Adversarial Policies Beat Superhuman Go AIs

Anonymous Authors[1]

Abstract

We attack the state-of-the-art Go-playing AI system KataGo by training adversarial policies that play against frozen KataGo victims. Our attack achieves a >99% win rate when KataGo uses no tree search, and a >97% win rate when KataGo uses enough search to be superhuman. We train our adversaries with a modified KataGo implementation, using less than 6% of the compute used to train the original KataGo. Notably, our adversaries do not win by learning to play Go better than KataGo—in fact, our adversaries are easily beaten by human amateurs. Instead, our adversaries win by tricking KataGo into making serious blunders. Our attack transfers zero-shot to other superhuman Go-playing AIs, and is interpretable to the extent that human experts can successfully implement it, without algorithmic assistance, to consistently beat superhuman AIs. Our results demonstrate that even superhuman AI systems may harbor surprising failure modes. Example games are available at `go-attack-icml.netlify.app`.

1. Introduction

Reinforcement learning from self-play has achieved superhuman performance in a range of games including Go (Silver et al., 2016), chess and shogi (Silver et al., 2016), and Dota (OpenAI et al., 2019). Moreover, idealized versions of self-play provably converge to Nash equilibria (Brown, 1951; Heinrich et al., 2015). Although realistic versions of self-play need not converge, their strong empirical performance seems to suggest this is rarely an issue in practice.

Nonetheless, prior work has found that seemingly highly capable continuous control policies trained via self-play can be exploited by *adversarial policies* (Gleave et al., 2020; Wu et al., 2021). This suggests that self-play may not be as robust as previously thought. However, although the victim agents are state-of-the-art for continuous control, they are still well below human performance. This raises the question: are adversarial policies a vulnerability of self-play policies in general, or simply an artifact of insufficiently capable policies?

To answer this, we study a domain where self-play has achieved very strong performance: Go. Specifically, we train adversarial policies end-to-end to attack KataGo (Wu, 2019), the strongest publicly available Go-playing AI system. Using less than 6% of the compute used to train KataGo, we obtain adversarial policies that win >99% of the time against KataGo with no search, and >97% of the time against KataGo with enough search to be superhuman.

Critically, our adversaries do not win by learning a generally capable Go policy. Instead, the adversaries trick KataGo into making serious blunders that result in KataGo losing the game (Figure 1). Despite being able to beat KataGo, our adversarial policies lose against even amateur Go players (see Appendix I.1). This is a clear example of non-transitivity, illustrated in Figure 2.

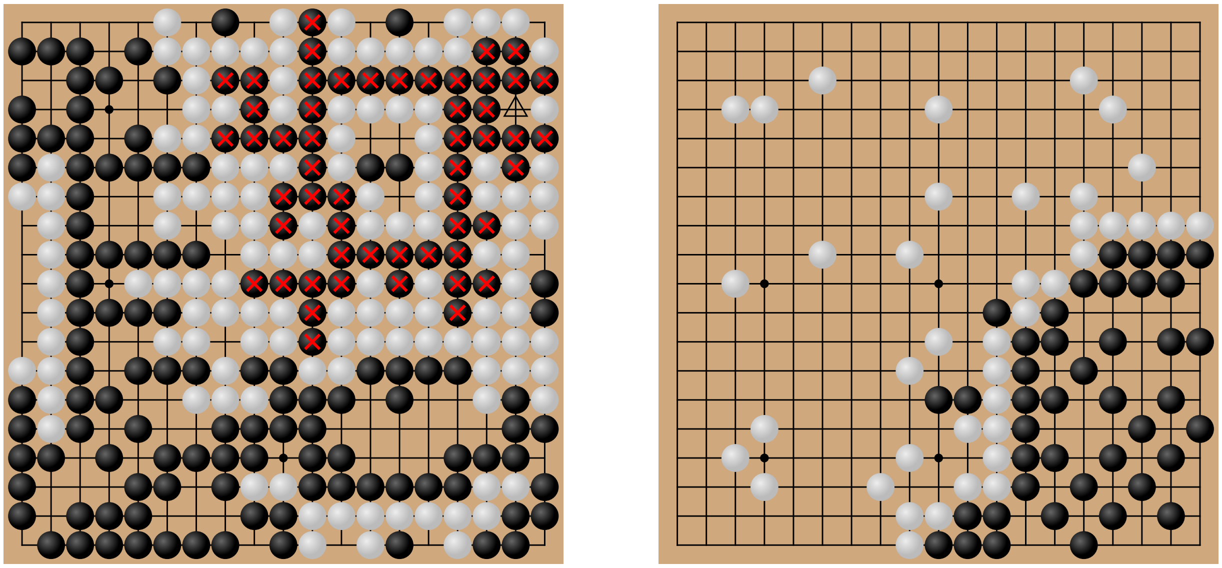
Our adversaries have no special powers: they can only place stones or pass, like a regular player. We do however give our adversaries access to the victim network they are attacking. In particular, we train our adversaries using an AlphaZero-style training process (Silver et al., 2018), similar to that of KataGo. The key differences are that we collect games with the adversary playing against the victim, and that we use the victim network to select victim moves during the *adversary's* tree search.

KataGo can play at a strongly superhuman level, winning (Wu, 2019, Section 5.1) against ELF OpenGo (Tian et al., 2019) and Leela Zero (Pascutto, 2019) that are themselves superhuman. In Appendix E, we estimate that KataGo without search plays at the level of a top-100 European player, and that KataGo is superhuman at or above 128 visits of search per move. Our attack scales far beyond this level, achieving a 76.7% win rate against KataGo playing with 10^7 visits of search per move.

Our paper makes three contributions. First, we propose a novel attack method, hybridizing the attack of Gleave et al. (2020) with AlphaZero-style training (Silver et al., 2018).

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.



(a) Our *cyclic-adversary* wins as white by capturing a cyclic group (X) that the victim (`Latest`$_{\texttt{def}}$, 10^7 visits) leaves vulnerable. Explore the game.

(b) Our *pass-adversary* wins as black by tricking the victim (`Latest`, no search) into passing prematurely, ending the game. Explore the game.

Figure 1: Games between the strongest KataGo network (which we refer to as `Latest`) and two different types of adversaries we trained. (a) Our *cyclic-adversary* beats KataGo even when KataGo plays with far more search than is needed to be superhuman. The adversary lures the victim into letting a large group of cyclic victim stones (X) get captured by the adversary's next move (Δ). Appendix I.2 has a detailed description of this adversary's behavior. (b) Our *pass-adversary* beats no-search KataGo by tricking it into passing. The adversary then passes in turn, ending the game with the adversary winning under the Tromp-Taylor ruleset for computer Go (Tromp, 2014) that KataGo was trained and configured to use (see Appendix A). The adversary gets points for its territory in the bottom-right corner (devoid of victim stones) whereas the victim does not get points for the territory in the top-left due to the presence of the adversary's stones.

Second, we demonstrate the existence of two distinct adversarial policies against the state-of-the-art Go AI system, KataGo. Finally, we provide a detailed empirical investigation into these adversarial policies, including showing they partially transfer to other Go AIs and learn interpretable strategies that can be replicated by experts under standard human playing conditions. Our open-source implementation is available at —anonymized—.

2. Related work

Our work is inspired by the presence of adversarial examples in a wide variety of models (Szegedy et al., 2014). Notably, many image classifiers reach or surpass human performance (Ho-Phuoc, 2018; Russakovsky et al., 2015; Shankar et al., 2020; Pham et al., 2021). Yet even these state-of-the-art image classifiers are vulnerable to adversarial examples (Carlini et al., 2019; Ren et al., 2020). This raises the question: could highly capable deep RL policies be similarly vulnerable?

One might hope that the adversarial nature of self-play training would naturally lead to robustness. This strategy works for image classifiers, where adversarial training is an effective if computationally expensive defense (Madry et al., 2018; Ren et al., 2020). This view is bolstered by idealized versions of self-play provably converging to a Nash equilibrium, which is unexploitable (Brown, 1951; Heinrich et al., 2015). However, our work finds that in practice even state-of-the-art and superhuman-level deep RL policies are still highly vulnerable to exploitation.

It is known that self-play may not converge in non-transitive games (Balduzzi et al., 2019) like rock-paper-scissors, where agent A beats B and B beats C yet C beats A. However, Czarnecki et al. (2020) has argued that real-world games like Go grow increasingly transitive as skill increases. This would imply that while self-play may struggle with non-transitivity early on during training, comparisons involving highly capable policies such as KataGo should be mostly transitive. By contrast, we find significant non-transitivity: our adversaries exploit KataGo agents that beat human professionals, yet lose to most amateur Go players (Appendix I.1).

Most prior work attacking deep RL has focused on perturbing observations (Huang et al., 2017; Ilahi et al., 2022). Concurrent work by Lan et al. (2022) shows that KataGo with $\leq$50 visits can be induced to play poorly by adding

Figure 2: A human amateur beats our adversarial policy (Appendix I.1) that beats KataGo. This non-transitivity shows the adversary is not a generally capable policy, and is just exploiting KataGo.

two adversarially chosen moves to a board, even though these moves do not substantially change the win rate estimated by KataGo with 800 visits. However, the perturbed input is highly off-distribution, as the move history seen by the KataGo network implies that it *chose* to play a seemingly poor move on the previous turn. Moreover, an attacker that can force the opponent to play a specific move has easier ways to win: it could simply make the opponent resign, or play a maximally bad move. We instead follow the threat model introduced by Gleave et al. (2020) of an adversarial *agent* acting in a shared environment.

Prior work on such *adversarial policies* has focused on attacking subhuman policies in simulated robotics environments (Gleave et al., 2020; Wu et al., 2021). In these environments, the adversary can often win just by causing the victim to make small changes to its actions. By contrast, our work focuses on exploiting superhuman-level Go policies that have a discrete action space. Despite the more challenging setting, we find these policies are not only vulnerable to attack, but also fail in surprising ways that are quite different from human-like mistakes.

Adversarial policies give a lower bound on the *exploitability* of an agent: how much expected utility a best-response policy achieves above the minimax value of the game. Exactly computing a policy's exploitability is feasible in some low-dimensional games (Johanson et al., 2011), but not in larger games such as Go with approximately 10^{172} possible states (Allis, 1994, Section 6.3.12). Prior work has lower bounded the exploitability in some poker variants using search (Lisý & Bowling, 2017), but the method relies on domain-specific heuristics that are not applicable to Go.

In concurrent work Timbers et al. (2022) developed the *approximate best response* (ABR) method to estimating exploitability. Whereas we exploit the open-source KataGo agent, they exploit a proprietary replica of AlphaZero from Schmid et al. (2021). They obtain a 90% win rate against no-search AlphaZero and 65% with 800 visits (Timbers et al., 2022, Figure 3). In Appendix E.3 we estimate that their AlphaZero victim with 800 visits plays at least at the level of a top-200 professional, and may be superhuman. That we were both able to exploit unrelated codebases confirms the vulnerability is in AlphaZero-style training as a whole, not merely an implementation bug.

Both Timbers et al. and our attacks use an AlphaZero-style training procedure adapted to use the *opponent's* policy during search, with a curriculum over the victim's search budget. However, our curriculum also varies the victim checkpoint. Furthermore, we trained our *cyclic-adversary* by first manually patching KataGo to protect against our initial *pass-adversary*, then repeating the attack.

Our main contribution lies in our experimental results: we find our attack beats victims playing with up to 10^7 visits whereas Timbers et al. only test up to 800 visits. Moreover, we find our *cyclic-adversary* is interpretable enough to be reliably replicated by a human expert. Additionally, we investigate modeling the victim's search process inside our adversary, show the victim's predicted win rate is miscalibrated, qualitatively analyse games played by the adversary, and investigate transfer of adversarial policies to other Go AIs.

3. Background

3.1. Threat model

Following Gleave et al. (2020), we consider the setting of a two-player zero-sum Markov game (Shapley, 1953). Our threat model assumes the attacker plays as one of the agents, which we will call the *adversary*, and seeks to win via standard play against some *victim* agent.

The key capability we grant to the attacker is gray-box access to the victim agent. That is, the attacker can evaluate the victim's neural network on arbitrary inputs. However, the attacker does not have direct access to the network weights. We furthermore assume the victim agent follows a fixed policy, corresponding to the common case of a pre-trained model deployed with static weights. Gray-box access to a fixed victim naturally arises whenever the attacker can run a copy of the victim agent, e.g., when attacking a commercially available or open-source Go AI system. However, we also weaken this assumption in some of our experiments, seeking to *transfer* the attack to an unseen victim agent—an extreme case of a black-box attack.

We know the victim must have some weak spots: optimal play is intractable in a game as complex as Go. However, these vulnerabilities could be quite hard to find, especially using only gray-box access. In particular, exploits that are easy to discover will tend to have already been found by self-play training, resulting in the victim being immunized against them.

Consequently, our two primary success metrics are the *win rate* of the adversarial policy against the victim and the adversary's *training and inference time*. We also track the mean score difference between the adversary and victim, but this is not explicitly optimized for by the attack. Tracking training and inference time rules out the degenerate "attack" of simply training KataGo for longer than the victim, or letting it search deeper at inference.

In principle, it is possible that a more sample-efficient training regime could produce a stronger agent than KataGo in a fraction of the training time. While this might be an important advance in computer Go, we would hesitate to classify it as an attack. Rather, we are looking for the adversarial policy to demonstrate *non-transitivity*, as this suggests the adversary is winning by exploiting a specific weakness in the opponent. That is, as depicted in Figure 2, the adversary beats the victim, the victim beats some baseline opponent, and that baseline opponent can in turn beat the adversary.

3.2. KataGo

We chose to attack KataGo as it is the strongest publicly available Go AI system at the time of writing. KataGo won against ELF OpenGo (Tian et al., 2019) and Leela Zero (Pascutto, 2019) after training for only 513 V100 GPU days (Wu, 2019, section 5.1). ELF OpenGo is itself superhuman, having won all 20 games played against four top-30 professional players. The latest networks of KataGo are even stronger than the original, having been trained for over 10,000 V100-equivalent GPU days. Indeed, even the policy network with *no search* is competitive with top professionals (see Appendix E.1).

KataGo learns via self-play, using an AlphaZero-style training procedure (Silver et al., 2018). The agent contains a neural network with a *policy head*, outputting a probability distribution over the next move, and a *value head*, estimating the win rate from the current state. It then conducts Monte-Carlo Tree Search (MCTS) using these heads to select self-play moves, described in Appendix B.1. KataGo trains the policy head to predict the outcome of this tree search, a policy improvement operator, and trains the value head to predict whether the agent wins the self-play game.

In contrast to AlphaZero, KataGo includes several additional heads predicting auxiliary targets such as the opponent's move on the following turn and which player "owns" a square on the board. These heads' outputs are not used for actual game play, serving only to speed up training via the addition of auxiliary losses. KataGo also introduces architectural improvements such as global pooling, and training process improvements such as playout cap randomization.

These modifications to KataGo improve its sample and compute efficiency by several orders of magnitude relative to prior work such as ELF OpenGo. For this reason, we choose to build our attack on top of KataGo, although in principle the same attack could be implemented on top of any AlphaZero-style training pipeline. We describe our extensions to KataGo in the following section.

4. Attack methodology

Prior works, such as KataGo and AlphaZero, train on self-play games where the agent plays many games against itself. We instead train on games between our adversary and a fixed victim agent, and only train the adversary on data from the turns where it is the adversary's move, since we wish to train the adversary to exploit the victim, not mimic it. We dub this procedure *victim-play*.

In regular self-play, the agent models its opponent's moves by sampling from its own policy network. This makes sense in self-play, as the policy *is* playing itself. But in victim-play, it would be a mistake to model the victim as playing from the *adversary's* policy network. We introduce two distinct families of *Adversarial MCTS* (A-MCTS) to address this problem. See Appendix C for the hyperparameter settings we used in experiments.

Adversarial MCTS: Sample (A-MCTS-S). In A-MCTS-S (Appendix B.2), we modify the adversary's search procedure to sample from the victim's policy head at *victim-nodes* in the Monte Carlo tree where it is the victim's move, and from the adversary's policy head at *adversary-nodes* where it is the adversary's turn. We also disable some KataGo optimizations, such as adding noise to the policy network at the root. Finally, we introduce a variant A-MCTS-S++ (Appendix B.3) that averages the victim policy network over board symmetries, to match the default behavior of KataGo.

Adversarial MCTS: Recursive (A-MCTS-R). A-MCTS-S underestimates the strength of victims that use search as it models the victim as sampling directly from the policy head. To resolve this, A-MCTS-R (Appendix B.4) runs MCTS for the victim at each victim node in the A-MCTS-R tree. Unfortunately, this change increases the computational complexity of both adversary training and inference by a factor equal to the victim's search budget. We include A-MCTS-R primarily as an upper bound to test how much benefit can be gained by resolving this misspecification.

Initialization. We randomly initialize the adversary's network. Note that we cannot initialize the adversary's weights to those of the victim as our threat model does not allow white-box access. Additionally, a random initialization encourages exploration to find weaknesses in the victim, rather than simply producing a stronger Go player. However, a randomly initialized network will almost always lose against a highly capable network, leading to a challenging initial learning problem. Fortunately, the adversary's network is able to learn something useful about the game even from matches that are lost due to KataGo's incorporation of auxiliary targets for prediction.

Curriculum. We use a curriculum that trains against successively stronger versions of the victim in order to help overcome the challenging random initialization. We switch to a more challenging victim agent once the adversary's win rate exceeds a certain threshold. We modulate victim strength in two ways. First, we train against successively later checkpoints of the victim agent, as KataGo releases its entire training history. Second, we increase the amount of search that the victim performs during victim-play.

5. Evaluation

We evaluate our attack method against KataGo (Wu, 2019). In Section 5.1 we use A-MCTS-S to train our *pass-adversary*, achieving a 99.9% win rate against `Latest` (the strongest KataGo network) playing without search. Notably `Latest` is very strong even without search: we find in Appendix E.1 that it is comparable to a top-100 European player. The pass-adversary beats `Latest` by tricking it into passing early and losing (Figure 1b).

In Section 5.2, we then add a *pass-alive defense* to the victim to defend against the aforementioned attack. The defended victim $\texttt{Latest}_{\texttt{def}}$ provably will not lose via accidentally passing. Moreover, $\texttt{Latest}_{\texttt{def}}$ is about as strong as `Latest`, winning 456/1000 games with no tree search and 461/1000 games with 2048 visits/move for both agents.

However, repeating the attack we find a *cyclic-adversary* that achieves a 100% win rate over 1048 games against $\texttt{Latest}_{\texttt{def}}$ playing without search. The cyclic-adversary succeeds against victims playing with search as well, achieving a 95.7% win rate against $\texttt{Latest}_{\texttt{def}}$ with 4096 visits in Section 5.3.

The cyclic-adversary is qualitatively very different from the pass-adversary as it does not use the pass-trick (Figure 1a). To check that our defense did not introduce any unintended weaknesses, we evaluate our cyclic-adversary against the unmodified `Latest`. The cyclic-adversary achieves a 100% win rate over 1000 games against `Latest` without search, a 97.3% win rate against `Latest` with 4096 visits, and a 76.7% win rate against `Latest` with 10^7 visits. In Appendix E.2, we estimate that `Latest` with 4096 visits is already much stronger than the best human Go players, and `Latest` with 10^7 visits far surpasses human players.

5.1. Attacking a victim without search

Our pass-adversary playing with 600 visits achieves a 99.9% win rate against `Latest` with no search. Notably, our pass-adversary wins despite being trained with just 0.04% of `Latest`'s training budget (Appendix D). Importantly, the pass-adversary does not win by playing a stronger game of Go than `Latest`. Instead, it follows a bizarre strategy illustrated in Figure 1b that loses even against human amateurs (see Appendix I.1). The strategy tricks the KataGo policy head into passing prematurely at a move where the adversary has more points under Tromp-Taylor Rules (Appendix A).

We trained our pass-adversary using A-MCTS-S and a curriculum, as described in Section 4. Our curriculum starts from a checkpoint `cp127` around a quarter of the way through KataGo's training, and ends with the `Latest` checkpoint corresponding to the strongest KataGo network (Appendix C.1 for details).

Appendix F contains further evaluation and analysis of our pass-adversary. Although this adversary was only trained on no-search victims, it transfers to very low search victims. Using A-MCTS-R the adversary achieves an 88% win rate against `Latest` playing with 8 visits. This win rate drops to 15% when the adversary uses A-MCTS-S.

5.2. Attacking a defended victim

We design a hard-coded defense for the victim against the attack found in Section 5.1: only passing when it cannot change the outcome of the game. Concretely, we only allow the victim to pass when its only legal moves are in its own *pass-alive territory*, a concept described in the official KataGo rules (Wu, 2021b) and which extends the traditional Go notion of a pass-alive group (see Appendix B.6 for a full description of the defense). Given a victim `V`, we let $\texttt{V}_{\texttt{def}}$ denote the victim with this defense applied. The defense completely thwarts the pass-adversary from Section 5.1; the pass-adversary loses every game out of 1000 against $\texttt{Latest}_{\texttt{def}}$.

We repeat our A-MCTS-S attack against the defended victim, obtaining our cyclic-adversary. The curriculum (Appendix C.2) starts from an early checkpoint $\texttt{cp39}_{\texttt{def}}$ with no search and continues until $\texttt{Latest}_{\texttt{def}}$. The curriculum

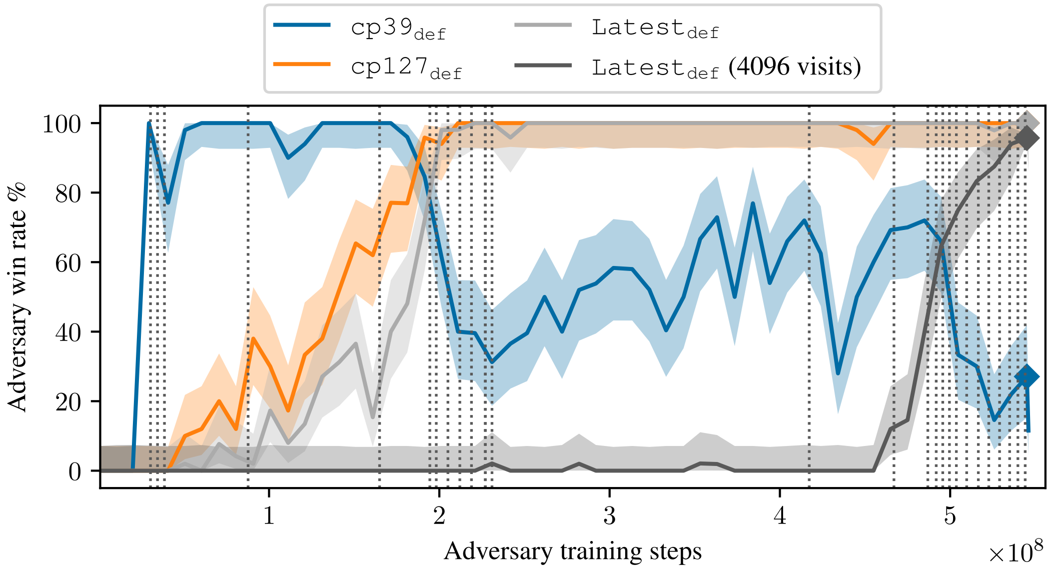


Figure 3: The win rate (y-axis) of the cyclic-adversary over time (x-axis) playing with 600 visits against four different victims. The strongest cyclic-adversary checkpoint (marked ⧫) wins 1048/1048 = 100% games against `Latest`$_{\texttt{def}}$ without search and 1007/1052 = 95.7% games against `Latest`$_{\texttt{def}}$ with 4096 visits. The shaded interval is a 95% Clopper-Pearson interval over 50 games per checkpoint. The cyclic-adversary is trained with a curriculum, starting from `cp39`$_{\texttt{def}}$ without search and ending at `Latest`$_{\texttt{def}}$ with 131,072 visits. Vertical dotted lines denote switches to stronger victim networks or to an increase in `Latest`$_{\texttt{def}}$'s search budget. See Appendix C.2 for the exact curriculum specification.

then starts increasing the number of victim visits.

In Figure 3 we evaluate the cyclic-policy against the policy networks of `cp39`$_{\texttt{def}}$, `cp127`$_{\texttt{def}}$, and `Latest`$_{\texttt{def}}$. We see that an attack that works against `Latest`$_{\texttt{def}}$ transfers well to `cp127`$_{\texttt{def}}$ but not to `cp39`$_{\texttt{def}}$, and an attack against `cp39`$_{\texttt{def}}$ early in training did not transfer well to `cp127`$_{\texttt{def}}$ or `Latest`$_{\texttt{def}}$. These results suggest that different checkpoints have unique vulnerabilities.

Our best cyclic-adversary checkpoint playing with 600 visits against `Latest`$_{\texttt{def}}$ playing with no search achieves a 100.0% win rate over 1048 games. The cyclic-adversary also still works against `Latest` with the defense disabled, achieving a 100.0% win rate over 1000 games. The cyclic-adversary is trained using roughly 5.82% of the compute used for training `Latest` (Appendix D). The cyclic-adversary still loses against human amateurs (see Appendix I.1).

5.3. Attacking a victim with search

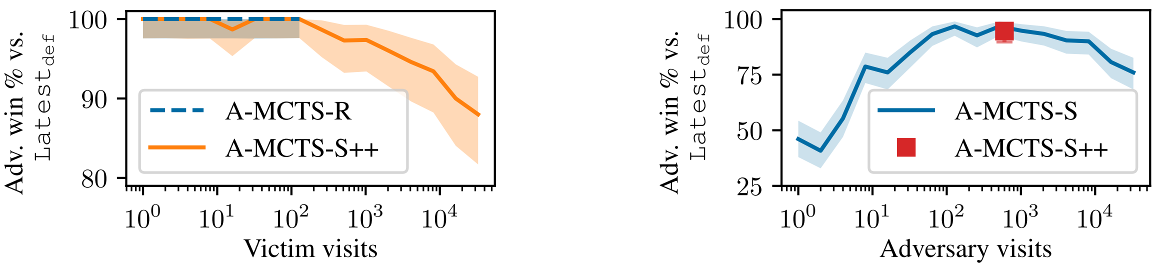
We evaluate the ability of our cyclic-adversary to exploit victims playing *with* search and find that it still achieves strong win rates by tricking its victims into making severe mistakes a human would avoid (see Appendix I.2).

The cyclic-adversary achieves a win rate of to 95.7% against `Latest`$_{\texttt{def}}$ with 4096 visits. The adversary also achieves a 97.3% win rate against an undefended `Latest` with 4096 visits, verifying that our adversary is not exploiting anomalous behavior introduced by the defense.

In Figure 4, we study the effects of varying both adversary and victim search. We find that for a fixed adversary search budget, victims with more search are harder to exploit. For a fixed victim search budget, the adversary does best at 128–600 visits, and A-MCTS-S++ performs no better than the computationally cheaper A-MCTS-S. Intriguingly, increasing adversary visits beyond 600 does not help and may even hurt performance, suggesting the adversary's strategy does not benefit from greater look-ahead.

In Figure 4a, we also plot the performance of A-MCTS-R (which correctly models the victim). In experiments with an earlier checkpoint of the cyclic-adversary, we saw A-MCTS-R outperform A-MCTS-S (which incorrectly models the victim as having no search) (see Figure 22 in the Appendix). With our current version of the cyclic-adversary, A-MCTS-S does so well that A-MCTS-R cannot provide any improvement up to 128 victim visits. The downside of A-MCTS-R is that it drastically increases the amount of compute, to the point that it is impractical to evaluate A-MCTS-R at higher visit counts.

Finally, we also tested our cyclic-adversary against `Latest` with substantially higher victim visits. The adversary (using A-MCTS-S with 600 visits/move) achieved an 82% win rate (over 50 games) against `Latest` with 10^6 visits/move, and a 76.7% win rate (over 30 games) against



(a) Win rate of cyclic-adversary (y-axis) playing with 600 visits/move vs. Latest$_{\text{def}}$ with varying amounts of search (x-axis). Victims with more search are harder to exploit.

(b) Win rate of cyclic-adversary (y-axis) playing with varying visits (x-axis). The victim Latest$_{\text{def}}$ plays with a fixed 4096 visits/move. Win rates are best with 128–600 adversary visits.

Figure 4: We evaluate the cyclic-adversary's win rate against Latest$_{\text{def}}$ with varying amounts of search (**left**: victim, **right**: adversary). Shaded regions and error bars denote 95% Clopper-Pearson confidence intervals over ~150 games.

Latest with 10^7 visits/move, using 10 and 1024 search threads respectively (see Appendix C). This demonstrates that search is not a practical defense against the attack: 10^7 visits is already prohibitive in many applications, taking over one hour per move to evaluate even on high-end consumer hardware (Yao, 2022). Indeed, Tian et al. (2019) used two orders of magnitude less search than this even in tournament games against human professionals.

That said, the adversary win rate does decrease with more victim search. This is even more apparent against a weaker adversary (Figure 22), and the victim tends to judge positions more accurately with more search (Appendix H). We conclude that search is a valid tool for improving robustness, but will not produce fully robust agents on its own.

5.4. Understanding the cyclic-adversary

Qualitatively, the cyclic-adversary we trained in Section 5.2 wins by coaxing the victim into creating a large group of stones in a circular pattern, thereby exploiting a weakness in KataGo's network which allows the adversary to capture the group. This causes the score to shift decisively and unrecoverably in the adversary's favor.

To better understand the attack, we examined the win rate predictions produced by both the adversary's and the victim's value networks at each turn of a game. Typically the victim predicts that it will win with over 99% confidence for most of the game, then suddenly realizes it will lose with high probability, often just *one move* before its circular group is captured. This trend is depicted in Figure 5a: the victim's prediction loss is elevated throughout the majority of the game, only dipping close to the self-play baseline around 40-50 moves from the end of the game. In some games, we observe that the victim's win rate prediction oscillates wildly before finally converging on certainty that it will lose (Figure 5b). This is in stark contrast to the adversary's own predictions, which change much more slowly and are less confident.

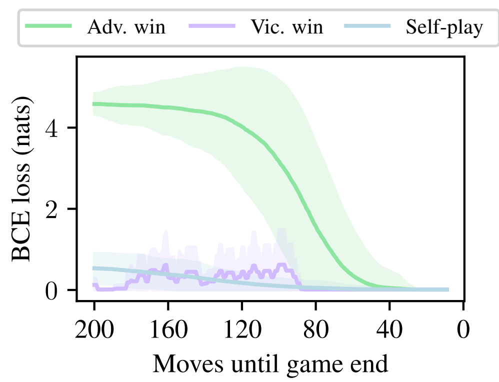
We test several hard-coded baseline attacks in Appendix F.5. We find that none of the attacks work well against Latest$_{\text{def}}$, although the *Edge* baseline playing as white wins almost half of the time against the unhardened Latest. This provides further evidence that Latest$_{\text{def}}$ is more robust than Latest, and that the cyclic-adversary has learned a relatively sophisticated exploit.

5.5. Transfer

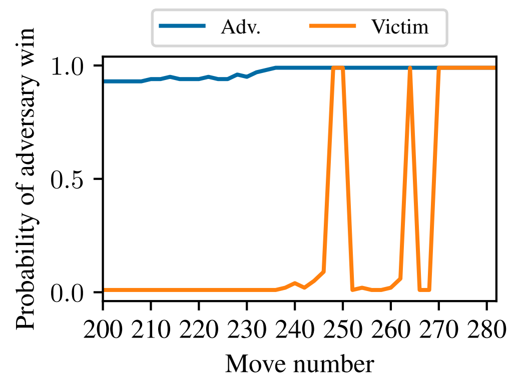
In Appendix G.1 we evaluate our cyclic-adversary (trained only on KataGo) in zero-shot transfer against two different superhuman Go agents, Leela Zero and ELF OpenGo. This setting is especially challenging because A-MCTS models the victim as being KataGo and will be continually surprised by the moves taken by the Leela or ELF opponent. Nonetheless, the adversary wins 6.1% of games against Leela and 3.5% of games against ELF.

In Appendix G.2 one of our authors, a Go expert, was able to learn from our adversary's game records to implement this attack without any algorithmic assistance. Playing in standard human conditions on the online Go server KGS they obtained a greater than 90% win rate against a top ranked bot that is unaffiliated with the authors. The author even won giving the bot 9 handicap stones, an enormous advantage: a human professional with this handicap would have a virtually 100% win rate against any opponent, whether human or AI. They also beat KataGo and Leela Zero playing with 100k visits each, which is normally far beyond human capabilities.

These results confirm that the cyclic vulnerability is present in a range of bots under a variety of configurations. They also further highlight the significance and interpretability of the exploit our algorithmic adversary finds. The adversary is not finding, for instance, just a special sequence of moves, but a strategy that a human can learn and act on. In addition, in both algorithmic and human transfer, the attacker does not have access to the victim model's weights,



(a) Binary cross-entropy loss of `Latest`'s predicted win rates at 4,096 visits playing against the cyclic-adversary. The green and purple curves are averaged over games won by the adversary and victim respectively. The blue curve is averaged over self-play games and serves as a baseline. Shaded regions denote ± 1 SD.



(b) Probability of adversary victory according to the adversary and victim value networks, for a portion of a randomly selected game. Note the sudden changes in win rate prediction between moves 248 and 272 during a ko fight. Explore the game.

Figure 5: Analysis of predicted win rate when `Latest` plays the cyclic-adversary.

policy network output, or even a large number of games to learn from. This increases the threat level and suggests, for example, that one could learn an attack on an open-source system and then transfer it to a closed-source model.

6. Limitations and future work

This paper has demonstrated that even superhuman agents can be vulnerable to adversarial policies. However, our results do not establish how common such vulnerabilities are: it is possible Go-playing AI systems are unusually vulnerable. A promising direction for future work is to evaluate our attack against strong AI systems in other games.

It is natural to ask how we can *defend* against adversarial policies. Fortunately, there are a number of promising multi-agent RL techniques. A key direction for future work is to evaluate policies trained with these approaches to determine whether they are also exploitable. One such technique is counterfactual regret minimization (Zinkevich et al., 2007, CFR), which can beat professional human poker players (Brown & Sandholm, 2018). CFR has difficulty scaling to high-dimensional state spaces, but regularization methods (Perolat et al., 2021) can natively scale to games such as Stratego with a game tree 10^{175} times larger than Go (Perolat et al., 2022). Alternatively, methods using populations of agents such as policy-space response oracles (Lanctot et al., 2017), AlphaStar's Nash league (Vinyals et al., 2019) or population-based training (Czempin & Gleave, 2022) may be more robust than self-play, at the cost of greater computational requirements.

Finally, we found it harder to exploit agents that use search, with our attacks achieving a lower win rate and requiring more computational resources. An interesting direction for future work is to look for more effective and compute-efficient methods for attacking agents that use large amounts of search, such as learning a computationally efficient model of the victim (Appendix B.5).

7. Conclusion

We trained adversarial policies that are able to exploit a superhuman Go AI. Notably, the adversaries do not win by playing a strong game of Go—in fact, they can be easily beaten by a human amateur. Instead, the adversaries win by exploiting particular blind spots in the victim agent. This result suggests that even highly capable agents can harbor serious vulnerabilities.

KataGo was published in 2019 and has since been used by many Go enthusiasts and professional players as a playing partner and analysis engine (Wu, 2019). Despite this public scrutiny, to the best of our knowledge the vulnerabilities discussed in this paper were never previously exploited. This suggests that learning-based attacks like the ones developed in this paper may be an important tool for uncovering hard-to-find vulnerabilities in AI systems.

Our results underscore that improvements in capabilities do not always translate into adequate robustness. Failures in Go AI systems are entertaining, but similar failures in safety-critical systems like automated financial trading or autonomous vehicles could have dire consequences. We believe that the ML research community should invest in improving robust training and adversarial defense techniques in order to produce models with the high levels of reliability needed for safety-critical systems.

Author contributions

Removed for double blind submission.

ACKNOWLEDGMENTS

Removed for double blind submission.

References

Allis, L. V. *Searching for Solutions in Games and Artificial Intelligence*. PhD thesis, Maastricht University, 1994.

Balduzzi, D., Garnelo, M., Bachrach, Y., Czarnecki, W., Pérolat, J., Jaderberg, M., and Graepel, T. Open-ended learning in symmetric zero-sum games. In *ICML*, 2019.

Baudiš, P. and Gailly, J.-l. PACHI: State of the art open source Go program. In *Advances in Computer Games*, 2012.

Baudiš, P. and Gailly, J.-l. PACHI readme, 2020. URL https://github.com/pasky/pachi/blob/a7c60ec10e1a071a8ac7fc51f7ccd62f006fff21/README.md.

Benson, D. B. Life in the game of Go. *Information Sciences*, 10(1):17–29, 1976.

Brown, G. W. Iterative solution of games by fictitious play. In *Activity Analysis of Production and Allocation*, volume 13, pp. 374, 1951.

Brown, N. and Sandholm, T. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

Bui, T. V., Mai, T., and Nguyen, T. H. Imitating opponent to win: Adversarial policy imitation learning in two-player competitive games. arXiv:2210.16915v1 [cs.LG], 2022.

Carlini, N., Athalye, A., Papernot, N., Brendel, W., Rauber, J., Tsipras, D., Goodfellow, I., Madry, A., and Kurakin, A. On evaluating adversarial robustness. arXiv:1902.06705v2 [cs.LG], 2019.

Coulom, R. Go ratings, 2022. URL https://archive.ph/H0VDl.

Czarnecki, W. M., Gidel, G., Tracey, B., Tuyls, K., Omidshafiei, S., Balduzzi, D., and Jaderberg, M. Real world games look like spinning tops. In *NeurIPS*, 2020.

Czempin, P. and Gleave, A. Reducing exploitability with population based training. In *ICML Workshop on New Frontiers in Adversarial Machine Learning*, 2022.

EGD. European Go database, 2022. URL https://www.europeangodatabase.eu/EGD/.

Federation, E. G. European pros, 2022. URL https://www.eurogofed.org/pros/.

Gleave, A., Dennis, M., Wild, C., Kant, N., Levine, S., and Russell, S. Adversarial policies: Attacking deep reinforcement learning. In *ICLR*, 2020.

Haoda, F. and Wu, D. J. summarize_sgfs.py, 2022. URL https://github.com/lightvector/KataGo/blob/c957055e020fe438024ddffd7c5b51b349e86dcc/python/summarize_sgfs.py.

Heinrich, J., Lanctot, M., and Silver, D. Fictitious self-play in extensive-form games. In *ICML*, volume 37, pp. 805–813, 2015.

Ho-Phuoc, T. CIFAR10 to compare visual recognition performance between deep neural networks and humans. arXiv:1811.07270v2 [cs.CV], 2018.

Huang, S. H., Papernot, N., Goodfellow, I. J., Duan, Y., and Abbeel, P. Adversarial attacks on neural network policies. arXiv:1702.02284v1 [cs.LG], 2017.

Ilahi, I., Usama, M., Qadir, J., Janjua, M. U., Al-Fuqaha, A., Hoang, D. T., and Niyato, D. Challenges and countermeasures for adversarial attacks on deep reinforcement learning. *IEEE TAI*, 3(2):90–109, 2022.

Johanson, M., Waugh, K., Bowling, M. H., and Zinkevich, M. Accelerating best response calculation in large extensive games. In *IJCAI*, 2011.

KGS. gnugo2 rank graph, 2022a. URL https://www.gokgs.com/graphPage.jsp?user=gnugo2.

KGS. Top 100 KGS players, 2022b. URL https://archive.ph/BbAHB.

Lan, L.-C., Zhang, H., Wu, T.-R., Tsai, M.-Y., Wu, I.-C., and Hsieh, C.-J. Are AlphaZero-like agents robust to adversarial perturbations? In *NeurIPS*, 2022.

Lanctot, M., Zambaldi, V., Gruslys, A., Lazaridou, A., Tuyls, K., Perolat, J., Silver, D., and Graepel, T. A unified game-theoretic approach to multiagent reinforcement learning. In *NeurIPS*, pp. 4190–4203, 2017.

Lisý, V. and Bowling, M. Equilibrium approximation quality of current no-limit poker bots. In *AAAI Workshop on Computer Poker and Imperfect Information Games*, 2017.

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., and Vladu, A. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018.

OpenAI, Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., de Oliveira Pinto, H. P., Raiman, J., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. Dota 2 with large scale deep reinforcement learning. arXiv:1912.06680v1 [cs.LG], 2019.

Pascutto, J. Leela Zero, 2019. URL https://zero.sjeng.org/.

Perolat, J., Munos, R., Lespiau, J.-B., Omidshafiei, S., Rowland, M., Ortega, P., Burch, N., Anthony, T., Balduzzi, D., De Vylder, B., Piliouras, G., Lanctot, M., and Tuyls, K. From Poincaré recurrence to convergence in imperfect information games: Finding equilibrium via regularization. In *ICML*, volume 139, pp. 8525–8535, 2021.

Perolat, J., de Vylder, B., Hennes, D., Tarassov, E., Strub, F., de Boer, V., Muller, P., Connor, J. T., Burch, N., Anthony, T., McAleer, S., Elie, R., Cen, S. H., Wang, Z., Gruslys, A., Malysheva, A., Khan, M., Ozair, S., Timbers, F., Pohlen, T., Eccles, T., Rowland, M., Lanctot, M., Lespiau, J.-B., Piot, B., Omidshafiei, S., Lockhart, E., Sifre, L., Beauguerlange, N., Munos, R., Silver, D., Singh, S., Hassabis, D., and Tuyls, K. Mastering the game of Stratego with model-free multiagent reinforcement learning. arXiv: 2206.15378v1 [cs.AI], 2022.

Pham, H., Dai, Z., Xie, Q., and Le, Q. V. Meta pseudo labels. In *CVPR*, June 2021.

Ren, K., Zheng, T., Qin, Z., and Liu, X. Adversarial attacks and defenses in deep learning. *Engineering*, 6(3):346–360, 2020.

Rob. NeuralZ06 bot configuration settings, 2022. URL https://discord.com/channels/417022162348802048/583775968804732928/983781367747837962.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A. C., and Fei-Fei, L. ImageNet large scale visual recognition challenge. *IJCV*, 115(3):211–252, dec 2015.

Schmid, M., Moravcik, M., Burch, N., Kadlec, R., Davidson, J., Waugh, K., Bard, N., Timbers, F., Lanctot, M., Holland, Z., Davoodi, E., Christianson, A., and Bowling, M. Player of games. arXiv: 2112.03178v1 [cs.LG], 2021.

Shankar, V., Roelofs, R., Mania, H., Fang, A., Recht, B., and Schmidt, L. Evaluating machine accuracy on ImageNet. In *ICML*, 2020.

Shapley, L. S. Stochastic games. *PNAS*, 39(10):1095–1100, 1953.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, 2017.

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

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I. J., and Fergus, R. Intriguing properties of neural networks. In *ICLR*, 2014.

Tian, Y., Ma, J., Gong, Q., Sengupta, S., Chen, Z., Pinkerton, J., and Zitnick, L. ELF OpenGo: an analysis and open reimplementation of AlphaZero. In *ICML*, 2019.

Timbers, F., Bard, N., Lockhart, E., Lanctot, M., Schmid, M., Burch, N., Schrittwieser, J., Hubert, T., and Bowling, M. Approximate exploitability: Learning a best response in large games. arXiv: 2004.09677v5 [cs.LG], 2022.

Tromp, J. The game of Go, 2014. URL https://tromp.github.io/go.html.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., Oh, J., Horgan, D., Kroiss, M., Danihelka, I., Huang, A., Sifre, L., Cai, T., Agapiou, J. P., Jaderberg, M., Vezhnevets, A. S., Leblond, R., Pohlen, T., Dalibard, V., Budden, D., Sulsky, Y., Molloy, J., Paine, T. L., Gulcehre, C., Wang, Z., Pfaff, T., Wu, Y., Ring, R., Yogatama, D., Wünsch, D., McKinney, K., Smith, O., Schaul, T., Lillicrap, T., Kavukcuoglu, K., Hassabis, D., Apps, C., and Silver, D. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 2019.

Wu, D. J. Accelerating self-play learning in Go. arXiv: 1902.10565v5 [cs.LG], 2019.

Wu, D. J. KataGo training history and research, 2021a. URL https://github.com/lightvector/KataGo/blob/master/TrainingHistory.md.

Wu, D. J. KataGo's supported Go rules (version 2), 2021b. URL https://lightvector.github.io/KataGo/rules.html.

Wu, D. J. KataGo - networks for kata1, 2022. URL https://katagotraining.org/networks/.

Wu, X., Guo, W., Wei, H., and Xing, X. Adversarial policy training against deep reinforcement learning. In *USENIX Security*, 2021.

Yao, D. KataGo benchmark, 2022. URL https://github.com/inisis/katago-benchmark/blob/5d1c70ea6cda46271d7d48770e4ef43918a8ab84/README.md.

Zinkevich, M., Johanson, M., Bowling, M., and Piccione, C. Regret minimization in games with incomplete information. In *NeurIPS*, volume 20, 2007.

A. Rules of Go used for evaluation

We evaluate all games with Tromp-Taylor rules (Tromp, 2014), after clearing opposite-color stones within pass-alive groups computed by Benson's algorithm (Benson, 1976). Games end after both players pass consecutively, or once all points on the board belong to a pass-alive group or pass-alive territory (defined in Appendix B.6). KataGo was configured to play using these rules in all our matches against it. Indeed, these rules simply consist of KataGo's version of Tromp-Taylor rules with `SelfPlayOpts` enabled (Wu, 2021b). We use a fixed Komi of 6.5.

We chose these *modified Tromp-Taylor* rules because they are simple, and KataGo was trained on (variants) of these rules so should be strongest playing with them. Although the exact rules used were randomized during KataGo's training, modified Tromp-Taylor made up a plurality of the training data. That is, modified Tromp-Taylor is at least as likely as any other configuration seen during training, and is more common than some other options.[1]

In particular, KataGo training randomized between area vs. territory scoring as well as ko, suicide, taxation and button rules from the options described in Wu (2021b). These configuration settings are provided as input to the neural network (Wu, 2019, Table 4), so the network should learn to play appropriately under a range of rule sets. Additionally, during training komi was sampled randomly from a normal distribution with mean 7 and standard deviation 1 (Wu, 2019, Appendix D).

A.1. Difference from typical human play

Although KataGo supports a variety of rules, all of them involve automatically scoring the board at the end of the game. By contrast, when a match between humans end, the players typically confer and agree which stones are dead, removing them from the board prior to scoring. If no agreement can be reached then either the players continue playing the game until the situation is clarified, or a referee arbitrates the outcome of the game.

KataGo has a variety of optional features to help it play well under human scoring rules. For example, KataGo includes an auxiliary prediction head for whether stones are dead or alive. This enables it to propose which stones it believes are dead when playing on online Go servers. Additionally, it includes hard-coded features that can be enabled to make it play in a more human-like way, such as `friendlyPassOk` to promote passing when heuristics suggest the game is nearly over.

These features have led some to speculate that the (undefended) victim passes prematurely in games such as those in Figure 1b because it has learned or is configured to play in a more human-like way. *Prima facie*, this view seems credible: a human player certainly might pass in a similar situation to our victim, viewing the game as already won under human rules. Although tempting, this explanation is not correct: the optional features described above were disabled in our evaluation. Therefore KataGo loses under the rules it was both trained and configured to use.

In fact, the majority of our evaluation used the `match` command to run KataGo vs. KataGo agents which naturally does not support these human-like game play features. We did use the `gtp` command, implementing the Go Text Protocol (GTP), for a minority of our experiments, such as when evaluating KataGo against other AI systems or human players and when evaluating our adversary against KataGo with 10^7 visits. In those experiments, we configured `gtp` to follow the same Tromp-Taylor rules described above, with any human-like extensions disabled.

B. Search algorithms

B.1. A review of Monte-Carlo tree search (MCTS)

In this section, we review the basic Monte-Carlo Tree Search (MCTS) algorithm as used in AlphaGo-style agents (Silver et al., 2016). This formulation is heavily inspired by the description of MCTS given in Wu (2019).

MCTS is an algorithm for growing a game tree one node at a time. It starts from a tree T_0 with a single root node x_0. It then goes through N *playouts*, where every playout adds a leaf node to the tree. We will use T_i to denote the game tree after i playouts, and will use x_i to denote the node that was added to T_{i-1} to get T_i. After MCTS finishes, we have a tree T_N with $N+1$ nodes. We then use simple statistics of T_N to derive a sampling distribution for the next move.

[1] In private communication, the author of KataGo estimated that modified Tromp-Taylor made up a "a few %" of the training data, "growing to more like 10% or as much as 20%" depending on differences such as "self-capture and ko rules that shouldn't matter for what you're investigating, but aren't fully the same rules as Tromp-Taylor".

B.1.1. MCTS PLAYOUTS

MCTS playouts are governed by two learned functions:

a. A value function estimator $\hat{V} : \mathcal{T} \times \mathcal{X} \to \mathbb{R}$, which returns a real number $\hat{V}_T(x)$ given a tree T and a node x in T (where $\mathcal{T}$ is the set of all trees, and $\mathcal{X}$ is the set of all nodes). The value function estimator is meant to estimate how good it is to be at x from the perspective of the player to move at the root of the tree.

b. A policy estimator $\hat{\pi} : \mathcal{T} \times \mathcal{X} \to \mathcal{P}(\mathcal{X})$, which returns a probability distribution over possible next states $\hat{\pi}_T(x)$ given a tree T and a node x in T. The policy estimator is meant to approximate the result of playing the optimal policy from x (from the perspective of the player to move at x).

For both KataGo and AlphaGo, the value function estimator and policy estimator are defined by two deep neural network heads with a shared backbone. The reason that $\hat{V}$ and $\hat{\pi}$ also take a tree T as an argument is because the estimators factor in the sequence of moves leading up to a node in the tree.

A playout is performed by taking a walk in the current game tree T. The walk goes down the tree until it attempts to walk to a node x' that either doesn't exist in the tree or is a terminal node.[2] At this point the playout ends and x' is added as a new node to the tree (we allow duplicate terminal nodes in the tree).

Walks start at the root of the tree. Let x be where we are currently in the walk. The child c we walk to (which may not exist in the tree) is given by

$$\begin{aligned}&\mathrm{walk}_T^{\mathrm{MCTS}}(x)\\&=\begin{cases}\underset{c}{\operatorname{argmax}} \quad \bar{V}_T(c) + \alpha \cdot \hat{\pi}_T(x)[c] \cdot \frac{\sqrt{S_T(x)-1}}{1+S_T(c)} & \text{if root player to move at } x,\\ \underset{c}{\operatorname{argmin}} \quad \bar{V}_T(c) - \alpha \cdot \hat{\pi}_T(x)[c] \cdot \frac{\sqrt{S_T(x)-1}}{1+S_T(c)} & \text{if opponent player to move at } x,\end{cases}\end{aligned} \tag{1}$$

where the argmin and argmax are taken over all children reachable in a single legal move from x. There are some new pieces of notation in Eq 1. Here is what they mean:

1. $\bar{V}_T : \mathcal{X} \to \mathbb{R}$ takes a node x and returns the average value of $\hat{V}_T$ across all the nodes in the subtree of T rooted at x (which includes x). In the special case that x is a terminal node, $\bar{V}_T(x)$ is the result of the finished game as given by the game-simulator. When x does not exist in T, we instead use the more complicated formula[3]

$$\bar{V}_T(x) = \bar{V}_T(\mathrm{par}_T(x)) - \beta \cdot \sqrt{\sum_{x' \in \mathrm{children}_T(\mathrm{par}_T(x))} \hat{\pi}_T(\mathrm{par}_T(x))[x']}\ ,$$

   where $\mathrm{par}_T(x)$ is the parent of x in T and β is a constant that controls how much we de-prioritize exploration after we have already done some exploration.

2. $\alpha \geq 0$ is a constant to trade off between exploration and exploitation.

3. $S_T : \mathcal{X} \to \mathbb{Z}_{\geq 0}$ takes a node x and returns the size of the subtree of T rooted at x. Duplicate terminal nodes are counted multiple times. If x is not in T, then $S_T(x) = 0$.

In Eq 1, one can interpret the first term as biasing the search towards exploitation, and the second term as biasing the search towards exploration. The form of the second term is inspired by UCB algorithms.

B.1.2. MCTS FINAL MOVE SELECTION

The final move to be selected by MCTS is sampled from a distribution proportional to

$$S_{T_N}(c)^{1/\tau}, \tag{2}$$

[2]A "terminal" node is one where the game is finished, whether by the turn limit being reached, one player resigning, or by two players passing consecutively.

[3]Which is used in KataGo and LeelaZero but not AlphaGo (Wu, 2019).

where c in this case is a child of the root node. The temperature parameter τ trades off between exploration and exploitation.[4]

B.1.3. Efficiently implementing MCTS

To efficiently implement the playout procedure one should keep running values of $\bar{V}_T$ and S_T for every node in the tree. These values should be updated whenever a new node is added. The standard formulation of MCTS bakes these updates into the algorithm specification. Our formulation hides the procedure for computing $\bar{V}_T$ and S_T to simplify exposition.

B.2. Adversarial MCTS: Sample (A-MCTS-S)

In this section, we describe in detail how our Adversarial MCTS: Sample (A-MCTS-S) attack is implemented. We build off of the framework for vanilla MCTS as described in Appendix B.1.

A-MCTS-S, just like MCTS, starts from a tree T_0 with a single root node and adds nodes to the tree via a series of N playouts. We derive the next move distribution from the final game tree T_N by sampling from the distribution proportional to

$$S_{T_N}^{\text{A-MCTS}}(c)^{1/\tau}, \quad \text{where } c \text{ is a child of the root node of } T_N. \tag{3}$$

Here, $S_T^{\text{A-MCTS}}$ is a modified version of S_T that measures the size of a subtree while ignoring non-terminal victim-nodes (at victim-nodes it is the victim's turn to move, and at self-nodes it is the adversary's turn to move). Formally, $S_T^{\text{A-MCTS}}(x)$ is the sum of the weights of nodes in the subtree of T rooted at x, with weight function

$$w_T^{\text{A-MCTS}}(x) = \begin{cases} 1 & \text{if } x \text{ is self-node,} \\ 1 & \text{if } x \text{ is terminal victim-node,} \\ 0 & \text{if } x \text{ is non-terminal victim-node.} \end{cases} \tag{4}$$

We grow the tree by A-MCTS playouts. At victim-nodes, we sample directly from the victim's policy π^v:

$$\text{walk}_T^{\text{A-MCTS}}(x) := \text{sample from } \pi_T^v(x). \tag{5}$$

This is a perfect model of the victim *without* search. However, it will tend to underestimate the strength of the victim when the victim plays with search.

At self-nodes, we instead take the move with the best upper confidence bound just like in regular MCTS:

$$\text{walk}_T^{\text{A-MCTS}}(x) := \operatorname*{argmax}_c \quad \bar{V}_T^{\text{A-MCTS}}(c) + \alpha \cdot \hat{\pi}_T(x)[c] \cdot \frac{\sqrt{S_T^{\text{A-MCTS}}(x) - 1}}{1 + S_T^{\text{A-MCTS}}(c)}. \tag{6}$$

Note this is similar to Eq 1 from the previous section. The key difference is that we use $S_T^{\text{A-MCTS}}(x)$ (a weighted version of $S_T(x)$) and $\bar{V}_T^{\text{A-MCTS}}(c)$ (a weighted version of $\bar{V}_T(c)$). Formally, $\bar{V}_T^{\text{A-MCTS}}(c)$ is the weighted average of the value function estimator $\hat{V}_T(x)$ across all nodes x in the subtree of T rooted at c, weighted by $w_T^{\text{A-MCTS}}(x)$. If c does not exist in T or is a terminal node, we fall back to the behavior of $\bar{V}_T(c)$.

B.3. Adversarial MCTS: More Accurate Sampling (A-MCTS-S++)

When computing the policy estimator $\hat{\pi}$ for the root node of a MCTS search (and when playing without tree-search, i.e. "policy-only"), KataGo will pass in different rotated/reflected copies of the game-board and average their results in order to obtain a more stable and symmetry-equivariant policy. That is

$$\hat{\pi}_{\text{root}} = \frac{1}{|S|} \sum_{g \in S \subseteq D_4} g^{-1} \circ \hat{\pi} \circ g,$$

where D_4 is the symmetry group of a square (with 8 symmetries) and S is a randomly sampled subset of D_4.[5]

[4] See `search.h::getChosenMoveLoc` and `searchresults.cpp::getChosenMoveLoc` to see how KataGo does this.

[5] See `searchhelpers.cpp::initNodeNNOutput` for how the symmetry averaging is implemented in KataGo. The size of $|S|$ is configured via the KataGo parameter `rootNumSymmetriesToSample`.

In A-MCTS, we ignore this symmetry averaging because modeling it would inflate the cost of simulating our victim by up to a factor of 8. By contrast, A-MCTS-S++ accurately models this symmetry averaging at the cost of increased computational requirements.

B.4. Adversarial MCTS: Recursive (A-MCTS-R)

In A-MCTS-R, we simulate the victim by starting a new (*recursive*) MCTS search. We use this simulation at victim-nodes, replacing the victim sampling step (Eq. 5) in A-MCTS-S. This simulation will be a perfect model of the victim when the MCTS search is configured to use the same number of visits and other settings as the victim. However, since MCTS search is stochastic, the (random) move taken by the victim may still differ from that predicted by A-MCTS-R. Moreover, in practice, simulating the victim with its full visit count at every victim node in the adversary's search tree can be prohibitively expensive.

B.5. Adversarial MCTS: Victim Model (A-MCTS-VM)

In A-MCTS-VM, we propose fine-tuning a copy of the victim network to predict the moves played by the victim in games played against the adversarial policy. This is similar to how the victim network itself was trained, but may be a better predictor as it is trained on-distribution. The adversary follows the same search procedure as in A-MCTS-S but samples from this predictive model instead of the victim.

A-MCTS-VM has the same inference complexity as A-MCTS-S, and is much cheaper than A-MCTS-R. However, it does impose a slightly greater training complexity due to the need to train an additional network. Additionally, A-MCTS-VM requires white-box access in order to initialize the predictor to the victim network.

In principle, we could randomly initialize the predictor network, making the attack black-box. Notably, imitating the victim has led to successful black-box adversarial policy attacks in other domains (Bui et al., 2022). However, a randomly initialized predictor network would likely need a large number of samples to imitate the victim. Bui et al. (2022) use tens of millions of time steps to imitate continuous control policies, and we expect this number to be still larger in a game as complex as Go.

B.6. Pass-alive defense

Our hard-coded defense modifies KataGo's C++ code to directly remove passing moves from consideration after MCTS, setting their probability to zero. Since the victim must eventually pass in order for the game to end, we allow passing to be assigned nonzero probability when there are no legal moves, *or* when the only legal moves are inside the victim's own pass-alive territory. We also do not allow the victim to play within its own pass-alive territory—otherwise, after removing highly confident pass moves from consideration, KataGo may play unconfident moves within its pass-alive territory, losing liberties and eventually losing the territory altogether. We use a pre-existing function inside the KataGo codebase, `Board::calculateArea`, to determine which moves are in pass-alive territory.

The term "pass-alive territory" is defined in the KataGo rules as follows (Wu, 2021b):

> A {maximal-non-black, maximal-non-white} region R is *pass-alive-territory* for {Black, White} if all {black, white} regions bordering it are pass-alive-groups, and all or all but one point in R is adjacent to a {black, white} pass-alive-group, respectively.

The notion "pass-alive group" is a standard concept in Go (Wu, 2021b):

> A black or white region R is a *pass-alive-group* if there does not exist any sequence of consecutive pseudolegal moves of the opposing color that results in emptying R.

KataGo uses an algorithm introduced by Benson (1976) to efficiently compute the pass-alive status of each group. For more implementation details, we encourage the reader to consult the official KataGo rules and the KataGo codebase on GitHub.

Hyperparameter	Value	Different from KataGo?
Batch Size	256	Same
Learning Rate Scale of Hard-coded Schedule	1.0	Same
Minimum Rows Before Shuffling	250,000	Same
Data Reuse Factor	4	Similar
Adversary Visit Count	600	Similar
Adversary Network Architecture	`b6c96`	Different
Gatekeeping	Disabled	Different
Auto-komi	Disabled	Different
Komi randomization	Disabled	Different
Handicap Games	Disabled	Different
Game Forking	Disabled	Different
Cheap Searches	Disabled	Different

Table 1: Key hyperparameter settings for our adversarial training runs.

C. Hyperparameter settings

We enumerate the key hyperparameters used in our training run in Table 1. For brevity, we omit hyperparameters that are the same as KataGo defaults and have only a minor effect on performance.

The key difference from standard KataGo training is that our adversarial policy uses a `b6c96` network architecture, consisting of 6 blocks and 96 channels. By contrast, the victims we attack range from `b6c96` to `b40c256` in size. We additionally disable a variety of game rule randomizations that help make KataGo a useful AI teacher in a variety of settings but are unimportant for our attack. We also disable gatekeeping, designed to stabilize training performance, as our training has proved sufficiently stable without it.

We train at most 4 times on each data row before blocking for fresh data. This is comparable to the original KataGo training run, although the ratio during that run varied as the number of asynchronous selfplay workers fluctuated over time. We use an adversary visit count of 600, which is comparable to KataGo, though the exact visit count has varied between their training runs.

In evaluation games we use a single search thread for KataGo unless otherwise specified. We used 10 and 1024 search threads for evaluation of victims with 10^6 and 10^7 visits in order to ensure games complete in a reasonable time frame. Holding visit count fixed, using more search threads tends to decrease the strength of an agent. However increasing search threads enables more visits to be used in practice, ultimately enabling higher agent performance. We also allowed players to resign in the 10^7 visit evaluation in order to reduce the length of games and speed up evaluation.

C.1. Configuration for curriculum against victim without search

In Section 5.1, we train using a curriculum over checkpoints, moving on to the next checkpoint when the adversary's win rate exceeds 50%. We ran the curriculum over the following checkpoints, all without search:

1. Checkpoint 127: `b20c256x2-s5303129600-d1228401921` (`cp127`).
2. Checkpoint 200: `b40c256-s5867950848-d1413392747`.
3. Checkpoint 300: `b40c256-s7455877888-d1808582493`.
4. Checkpoint 400: `b40c256-s9738904320-d2372933741`.
5. Checkpoint 469: `b40c256-s11101799168-d2715431527`.
6. Checkpoint 505: `b40c256-s11840935168-d2898845681` (`Latest`).

These checkpoints can all be obtained from Wu (2022).

We start with checkpoint 127 for computational efficiency: it is the strongest KataGo network of its size, 20 blocks or `b20`. The subsequent checkpoints are all 40 block networks, and are approximately equally spaced in terms of training time steps. We include checkpoint 469 in between 400 and 505 for historical reasons: we ran some earlier experiments against checkpoint 469, so it is helpful to include checkpoint 469 in the curriculum to check performance is comparable to prior experiments.

Checkpoint 505 is the latest *confidently rated* network. There are some more recent, larger networks (`b60` = 60 blocks) that may have an improvement of up to 150 Elo. However, they have had too few rated games to be confidently evaluated.

C.2. Configuration for curriculum against victim with passing defense

In Section 5.2, we ran the curriculum over the following checkpoints, all with the pass-alive defense enabled:

1. Checkpoint 39: `b6c96-s45189632-d6589032` ($\texttt{cp39}_{\texttt{def}}$), no search
2. Checkpoint 49: `b6c96-s69427456-d10051148`, no search.
3. Checkpoint 63: `b6c96-s175395328-d26788732`, no search.
4. Checkpoint 79: `b10c128-s197428736-d67404019`, no search.
5. Checkpoint 99: `b15c192-s497233664-d149638345`, no search.
6. Checkpoint 127: `b20c256x2-s5303129600-d1228401921`, no search ($\texttt{cp127}_{\texttt{def}}$).
7. Checkpoint 200: `b40c256-s5867950848-d1413392747`, no search
8. Checkpoint 300: `b40c256-s7455877888-d1808582493`, no search.
9. Checkpoint 400: `b40c256-s9738904320-d2372933741`, no search.
10. Checkpoint 469: `b40c256-s11101799168-d2715431527`, no search.
11. Checkpoint 505: `b40c256-s11840935168-d2898845681` ($\texttt{Latest}_{\texttt{def}}$), no search (1 visit).
12. Checkpoint 505: `b40c256-s11840935168-d2898845681` ($\texttt{Latest}_{\texttt{def}}$), 2 visits.
13. Checkpoint 505: `b40c256-s11840935168-d2898845681` ($\texttt{Latest}_{\texttt{def}}$), 4 visits.
14. Checkpoint 505: `b40c256-s11840935168-d2898845681` ($\texttt{Latest}_{\texttt{def}}$), 8 visits.
15. Checkpoint 505: `b40c256-s11840935168-d2898845681` ($\texttt{Latest}_{\texttt{def}}$), 16 visits.

16–27. ...

28. Checkpoint 505: `b40c256-s11840935168-d2898845681` ($\texttt{Latest}_{\texttt{def}}$), $2^{17} = 131072$ visits.

We move on to the next checkpoint when the adversary's win rate exceeds 50% until we reach $\texttt{Latest}_{\texttt{def}}$ with 2 visits, at which point we increase the win rate threshold to 75%.

D. Compute estimates

In this section, we estimate the amount of compute that went into training our adversary and the amount of compute that went into training KataGo.

We estimate it takes at most ~7 V100 GPU days to train our strongest pass-adversary, at most ~924 V100 GPU days to train our strongest cyclic-adversary, and at least 15,881 V100 GPU days to train the `Latest` KataGo checkpoint. Thus our pass-adversary and cyclic-adversary can be trained using 0.044% and 5.82% (respectively) of the compute it took to train KataGo.

Network	FLOPs / forward pass
`b6c96`	7.00×10^{8}
`b10c128`	2.11×10^{9}
`b15c192`	7.07×10^{9}
`b20c256`	1.68×10^{10}
`b40c256`	3.34×10^{10}

Table 2: Inference compute costs for different KataGo neural network architectures. These costs were empirically measured using `ptflops` and `thop`, and the reported numbers are averaged over the two libraries.

D.1. Estimating the compute used by our attack

We assume the primary cost of training our adversaries is in generating victim-play games. This underestimates the compute used by at most a factor of 14.3%: during our training runs we used only 1 GPU for training the adversary neural network while we used 7-51 GPUs (fluctuating over time due to cluster demand) for generating victim-play games. Moreover, our training GPU was often completely idle and waiting for more victim-play games to be generated, whereas our victim-play GPUs were always maxed out in power consumption.

To estimate the cost of generating victim-play games, we first upper-bounded the total number of neural network evaluations used to generate victim-play games. To do so, we summed the search budget used per move across all the moves in our generated victim-play games. We were only able to compute an upper-bound here because the victim-play code utilizes a neural network cache which remembers the result of evaluating the adversary and victim networks on commonly encountered moves, so not every move in our victim-play games counted towards our overall compute costs.

Critically, we break down the number of neural network evaluations by neural network type. This is necessary because different KataGo network architectures have different inference costs (Table 2). To arrive at our final compute estimate, we multiply the number of neural network evaluations for each neural network type by the inference cost of the corresponding net, and then add.

Per Figure 6, we find that our strongest pass-adversary used 6.33×10^{19} FLOPs to generate victim-play games, and our strongest cyclic-adversary used 9.58×10^{21} FLOPs to generate victim-play games. Inflating these numbers by 14.3% to account for the training GPU yields an upper-bound of 7.24×10^{19} FLOPs to train our strongest pass-adversary, and an upper-bound of 1.09×10^{22} FLOPs to train our strongest cyclic-adversary.

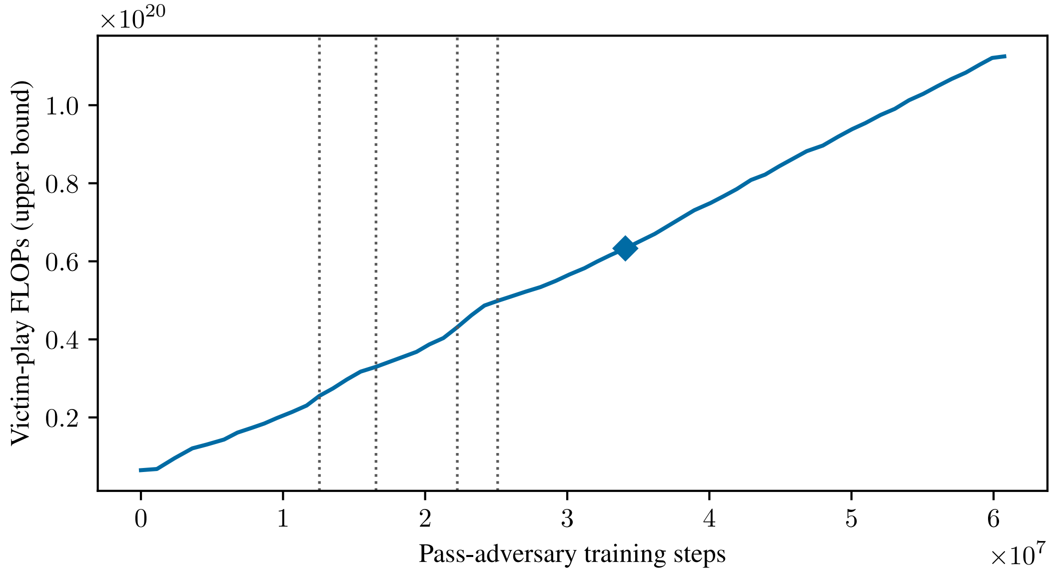
Assuming a conversion factor of ~120 teraFLOPs[6] per Nvidia V100 GPU per second, our upper bounds tell us that training our strongest pass-adversary takes ~7 V100 GPU days, and training our strongest cyclic-adversary takes ~924 V100 GPU days.

D.2. Estimating the compute used to train the `Latest` KataGo checkpoint

The `Latest` KataGo checkpoint was obtained via distributed (i.e. crowdsourced) training starting from the strongest checkpoints in KataGo's "third major run" (Wu, 2021a). The KataGo repository documents the compute used to train the strongest network of this run as: 14 days of training with 28 V100 GPUs, 24 days of training with 36 V100 GPUs, and 119 days of training with 46 V100 GPUs. This totals to $14 \times 28 + 24 \times 36 + 119 \times 46 = 6730$ V100 GPU days of compute.

To lower-bound the remaining compute used by distributed training, we make the assumption that the average row of training-data generated during distributed training was more expensive to generate than the average row of data for the "third major run". We base this assumption off the fact that the "third major run" used `b6`, `b10`, `b20`, and `b40` nets while distributed training used only `b40` nets and larger, with larger nets being more costly to run (Table 2). `Latest` was trained with 2,898,845,681 data rows, while the strongest network of the "third major run" used 1,229,425,124 data rows. We thus lower bound the compute cost of training `Latest` at $2898845681/1229425124 \times 6730 \approx 15881$ V100 GPU days.

[6] Per `https://www.nvidia.com/en-us/data-center/v100/`.



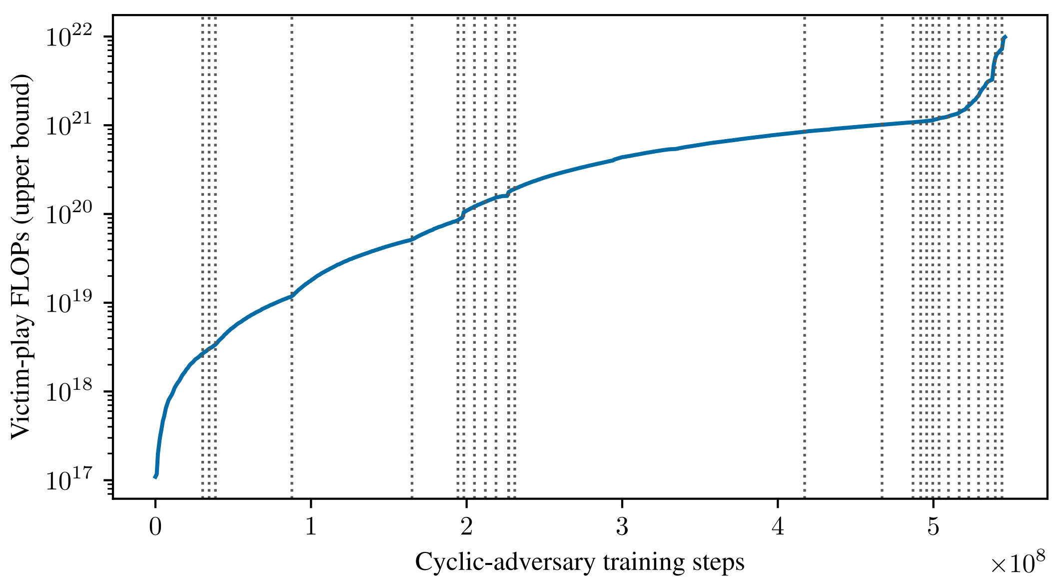


Figure 6: The compute used to generate victim-play games (y-axis) as a function of the number of adversary training steps taken (x-axis). The plots here mirror the structure of Figure 8 and Figure 3. The compute of the pass-adversary is a linear function of its training steps because the pass-adversary was trained against victims of similar size all of which used no search (Appendix C.1). In contrast, the compute of the cyclic-adversary is highly non-linear due to training against a wider range of victim sizes and the exponential ramp up of victim search at the end of its curriculum (Appendix C.2).

KGS handle	Is KataGo?	KGS rank	EGF rank	EGD Profile
Fredda		22	25	Fredrik Blomback
cheater		25	6	Pavol Lisy
TeacherD		26	39	Dominik Boviz
NeuralZ03	✓	31		
NeuralZ05	✓	32		
NeuralZ06	✓	35		
ben0		39	16	Benjamin Drean-Guenaizia
sai1732		40	78	Alexandr Muromcev
Tichu		49	64	Matias Pankoke
Lukan		53	10	Lukas Podpera
HappyLook		54	49	Igor Burnaevskij

Table 3: Rankings of various humans and no-search KataGo bots on KGS (KGS, 2022b). Human players were selected to be those who have European Go Database (EGD) profiles (EGD, 2022), from which we obtained the European Go Federation (EGF) rankings in the table. The KataGo bots are running with a checkpoint slightly weaker than `Latest`, specifically Checkpoint 469 or `b40c256-s11101799168-d2715431527` (Rob, 2022). Per Wu (2022), the checkpoint is roughly 10 Elo weaker than `Latest`.

E. Strength of Go AI systems

In this section, we estimate the strength of KataGo's `Latest` network with and without search and the AlphaZero agent from Schmid et al. (2021) playing with 800 visits.

E.1. Strength of KataGo without search

First, we estimate the strength of KataGo's `Latest` agent playing without search. We use two independent methodologies and conclude that `Latest` without search is at the level of a weak professional.

One way to gauge the performance of `Latest` without search is to see how it fares against humans on online Go platforms. Per Table 3, on the online Go platform KGS, a slightly earlier (and weaker) checkpoint than `Latest` playing without search is roughly at the level of a top-100 European player. However, some caution is needed in relying on KGS rankings:

1. Players on KGS compete under less focused conditions than in a tournament, so they may underperform.
2. KGS is a less serious setting than official tournaments, which makes cheating (e.g., using an AI) more likely. Thus human ratings may be inflated.
3. Humans can play bots multiple times and adjust their strategies, while bots remain static. In a sense, humans are able to run adversarial attacks on the bots, and are even able to do so in a white-box manner since the source code and network weights of a bot like KataGo are public.

Another way to estimate the strength of `Latest` without search is to compare it to other AIs with known strengths and extrapolate performance across different amounts of search. Our analysis critically assumes the transitivity of Elo at high levels of play. We walk through our estimation procedure below:

1. Our anchor is ELF OpenGo at 80,000 visits per move, which won all 20 games played against four top-30 professional players, including five games against the now world number one (Tian et al., 2019). We assume that ELF OpenGo at 80,000 visits is strongly superhuman, meaning it has a 90%+ win rate over the strongest current human.[7] At the time of writing, the top ranked player on Earth has an Elo of 3845 on goratings.org (Coulom, 2022). Under our assumption, ELF OpenGo at 80,000 visits per move would have an Elo of 4245+ on goratings.org.

[7] This assumption is not entirely justified by statistics, as a 20:0 record only yields a 95% binomial lower confidence bound of an 83.16% win rate against top-30 professional players in 2019. It does help however that the players in question were rated #3, #5, #23, and #30 in the world at the time.

2. The strongest network in the original KataGo paper was shown to be slightly stronger than ELF OpenGo (Wu, 2019, Table 1) when both bots were run at 1600 visits per move. From Figure 7, we see that the relative strengths of KataGo networks is maintained across different amounts of search. We thus extrapolate that the strongest network in the original KataGo paper with 80,000 visits would also have an Elo of 4245+ on goratings.org.

3. The strongest network in the original KataGo paper is comparable to the `b15c192-s1503689216-d402723070` checkpoint on katagotraining.org (Wu, 2022). We dub this checkpoint `Original`. In a series of benchmark games, we found that `Latest` without search won 27/3200 games against `Original` with 1600 visits. This puts `Original` with 1600 visits ~823 Elo points ahead of `Latest` without search.

4. Finally, log-linearly extrapolating the performance of `Original` from 1600 to 80,000 visits using Figure 7 yields an Elo difference of ~834 between the two visit counts.

5. Combining our work, we get that `Latest` without search is roughly $823 + 834$ = ~1657 Elo points weaker than ELF OpenGo with 80,000 visits. This would give `Latest` without search an Elo rating of $4245 - 1657$ = ~2588 on goratings.org, putting it at the skill level of a weak professional.

As a final sanity check on these calculations, the raw AlphaGo Zero neural network was reported to have an Elo rating of 3,055, comparable to AlphaGo Fan's 3,144 Elo.[8] Since AlphaGo Fan beat Fan Hui, a 2-dan professional player (Silver et al., 2017), this confirms that well-trained neural networks can play at the level of human professionals. Although there has been no direct comparison between KataGo and AlphaGo Zero, we would expect them to be not wildly dissimilar. Indeed, if anything the latest versions of KataGo are likely stronger, benefiting from both a large distributed training run (amounting to over 10,000 V100 GPU days of training) and four years of algorithmic progress.

E.2. Strength of KataGo with search

In the previous section, we established that `Latest` without search is at the level of a weak professional with rating around ~2588 on goratings.org.

Assuming Elo transitivity, we can estimate the strength of `Latest` by utilizing Figure 7. Our evaluation results tell us that `Latest` with 8 playouts/move is roughly 325 Elo stronger than `Latest` with no search. This puts `Latest` with 8 playouts/move at an Elo of ~2913 on goratings.org—within the top 750 in the world. Beyond 128 playouts/move, `Latest` plays at a superhuman level. `Latest` with 512 playouts/move, for instance, is roughly 1762 Elo stronger than `Latest` with no search, giving an Elo of ~4350, over 500 points higher than the top player on goratings.org.

E.3. Strength of AlphaZero

Prior work from Timbers et al. (2022) described in Section 2 exploited the AlphaZero replica from Schmid et al. (2021) playing with 800 visits. Unfortunately, this agent has never been evaluated against KataGo or against any human player, making it difficult to directly compare its strength to those of our victims. Moreover, since it is a proprietary model, we cannot perform this evaluation ourselves. Accordingly, in this section we seek to estimate the strength of these AlphaZero agents using three anchors: GnuGo, Pachi and Lee Sedol. Our estimates suggest AlphaZero with 800 visits ranges in strength from the top 300 of human players, to being slightly superhuman.

We reproduce relevant Elo comparisons from prior work in Table 4. In particular, Table 4 of Schmid et al. (2021) compares the victim used in Timbers et al. (2022), AlphaZero(s=800,t=800k), to two open-source AI systems, GnuGo and Pachi. It also compares it to a higher visit count version AlphaZero(s=16k, t=800k), from which we can compare using Silver et al. (2018) to AG0 3-day and from there using Silver et al. (2017) to AlphaGo Lee which played Lee Sedol.

Our first strength evaluation uses the open-source anchor point provided by Pachi(s=10k). The authors of Pachi (Baudiš & Gailly, 2012) report it achieves a 2-dan ranking on KGS (Baudiš & Gailly, 2020) when playing with 5000 playouts and using up to 15,000 when needed. We conservatively assume this corresponds to a 2-dan EGF player (KGS rankings tend to be slightly inflated compared to EGF), giving Pachi(s=10k) an EGF rating of 2200 GoR.[9] The victim Alp-

[8] The Elo scale used in Silver et al. (2017) is not directly comparable to our Elo scale, although they should be broadly similar as both are anchored to human players.

[9] GoR is a special rating system (distinct from Elo) used by the European Go Federation. The probability that a player A with a GoR of G_A beats a player B with a GoR of G_B is $1/(1 + \left(\frac{3300-G_A}{3300-G_B}\right)^7)$.

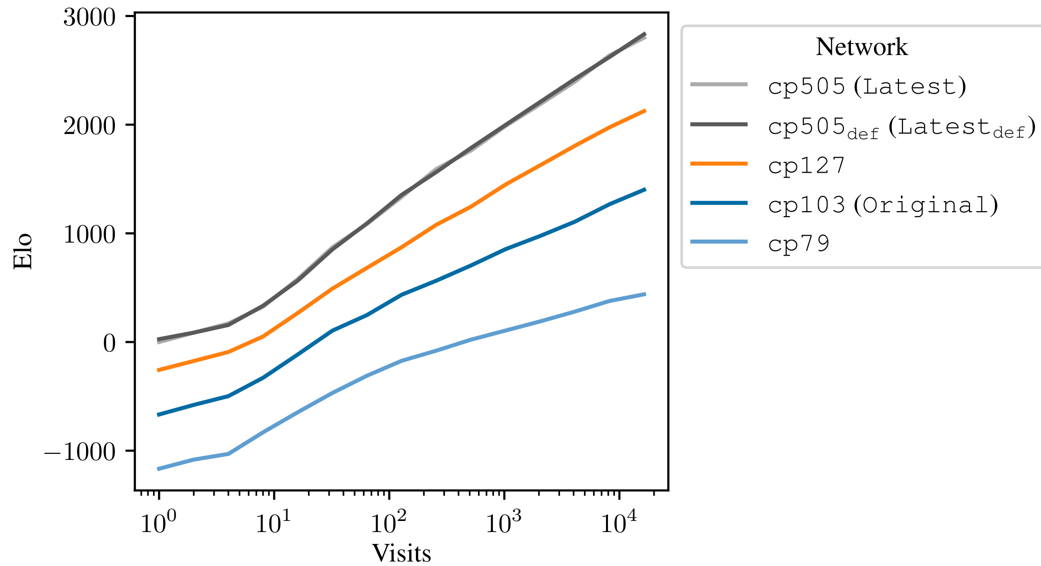


Figure 7: Elo ranking (y-axis) of networks (different colored lines) by visit count (x-axis). The lines are approximately linear on a log x-scale, with the different networks producing similarly shaped lines vertically shifted. This indicates that there is a *consistent* increase in Elo, regardless of network strength, that is logarithmic in visit count. Elo ratings were computed from self-play games among the networks using a Bayesian Elo estimation algorithm (Haoda & Wu, 2022).

Agent	Victim?	Elo (rel GnuGo)	Elo (rel victim)
AlphaZero(s=16k, t=800k)		+3139	+1040
AG0 3-day(s=16k)		+3069	+970
AlphaGo Lee(time=1sec)		+2308	+209
AlphaZero(s=800,t=800k)	✓	**+2099**	0
Pachi(s=100k)		+869	-1230
Pachi(s=10k)		+231	-1868
GnuGo(l=10)		+0	-2099

Table 4: Relative Elo ratings for AlphaZero, drawing on information from Schmid et al. (2021, Table 4), Silver et al. (2018) and Silver et al. (2017). s stands for number of steps, time for thinking time, and t for number of training steps.

haZero(s=800,t=800k) is 1868 Elo stronger than Pachi(s=10k), so assuming transitivity, AlphaZero(s=800,t=800k) would have an EGF rating of 3063 GoR.[10] The top EGF professional Ilya Shiskin has an EGF rating of 2830 GoR (Federation, 2022) at the time of writing, and 2979 Elo on goratings.org (Coulom, 2022). Using Ilya as an anchor, this would give AlphaZero(s=800,t=800k) a rating of 3813 Elo on goratings.org. This is near-superhuman, as the top player at the time of writing has an rating of 3845 Elo on goratings.org.

However, some caution is needed here—the Elo gap between Pachi(s=10k) and AlphaZero(s=800,t=800k) is huge, making the exact value unreliable. The gap from Pachi(s=100k) is smaller, however unfortunately to the best of our knowledge there is no public evaluation of Pachi at this strength. However, the results in Baudiš & Gailly (2020) strongly suggest it would perform at no more than a 4-dan KGS level, or at most a 2400 GoR rating on EGF.[11] Repeating the analysis above then gives AlphaZero(s=800,t=800k) a rating of 2973 GoR on EGF and a rating of 3419 Elo on goratings.org. This is a step below superhuman level, and is roughly at the level of a top-100 player in the world.

If we instead take GnuGo level 10 as our anchor, we get a quite different result. It is known to play between 10 and 11kyu on KGS (KGS, 2022a), or at an EGF rating of 1050 GoR. This gives AlphaZero(s=800,t=800k) an EGF rating of 2900 GoR, or a goratings.org rating of 3174 Elo. This is still strong, in the top ~300 of world players, but is far from superhuman.

The large discrepancy between these results led us to seek a third anchor point: how AlphaZero performed relative to previous AlphaGo models that played against humans. A complication is that the version of AlphaZero that Timbers et al. use differs from that originally reported in Silver et al. (2018), however based on private communication with Timbers et al. we are confident the performance is comparable:

> These agents were trained identically to the original AlphaZero paper, and were trained for the full 800k steps. We actually used the original code, and did a lot of validation work with Julian Schrittweiser & Thomas Hubert (two of the authors of the original AlphaZero paper, and authors of the ABR paper) to verify that the reproduction was exact. We ran internal strength comparisons that match the original training runs.

Table 1 of Silver et al. (2018) shows that AlphaZero is slightly stronger than AG0 3-day (AlphaGo Zero, after 3 days of training), winning 60 out of 100 games giving an Elo difference of +70. This tournament evaluation was conducted with both agents having a thinking time of 1 minute. Table S4 from Silver et al. (2018) reports that 16k visits are performed per second, so the tournament evaluation used a massive 960k visits–significantly more than reported on in Table 4. However, from Figure 7 we would expect the *relative* Elo to be comparable between the two systems at different visit counts, so we extrapolate AG0 3-day at 16k visits as being an Elo of $3139 - 70 = 3069$ relative to GnuGo.

Figure 3a from Silver et al. (2017) report that AG0 3-day achieves an Elo of around 4500. This compares to an Elo of 3,739 for AlphaGo Lee. To the best of our knowledge, the number of visits achieved per second of AlphaGo Lee has not been reported. However, we know that AG0 3-day and AlphaGo Lee were given the same amount of thinking time, so we can infer that AlphaGo Lee has an Elo of -761 relative to AG0 3-day. Consequently, AlphaGo Lee(time=1sec) thinking for 1 second has an Elo relative to GnuGo of $3069 - 761 = 2308$.

Finally, we know that AlphaGo Lee beat Lee Sedol in four out of five matches, giving AlphaGo Lee a +240 Elo difference relative to Lee Sedol, or that Lee Sedol has an Elo of 2068 relative to Gnu Go level 10. This would imply that the victim is slightly stronger than Lee Sedol. However, this result should be taken with some caution. First, it relies on transitivity through many different versions of AlphaGo. Second, the match between AlphaGo Lee and Lee Sedol was played under two hours of thinking time with 3 byoyomi periods of 60 seconds per move Silver et al. (2018, page 30). We are extrapolating from this to some hypothetical match between AlphaGo Lee and Lee Sedol with only 1 second of

[10] This is a slightly nontrivial calculation: we first calculated the win-probability x implied by an 1868 Elo difference, and then calculated the GoR of AlphaZero(s=800,t=800k) as the value that would achieve a win-probability of x against Pachi(s=10k) with 2200 GoR. We used the following notebook to perform this and subsequent Elo-GoR conversion calculations: —anonymized—.

[11] In particular, Baudiš & Gailly (2020) report that Pachi achieves a 3-dan to 4-dan ranking on KGS when playing on a cluster of 64 machines with 22 threads, compared to 2-dan on a 6-core Intel i7. Figure 4 of Baudiš & Gailly (2012) confirms playouts are proportional to the number of machines and number of threads, and we'd therefore expect the cluster to have 200x as many visits, or around a million visits. If 1 million visits is at best 4-dan, then 100,000 visits should be weaker. However, there is a confounder: the 1 million visits was distributed across 64 machines, and Figure 4 shows that distributed playouts do worse than playouts on a single machine. Nonetheless, we would not expect this difference to make up for a 10x difference in visits. Indeed, Baudiš & Gailly (2012, Figure 4) shows that 1 million playouts spread across 4 machines (red circle) is substantially better than 125,000 visits on a single machine (black circle), achieving an Elo of around 150 compared to -20.

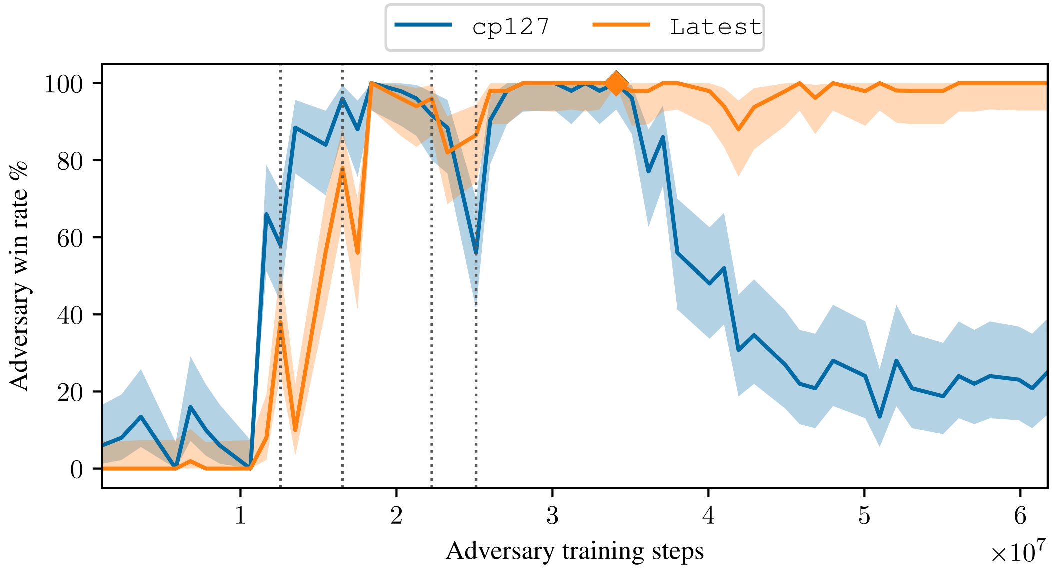


Figure 8: The win rate (y-axis) of the pass-adversary from Section 5.1 over time (x-axis) against the `cp127` and `Latest` victim policy networks playing without search. The strongest adversary checkpoint (marked ⧫) wins 1047/1048 games against `Latest`. The adversary overfits to `Latest`, winning less often against `cp127` over time. Shaded interval is a 95% Clopper-Pearson interval over $n = 50$ games per checkpoint. The adversarial policy is trained with a curriculum, starting from `cp127` and ending at `Latest` (see Appendix C.1). Vertical dashed lines denote switches to a later victim policy.

thinking time per player. Although the Elo rating of Go AI systems seems to improve log-linearly with thinking time, it is unlikely this result holds for humans.

F. More evaluations of adversaries against KataGo

In this section we provide more evaluations of our attacks from Section 5.

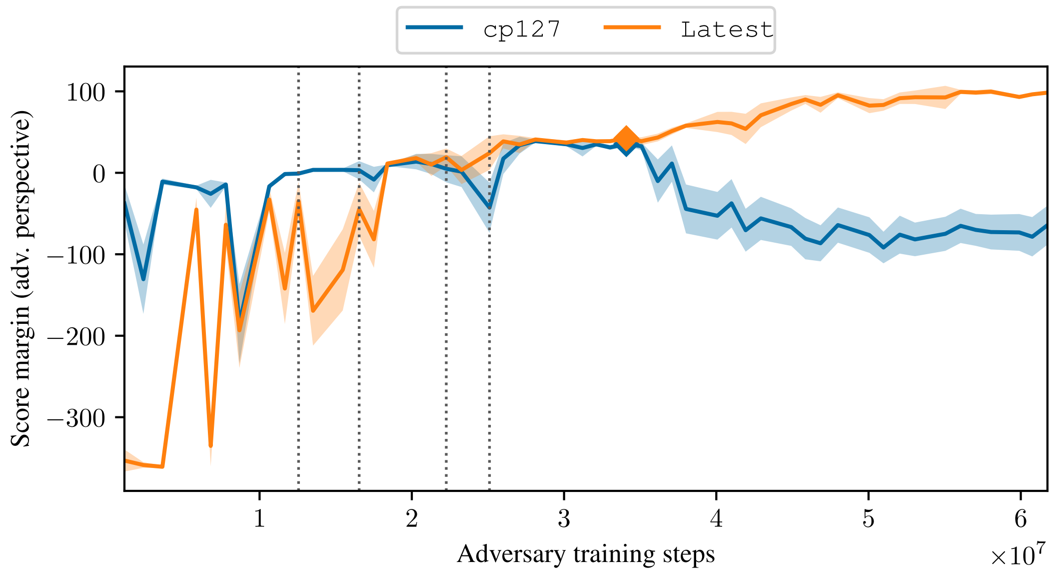
F.1. Evolution of pass-adversary over training

In Figure 8 we evaluate the pass-adversary from Section 5.1 against `cp127` and `Latest` throughout the training process of the adversary. We find the pass-adversary attains a large (>90%) win rate against both victims throughout much of training. However, over time the adversary overfits to `Latest`, with the win rate against `cp127` falling to around 22%.

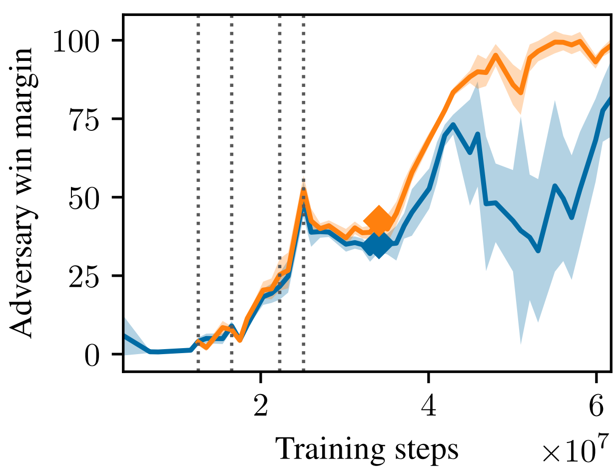
In Figure 9, the context is the same as the preceding figure but instead of win rate we report the margin of victory. In the win-only and loss-only subfigures, we plot only points with at least 5 wins or losses. Note that standard Go has no incentives for winning by a larger margin; we examine these numbers for solely additional insight into the training process of our adversary. We see that even after win rate is near 100% against `Latest`, the win margin continues to increase, suggesting the adversary is still learning.

F.2. Score margin of the cyclic-adversary

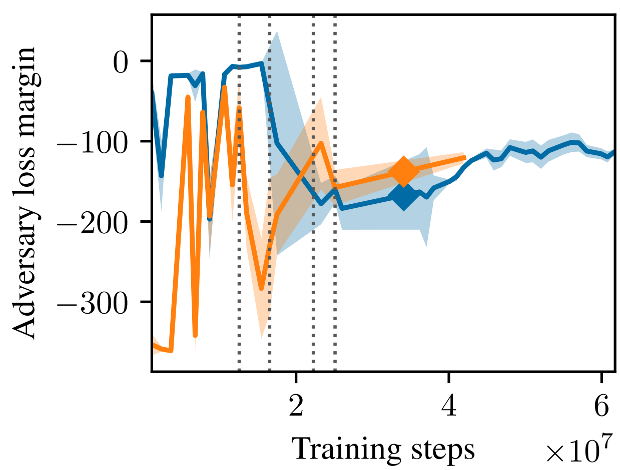
In Figure 10, we show the margin of victory over the training process of the cyclic-adversary from Section 5.2 against victims with the pass-alive defense. The corresponding win rate is shown in Figure 3. Compared to Figure 9, we see that the margin of victory is typically larger. This is likely because the cyclic-adversary either captures a large group or gives up almost everything in a failed attempt. After approximately 250 million training steps, the margins are relatively stable, but we do see a gradual reduction in the loss margin against $\texttt{Latest}_{\texttt{def}}$ with 2048 visits (preceding the eventual spike in win rate against that victim).



(a) Final score margin from adversary's perspective (i.e. adversary score − victim score) on y-axis vs. adversary training steps on x-axis.

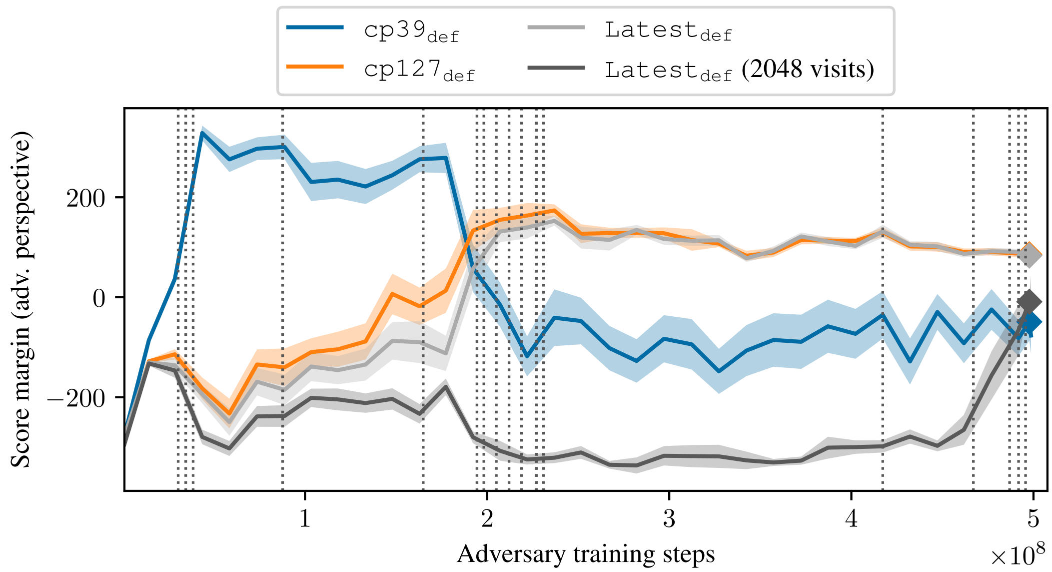


(b) Score margin, restricted to games adversary won.

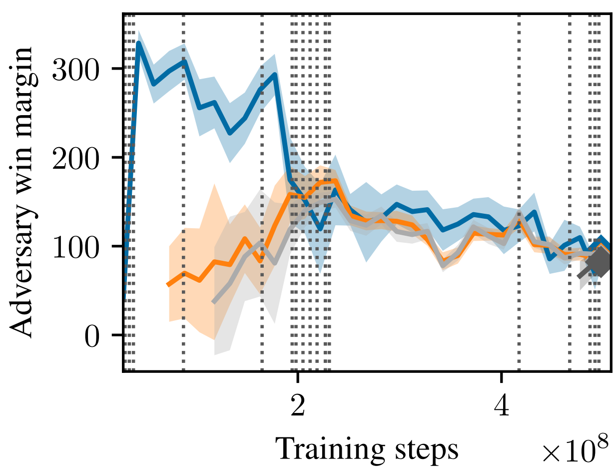


(c) Score margin, restricted to games adversary lost.

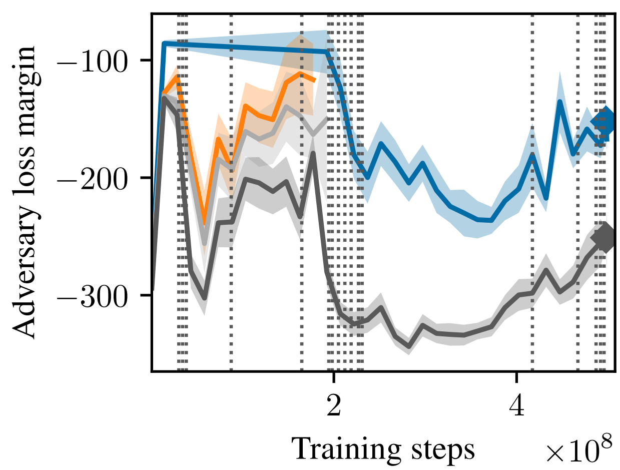
Figure 9: We evaluate the average margin of victory for the pass-adversary from Section 5.1 against `Latest` without search as the training process progresses. Shaded regions are 95% T-intervals over $n = 50$ games per checkpoint.



(a) Final score margin from adversary's perspective (i.e. adversary score − victim score) on y-axis vs. adversary training steps on x-axis.

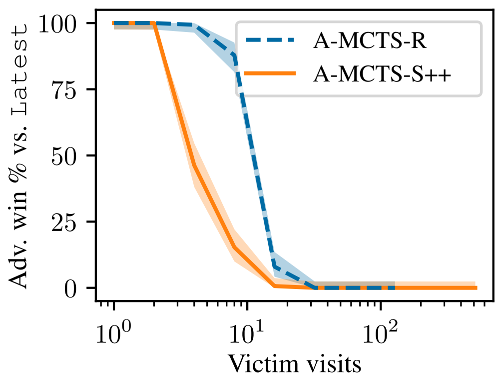


(b) Score margin, restricted to games adversary won.

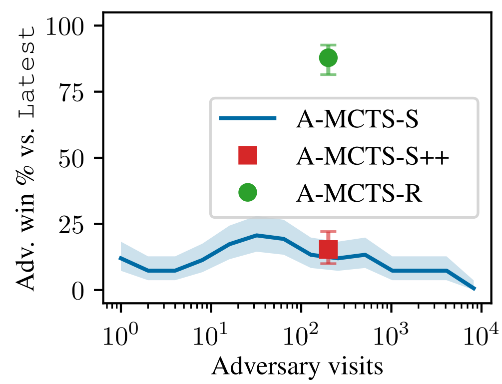


(c) Score margin, restricted to games adversary lost.

Figure 10: We evaluate the average margin of victory for the cyclic-adversary from Section 5.2 against `Latest` with 2048 visits as the training process progresses. Shaded regions are 95% T-intervals over $n = 50$ games per checkpoint.



(a) Win rate by number of victim visits (x-axis) for A-MCTS-S and A-MCTS-R. The adversary is run with 200 visits. The adversary is unable to exploit `Latest` when it plays with at least 32 visits.



(b) Win rate by number of adversary visits with A-MCTS-S, playing against `Latest` with 8 visits. In this case, scaling up the number of adversary visits does not lead to stronger attack.

Figure 11: We evaluate the ability of the pass-adversary from Section 5.1 trained against `Latest` without search to transfer to `Latest` with search.

F.3. Pass-adversary vs. victims with search

We evaluate the ability of the pass-adversary to exploit `Latest` playing *with* search (the pass-adversary was trained only against no-search victims). Although the pass-adversary with A-MCTS-S and 200 visits achieves a win rate of 100% over 160 games against `Latest` without search, in Figure 11a we find the win rate drops to 15.3% at 8 victim visits. However, A-MCTS-S models the victim as having no search at both training and inference time. We also test A-MCTS-R, which correctly models the victim at inference by performing an MCTS search at each victim node in the adversary's tree. We find that our pass-adversary with A-MCTS-R performs somewhat better, obtaining an 87.8% win rate against `Latest` with 8 visits, but performance drops to 8% at 16 visits.

Of course, A-MCTS-R is more computationally expensive than A-MCTS-S. An alternative way to spend our inference-time compute budget is to perform A-MCTS-S with a greater *adversary* visit count. We see in Figure 11b, however, that this does not increase the win rate of the pass-adversary against `Latest` with 8 visits. It seems that `Latest` at a modest number of visits quickly becomes resistant to our pass-adversary, no matter how we spend our inference-time compute budget.

F.4. Transferring attacks between checkpoints

In Figure 12, we train adversaries against the `Latest` and `cp127` checkpoints respectively and evaluate against both checkpoints. An adversary trained against `Latest` does better against `Latest` than `cp127`, despite `Latest` being a stronger agent. The converse also holds: an agent trained against `cp127` does better against `cp127` than `Latest`. This pattern holds across visit counts. These results support the conclusion that different checkpoints have unique vulnerabilities.

F.5. Baseline attacks

We also test *hard-coded* baseline adversarial policies. These baselines were inspired by the behavior of our trained adversary. The *Edge* attack plays random legal moves in the outermost ℓ^∞-box available on the board. The *Spiral* attack is similar to the *Edge* attack, except that it plays moves in a deterministic counterclockwise order, forming a spiral pattern. The *Random* attack plays uniformaly random legal moves. Finally, we also implement *Mirror Go*, a classic strategy that plays the opponent's last move reflected about the $y = x$ diagonal. If the opponent plays on $y = x$, Mirror Go plays that move reflected along the $y = -x$ diagonal. If the mirrored vertex is taken, Mirror Go plays the closest legal move by ℓ^1 distance.

For each of these baseline policies, if the victim passes, then the policy will pass to end the game if passing is a winning move.

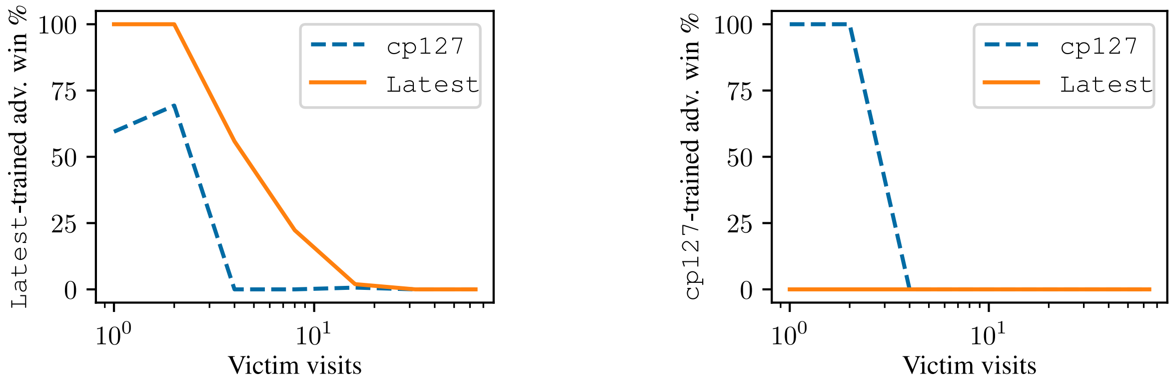
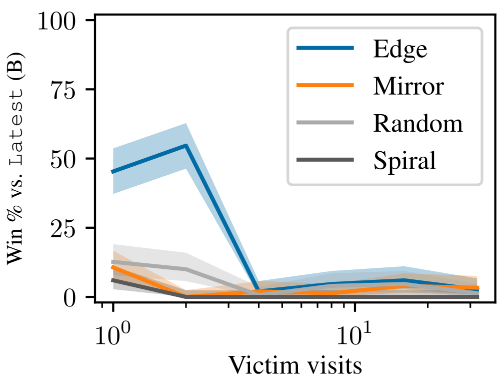
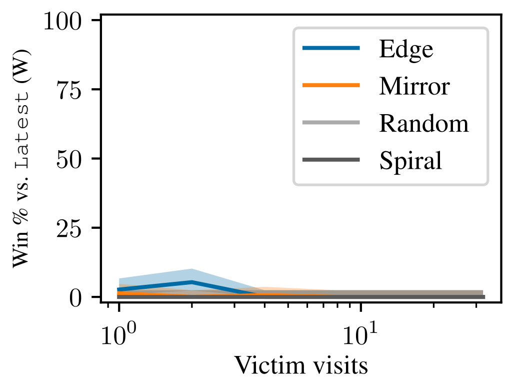


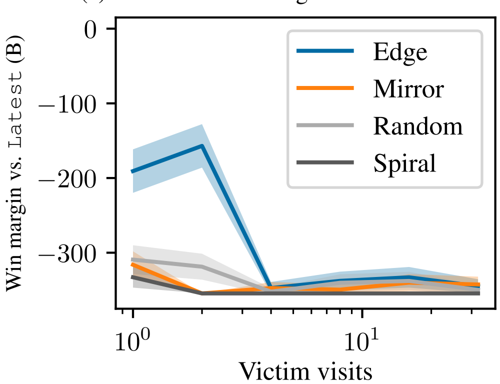
Figure 12: An adversary trained against `Latest` (left) or `cp127` (right), evaluated against both `Latest` and `cp127` at various visit counts. The adversary always uses 600 visits/move.



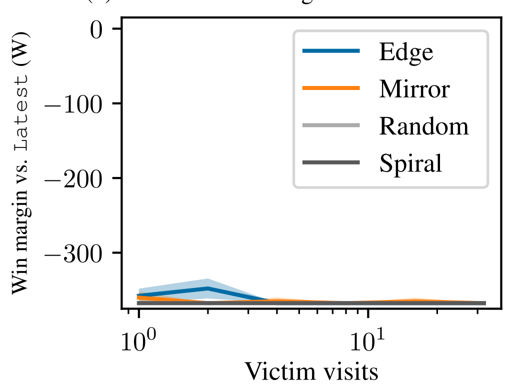
(a) Win rate as white against `Latest` as black.



(b) Win rate as black against `Latest` as white.

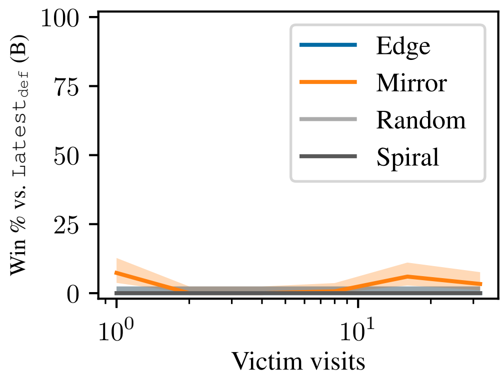


(c) Win margin as white against `Latest` as black.

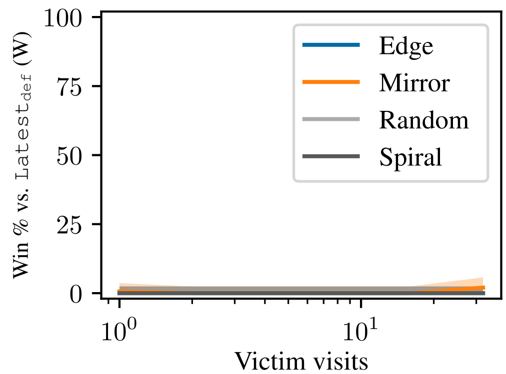


(d) Win margin as white against `Latest` as black.

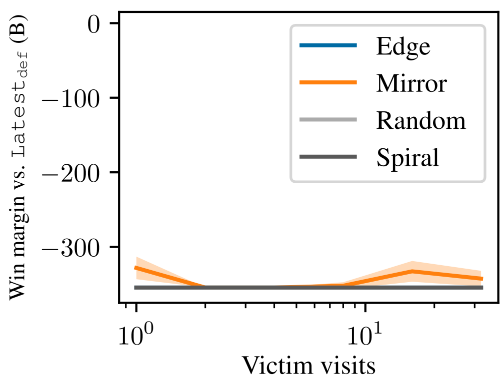
Figure 13: Win rates and win margins of different baseline attacks versus `Latest` at varying visit counts (x-axis). 95% confidence intervals are shown. The win margins are negative, indicating that on average the victim gains more points than the attack does.



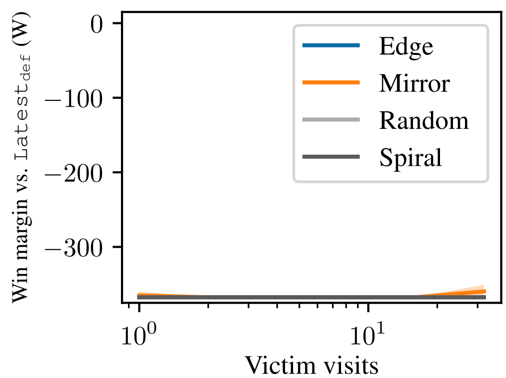
(a) Win rate as white against `Latest`$_{\mathrm{def}}$ as black.



(b) Win rate as black against `Latest`$_{\mathrm{def}}$ as white.

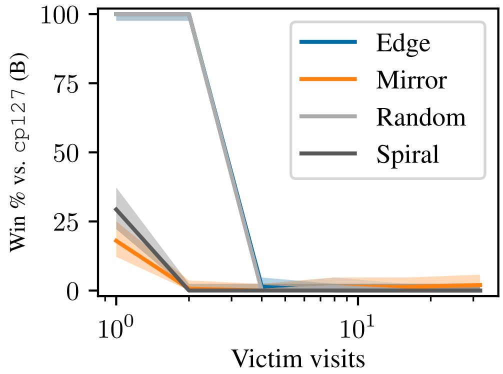


(c) Win margin as white against `Latest`$_{\mathrm{def}}$ as black.

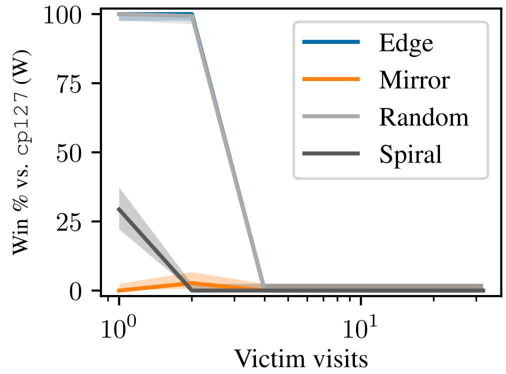


(d) Win margin as white against `Latest`$_{\mathrm{def}}$ as black.

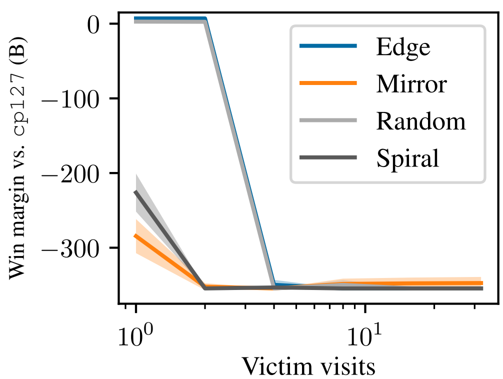
Figure 14: Win rates and win margins of different baseline attacks versus `Latest`$_{\mathrm{def}}$ at varying visit counts (x-axis). 95% confidence intervals are shown. None of the attacks see much success.



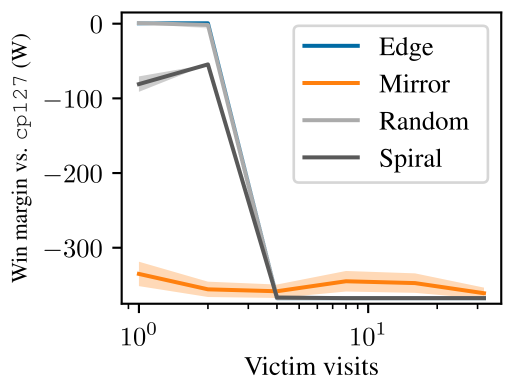
(a) Win rate as white against `cp127` as black.



(b) Win rate as black against `cp127` as white.



(c) Win margin as white against `cp127` as black.



(d) Win margin as white against `cp127` as black.

Figure 15: Win rates and win margins of different baseline attacks versus `cp127` at varying visit counts (x-axis). 95% confidence intervals are shown.

In Figure 13, we plot the win rate and win margin of the baseline attacks against the KataGo victim `Latest`. The edge attack is the most successful, achieving a 45% win rate when `Latest` plays as black with no search. None of the attacks work well once `Latest` is playing with at least 4 visits.

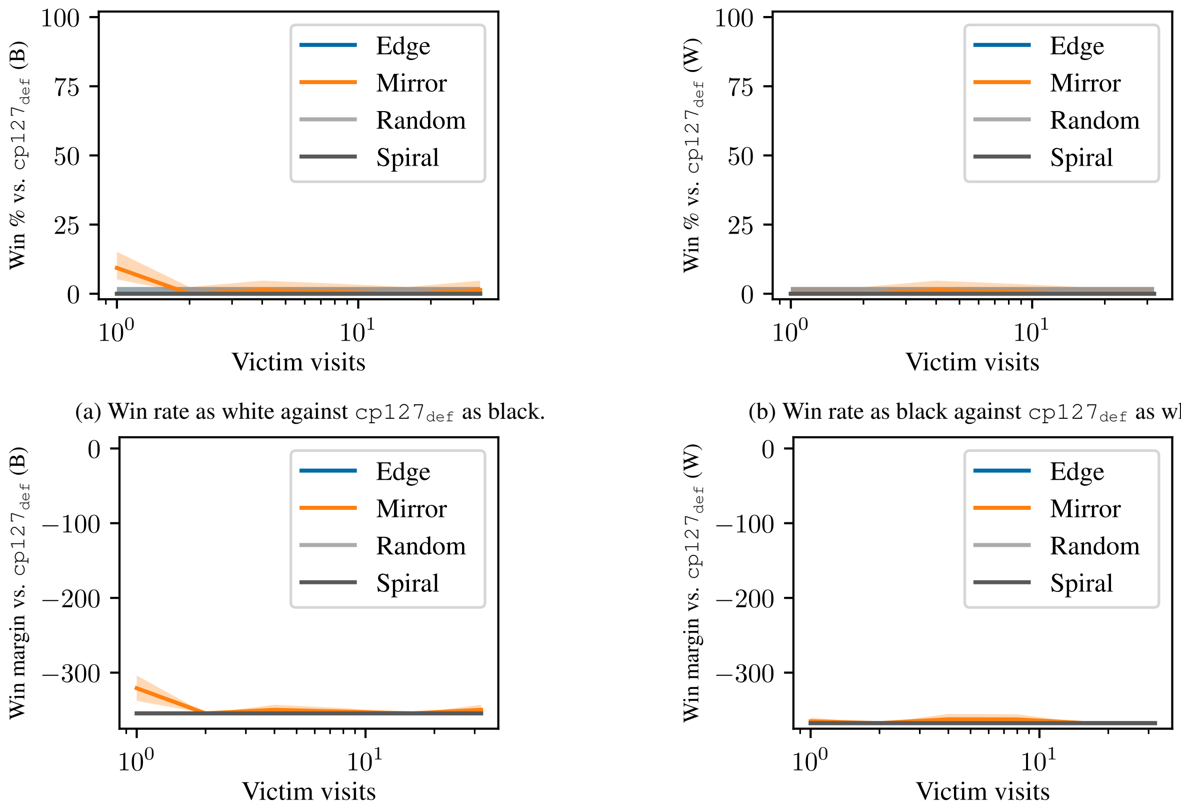
In Figure 14, we plot the win rate and win margin against $\texttt{Latest}_{\texttt{def}}$. In this setting, none of the attacks work well even when $\texttt{Latest}_{\texttt{def}}$ is playing with no search, though the mirror attack wins very occasionally.

We also run the baseline attacks against the weaker `cp127`, with Figure 15 plotting the win rate and win margin of the baseline attacks against `cp127` and Figure 16 plotting the same statistics against $\texttt{cp127}_{\texttt{def}}$. `cp127` without search is shockingly vulnerable to simple attacks, losing all of its games against the edge and random attacks. Still, like `Latest`, `cp127` becomes much harder to exploit once it is playing with at least 4 visits, and $\texttt{cp127}_{\texttt{def}}$ only suffers losses to the mirror attack.

F.6. Understanding the pass-adversary

We observed in Figure 1b that the pass-adversary appears to win by tricking the victim into passing prematurely, at a time favorable to the adversary. In this section, we seek to answer three key questions. First, *why* does the victim pass even when it leads to a guaranteed loss? Second, is passing *causally* responsible for the victim losing, or would it lose anyway for a different reason? Third, is the adversary performing a *simple* strategy, or does it contain some hidden complexity?

Evaluating the `Latest` victim without search against the pass-adversary over $n = 250$ games, we find that `Latest` passes (and loses) in 247 games and does not pass (and wins) in the remaining 3 games. In all cases, `Latest`'s value head estimates a win probability of greater than 99.5% after the final move it makes, although its true win percentage is only 1.2%. `Latest` predicts it will *win* by $\mu = 134.5$ points ($\sigma = 27.9$) after its final move, and passing would be reasonable



(a) Win rate as white against $\texttt{cp127}_{\texttt{def}}$ as black.

(b) Win rate as black against $\texttt{cp127}_{\texttt{def}}$ as white.

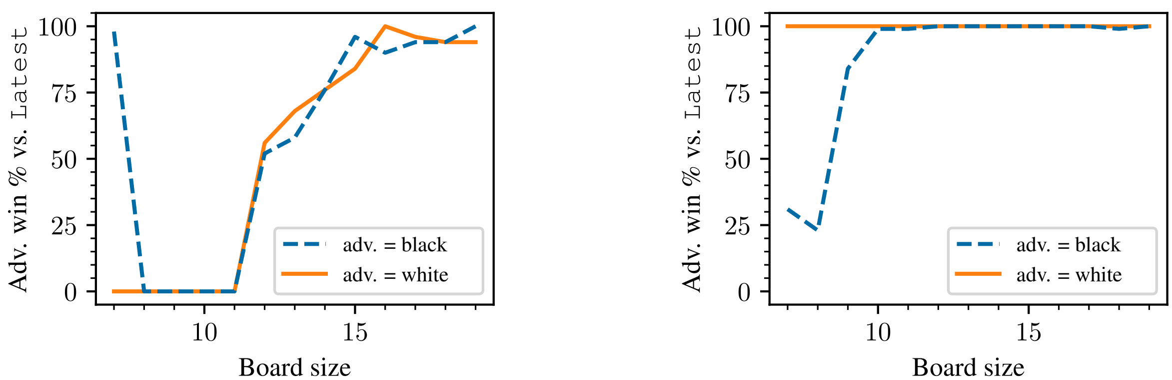
(c) Win margin as white against $\texttt{cp127}_{\texttt{def}}$ as black.

(d) Win margin as white against $\texttt{cp127}_{\texttt{def}}$ as black.

Figure 16: Win rates and win margins of different baseline attacks versus $\texttt{cp127}_{\texttt{def}}$ at varying visit counts (x-axis). 95% confidence intervals are shown.

Board size ($n \times n$)	7	8	9	10	11	12	13	14	15	16	17	18	19
Training frequency (%)	1	1	4	2	3	4	10	6	7	8	9	10	35

Table 5: Percentage of games played at each board size throughout the training of our adversaries. These percentages are the defaults for KataGo training.



(a) Cyclic-adversary with 600 visits versus `Latest` with 8192 visits.

(b) Pass-adversary with 600 visits versus `Latest` without search.

Figure 17: Win rate of our adversaries playing as each color against `Latest` on different board sizes.

if it were so far ahead. But in fact it is just one move away from losing by an average of 86.26 points.

We conjecture that the reason why the victim's prediction is so mistaken is that the games induced by playing against the adversarial policy are very different from those seen during the victim's self-play training. Certainly, there is no fundamental inability of neural networks to predict the outcome correctly. The adversary's value head achieves a mean-squared error of only 3.18 (compared to 49,742 for the victim) on the adversary's penultimate move. The adversary predicts it will win 98.6% of the time—extremely close to the true 98.8% win rate in this sample.

To verify whether this pathological passing behavior is the reason the adversarial policy wins, we design a hard-coded defense for the victim, the pass-alive defense described in Section 5.2. Whereas the pass-adversary won greater than 99% of games against vanilla `Latest`, we find that it *loses* all 1600 evaluation games against $\texttt{Latest}_{\texttt{def}}$. This confirms the pass-adversary wins via passing.

Unfortunately, this "defense" is of limited effectiveness: as we saw in Section 5.2, repeating the attack method finds the cyclic-adversary that can beat it. Moreover, the defense causes KataGo to continue to play even when a game is clearly won or lost, which is frustrating for human opponents. The defense also relies on hard-coded knowledge about Go, using a search algorithm to compute the pass-alive territories.

Finally, we seek to determine if the adversarial policy is winning by pursuing a simple high-level strategy, or via a more subtle exploit such as forming an adversarial example by the pattern of stones it plays. We start by evaluating the hard-coded baseline adversarial policies described in Appendix F.5. In Figure 13, we see that all of our baseline attacks perform substantially worse than our pass-adversary (Figure 11a). Moreover, when our baseline attacks do win it is usually due to the komi bonus given to white (as compensation for playing second), and therefore they almost never win as black. By contrast, our pass-adversary wins playing as either color, and often by a large margin (in excess of 50 points).

F.7. Performance of adversaries on other board sizes

Throughout this paper, we have been only reporting on the performance of our adversaries on 19×19 boards. During training, however, our adversaries played games on different board sizes from 7×7 up to 19×19 with the default KataGo training frequencies listed in Table 5, so our adversaries are also able to play on smaller board sizes.

Figure 17a plots the win rate of the cyclic-adversary against `Latest` playing with 8192 visits, and Figure 17b plots the win rate of the pass-adversary against `Latest` playing without search. The komi is 8.5 for 7×7 boards, 9.5 for 8×8

boards, and 6.5 otherwise. These values were taken from analysis by David Wu, creator of KataGo, on fair komis for different board sizes under Chinese Go rules.[12] These are the same komi settings we used during training, except that we had a configuration typo that swapped the komis for 8×8 boards and 9×9 boards during training.

The cyclic-adversary sets up the cyclic structure on board sizes of at least 12×12, and not coincidentally, those are board sizes on which the cyclic-adversary achieves wins. The pass-adversary achieves wins on all board sizes via getting the victim to pass early, but on board sizes of 12×12 and smaller, the adversary sometimes plays around the edge of the board instead of playing primarily in one corner.

G. Transfer of cyclic-adversary to other Go systems

G.1. Algorithmic transfer

The cyclic-adversary transferred zero-shot to attacking Leela Zero and ELF OpenGo. Note that in this transfer, not only are the weights of the adversary trained against a different model (i.e. KataGo), the simulated victim in the search (A-MCTS-S simulating KataGo) is also different from the actual victim.

We ran ELF OpenGo with its final model and 80,000 rollouts. A weaker model with 80,000 rollouts was already strong enough to consistently defeat several top-30 Go players (Tian et al., 2019). We ran Leela Zero with its final model (released February 15, 2021), unlimited time, and a maximum of 40,000 visits per move. We turned off resignation for both ELF and Leela. We expect that ELF and Leela play at a superhuman level with these parameters. Confirming this, we found that ELF and Leela with these parameter settings defeat `Latest` with 128 visits a majority of the time, and we estimate in Appendix E.2 that `Latest` with 128 visits plays at a superhuman level.

Our adversary won 8/132 = 6.1% games against Leela Zero and 5/142 = 3.5% games against ELF OpenGo. Although this win rate is considerably lower than that attained against KataGo, to beat these systems at all zero-shot is significant given that even the best human players almost never win against these systems.

G.2. Human transfer

The cycle attack discovered by our algorithmic adversaries can also be implemented by humans. An author, who is a Go expert, successfully attacked a variety of Go programs including KataGo and Leela Zero playing with 100,000 visits, both of which even top professional players are normally unable to beat. They also won 14/15 games against `JBXKata005`, a custom KataGo implementation not affiliated with the authors, which was the strongest bot available to play on the KGS Go Server at the time of the test. In addition, they also tested giving JBXKata005 3, 5, and 9 handicap stones (additional moves at the beginning of the game), and won in all cases.

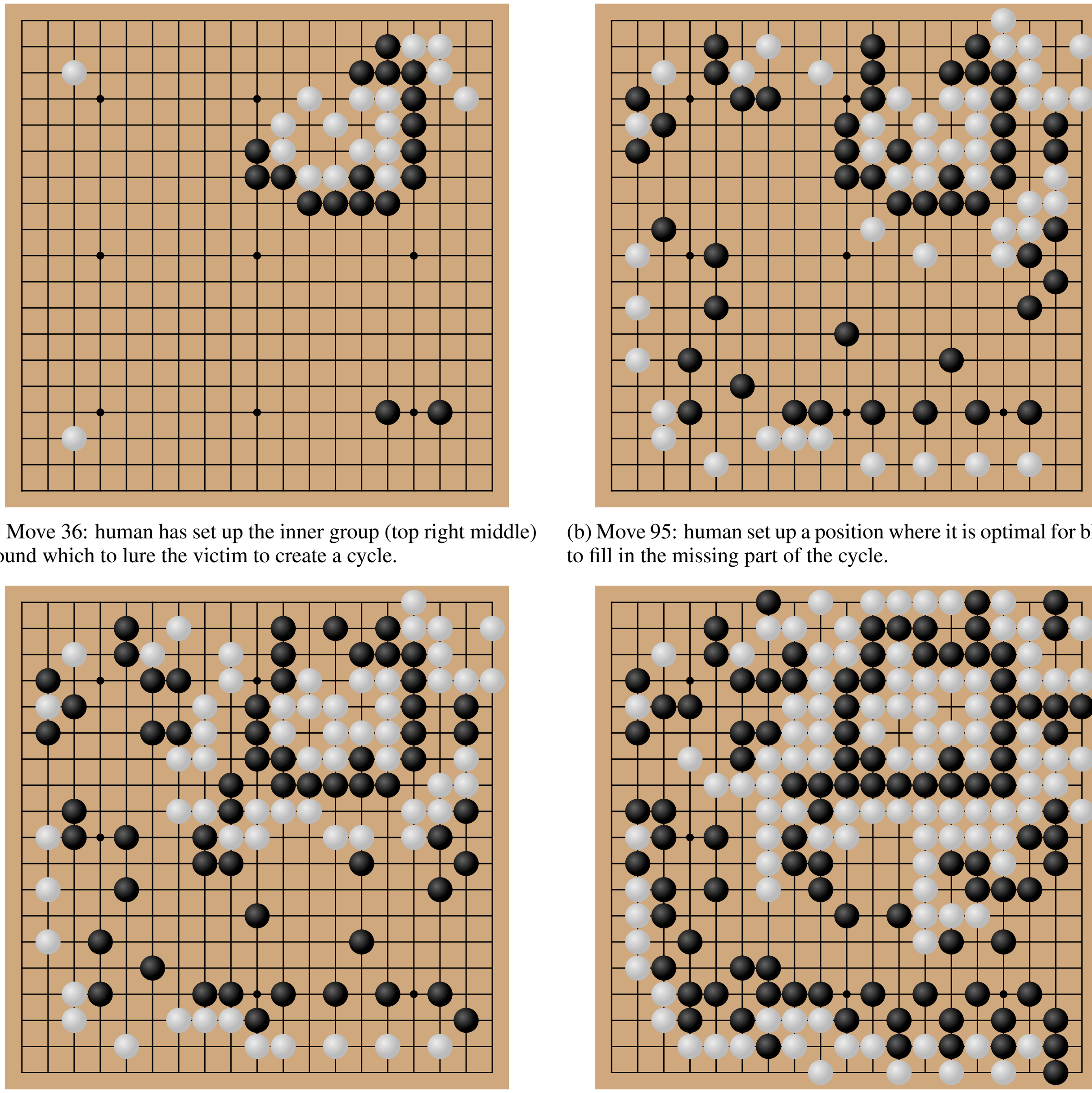
In the following figures we present selected positions from the games. The full games are available on our website. First, Figure 18 shows key moments in a game against KataGo with 100k visits. Figure 19 shows the same against LeelaZero with 100k visits. Figure 20 shows a normal game against JBXKata005, while Figure 21 shows a game where JBXKata005 received the advantage of a 9 stone handicap at the start. In each case the strategy is roughly the following: first, set up an "inside" group and let or lure the victim to surround it, creating a cyclic group. Second, surround the cyclic group. Third, guarantee the capture before the victim realizes it is in danger and defends. In parallel to all these steps, one must also make sure to secure enough of the rest of the board that capturing the cyclic group will be enough to win.

H. The Role of Search in Robustness

Asymptotically, search leads to robustness: with infinite search, a model could perfectly evaluate every possible move and never make a mistake. However, this level of search is computationally impossible. Our results show that in computationally feasible regimes, search is insufficient to produce full robustness. Does search have a meaningful impact at all on robustness in realistic regimes? In this appendix we show that it does substantially improve robustness, and consequently, while not a full solution, it is nonetheless a practical tool for creating more robust models and pipelines.

In results discussed previously, we see that for a fixed adversary, increasing victim search improves its win rate (e.g. Figure 4a). This provides evidence search increases robustness. However, there are potential confounders. First, the

[12] The komi analysis is at `https://lifein19x19.com/viewtopic.php?p=259358#p259358`.



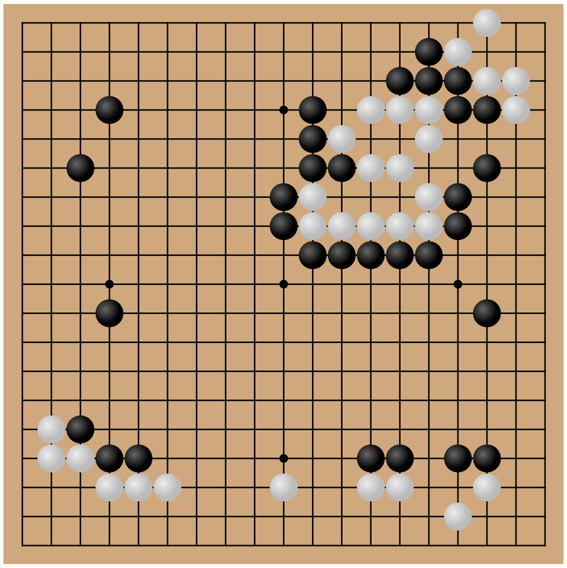
(a) Move 36: human has set up the inner group (top right middle) around which to lure the victim to create a cycle.

(b) Move 95: human set up a position where it is optimal for black to fill in the missing part of the cycle.

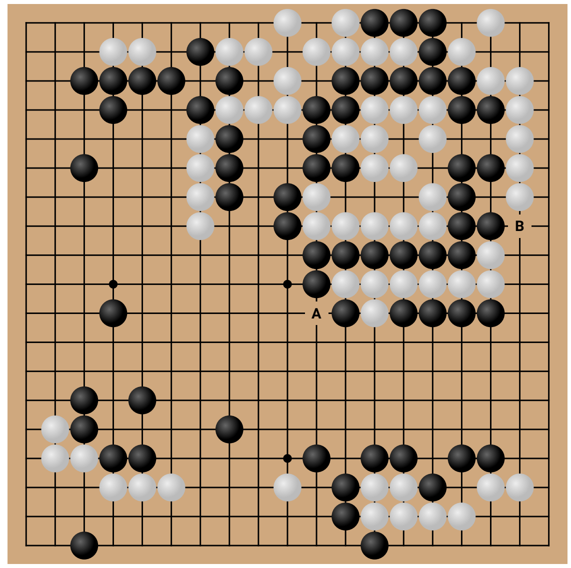
(c) Move 122: victim's cycle group is now surrounded. It remains to capture it before the victim catches on and defends.

(d) Move 210: by now none of the victim's stones in the top right can avoid capture. The victim finally realizes and resigns.

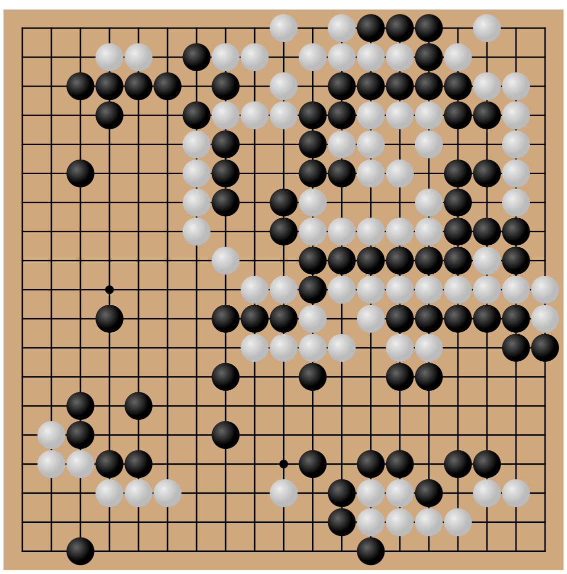
Figure 18: Human (white) beats KataGo with 100k visits (black).



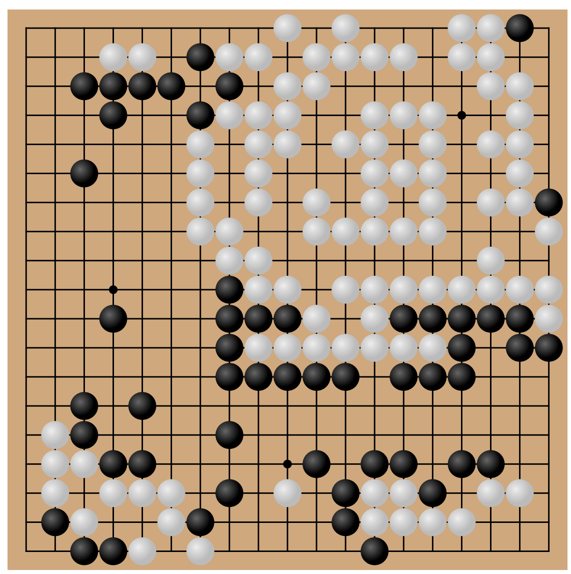
(a) Move 61: human has set up the inner group (top right middle) around which to lure the victim to create a cycle.



(b) Move 95: human next plays A, instead of the safer connecting move at B, to attempt to encircle victim's cyclic group.

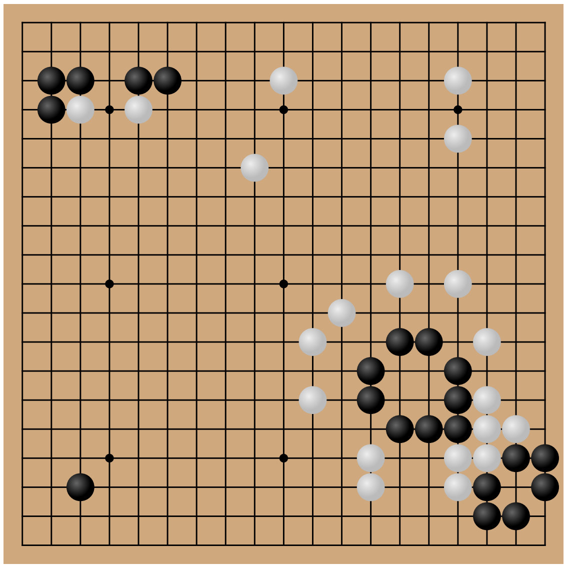


(c) Move 156: the encirclement is successful. Victim could survive by capturing one of the encircling groups, but will it?

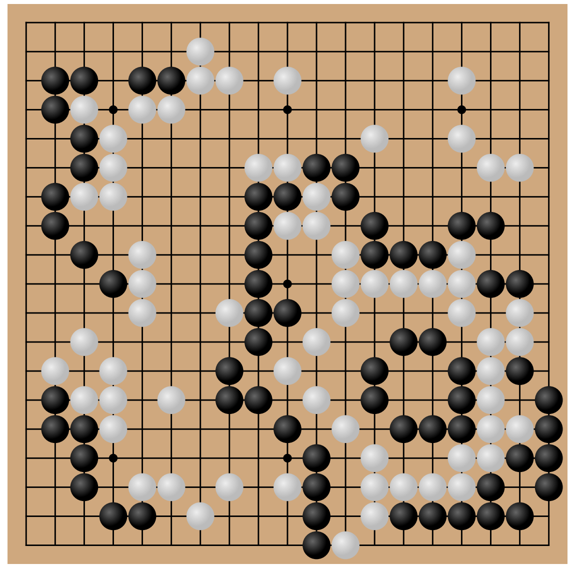


(d) Move 199: the victim failed to see the danger in time, was captured, and resigns.

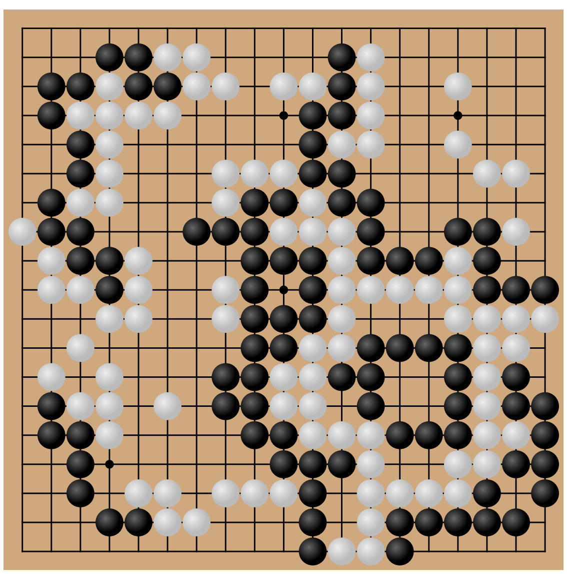
Figure 19: Human (white) beats Leela Zero with 100k visits (black).



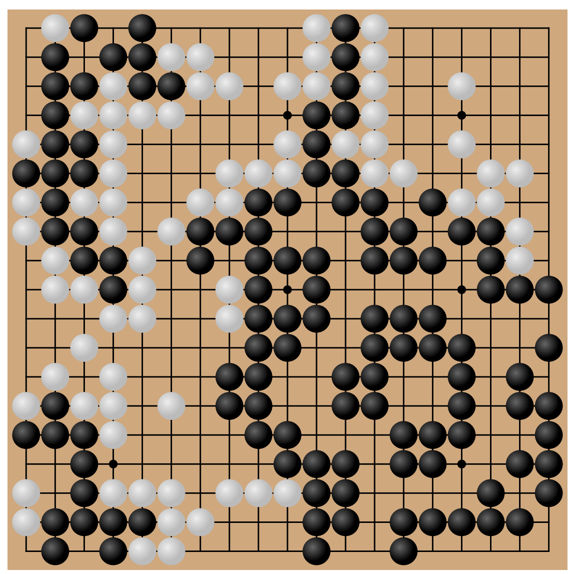
(a) Move 41: the frame of the cyclic group is set up (lower right middle)



(b) Move 133: human has completed loose encirclement of the victim's cyclic group.

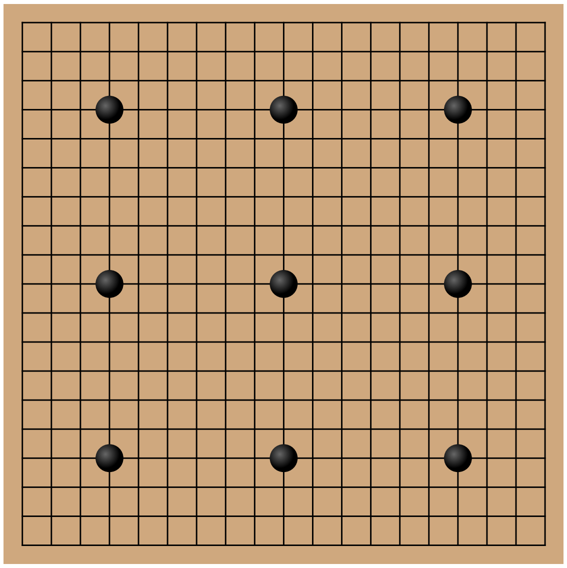


(c) Move 189: although victim's cyclic group has a number of liberties left, it can no longer avoid capture and the game is decided.

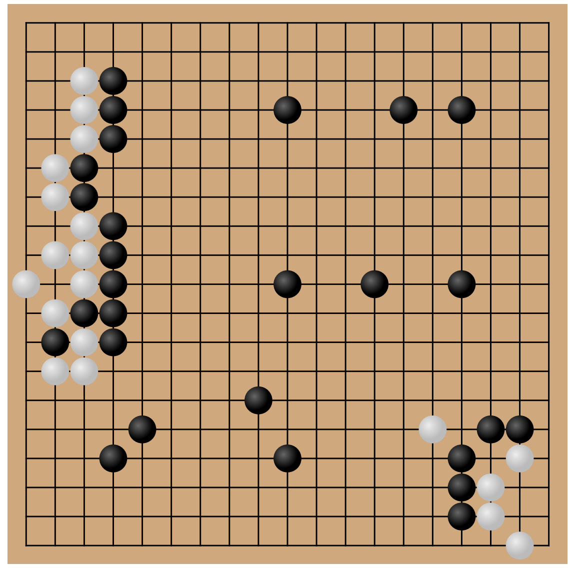


(d) Move 237: after nearly 40 more moves the cyclic group is captured. Victim realizes game is lost and resigns.

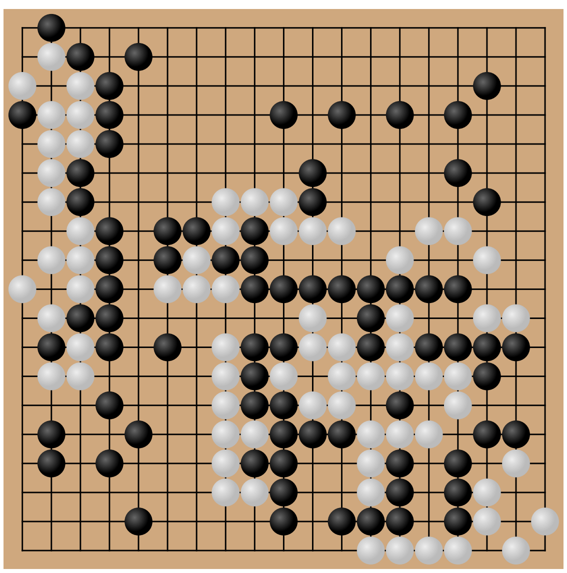
Figure 20: Human (black) beats JBXKata005 (white).



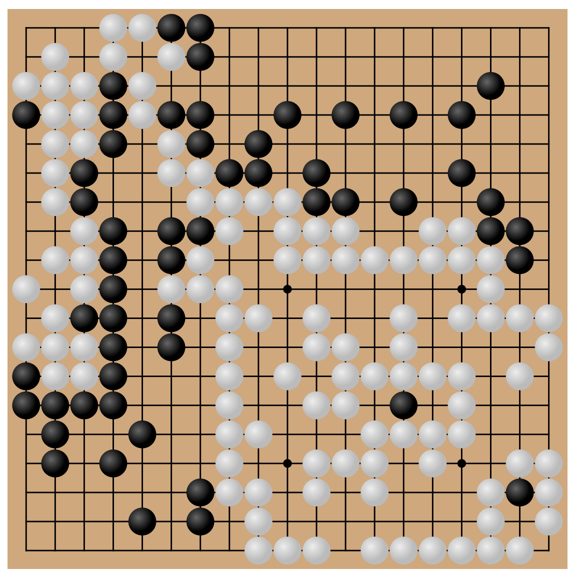
(a) Move 0: starting board position. In contrast to a normal game starting with an empty board, here the victim received 9 handicap stones, giving it an enormous initial advantage.



(b) Move 38: setting up the inside group is slightly more challenging here, since the victim has it surrounded from the start.



(c) Move 141: an encirclement is complete, but there are numerous defects. Victim could easily live inside or capture key stones.



(d) Move 227: victim fails to grasp any option to survive. Its group is captured and it resigns.

Figure 21: Human (white) beats JBXKata005 (black), giving it 9 handicap stones.

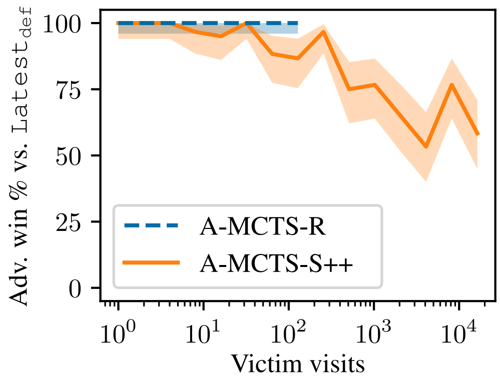


Figure 22: We evaluate the win rate of an older version of our cyclic-adversary playing with 200 visits / move against $\texttt{Latest}_{\texttt{def}}$ playing with varying amounts of search. The cyclic-adversary was trained for 498 million steps and had a curriculum whose victims only went up to 256 visits of search. Shaded regions and error bars denote 95% Clopper-Pearson confidence intervals over 60 games for A-MCTS-S++ and 90 games for A-MCTS-R. The adversary does better and wins all its games when performing A-MCTS-R, which models the victim perfectly.

Visits	**Game 0**	**Game 1**	**Game 2**	**Game 3**	**Game 4**	**Game 5**	**Game 6**	**Game 7**	**Game 8**	**Game 9**
1.6k	✗	✗	✗	✗	✗	✗	✗	✗	✗	✗
5k	✗	✓	?	✗	✗	✓	✗	✗	✗	✗
50k	✓	✓	?	?	✗	✓	✗	✓	✗	✓
100k	✓	✓	?	?	✓	✓	✗	?	✗	✓
250k	✓	✓	✓	?	✓	✓	✗	✓	✗	✓
500k	✓	✓	✓	?	✓	✓	✓	✓	✗	✓

Table 6: Examining how much search is needed to make the correct move in deciding positions. The original victim, which played the wrong move and consequently lost, used 1.6k visits of search. Higher visits leads to more correct moves in these positions, suggesting improved robustness.

approximation that A-MCTS-S and A-MCTS-S++ makes of the victim becomes less accurate the more search the victim has, making it harder to exploit for this algorithm regardless of its general change in robustness. (Indeed, we see in Figure 22 that A-MCTS-R, which perfectly models the victim, achieves a higher win rate than A-MCTS-S++.) Second, for a fixed adversary, the further the victim search diverges from the training, the more out-of-distribution the victim becomes. Third, it is possible that higher search improves winrate not through improved robustness or judgment but because it simply has less tendency to create cyclic positions. A person who hates mushrooms is less likely to eat a poisonous one, regardless of their judgment identifying them or towards risk in general.

In order to remove some of these confounders, we analyze board positions from games of the cyclic-adversary vs. victim which the victim lost. The cyclic-adversary examined here was trained for 498 million steps, making it an older version whose curriculum only went up to victims with 256 visits. The positions were selected manually by an author who is an expert Go player to be the last opportunity for the victim to win the game with subsequent best play by both sides. To facilitate accurate and informative analysis, they selected positions with relatively decisive correct and incorrect moves. This is necessary because many positions have numerous inconclusive moves that postpone resolution of the conflict until a later move (typically, a strong threat which requires an immediate answer, but does not change the overall situation). We vary the level of victim search from the original 1.6k visits, to 5k, 50k, 100k, 250k, and 500k and examine how much search is needed for the victim to rank a correct move as the best one. This corresponds roughly (ignoring stochasticity in the search and in the move choice which depends on the chosen temperature hyperparameter) to asking "how much search is needed for the victim to play correctly in this position?"

Results are shown in Table 6. "✓" indicates a correct move that should lead to victory was ranked #1, "✗" indicates a wrong move that should lead to defeat was #1, while "?" indicates an inconclusive move ranked #1.

We see that in 8 out of 10 positions, 500k visits leads to a winning move, and in many of the positions a winning move is found with substantially fewer visits. These search numbers are well within a feasible range. Although the sample size is limited due to the substantial manual analysis needed for each position, the results provide consistent evidence that adding a reasonable amount of search is indeed beneficial for robustness.

We show the board positions analyzed in Figures 23, 24, and 25. Moves are marked according to the preceding table, though note the markings for wrong and inconclusive moves are non-exhaustive. Full game records are available on our website.

We also investigated games played by the fully-trained adversary (i.e. the adversary whose curriculum goes up to 131k visits) against KataGo with 10 million visits. We find that when the adversary wins in this setting, the decisive move is played a greater number of moves before the cyclic group is captured than in the previous setting. This means that more victim search is needed to see the correct result. The adversary has likely learned to favor such positions during the additional training against higher search victims. There is also likely a selection bias, as the victim will likely win when the attack is less concealed, although as the adversary achieves a 76.7% win rate this effect cannot be substantial.

To test this impression quantitatively, we randomly sampled 25 games in which the adversary wins from each set of opponents. We resampled 1 outlier involving an abnormal, very complicated triple ko. For each game, we determined the last move from which the victim could have won. We then measure the number of moves from that move until the cyclic group is captured. For this measurement we consider the fastest possible sequence to capture, which might slightly differ from the actual game, because in the actual game the victim might resign before the final capture, or the adversary might not capture immediately if there is no way for the victim to escape. We include in the count any moves which postpone the capture that the victim actually played. This represents a middle ground: including all possible moves to postpone the capture could result in counting many moves that were irrelevant to the search (e.g. moves that require an answer but have no effect on the final result, which the victim realized without significantly affecting the search). On the other hand, removing all moves that postpone the capture might ignore moves that the victim thought were beneficial and had a significant effect on the search. Since the goal is to determine if there is a difference in how well hidden the attack is vis-a-vis the search, this middle ground is the most informative.

We find the lower search games had a mean moves-to-capture of 6.36 moves with a standard deviation of 2.87, while the higher search games had a mean of 8.36 with a standard deviation of 2.69. With a standard t-test for difference in means, this is significant at the 5% level ($p = 0.0143$). This also matches a qualitative assessment that the higher visit positions are more complex and have more potential moves, even if they are not part of the optimal sequence. Overall, this suggests that increased search leads to increased robustness, but that the adversary is able to partially combat this by setting up complex positions.

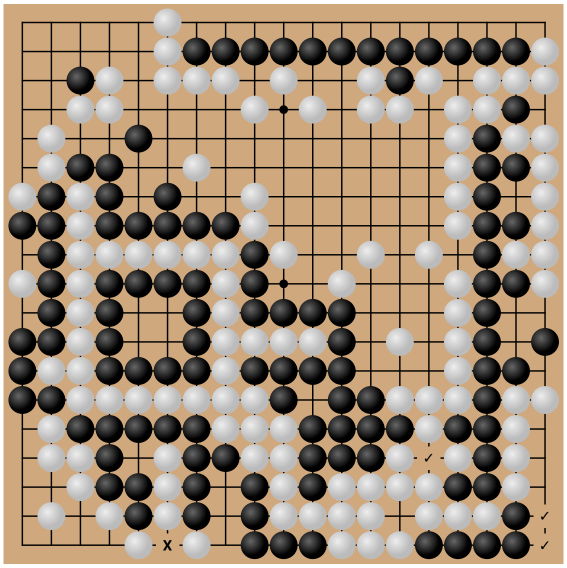
We observe that with lower search, there are 6 games which have a 3 move difference between the deciding move and the capture, while with higher search there are none less than 5. Is a 3 move trap too few to catch a high search victim? We examine these 6 positions (shown in Figures 26 and 27) further by varying the amount of search, as in the preceding experiment. Results are shown in Table 7. Similar to the positions examined previously, higher search typically leads to a correct move, although there is one exception where none of the visit levels tested fixed the victim's mistake. We tested this one position with 1 million, 2.5 million, and 10 million visits, and found that 1 million is still insufficient but 2.5 million and 10 million find the correct move. Therefore, it does seem these positions are not enough to fool a high search victim. Once again, this indicates overall that search does not give full robustness but still yields improvements.

I. Human experiments and analysis

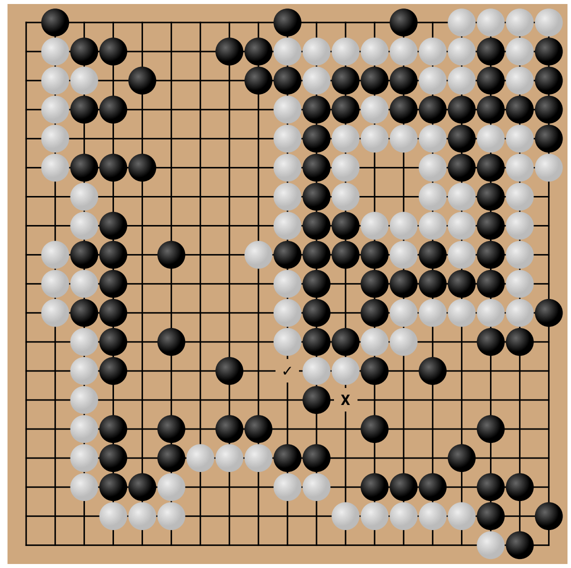
I.1. Humans vs. adversarial policies

An author who is a Go novice (strength weaker than 20kyu) played manual games against both the strongest cyclic-adversary from Figure 3 and the strongest pass-adversary from Figure 8. In the games against the pass-adversary, the author was able to achieve an overwhelming victory. In the game against the cyclic-adversary, the author won but with a much smaller margin. See Figure 28 for details.

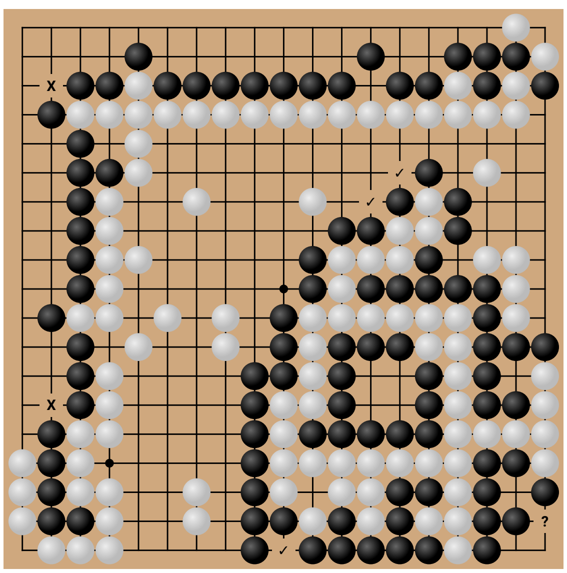
Our evaluation is imperfect in one significant way: the adversaries did not play with an accurate model of the author (rather they modeled the author as `Latest` with 1 visit). However, given the limited transferability of our adversaries to different KataGo checkpoints (see Figure 3, Figure 8, and Appendix F.4), we conjecture that our adversaries would not win even if



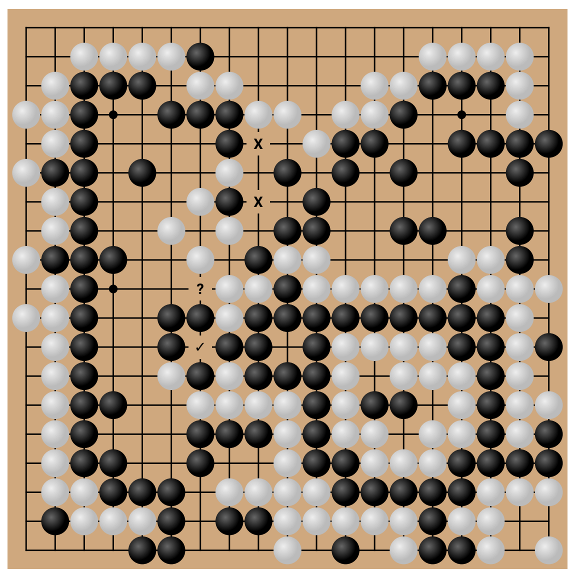
(a) White to play.



(b) Black to play.

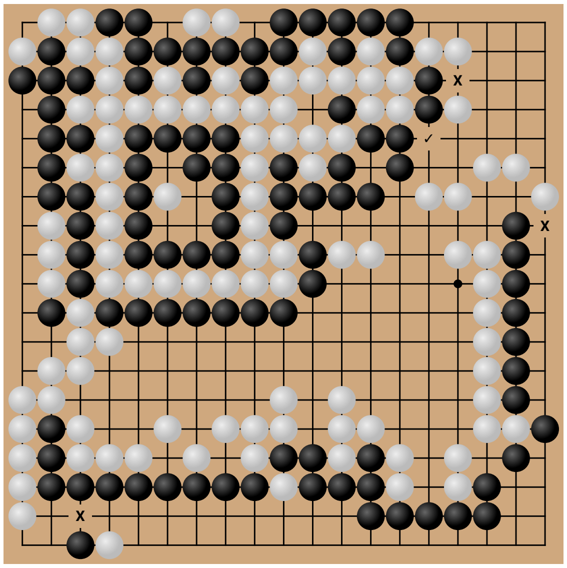


(c) White to play.

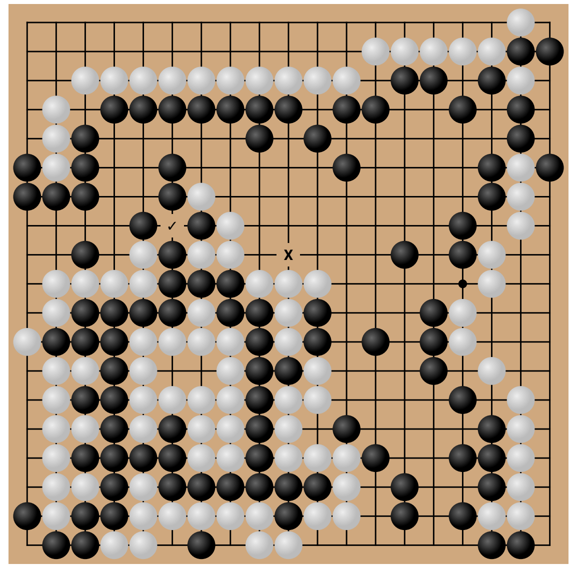


(d) Black to play.

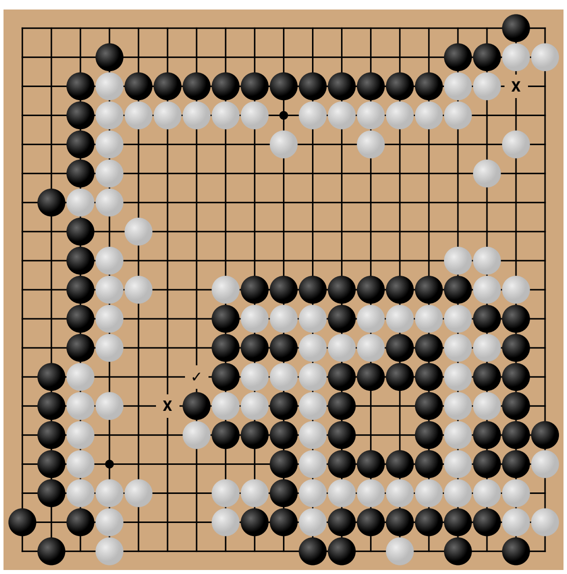
Figure 23: Part 1 of positions analyzed with varying levels of search. Correct moves are marked "✓", and non-exhaustive examples of incorrect and inconclusive moves that the victim likes to play are marked with "✗" and "?" respectively.



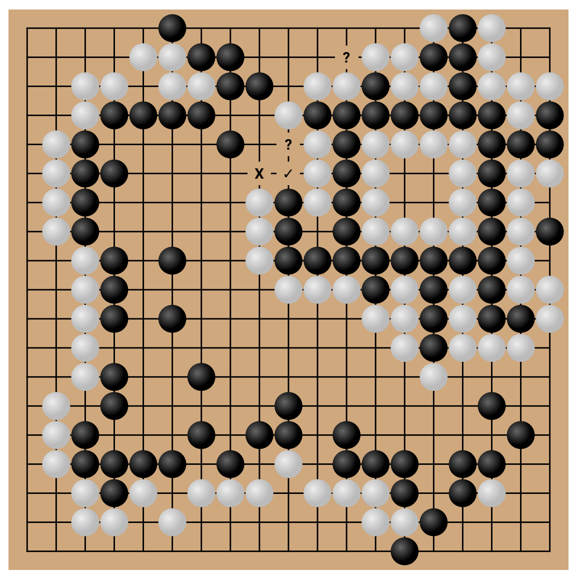
(a) White to play.



(b) Black to play.

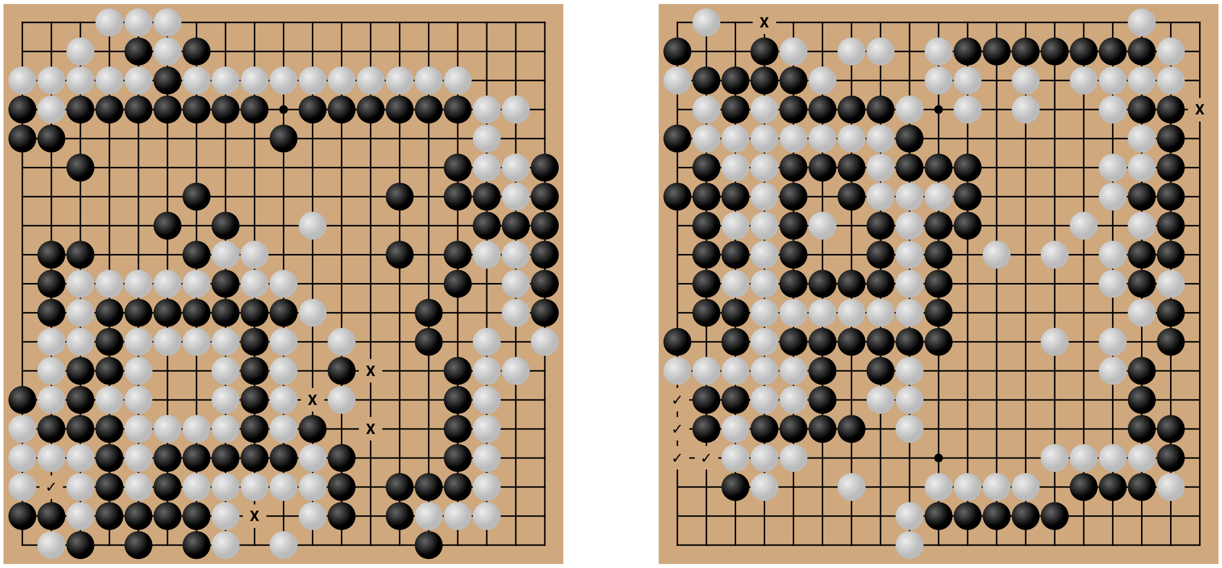


(c) White to play.



(d) Black to play.

Figure 24: Part 2 of positions analyzed with varying levels of search. Correct moves are marked “✓”, and non-exhaustive examples of incorrect and inconclusive moves that the victim likes to play are marked with “✗” and “?” respectively.



(a) Black to play. (b) White to play.

Figure 25: Part 3 of positions analyzed with varying levels of search. Correct moves are marked "✓", and non-exhaustive examples of incorrect and inconclusive moves that the victim likes to play are marked with "✗" and "?" respectively.

Visits	**Game 0**	**Game 1**	**Game 2**	**Game 3**	**Game 4**	**Game 5**
1.6k	✗	✗	✗	✗	✗	✗
5k	✗	✗	✗	✗	✗	✗
50k	✓	✓	✗	✗	✗	✗
100k	✓	✓	✗	✗	✗	✗
250k	✓	✓	✓	✗	✗	✓
500k	✓	✓	✓	✗	✓	✓

Table 7: Examining how much search is needed to make the correct move in positions with a 3 move difference between the deciding move and capture. Similar to the preceding table, the original victim had 1.6k visits. Higher visits again leads to more correct moves and improved robustness.

they had access to an accurate model of the author.

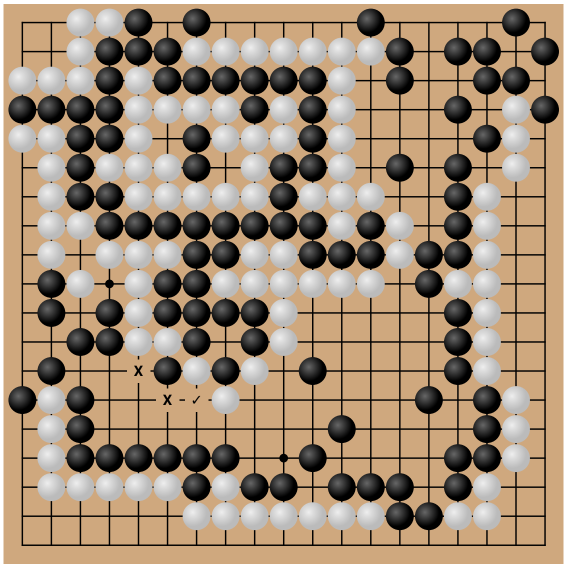
I.2. Human analysis of the cyclic-adversary

In the following we present human analysis of games with the cyclic-adversary (the type shown in Figure 1a) playing against $\texttt{Latest}_{\texttt{def}}$ with 1600 visits. This analysis was done by an expert-level Go player on our team. We first analyze in detail a game where the adversary won. We then summarize a sample of games where the adversary lost.

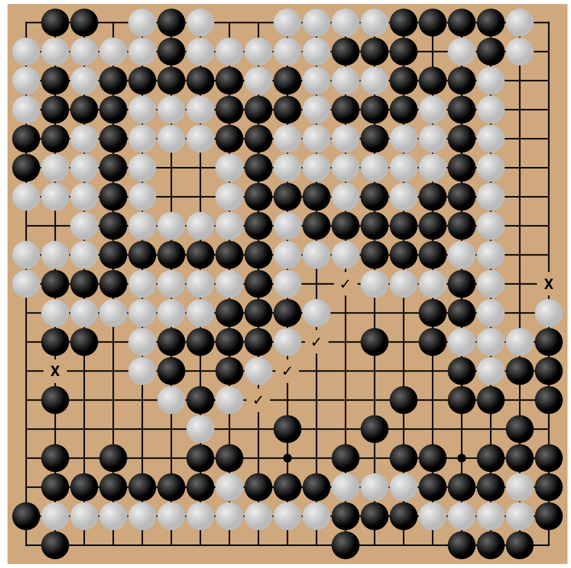
Adversary win analysis The game in Figure 29 shows typical behavior and outcomes with this adversary: the victim gains an early and soon seemingly insurmountable lead. The adversary sets a trap that would be easy for a human to see and avoid. But the victim is oblivious and collapses.

In this game the victim plays black and the adversary white. The full game is available on our website. We see in Figure 29a that the adversary plays non-standard, subpar moves right from the beginning. The victim's estimate of its win rate is over 90% before move 10, and a human in a high-level match would likewise hold a large advantage from this position.

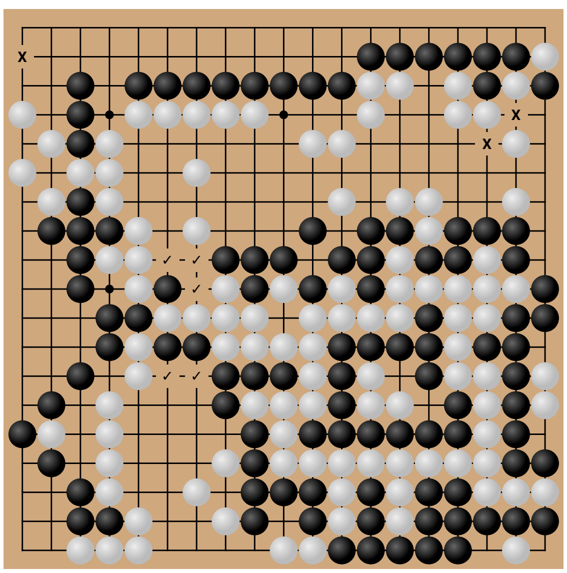
On move 20 (Figure 29b), the adversary initiates a tactic we see consistently, to produce a "dead" (at least, according to normal judgment) square 4 group in one quadrant of the board. Elsewhere, the adversary plays low, mostly second and third line moves. This is also common in its games, and leads to the victim turning the rest of the center into its sphere of



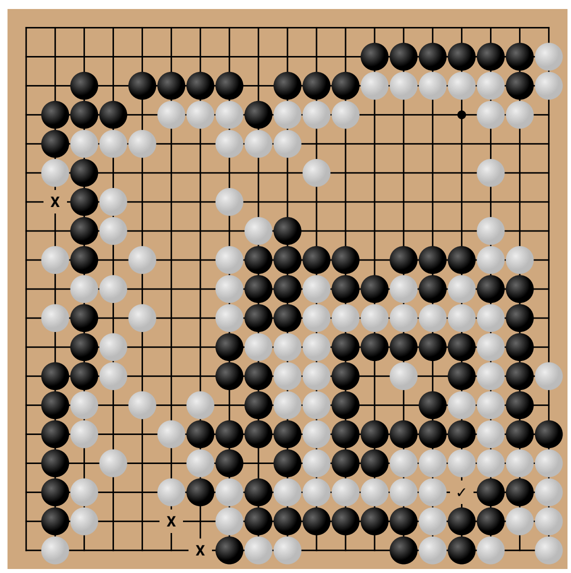
(a) Black to play.



(b) White to play.

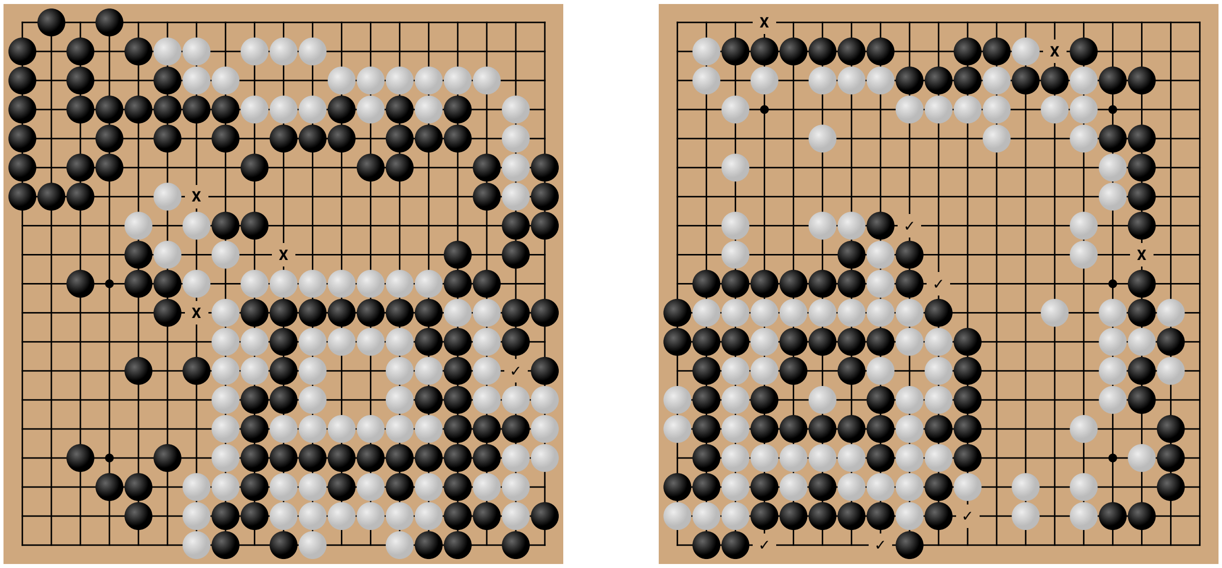


(c) White to play.



(d) White to play.

Figure 26: Positions with a 3 move difference between deciding move and capture, analyzed with varying levels of search. Correct moves are marked "✓", and non-exhaustive examples of incorrect and inconclusive moves that the victim likes to play are marked with "✗" and "?" respectively.



(a) Black to play. (b) White to play.

Figure 27: Part 2 of positions with a 3 move difference between deciding move and capture, analyzed with varying levels of search. Correct moves are marked "✓", and non-exhaustive examples of incorrect and inconclusive moves that the victim likes to play are marked with "✗" and "?" respectively.

influence. We suspect this helps the adversary later play moves in that area without the victim responding directly, because the victim is already strong in that area and feels confident ignoring a number of moves.

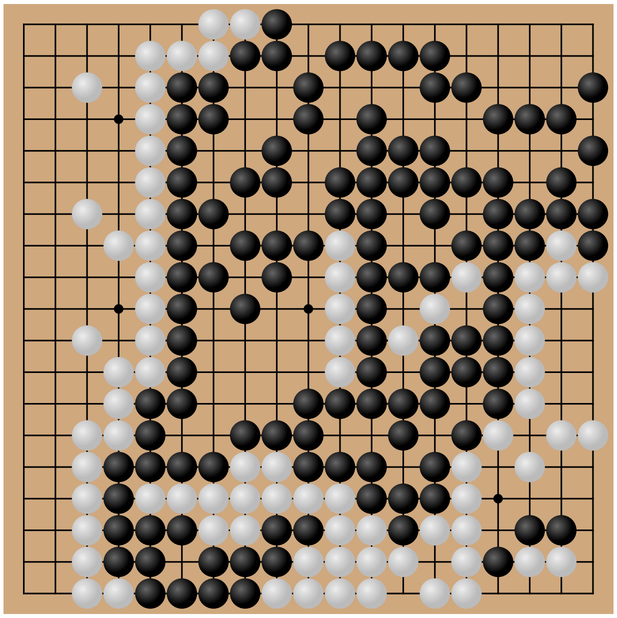
On move 74 (Figure 29c), the adversary begins mobilizing its "dead" stones to set up an encirclement. Over the next 100+ moves, it gradually surrounds the victim in the top left. A key pattern here is that it leads the victim into forming an isolated group that loops around and connects to itself (a group with a cycle instead of tree structure). David Wu, creator of KataGo (Wu, 2019), suggested Go-playing agents like the victim struggle to accurately judge the status of such groups, but they are normally very rare. This adversary seems to produce them consistently.

Until the adversary plays move 189 (Figure 29d), the victim could still save that cycle group (marked with X), and in turn still win by a huge margin. There are straightforward moves to do so that would be trivial to find for any human playing at the victim's normal level. Even a human who has only played for a few months or less might find them. For instance, on 189 it could have instead played at the place marked "A." But after 189, it is impossible to escape, and the game is reversed. The victim seems to have been unable to detect the danger. Play continues for another 109 moves but there is no chance for the victim (nor would there be for a human player) to get out of the massive deficit it was tricked into.

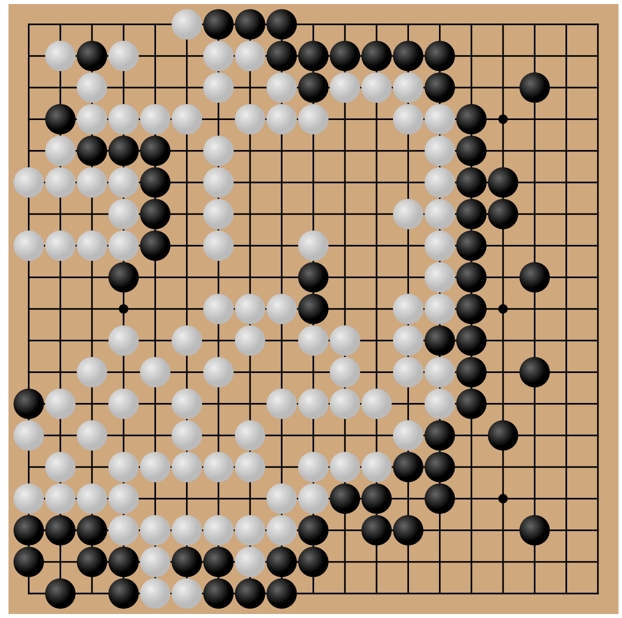
Adversary loss analysis In all cases examined where the adversary lost, it did set up a cycle group, or a cycle group with one stone missing, which is likely still a cycle as perceived by the neural net of the victim (see Figure 29d for an example where it misjudges such a position).

In four out of ten cases, the adversary could either immediately capture the cycle group or could capture it on its next turn if it played correctly. An example is shown in Figure 30. But instead it allowed the victim to save the group and win the game. We found this is due in some situations to imperfect modeling of the victim, i.e., modeling a victim without search in A-MCTS-S++ even though the true victim has search. This can lead to the adversary thinking the victim will not defend, and therefore there is no need to capture immediately, while in reality the victim is about to defend and the opportunity will disappear. In such cases, A-MCTS-R leads to the correct move. Besides this search limitation, other contributing factors potentially include the adversary itself not being completely immune to misjudging cycle groups, or the adversary's skill at Go in general being too low, resulting in many mistakes.

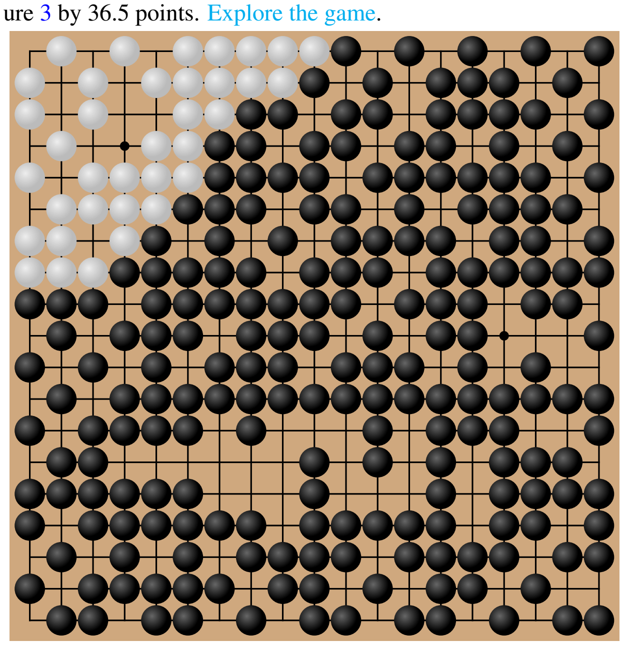
In the other six cases the adversary never has any clear opportunity to capture the cycle group. This is because the victim breaks through the attempted encirclement in some fashion, either by capturing some surrounding stones or simply



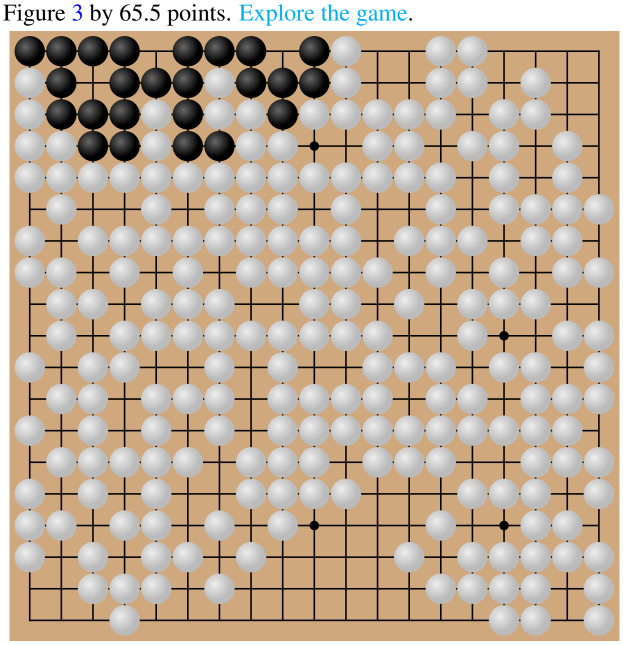
(a) An author (B) defeats the strongest cyclic-adversary from Figure 3 by 36.5 points. Explore the game.



(b) An author (W) defeats the strongest cyclic-adversary from Figure 3 by 65.5 points. Explore the game.

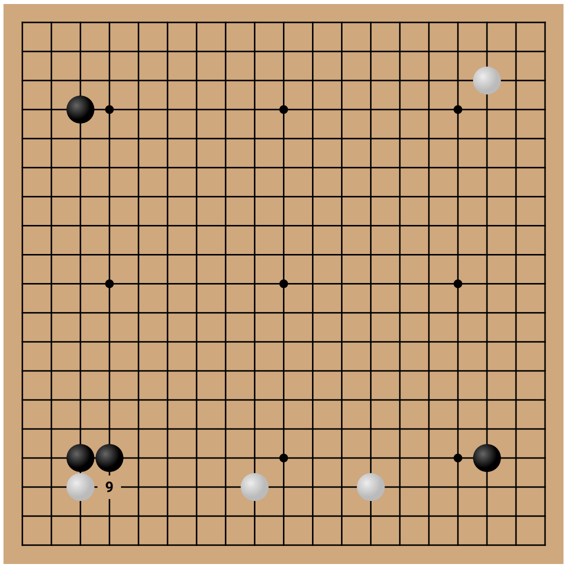


(c) An author (B) defeats the strongest pass-adversary from Figure 8. Explore the game.

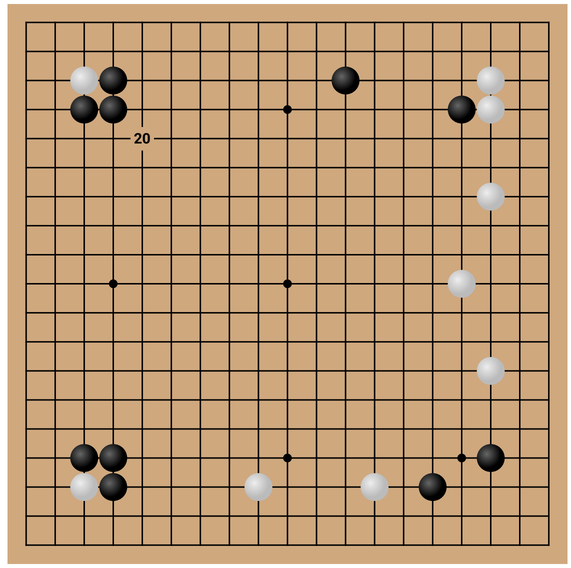


(d) An author (W) defeats the strongest pass-adversary from Figure 8 using A-MCTS-S++. Explore the game.

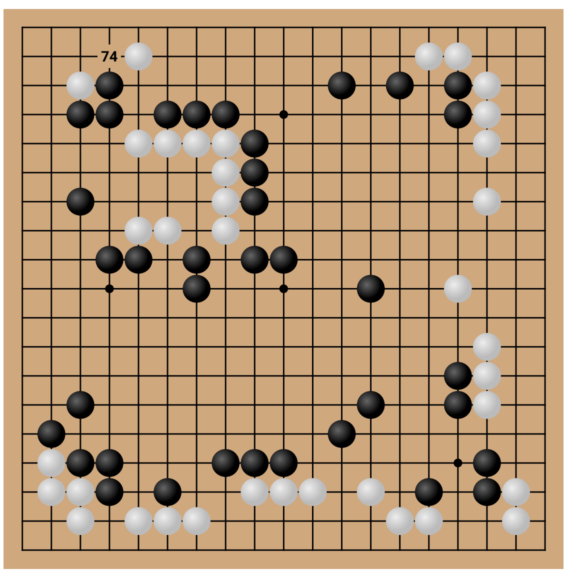
Figure 28: Games between an author of this paper (who is a Go amateur) and the strongest adversaries from Figure 3 and Figure 8. In all games, the author achieves a victory. The adversaries used 600 playouts / move and used `Latest` as the model of its human opponent. The adversaries used A-MCTS-S for all games except the one marked otherwise.



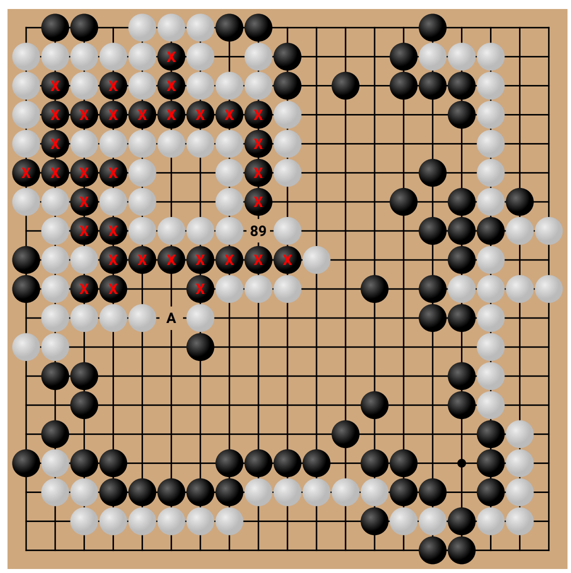
(a) Move 9: after this move victim already has the advantage, if it were robust.



(b) Move 20: adversary initiates a key tactic to create a cycle group.



(c) Move 74: adversary slowly begins to surround victim.



(d) Move 189 (89): victim could have saved X group by playing at "A" instead, but now it will be captured.

Figure 29: The cyclic-adversary (white) exploiting a KataGo victim (black) by capturing a large group that a human could easily save. The subfigures show different moves in the game. Explore the full game.

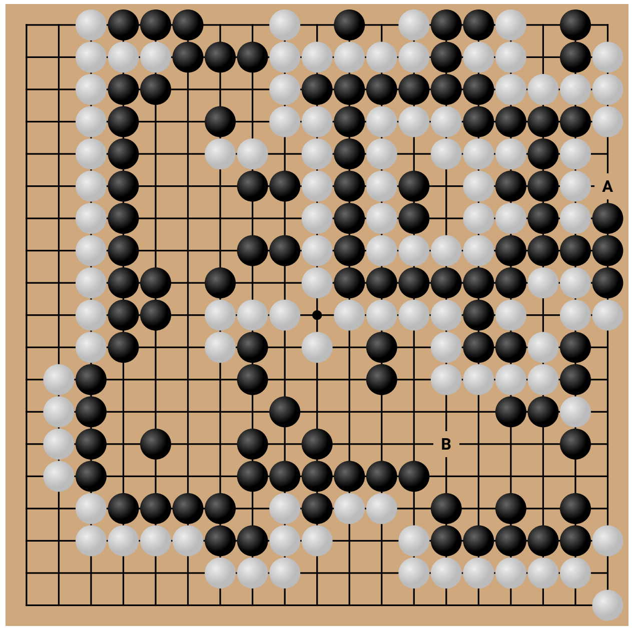


Figure 30: A game the cyclic-adversary (white) lost. The adversary could take a decisive lead by capturing at A, but instead plays B and lets the victim (black) save their group.

connecting to one of its other groups. Although this could indicate the victim recognized the danger to the cycle group, the moves are typically also consistent with generic plays to wrap up the game with the large lead that it has established.

J. Adversarial board state

This paper focuses on training an *agent* that can exploit Go-playing AI systems. A related problem is to find an adversarial *board state* which could be easily won by a human, but which Go-playing AI systems will lose from. In many ways this is a simpler problem, as an adversarial board state need not be a state that the victim agent would allow us to reach in normal play. Nonetheless, adversarial board states can be a useful tool to probe the blind spots that Go AI systems may have.

In Figure 31 we present a manually constructed adversarial board state. Although quite unlike what would occur in a real game, it represents an interesting if trivial (for a human) problem. The black player can always win by executing a simple strategy. If white plays in between two of black's disconnected groups, then black should immediately respond by connecting those groups together. Otherwise, the black player can connect any two of its other disconnected groups together. Whatever the white player does, this strategy ensures that blacks' groups will eventually all be connected together. At this point, black has surrounded the large white group on the right and can capture it, gaining substantial territory and winning.

Although this problem is simple for human players to solve, it proves quite challenging for otherwise sophisticated Go AI systems such as KataGo. In fact, KataGo playing against a copy of itself *loses* as black 40% of the time. We conjecture this is because black's winning strategy, although simple, must be executed flawlessly and over a long horizon. Black will lose if at any point it fails to respond to white's challenge, allowing white to fill in both empty spaces between black's groups. This problem is analogous to the classical cliff walking reinforcement learning task (Sutton & Barto, 2018, Example 6.6).

K. Known failures of Go-playing agents

The following draws largely on discussion with David Wu, creator of KataGo.

Ladders A "ladder" is often the first tactic a beginner learns. An example is shown in Figure 32. In this pattern, the defending side only has a single move to attempt to escape, while the attacking side only has a single move to continue threatening the defender. After each move in the pattern, the same situation recurs, shifted one space over diagonally. The

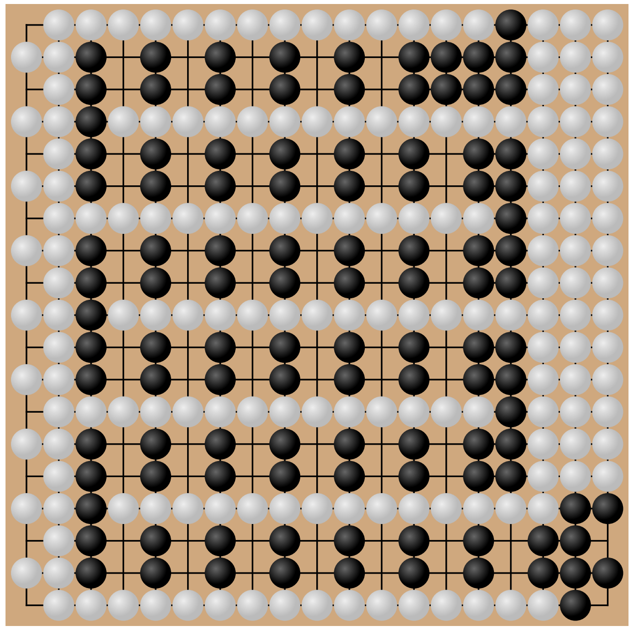


Figure 31: A hand-crafted adversarial example for KataGo and other Go-playing AI systems. It is black's turn to move. Black can guarantee a win by connecting its currently disconnected columns together and then capturing the large white group on the right. However, KataGo playing against itself from this position loses 40% of the time as black.

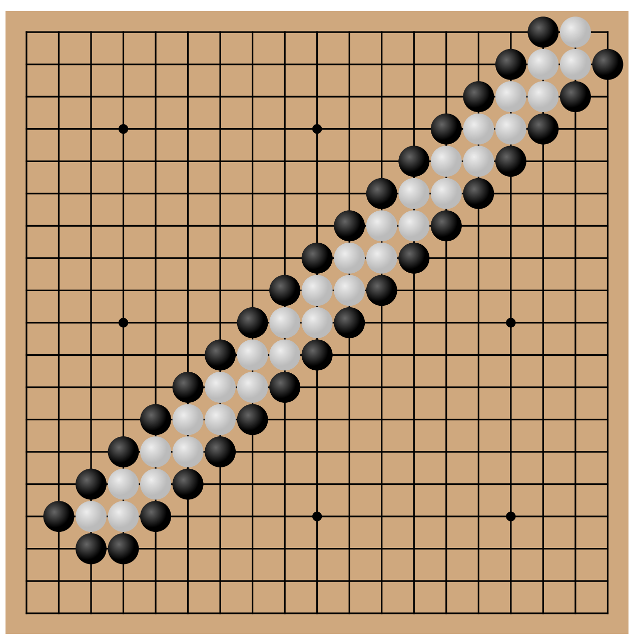


Figure 32: Illustration of a ladder. White ran across the board, but has hit the top edge and has nowhere left to run. Black can capture on the next move by playing in the top right corner.

chain continues until either the defender runs into the edge of the board or more enemy stones, at which time there is no more room to escape, or conversely into the defender's allied stones, in which case the defender escapes and the attacker is usually left disastrously overextended. Consequently, it is a virtually unbranching pattern, but one that takes many moves (often dozens), depends on the position all the way on the other side of the board, and can decide the result of the game.

Bots struggle to understand when escape and capture is possible, especially fairly early in the game. This issue occurs across many different models. It is especially prevalent early in training and with less search, but even with thousands of playouts per move it still occurs.

This issue has been solved in KataGo by adding a separate, hardcoded ladder module as an input feature. Such an approach, however, would not work for flaws one is unaware of, or where hardcoded solutions are prohibitively difficult.

Liberty Counts Even without a long variation or consistent pattern, bots may occasionally fail to see that something can be captured on their move or their opponent's next move. Known examples of this occurred with very large groups in slightly unusual situations, but nonetheless where an intermediate human would easily make the correct judgment.

This is again mitigated in KataGo through a hardcoded auxiliary algorithm that provides input features (liberty counts) to the main network.

Complicated Openings There are some extremely complicated opening variations, especially variations of Mi Yuting's Flying Dagger joseki, which have crucial, unusual moves required to avoid a disadvantage. Once again, KataGo solved this with a manual intervention. Here it was through directly adding a large collection of variations to the training. Other bots still play certain variations poorly.

Cyclic Topology This is a position with a loop, such as the marked group in Figure 1a. It is possible but very uncommon in normal play. David Wu's hypothesis is that information propagates through the neural network in a way analagous to going around the cycle, but it cannot tell when it has reached a point it has "seen" before. This leads to it counting each liberty multiple times and judging such groups to be very safe regardless of the actual situation.

Mirror Go This is where one player copies the other player's moves, mirroring them across the board diagonally. This is typically not part of training nor other aspects of agents' construction. However, even without specific counter strategies, there is a long time over the course of the game to stumble into a position where a generically good move also breaks the mirror. So this strategy is not a consistent weakness, but can occasionally win games if no such good mirror-breaking move happens to come up.

Other Finally, there are also other mistakes bots make that are more complex and more difficult to categorize. Even though the best bots are superhuman, they are certainly still a ways away from perfect play, and it is not uncommon for them to make mistakes. In some positions these mistakes can be substantial, but fixing them may be not so much about improving robustness as it is about building an overall stronger agent.

Summary There are a number of situations that are known to be challenging for computer Go players. Some can be countered through targeted modifications and additions to the model architecture or training, however, as we see with Cyclic Topology, it is difficult to design and implement solutions one-by-one to fix every possibility. Further, the weaknesses may be unknown or not clearly understood – for instance, Cyclic Topology is normally rare, but through our work we now know it can be produced consistently. Thus, it is critical to develop algorithmic approaches for detecting weaknesses, and eventually for fixing them.