0, \quad W(s_L, a) = 0, \quad Q(s_L, a) = 0, \quad P(s_L, a).$$

If s_L is terminal, skip expansion and set $v(s_L)$ to $+1/-1/0$ for win/loss/draw.

Evaluation If a network evaluation was performed at the new leaf, use the returned scalar $v(s_L)$ as the value for that position. Otherwise, the terminal-state outcome is used directly.

Backpropagation Starting from s_L and moving back to the root, for each visited edge (s_t, a_t) update:

$$N(s_t, a_t) \;\leftarrow\; N(s_t, a_t){+}1, \quad W(s_t, a_t) \;\leftarrow\; W(s_t, a_t) + (-1)^t\, v(s_L), \quad Q(s_t, a_t) \;=\; \tfrac{W(s_t, a_t)}{N(s_t, a_t)}.$$

The factor $(-1)^t$ flips the sign at alternating plies so that each value is always from the perspective of the acting player at that node.

Move Selection After all 500 simulations, let the improved policy $\pi(a) \propto N(s_0, a)$ be the normalized visit counts at the root. In training, π serves as the target policy for the network; in evaluation, the chosen move is

$$a^* \;=\; \arg\max_a N(s_0, a).$$

3.4 Training Loop

Agents learn solely from self-play:

1. Generate N ($N = 400$ for 6×6, and $N = 800$ for 7×7) *self-play games* using the current network + MCTS; store (s_t, π_t, z) into a data buffer, where π_t is the search-improved policy and $z \in \{-1, 0, 1\}$ is the game outcome from the current player's view.
2. Once the data buffer contains at least 512 game states, randomly select a batch of 512 game states from the data buffer after every game. Update network parameters by minimizing

   $$\mathcal{L} = (z - v_\theta)^2 - \pi_t^\top \log \pi_\theta + 10^{-4} \|\theta\|_2^2$$

   on the batch of game states with Adam (η=10^{-3}, batch = 256).

3. Replace the "best" network when its win rate in 1000 games against a pure MCTS agent reaches a higher ratio. Once the agent successfully defeats the pure MCTS agent in all of its games, we augment the pure MCTS agent by increasing its simulation number per move by 1000.

Training proceeds for **50 k** self-play games (≈**5 M** positions), or until the evaluation plateaued.

3.5 Reward Function

Throughout the training process, we experimented with various reward functions. For example, one iteration of the reward function included an offensive reward:

$$\text{reward} = w2 \cdot \Delta_2 + w3 \cdot \Delta_3 + w4 \cdot \Delta_4$$

where Δ_i denotes the number of i pieces in a row. The idea here was that in order to place 4 pieces in a row, we must first place 2 in a row, and then 3, etc. If we play the games ourselves as humans, we follow a similar framework of trying to get smaller pieces in line first. Our specific weights were $w2 = 0.1, w3 = 0.5, w4 = 2.0$.

We also incorporated a defensive penalty. That is,

$$\text{reward} = w2 \cdot \Delta_2 + w3 \cdot \Delta_3 + w4 \cdot \Delta_4 - w_{opp} \cdot \text{opponent reward}$$

We set $w_{opp} = 0.8$. This penalizes the agent if its opponent is able to places pieces in a row and the agent fails to block these connections.

On a small board with a simple game (5×5 with a win length of 3) this gave relatively good results. We saw that the model successfully avoided rookie mistakes such as seeing 3 in a row and not taking the win or having its opponent connect 3 in a row and not blocking. Additionally, this helped incentivize the model to place pieces closer together which tends to be necessary in winning a game of Gomoku.

However, when we scaled the model up to a larger setting of 6×6 with a win length of 3, this did not perform as desired. Instead, what we saw happening was the model began to reward hack by simply placing pieces next to each other but not considering the final goal of winning. Though we could have invested more time into tuning the weights of this reward function, the possibility of reward hacking showed us that it would be more natural to implement a simpler reward model. Thus, our final reward function was

$$\text{reward} = \begin{cases} 1 & \text{win} \\ 0 & \text{tie} \\ -1 & \text{loss} \end{cases}$$

This simpler framework avoided unnecessary noise in our reward function. This showed us that though a more robust reward function can help "hold the hand" of our models in simpler settings, this may become unnecessarily noisy as the environments become more complex.

3.6 Five in a Row with `SigmaZero`

Of course, our original goal was to train for a 15×15 gameboard with a winning condition of 5 in a row. However, increasing the board by this size would exponentially increase the training time, since every MCTS root-to-leaf path would become longer, on the order of $\mathcal{O}(n^2)$, where n is the board width. Furthermore, due to the exponentially increased number of gamestates (on the order of $\mathcal{O}(2^{n^2})$), we'd need to run more MCTS simulations per move in order to provide a sufficiently noiseless training signal to the model. With our limited compute and training time, we made the executive decision to pursue a more achievable goal.

Due to our limited resources and time, we therefore decided to train only our most advanced architecture, the HybridNet, with self-play on a 7×7 board with a winning condition of 5 in a row. The architecture in the 7×7 case was almost exactly the same as for the 6×6, except that we increased the height and width of the board by 1, so the fully-connected linear layers in the policy and value heads were expanded.

Due to the exceptional nature of this model (and the fact that it plays on a 7×7 board with slightly different rules), it was incompatible to evaluate against our other trained models. However, we dedicate a section in the results to evaluating it with human play. We'll refer to this advanced 7×7, 5-in-a-row model as `SigmaZero`.

4 Experimental Setup and Evaluation

Our experimental task is self-play reinforcement learning in Gomoku. For tractability, we reduced the board size to 6×6 with a win condition of 4-in-a-row, and used a 7×7 board with a win condition of 5-in-a-row for advanced evaluation with `SigmaZero`. Each game state is encoded as a 4-plane tensor (current player's stones, opponent's stones, last move indicator, and player-to-move). Agents are trained via MCTS-guided self-play, using search-improved policies as training targets.

We benchmark each architecture on:

- **Baseline comparison** versus (i) a random policy and (ii) the `gym-gomoku` naive heuristic.
- **Head-to-head win rate** in a round-robin of 100 games.
- **Qualitative analysis**: manual review of games to categorize tactical motifs captured or missed (fork defence, diagonal setups, long traps).

All 6×6 experiments were run on a single NVIDIA A100 GPU (40 GB), with wall-clock training time capped at 48 hours per model.

Due to the exponentially increased number of gameboard states and the increased number of MCTS simulations per move, we ran the 7×7 HybridNet experiment on a single NVIDIA A100 for 7 days.

5 Results

5.1 Quantitative Results

When playing all of our models against both baselines (random policy and naive policy), our models consistently won. This shows that the models were able to learn some degree of strategy during the training process.

After running a round robin tournament (100 games) where the models alternated playing first/second, we found that:

Winner ↓ / Loser →	**Vanilla**	**TemporalNet**	**Hybrid**	**ResNet**	**PureMCTS**
Vanilla CNN	–	62.5%	50.0%	100.0%	76.9%
TemporalNet	37.5%	–	0.0%	83.3%	57.1%
HybridNet	50.0%	100.0%	–	100.0%	66.7%
ResNet	0.0%	16.7%	0.0%	–	41.7%
PureMCTS	23.1%	42.9%	33.3%	58.3%	–

Table 1: Win rate matrix between different agents.

Model	**Average Win Rate**
Hybrid	79.17%
Vanilla	72.35%
Temporal	44.48%
PureMCTS	39.40%
ResNet	14.60%

Table 2: Average win rate of each agent across all opponents.

Evidently, the HybridNet outperformed the other models. On average, the Vanilla did the second best, followed by the Temporal Model, then PureMCTS and lastly the ResNet.

5.2 Qualitative Results

Figure 1 shows the final state of the board across various runs between the different models. From these final board states, we can see that most models are able to demonstrate rather sophisticated understanding of the game. Especially in the games between the HybridNet vs TemporalNet and HybridNet vs Vanilla CNN, we see that both sides are playing both offensive and defensive strategies.

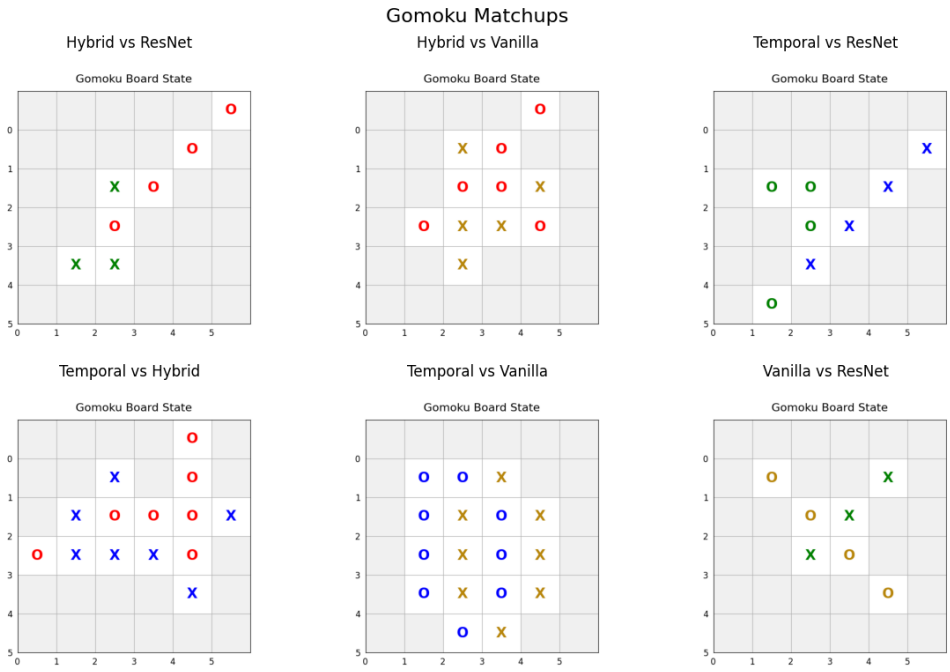

Figure 1: Final Board State for various match ups.

By reviewing the games and playing with the models ourselves, we found that the HybridNet was able to consistently put its opponent in a situation where it was guaranteed a win. For example, if we look at the Temporal vs Hybrid game, there comes a point where it had two opportunities to connect 4, rending it impossible for the opponent to block both at the same time.

On the other hand, there are a few cases where the ResNet showed "rookie mistakes". This includes having 3 in a row but not directly taking the win, or seeing it's opponent have 3 in a row but not blocking. However, these mistakes were not seen in the other models, indicating the sophistication of their policies.

5.3 `SigmaZero` V.S. Eric H. Chen

After a week of training on a 7×7 gameboard with a winning condition of 5 in a row, `SigmaZero` became a truly formidable opponent. Since `SigmaZero` had no contemporary models to play against, we matched it with the best proxy evaluation metric we could find—ourselves.

Eric played 10 games against `SigmaZero`—5 where `SigmaZero` went first, and 5 where he went first. In the 5 where Eric went first, he lost 2 and tied 3. In the 5 where Eric went second, he lost 4 and tied 1. In informal playing, none of our three team members have won a single game against `SigmaZero`, reaching at most a tie.

We observe that `SigmaZero` had, through self-play training through millions of games, learned how to both lay traps and avoid them. Its"trapping" behavior is displayed in Figure 2. Furthermore, it displays much more confidence over the board, such as knowing when it is not necessary to block

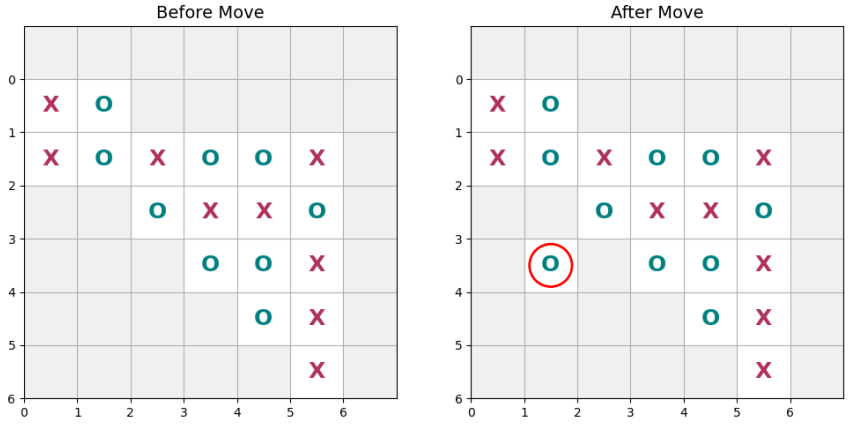

Figure 2: SigmaZero (playing with 'O') plays against Eric (playing with 'X'). SigmaZero successfully sets and springs a trap. Once it places the 'O' in slot $(2, 1)$, (row 2, column 1), it's guaranteed to win. Its opponent (Eric) can only block the diagonal three-in-a-row or prevent the vertical four-in-a-row, but not both.

a 3-in-a-row (such as when it's stuck against the edge of the board). It even displays a level of cockiness; if given the chance (i.e. if Eric plays poorly on purpose), it elects to complete multiple 4-in-a-rows before taking its win. This "BM" behavior naturally arose during its millions of self-play games, though we do not know why. We suspect that this behavior arises because we do not discount the reward, so the model is not punished for attaining its reward later. Perhaps the model was bored in its millions of self-play games and wanted to flex on its opponents.

It is notable that `SigmaZero` has attained human-level or even superhuman performance on a 7×7 board with just a week's training time.

6 Discussion

Ultimately, we see considerable success across most of the models, with the HybridNet being the best. We believe that the HybridNet is successful as its self attention layer allows it to assess relationships between any two positions on the board in a single step. This gives the model spatial awareness which is especially important in a game like Gomoku. On the other hand, the purely convolutional models such as Vanilla and ResNet must rely on multiple stacked layers to "see" across the board, making them less efficient at capturing long-range threats.

On the other hand, while the TemporalNet is able to capture past states, it is unable to fully capture the **current** state of the board as well as the VanillaCNN or HybridNet do. In a game like Gomoku, knowing what stones are on the board and *where* they are matters more than *when* they were placed. Additionally, the TemporalNet is bottle necked by the fact that it has memory, but not enough spatial resolution per frame to make full use of it.

In a game like Gomoku where winning and losing ultimately depend on where pieces are being placed, our experiments show that spatial awareness is more valuable than history. The HybridNet's ability to attend to the entire board and capture long-range spatial dependencies on the current board (e.g., multi-cell threats and forks) allows it to outperform the other models.

It is valuable to point out that connect 4 on a 6×6 board is sufficiently simple that it is a solved game. After playing against our HybridNet model, we believe that it has found the optimal strategy, making it unbeatable if the model plays first (we have been offering \$10 to anyone who can beat the model when going second, and have so far not lost any money).

Furthermore, the performance of `SigmaZero` indicates that the HybridNet architecture with a single self-attention layer is complex enough to solve even larger board sizes and win lengths. We are optimistic that, given enough time and training resources, the HybridNet will make a superhuman opponent on the full 15×15 board with 5 in a row.

6.1 Limitations

We are of course limited by our compute resources and our short time frame of completing this project. We attempted to train `SigmaZero` on a larger 10×10 board, but quickly realized that each training step would take $\sim$30 minutes to complete, due to the increased number of MCTS simulations

7 Conclusion

By combining Monte Carlo Tree Search with various model architectures for the policy-value model, we were able to train models that more or less "solved" the game of Gomoku in our given environment. We discovered that self attention gives the model the ability to understand and act on long-range spatial dependencies. Incorporating self attention proved to be a huge advantage. This concept is not limited to the game of Gomoku. Understanding when to use temporal attention versus spatial attention in different environments can greatly improve the performance of models.

Moving forward, we intend on scaling our models to play in a more complicated environment (e.g. 15×15 board with a win length of 5.) Another natural next step is combining our two results into one larger hybrid model. That is, building a hybrid model that includes both spatial self-attention and temporal attention across move histories. This could enable the agent to track recurring tactical themes while maintaining global spatial awareness.

8 Team Contributions

All three members collaborated and discussed on all aspects of the project.

- **Emily Xia** focused on creating and training the TemporalNet. She adapted the publicly-available Gomoku environment (Song, 2025) to be compatible with training the TemporalNet.
- **Eric Chen** focused on creating, tuning, and training the HybridNet architecture. He adopted and modernized the training script from (Song, 2025). He helped experiment with partial offensive/defensive reward functions. He also trained and evaluated `SigmaZero` on a 7×7 board with a winning condition of 5-in-a-row.
- **Robin Li** focused on designing and training the ResNet architecture. She helped modernized the training pipeline from (Song, 2025) to support multiple backbones and custom evaluation logic. She also collaborated on architectural innovations across models.

All three members worked together on the evaluation process, writing the paper, and making the project poster.

Changes from Proposal During the proposal, we were using a naive MCTS model. However, we were planning on improving the model by adding more complexity via CNNs, Transformers, etc. in the policy value model.

Furthermore, we did not incorporate an entire transformer architecture in our HybridNet—only the crucial self-attention layer of transformers. This is because we wanted to avoid the complex and error-prone process of tokenizing the positions of stones on a board. Instead combining a CNN with a self-attention layer incorporates the simplicity of a CNN with the power of attention.

Our proposal focused only on temporal attention, which we implemented with our TemporalNet. The HybridNet implements attention across an orthogonal dimension—the spacial dimensions of the board itself. This form of attention turned out more potent than our originally proposed temporal attention, as demonstrated in our results.

We went beyond our original expectation of training one model by experimenting with multiple different structures.

References

Claire Glanois, Paul Weng, Matthieu Zimmer, Dong Li, Tianpei Yang, Jianye Hao, and Wulong Liu. 2022. A Survey on Interpretable Reinforcement Learning. arXiv:2112.13112 [cs.LG] `https://arxiv.org/abs/2112.13112`

Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*. Springer, 282–293.

Wenzhe Li, Hao Luo, Zichuan Lin, Chongjie Zhang, Zongqing Lu, and Deheng Ye. 2023. A Survey on Transformers in Reinforcement Learning. arXiv:2301.03044 [cs.LG] `https://arxiv.org/abs/2301.03044`

OpenAI, :, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique P. d. O. Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. 2019. Dota 2 with Large Scale Deep Reinforcement Learning. arXiv:1912.06680 [cs.LG] `https://arxiv.org/abs/1912.06680`

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. 2020. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature* 588, 7839 (Dec. 2020), 604–609. `https://doi.org/10.1038/s41586-020-03051-4`

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484–489.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2017a. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815* (2017).

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017b. Mastering the game of go without human knowledge. *nature* 550, 7676 (2017), 354–359.

Junxiao Song. 2025. AlphaZero_Gomoku: An implementation of the AlphaZero algorithm for Gomoku. `https://github.com/junxiaosong/AlphaZero_Gomoku` GitHub repository, last accessed June 7, 2025.

Gerald Tesauro et al. 1995. Temporal difference learning and TD-Gammon. *Commun. ACM* 38, 3 (1995), 58–68.

A Implementation Details

Our models were trained using the framework established by Song (2025). We decided to leverage existing framework as to avoid reinventing the wheel and instead focus our efforts on innovating and trying new ideas.