Adversarial Policies Beat Superhuman Go AIs

Tony T Wang [*1] **Adam Gleave** [*2 3] **Tom Tseng** [3] **Kellin Pelrine** [3 4] **Nora Belrose** [3] **Joseph Miller** [3]
Michael D Dennis [2] **Yawen Duan** [2] **Viktor Pogrebniak** **Sergey Levine** [2] **Stuart Russell** [2]

Abstract

We attack the state-of-the-art Go-playing AI system KataGo by training adversarial policies against it, achieving a >97% win rate against KataGo running at superhuman settings. Our adversaries do not win by playing Go well. Instead, they trick KataGo into making serious blunders. Our attack transfers zero-shot to other superhuman Go-playing AIs, and is comprehensible to the extent that human experts can implement it without algorithmic assistance to consistently beat superhuman AIs. The core vulnerability uncovered by our attack persists even in KataGo agents adversarially trained to defend against our attack. Our results demonstrate that even superhuman AI systems may harbor surprising failure modes. Example games are available at goattack.far.ai.

1. Introduction

The average-case performance of AI systems has grown rapidly in recent years, from RL agents achieving superhuman performance in competitive games (Silver et al., 2016; 2018; OpenAI et al., 2019) to generative models showing signs of general intelligence (OpenAI, 2023; Bubeck et al., 2023). However, designing AI systems with good *worst-case* performance remains an open problem. One key question is whether average-case performance gains can be translated into worst-case robustness. If so, then efforts to increase average-case performance such as through scaling models would naturally lead to robustness. We find that even superhuman systems can fail catastrophically, suggesting that capabilities are not enough: a dedicated effort will be needed to make systems robust.

In particular, we find vulnerabilities in KataGo (Wu, 2019), the strongest publicly available Go-playing AI system. We find these vulnerabilities by training adversarial policies to beat KataGo. Using less than 14% 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 playing Go well.[1] Instead, they trick KataGo into making serious blunders that cause it to lose the game (Figure 1.1).

Our adversaries transfer zero-shot to other superhuman Go-playing AIs, and the strategy they use can be replicated by human experts to consistently beat many different superhuman AIs (Appendix O). Moreover, the core vulnerability uncovered by our attack persists even in KataGo agents adversarially trained to defend against our attack, suggesting that the vulnerability is non-trivial to patch.

We chose to attack KataGo as we expected it to be unusually challenging to exploit, such that a successful attack suggests that a broad swathe of other systems will be vulnerable. In particular, KataGo's capabilities are superhuman by a large margin, whereas the state-of-the-art in broader domains like language modeling are still subhuman at many tasks. Moreover, Go is naturally an adversarial setting, such that average-case performance should be predictive of worst-case performance.

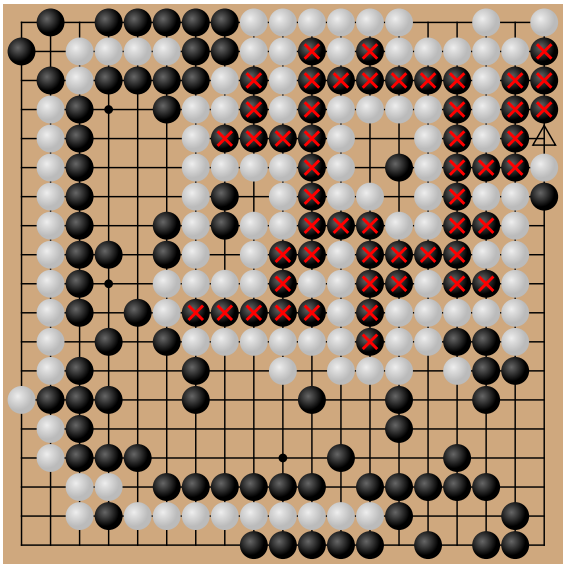
Most prior work on robustness has focused on ML systems in isolation. However, techniques such as simulation of alternatives at inference time (Yao et al., 2023) and self-reflection (Bai et al., 2022) can improve system robustness. KataGo performs substantial simulation and self-reflection in the form of Monte-Carlo Tree Search (Coulom, 2007), but our attack still wins more often than not even when KataGo searches 10 million nodes per move.

Our adversaries have no special powers: they can only place stones or pass, like a regular player. We do however give our adversaries gray-box access to the victim network they are attacking (Section 3.1). In particular, we train our adversaries using an AlphaZero-style training process (Silver et al., 2018), similar to that of KataGo. The key differ-

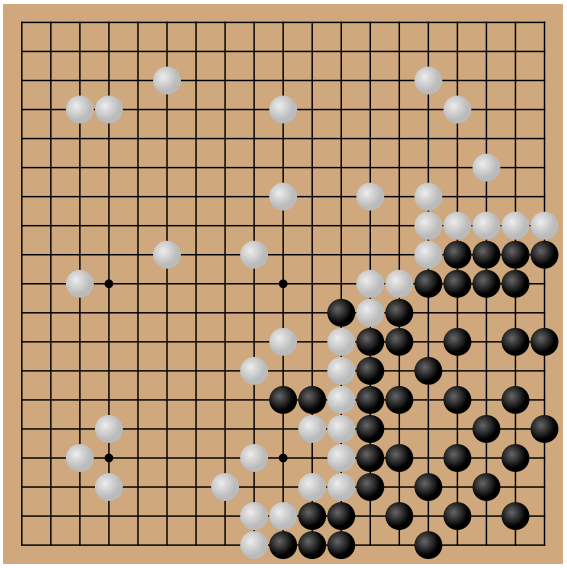
[*]Equal contribution [1]MIT [2]UC Berkeley [3]FAR AI [4]McGill University; Mila. Correspondence to: Tony T Wang <twang6@mit.edu>, Adam Gleave <adam@far.ai>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

[1]Despite being able to beat KataGo, our adversarial policies lose against even amateur Go players (Appendix J.1).



(a) Our *cyclic-adversary* wins as white by capturing a cyclic group (×) that the victim (`Latest`, 10 million 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 1.1. Games between the strongest KataGo network at the time of conducting this research (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 (×) get captured by the adversary's next move (△). Appendix J.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.

ences 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.

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). 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. Our open-source implementation is available at GitHub.

2. Related Work

Our work is inspired by the presence of adversarial examples in a wide variety of models (Szegedy et al., 2014). Notably, though not consistently superhuman (Shankar et al., 2020), many image classifiers reach and sometimes surpass human performance in a number of contexts (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 a somewhat 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 fact, 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 A beats B and B beats C yet C beats A. However, Czarnecki et al. (2020) argues real-world games like Go grow increasingly transitive as skill increases. This would imply that while self-play may struggle with non-transitivity early in training, comparisons involving highly capable policies such as KataGo should be mostly transitive. But we find significant non-transitivity: our adversaries exploit KataGo agents that beat human professionals, yet lose to most amateur Go players (Appendix J.1).

Most prior work attacking deep RL has focused on perturbing observations (Huang et al., 2017; Ilahi et al., 2022).

Figure 2.1. A human amateur beats our adversarial policy (Appendix J.1) that beats KataGo. This non-transitivity shows the adversary is not a generally capable policy, and is just exploiting KataGo.

Concurrent work by Lan et al. (2022) shows that KataGo with ≤ 50 visits can be induced to play poorly by adding 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 unrealistic, 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 estimate exploitability. Whereas we exploit an open-source system KataGo, they exploit a proprietary replica of AlphaZero from Schmid et al. (2021). Both Timbers et al. and our attacks use AlphaZero-style training modified 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 patching KataGo to protect against our initial *pass-adversary*, then repeating the attack.

Our main contribution lies in our experimental results. Timbers et al. 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. But we show our attack beats victims playing with an unquestionably superhuman 10^7 visits. Furthermore, our experiments give an in-depth investigation of this vulnerability, and include insights on defense, transfer, both human and mechanistic interpretability, the role of search for both victim and adversary, the evolution of the attack over training, and more.

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 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. 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 result, 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 2.1, 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 15,000 V100-equivalent GPU days (Appendix D.2). 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 its policy head to mimic the outcome of this tree search, and its value head to predict whether the agent wins the self-play game. Each step of training is designed to act as a policy-improvement operator.

In contrast to AlphaZero, KataGo has several additional heads that predict auxiliary targets such as the opponent's next move and which player "owns" a square on the board. The outputs of these heads 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, training process improvements such as playout cap randomization, and hand-engineered input features such as a ladderable stones mask.

These modifications to KataGo improve its sample and compute efficiency by several orders of magnitude over prior work such as ELF OpenGo, and protect KataGo from some previously known vulnerabilities in neural-net-based Go AIs (Appendix N). For these reasons, we choose to build our attack on top of KataGo, adopting its various architecture and training improvements and hand-engineered features. In principle though, the same attack could be implemented on top of any AlphaZero-style training pipeline.

4. Attack Methodology

Prior works, such as KataGo and AlphaZero, train on self-play games where an 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*.

Adversarial MCTS. In regular self-play, an agent models its opponent's moves by sampling from its own policy network. This makes sense, as in this case the policy *is* playing itself. But in victim-play, it would be a mistake to model the victim's behavior using the adversary's policy network. We introduce *Adversarial MCTS* (A-MCTS) to address this problem (Figure 4.1).

We experiment with three variants of A-MCTS. The *sample* variant (A-MCTS-S) models a computationally bounded version of the victim that plays moves directly from its policy head. A-MCTS-S++ improves upon this by averaging the victim policy head over board symmetries to match the default behavior of KataGo. Finally, the *recursive* variant (A-MCTS-R) models the victim perfectly at the cost of increased computational complexity; the cost of adversary training and inference is increased by a factor equal to the victim's search budget. We use A-MCTS-R to study the benefits of using a more accurate model of the victim.

Initialization. We randomly initialize the adversary's network. We cannot initialize the adversary's weights to those of the victim as our threat model does not allow white-box access. A random initialization also 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

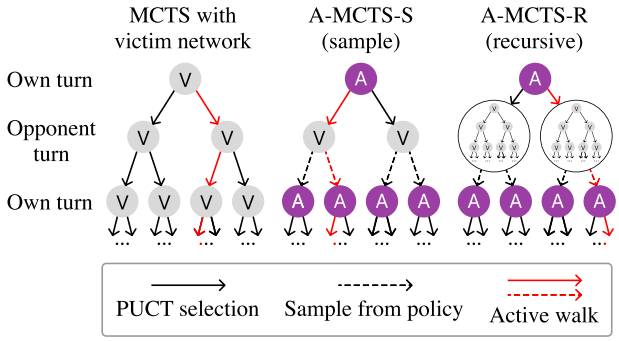


Figure 4.1. MCTS (left) builds a search tree one node at a time. To add a node, it walks down the tree until a new leaf is reached (red arrows). At a node x, the next step of the walk is determined by a PUCT (Rosin, 2011) algorithm (solid arrows) which takes into account neural network evaluations of each node in the subtree of x. A-MCTS-S (middle) walks down the tree by using a modified PUCT algorithm at adversary nodes, and sampling directly from the victim's policy network (dashed arrows) at victim nodes. A-MCTS-R (right) performs a full simulation of the victim as opposed to sampling from the victim's policy net. Search trees are depicted as binary for illustrative purposes only. See Appendix B for full details.

learn something useful about the game even from games that are lost, due to KataGo's auxiliary loss functions.

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 against KataGo (Wu, 2019), focusing on the `b40c256-s11840935168` network, which was the strongest KataGo network at the time of our main experiments, and which we refer to as `Latest`. In Section 5.1 we use A-MCTS-S with 600 adversary visits to train our *pass-adversary*, achieving a 99.9% win rate against `Latest` playing without search. Even without search `Latest` is comparable to a top-100 European player (Appendix E.1). The pass-adversary beats `Latest` by tricking it into passing early and losing (Figure 1.1b).

In Section 5.2, we add a *pass-alive defense* to the victim to defend against the pass-adversary. The defended victim $\texttt{Latest}_{\texttt{def}}$ cannot lose via accidentally passing, and is about as strong as `Latest` (it beats `Latest` 456/1000 games when both agents use no tree search, and 461/1000 games when both use 2048 visits/move of search).

Repeating the A-MCTS-S attack against $\texttt{Latest}_{\texttt{def}}$ yields our *cyclic-adversary*,[2] which is qualitatively very different from the pass-adversary as it does not use the pass-trick (Figure 1.1a) 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 (detailed in Section 5.3), achieving a 95.7% win rate against $\texttt{Latest}_{\texttt{def}}$ with 4096 visits and a 72% win rate against `Latest` with 10^7 visits.[3] 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 them.

In the remaining results sections, we provide a deeper understanding of the cyclic adversary and vulnerability, looking at defense (Section 5.4), how the attack works and the victim fails (Section 5.5), and transfer (Section 5.6).

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 for just 20.4 V100 GPU days, which is 0.13% 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 1.1b that loses even against human amateurs (see Appendix J.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 (see 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: prohibiting passing until it cannot change the game outcome. Concretely, we only al-

[2]Unless otherwise specified, the "cyclic-adversary" refers to the strongest checkpoint indicated in Figure 5.1. Likewise "pass-adversary" refers to the strongest checkpoint in Figure F.1.

[3]In the latter case, we manually verified it is not winning any games with the pass-trick.

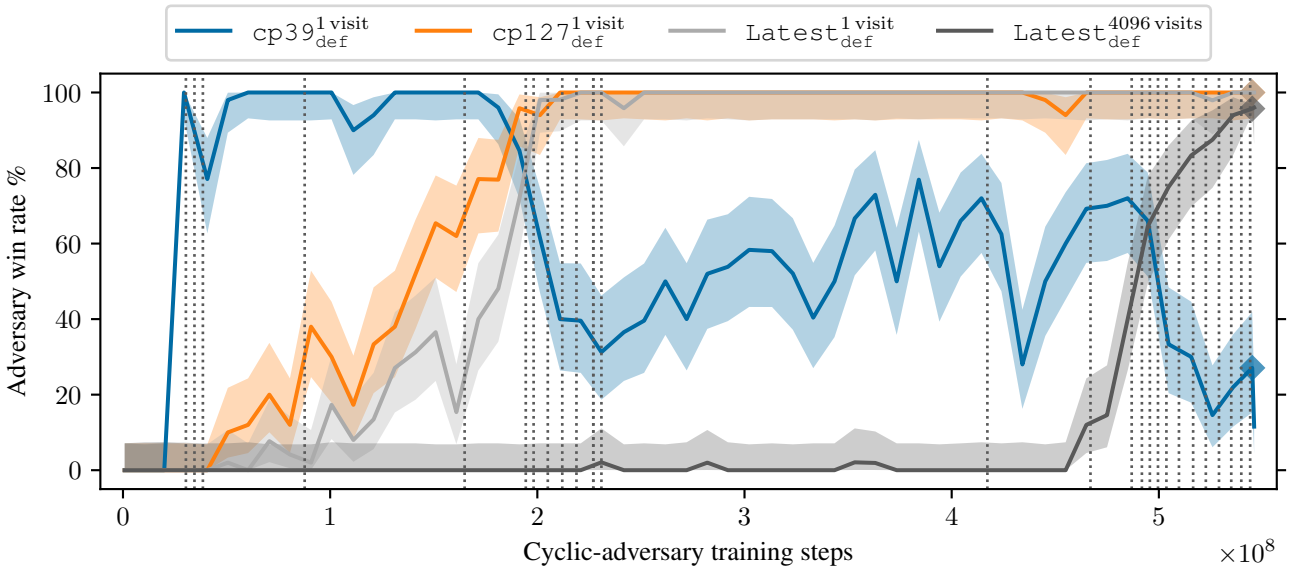


Figure 5.1. The win rate (y-axis) of the cyclic-adversary over time (x-axis) playing with 600 visits against four different victims. The strongest cyclic-adversary checkpoint (marked ♦) wins 1048/1048 = 100% games against $\texttt{Latest}_{\texttt{def}}$ without search and 1007/1052 = 95.7% games against $\texttt{Latest}_{\texttt{def}}$ 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 $\texttt{cp39}_{\texttt{def}}$ without search and ending at $\texttt{Latest}_{\texttt{def}}$ with 131,072 visits. Vertical dotted lines denote switches to stronger victim networks or to an increase in $\texttt{Latest}_{\texttt{def}}$'s search budget.

low the victim to pass when all its legal moves are in its own *pass-alive territory*, a concept described in the official KataGo rules (Wu, 2021b) that extends the traditional Go notion of a pass-alive group (see Appendix B.6 for full defense details). Given a victim $\texttt{V}$, we denote the victim with this defense applied $\texttt{V}_{\texttt{def}}$. 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 then starts increasing the number of victim visits.

In Figure 5.1 we evaluate various cyclic-adversary checkpoints against the policy networks of $\texttt{cp39}_{\texttt{def}}$, $\texttt{cp127}_{\texttt{def}}$, and $\texttt{Latest}_{\texttt{def}}$. We see that an attack that works against $\texttt{Latest}_{\texttt{def}}$ transfers well to $\texttt{cp127}_{\texttt{def}}$ but not to $\texttt{cp39}_{\texttt{def}}$, and an attack against $\texttt{cp39}_{\texttt{def}}$ early in training did not transfer well to $\texttt{cp127}_{\texttt{def}}$ or $\texttt{Latest}_{\texttt{def}}$. These results suggest that different checkpoints have unique vulnerabilities. We analyze the evolution of our cyclic-adversary's strategy in Appendix J.3.

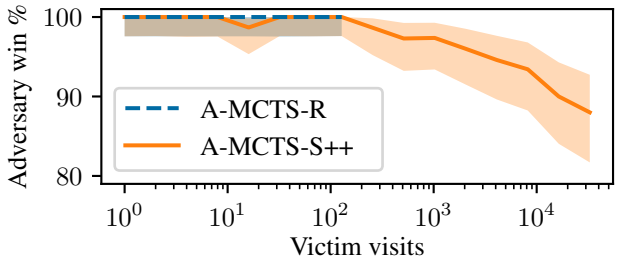
Our best cyclic-adversary checkpoint playing with 600 visits against $\texttt{Latest}_{\texttt{def}}$ playing with no search achieves a 100.0% win rate over 1048 games. It also still works against $\texttt{Latest}$ with the defense disabled, achieving a 100.0% win rate over 1000 games. The cyclic-adversary is trained for 2223.2 V100 GPU days, which is roughly 14.0% of the compute used for training $\texttt{Latest}$ (Appendix D). The cyclic-adversary still loses against human amateurs (Appendix J.1).

5.3. Attacking a Victim With Search

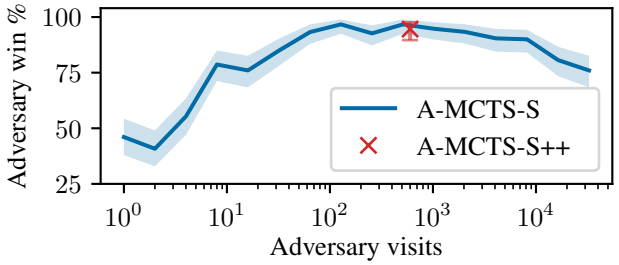
We evaluate the ability of our cyclic-adversary to exploit victims playing *with* search and find that it still achieves high win rates by tricking its victims into making severe mistakes a human would avoid (see Appendix J.2).

The cyclic-adversary achieves a win rate of 95.7% (over 1052 games) against $\texttt{Latest}_{\texttt{def}}$ with 4096 visits. The adversary also achieves a 97.3% win rate (over 1000 games) against an undefended $\texttt{Latest}$ with 4096 visits, verifying that our adversary is not exploiting anomalous behavior introduced by the pass-alive defense.

We also tested our cyclic-adversary against $\texttt{Latest}$ with substantially higher victim visits. The adversary (using A-MCTS-S with 600 visits/move) achieved an 82% win rate over 50 games against $\texttt{Latest}$ with 10^6 visits/move, and a 72% win rate over 50 games against $\texttt{Latest}$ 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.



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

Figure 5.2. We evaluate the cyclic-adversary's win rate against $\texttt{Latest}_{\texttt{def}}$ with varying amounts of search (**left**: victim, **right**: adversary). Shaded regions and error bars denote 95% Clopper-Pearson confidence intervals over ~150 games.

That said, the adversary win rate does decrease with more victim search. This is further shown in Figure 5.2a, and is even more apparent against a weaker adversary (Figure H.1). The victim also tends to judge decisive 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.

In Figure 5.2b we examine adversary search. 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.

We also plot in Figure 5.2a 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 H.1). 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 in this context to evaluate A-MCTS-R at higher visit counts. However, we do find indications that A-MCTS-R helps in high-victim-visit regimes, with the benefits being visible even with very limited recursive victim simulation. We include an initial analysis of this phenomenon in Appendix I.

5.4. Defense

In mid-December 2022, KataGo's official distributed training run was modified so that 0.08% of its self-play games start from positions where the cyclic-exploit is in the process of being carried out. This mild form of adversarial training is designed to improve KataGo's understanding of cyclic positions while preserving KataGo's strength in normal games. The performance of our cyclic-

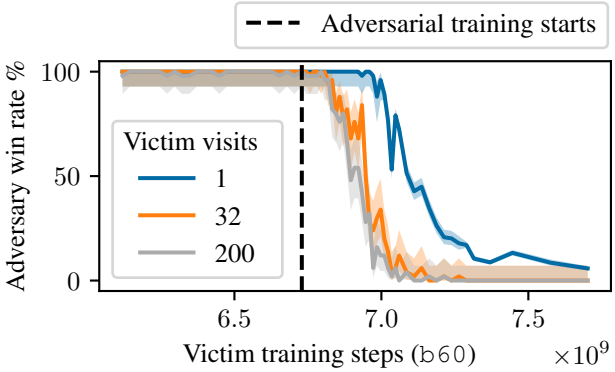


Figure 5.3. Win rate of our cyclic-adversary$^{600\text{ visits}}$ vs. different 60-block KataGo nets, including ones trained on adversarial positions surfaced by our cyclic-adversary. After adversarial training starts, the win rate of our cyclic-adversary steadily decreases. But it recovers with fine-tuning (Section 5.4 and Appendix L).

adversary$^{600\text{ visits}}$ dropped steadily after this was introduced (Figure 5.3), reaching a low of 0 / 50 won games against the $\texttt{b60-s7702m}^{32\text{ visits}}$ KataGo agent[4] and 119 / 2050 won games against $\texttt{b60-7702m}^{1\text{ visit}}$.

However, after fine-tuning the cyclic-adversary for an additional 1154.9 V100 GPU days against adversarially trained networks, it recovers its exploitation abilities and achieves a 47% win rate over 400 games against $\texttt{b60-s7702m}^{4096\text{ visits}}$ and a 17.5% win rate over 40 games against $\texttt{b60-s7702m}^{100{,}000\text{ visits}}$. These wins still rely on the cyclic-exploit, although carried out in a slightly different way. See Appendix L for a sample game and details on KataGo's defense and our cyclic-adversary's fine-tuning.

These results demonstrate that while a small amount of training on adversarial positions is enough to robustly defend against a fixed adversary, such a defense does not generalize and can be broken again by fine-tuning the fixed adversary. However, such fine-tuning requires more

[4] $\texttt{b60-s7702m}$ refers to the $\texttt{b60c320-s7701878528}$ network released on May 17th, 2023. This was the most recent 60-block network available at the time of conducting our research.

compute per unit of win rate compared to attacking non-adversarially trained networks (compare Figure L.5 with Figure D.3), so it is plausible that with much more adversarial training, KataGo could become computationally infeasible to exploit. Computing more precise scaling laws for this type of adversarial training is a fruitful direction for future work.

5.5. 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.

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

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 cyclic group is captured. This trend is depicted in Figure 5.4: in games the adversary wins, 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 5.5). This is in stark contrast to the adversary's own predictions, which change much more slowly and are less confident.

Why does the victim misjudge these cyclic positions so severely? To understand this, we studied the differences in the activations of the victim between cyclic and minimally perturbed non-cyclic positions. We found that a few channels at layer 26 show a clear divergence between cyclic and non-cyclic positions, as illustrated in Figure 5.6, whereas earlier layers showed no comparable trend. Moreover, we found that the difference in activations between `Latest` and the adversarially trained `cp580` shows a similar pattern, suggesting that adversarial training preferentially changes the behavior of the network on cyclic positions at these channels. These results provide a clear area for further investigation that could lead to a more detailed mechanistic understanding of this and similar vulnerabilities. Further analysis is available in Appendix K.

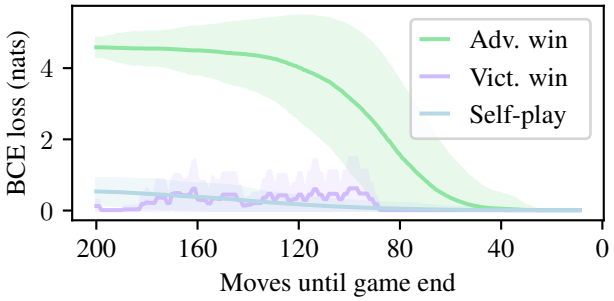


Figure 5.4. Binary cross-entropy loss of $\texttt{Latest}^{4096\text{ visits}}$'s prediction of the game result over the course of the games played against the cyclic-adversary$^{600\text{ visits}}$. The green and purple curves are averaged over games won by the adversary and victim respectively. The blue curve is averaged over self-play games and serves as a baseline. Shaded regions denote ± 1 SD.

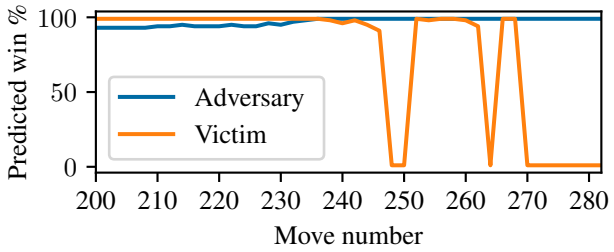
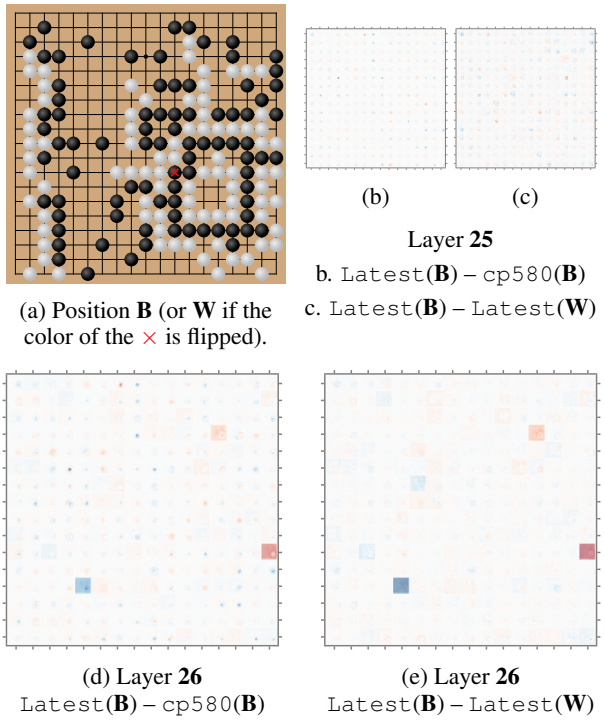


Figure 5.5. Probability of victory as predicted by the cyclic-adversary and `Latest` for a portion of a randomly selected game. Note the sudden changes in win rate prediction between moves 248 and 272 during a ko fight. Explore the game.

5.6. 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 KataGo 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. Other humans have since used cyclic attacks to beat a variety of other top Go AIs (Appendix O).



(a) Position **B** (or **W** if the color of the × is flipped).

Figure 5.6. Comparison of activations of `Latest` and `cp580` (a checkpoint adversarially trained to defend against the cyclic adversary) on a cyclic (**B**) and non-cyclic position (**W**) which differ by a single stone (a). We plot differences in activations (b-e); brighter colors indicate larger differences. In layer 25 (b,c) the activations are fairly similar. In layer 26 (d,e) there are strong differences localized to a few channels. Adversarial training (d) changes these channel activations in a similar manner to breaking the cycle **B** → **W** (e), suggesting these channels are linked to the cyclic vulnerability.

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, 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

We demonstrate that even superhuman agents can be vulnerable to adversarial policies. However, it is possible Go-playing AI systems are unusually vulnerable. Thus a promising direction for future work is to evaluate our attack against strong AI systems in other games and settings.

It is also natural to ask how we can *defend* against adversarial policies. A first attempt was made by the KataGo team after we published an earlier version of this work, but we show in Section 5.4 that this defense is as of yet inadequate. Fortunately, there are a number of other promising multi-agent RL techniques. 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 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 computation.

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 exploit superhuman Go AIs. Notably, our adversaries do not win by playing a strong game of Go. Instead, they exploit blind spots in their victims. 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.

See Appendix P for acknowledgements.

References

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

Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., Chen, A., Goldie, A., Mirhoseini, A., McKinnon, C., Chen, C., Olsson, C., Olah, C., Hernandez, D., Drain, D., Ganguli, D., Li, D., Tran-Johnson, E., Perez, E., Kerr, J., Mueller, J., Ladish, J., Landau, J., Ndousse, K., Lukosuite, K., Lovitt, L., Sellitto, M., Elhage, N., Schiefer, N., Mercado, N., DasSarma, N., Lasenby, R., Larson, R., Ringer, S., Johnston, S., Kravec, S., Showk, S. E., Fort, S., Lanham, T., Telleen-Lawton, T., Conerly, T., Henighan, T., Hume, T., Bowman, S. R., Hatfield-Dodds, Z., Mann, B., Amodei, D., Joseph, N., McCandlish, S., Brown, T., and Kaplan, J. Constitutional AI: Harmlessness from AI feedback, 2022.

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.

Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y. T., Li, Y., Lundberg, S., Nori, H., Palangi, H., Ribeiro, M. T., and Zhang, Y. Sparks of artificial general intelligence: Early experiments with GPT-4, 2023.

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. Efficient selectivity and backup operators in Monte-Carlo tree search. In *Computers and Games*, pp. 72–83. Springer, 2007.

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.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

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. Gpt-4 technical report, 2023.

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, G.-C. 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.

Rosin, C. D. Multi-armed bandits with episode context. *Annals of Mathematics and Artificial Intelligence*, 61(3):203–230, March 2011. ISSN 1573-7470. doi: 10.1007/s10472-011-9258-6.

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. Discord comment on the b18c384 katago architecture, 5 2022a. URL https://discord.com/channels/417022162348802048/583775968804732928/970572391325532200.

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, 2022b. 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.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models, 2023.

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.[5]

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 1.1b 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.

[5] 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. 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.

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.[6] 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

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

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[7]

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

   where $\text{par}_T(x)$ 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.

[6] 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.

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

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

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

where c in this case is a child of the root node. The temperature parameter τ trades off between exploration and exploitation.[8]

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.

In addition, neural network evaluations of each node should only be performed once and a cached evaluation should be used when revisiting a node during a subsequent walk down the tree.

Our adversarial variants of MCTS use both of the above speedups.

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:

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

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

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.[9]

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.

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

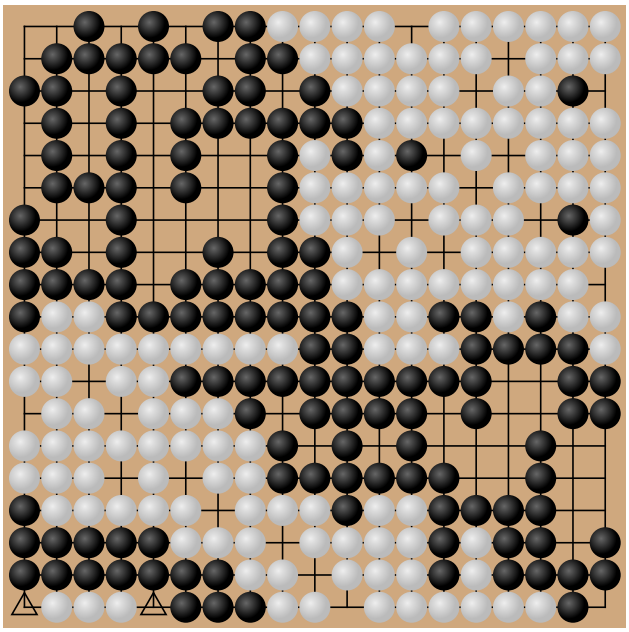


Figure B.1. Black moves next in this game. There is a seki in the bottom left corner of the board. Neither black nor white should play in either square marked with Δ, or else the other player will play in the other square and capture the opponent's stones. If `Latest` with 128 visits plays as black, it will pass. On the other hand, $\texttt{Latest}_{\texttt{def}}$ with 128 visits playing as black will play in one of the marked squares and lose its stones.

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.

B.6.1. Vulnerability of Defense in Seki Situations

Training against defended victims resulted in the cyclic-adversary which successfully exploits both the defended $\texttt{Latest}_{\texttt{def}}$ and the undefended `Latest`, but adding the defense to victims in fact adds a vulnerability that undefended victims do not have. Because defended victims are usually not allowed to pass, they blunder seki situations where it is better to pass than play.

For instance, consider the board shown in Figure B.1. Black is next to play. At this point, the game is over unless one of the players severely blunders. White cannot capture black's large group stretching from the top-left corner to the top-right corner, and black cannot capture white's two large groups. There is a seki in the bottom-left corner of the board, where neither player wants to play in either of the two squares marked with Δ since then the other player could play in the other marked square and capture the opponent's stones. Black is winning and should pass and wait for white to also pass or resign. Indeed, `Latest` with 128 visits playing as black passes and eventually wins by 8.5 points.

$\texttt{Latest}_{\texttt{def}}$, however, is not allowed to pass, and instead plays in one of the squares marked by Δ. White can then play in

the other marked square to capture black's stones. Then white owns all the territory in the bottom-left corner and wins by 25.5 points.

We discovered this weakness of the pass-alive defense when we trained an adversary against $\texttt{Latest}_{\texttt{def}}$ with the adversary's weights initialized to `cp63`, an early KataGo checkpoint. The adversary consistently set up similar seki situations to defeat $\texttt{Latest}_{\texttt{def}}$, but it would lose against the undefended `Latest`.

C. Hyperparameter Settings

We enumerate the key hyperparameters used in our training run in Table C.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 self-play 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.

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 C.1. Key hyperparameter settings for our adversarial training runs.

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 (2022b).

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.

16–20. ...

21. `b40c256-s11840935168-d2898845681` ($\texttt{Latest}_{\texttt{def}}$), 1024 visits.
22. `b40c256-s11840935168-d2898845681` ($\texttt{Latest}_{\texttt{def}}$), 1600 visits.
23. `b40c256-s11840935168-d2898845681` ($\texttt{Latest}_{\texttt{def}}$), 4096 visits.
24. `b40c256-s11840935168-d2898845681` ($\texttt{Latest}_{\texttt{def}}$), 8192 visits.

25–27. ...

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 ~20.4 V100 GPU days to train our strongest pass-adversary, ~2223.2 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 0.13% and 14.0% (respectively) of the compute it took to train KataGo. Moreover, an earlier checkpoint of the cyclic-adversary trained using only 7.6% of the compute to train KataGo already achieves a 94% win rate against $\texttt{Latest}_{\texttt{def}}$ with 4096 visits.

As another point of reference, our strongest pass-adversary took 9.18×10^4 self-play games to train, our strongest cyclic-adversary took 1.01×10^6 self-play games to train, and `Latest` took 5.66×10^7 self-play games to train.[10]

Note that training our cyclic-adversary used 14% of `Latest`'s compute, but less than 2% of `Latest`'s games. This is because our cyclic-adversary was trained against high-visit count versions of `Latest` towards the end of its curriculum, and the compute required to generate a victim-play game scales proportionally with the amount of victim visits. See Figure D.1 for a visual illustration of this effect.

D.1. Estimating the Compute Used by Our Attack

To train our adversaries, we used A4000, A6000, A100 40GB, and A100 80GB GPUs. The primary cost of training is in generating victim-play games, so we estimated GPU-day conversions between these GPUs by benchmarking how fast the GPUs generated games.

We estimate that one A4000 GPU-day is 0.627 A6000 GPU-days, one A100 40GB GPU-day is 1.669 A6000 GPU-days, and one A100 80GB GPU-day is 1.873 A6000 GPU-days. We estimate one A6000 GPU-day is 1.704 V100 GPU-days.

Figure D.1 plots the amount of compute used against the number of adversary training steps. To train the pass-adversary, we used 12.001 A6000 GPU-days, converting to 20.4 V100 GPU-days. To train the cyclic-adversary, we used 61.841 A4000 GPU-days, 348.582 A6000 GPU-days, 299.651 A100 40GB GPU-days, and 222.872 A100 80GB GPU-days, converting to 2223.2 V100 GPU-days.

The cyclic-adversary was already achieving high win rates against $\texttt{Latest}_{\texttt{def}}$ with smaller amounts of training. In Figure D.3, earlier checkpoints of the cyclic-adversary achieved a win rate of 64.6% against $\texttt{Latest}_{\texttt{def}}$ with 4096 victim visits using 749.6 V100 GPU-days of training (4.7% of the compute to train `Latest`) and a win rate of 94% using 1206.2 V100 GPU-days of training (7.6% of the compute to train `Latest`), compared to a win rate of 95.7% using 2223.2 V100 GPU-days of training.

[10] To estimate the number of games for KataGo, we count the number of training games at `katagotraining.org/games` (only for networks prior to `Latest`) and `katagoarchive.org/g170/selfplay/index.html`.

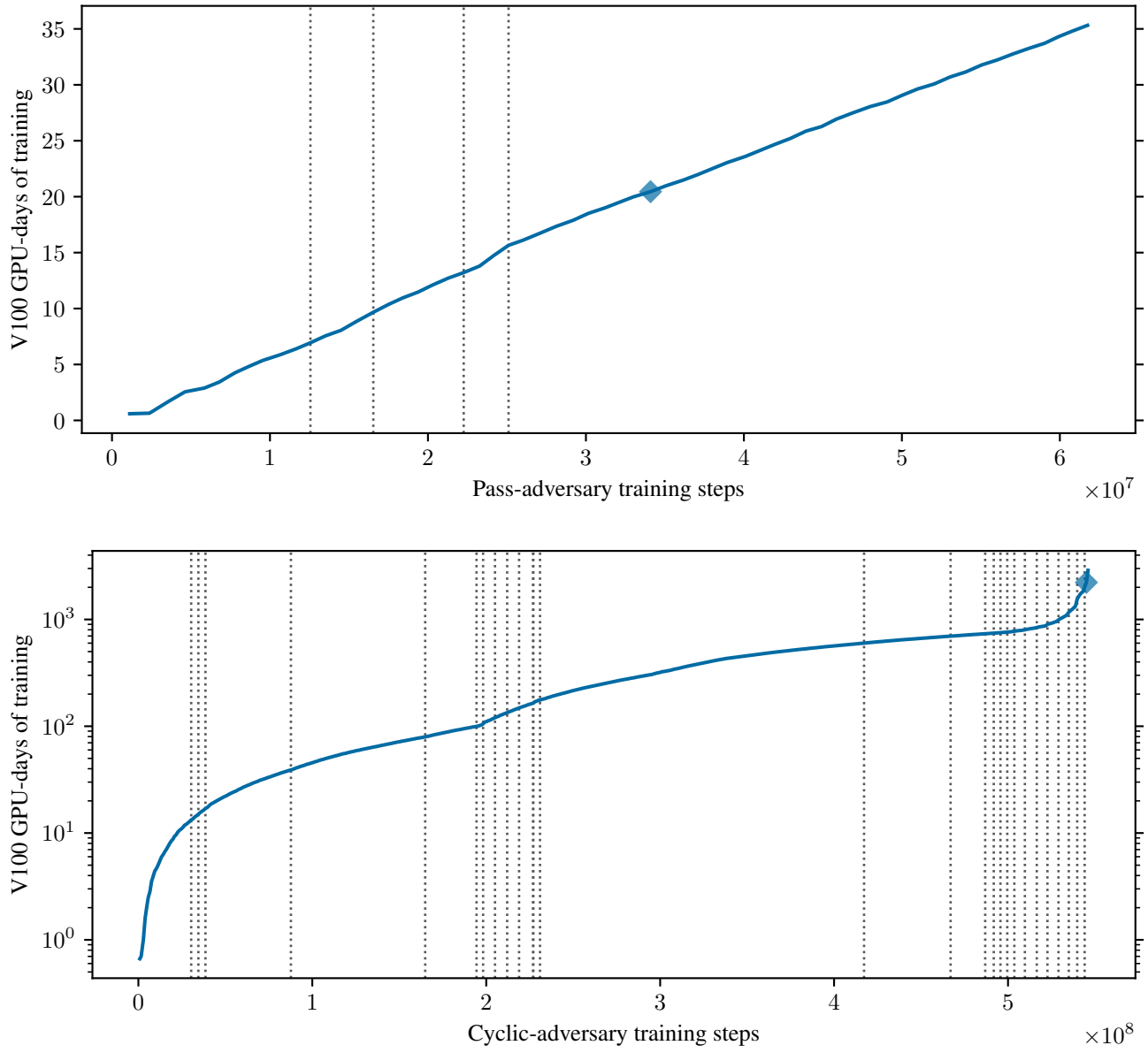


Figure D.1. The compute used for adversary training (y-axis) as a function of the number of adversary training steps taken (x-axis). The plots here mirror the structure of Figure F.1 and Figure 5.1. Top: The compute of the pass-adversary is a linear function of its training steps because the pass-adversary was trained against victims of similar size, all of which used no search (Appendix C.1). Bottom: 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).

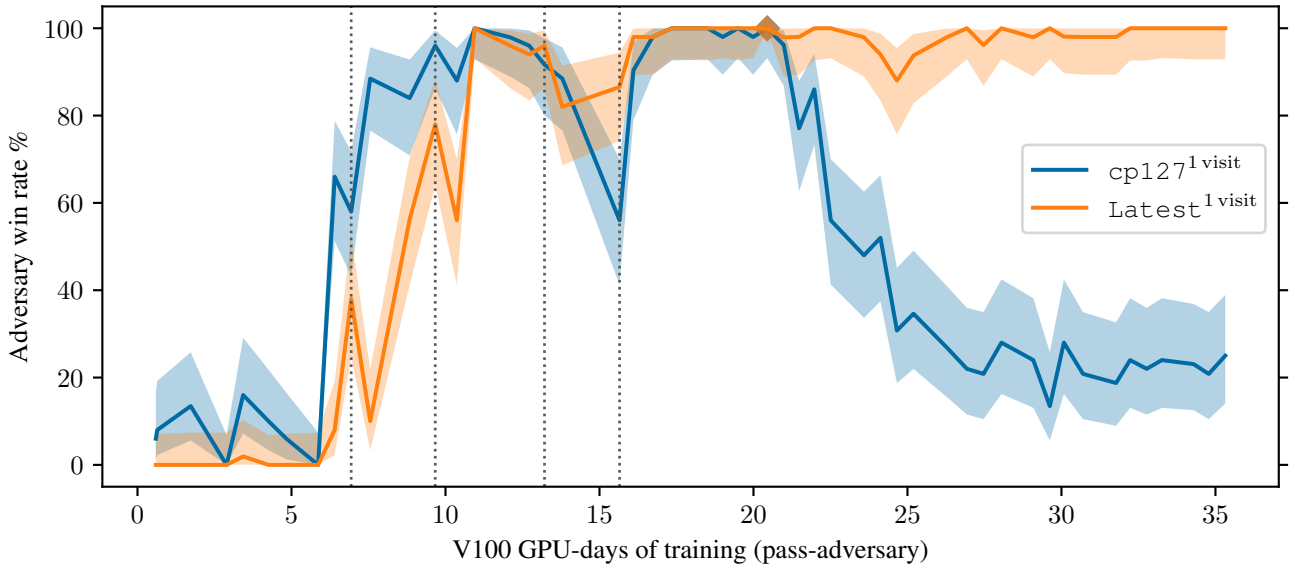


Figure D.2. The win rate achieved by the pass-adversary throughout training (y-axis) as a function of the training compute used (x-axis). This figure is the same as Figure F.1 but with V100 GPU-days on the x-axis instead of adversary training steps.

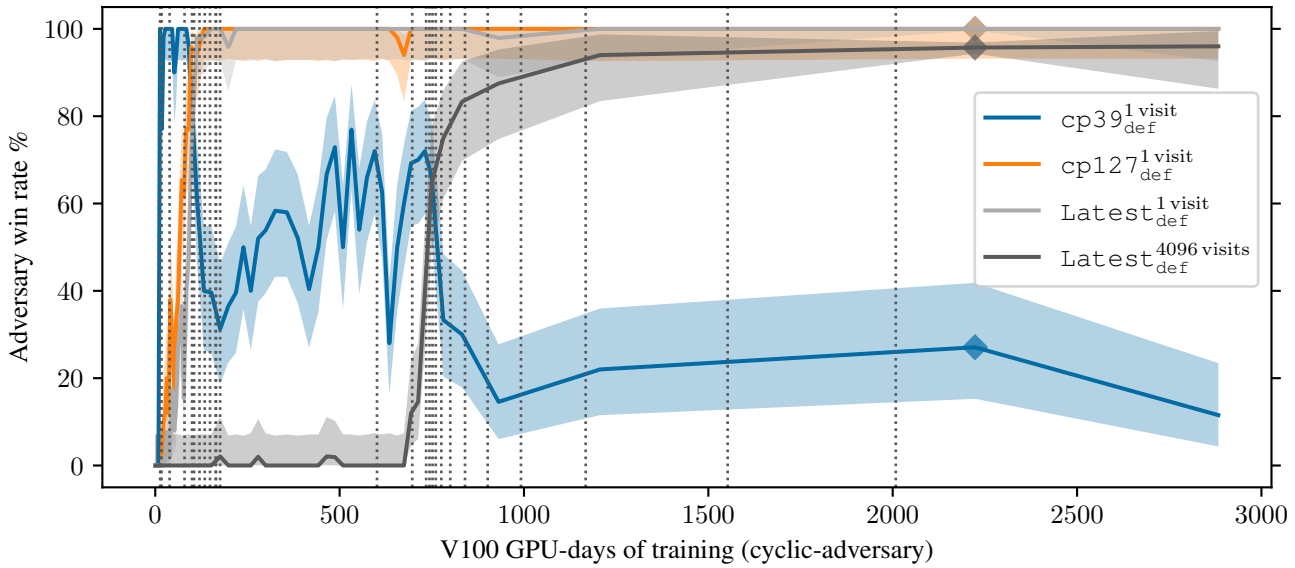


Figure D.3. The win rate achieved by the cyclic-adversary throughout training (y-axis) as a function of the training compute used (x-axis). This figure is the same as Figure 5.1 but with V100 GPU-days on the x-axis instead of adversary training steps.

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 D.1. 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.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 justify this assumption based on the following factors:[11]

1. The "third major run" used `b6`, `b10`, `b20`, `b30`, and `b40` nets while distributed training used only `b40` nets and larger, with larger nets being more costly to run (Table D.1).
2. The "third major run" used less search during self-play than distributed training. Source: the following message from David Wu (the creator and primary developer of KataGo).

   > KataGo used 600 full / 100 cheap [visits] for roughly the first 1-2 days of training (roughly up through b10c128 and maybe between 1/2 and 1/4 of b15c192), 1000 full / 200 cheap [visits] for the rest of g170 (i.e. all the kata1 models that were imported from the former run g170 that was done on private hardware alone, before that run became the prefix for the current distributed run kata1), and then 1500 full / 250 cheap [visits] for all of distributed training so far.

`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.

[11] The biggest potential confounding factor is KataGo's neural network cache, which (per David Wu in private comms) "is used if on a future turn you visit the same node that you already searched on the previous turn, or if multiple move sequences in a search lead to the same position". Moreover, "this [cache] typically saves somewhere between 20% and 50% of the cost of a search relative to a naive estimate based on the number of visits". It is possible that distributed training has a significantly higher cache hit-rate than the "third major run", in which case our bound might be invalid. We assume that the stated factors are enough to overcome this and other potential confounding effects to yield a valid lower-bound.

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 E.1, 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.

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 E.1. 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 (2022b), the checkpoint is roughly 10 Elo weaker than `Latest`.

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 using its "prototype" model, 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.[12] 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.

[12]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. ELF OpenGo's "final" model is about 150 Elo stronger than its prototype model (Tian et al., 2019), giving an Elo of 4395+ at 80,000 visits per move.

3. The strongest network in the original KataGo paper was shown to be slightly stronger than ELF OpenGo's final network (Wu, 2019, Table 1) when both bots were run at 1600 visits per move. From Figure E.1, 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 4395+ on goratings.org.

4. The strongest network in the original KataGo paper is comparable to the `b15c192-s1503689216-d402723070` checkpoint on katagotraining.org (Wu, 2022b). 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.

5. Finally, log-linearly extrapolating the performance of `Original` from 1600 to 80,000 visits using Figure E.1 yields an Elo difference of ~834 between the two visit counts.

6. 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 $4395 - 1657$ = ~2738 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.[13] 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.

[13] 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.

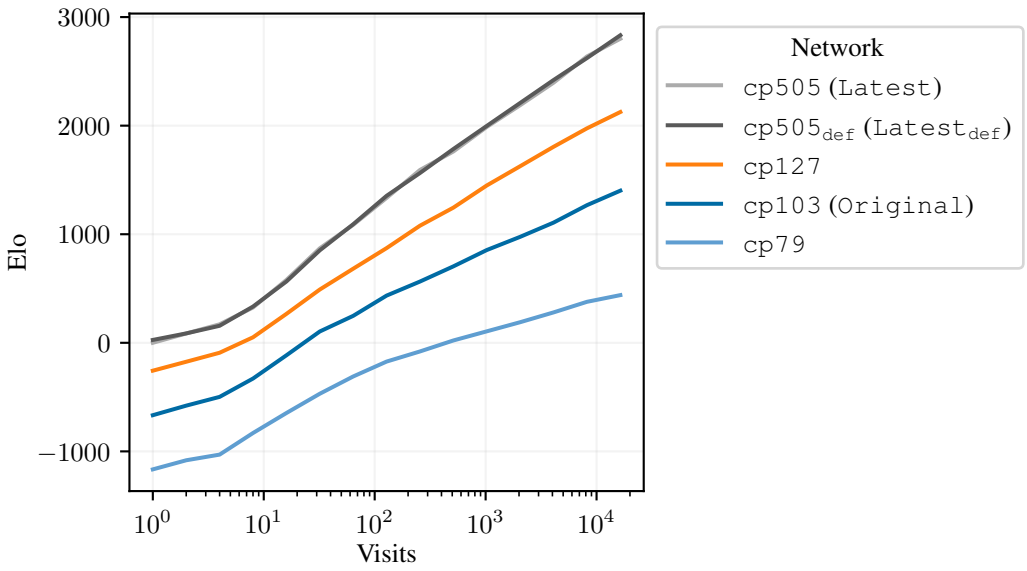


Figure E.1. Elo ranking (y-axis) of networks (different colored lines) by visit count (x-axis). The lines are approximately linear on a log x-scale, with the different networks producing similarly shaped lines vertically shifted. This indicates that there is a *consistent* increase in Elo, regardless 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).

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 ~2738 on goratings.org.

Assuming Elo transitivity, we can estimate the strength of `Latest` by utilizing Figure E.1. 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 ~3063 on goratings.org—within the top 500 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 4500, over 600 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 E.2. 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.[14] The victim Alp-

[14]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 $1/(1+\left(\frac{3300-G_A}{3300-G_B}\right)^7)$.

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 E.2. 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.[15] 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.[16] 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 E.2. However, from Figure E.1 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.

[15] This is a slightly non-trivial 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 subsquent Elo-GoR conversion calculations: Colab notebook link.

[16] 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 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, and that Lee Sedol has an Elo of 2068 relative to GnuGo 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 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 F.1 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 F.2, 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.

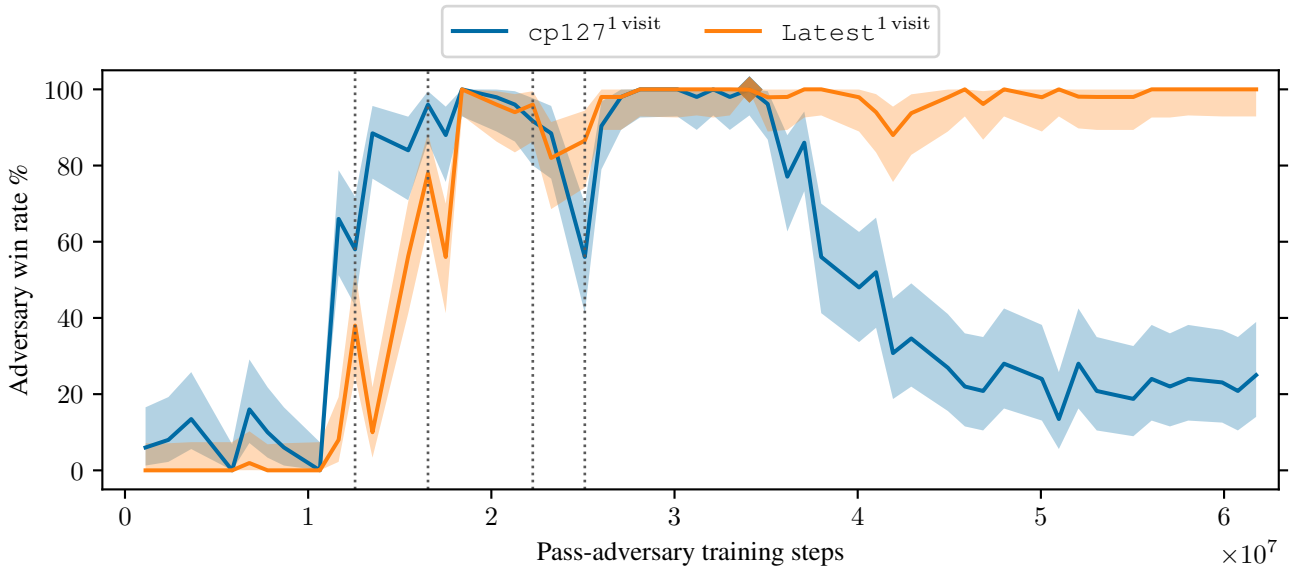
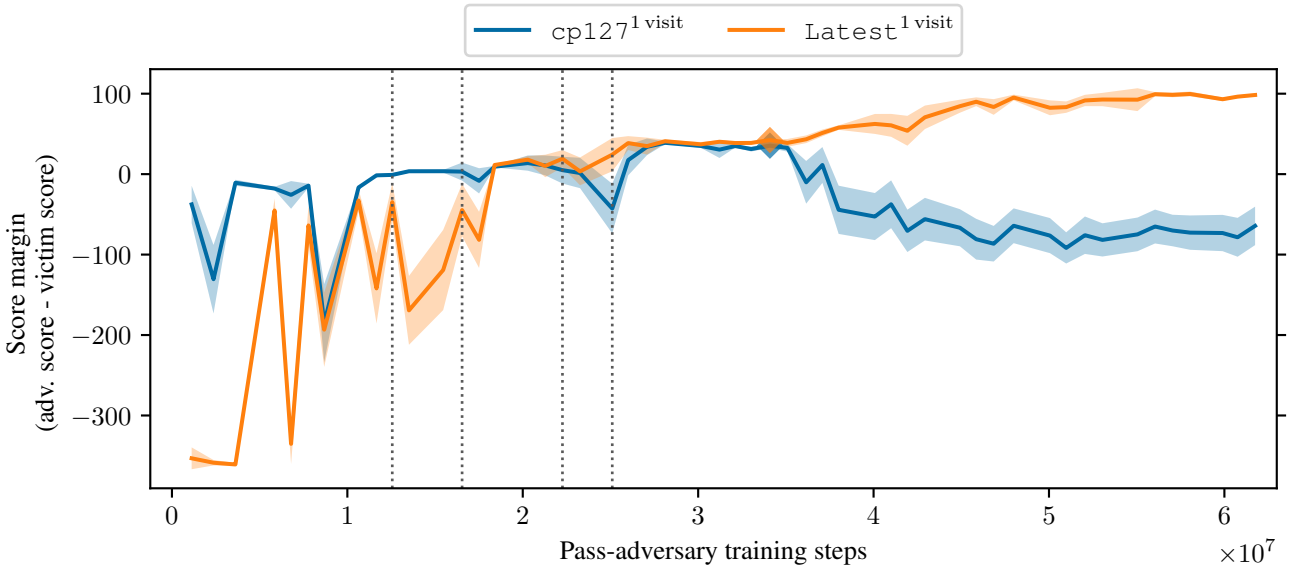
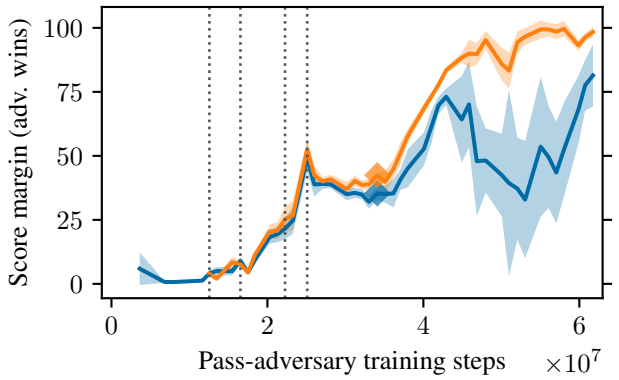


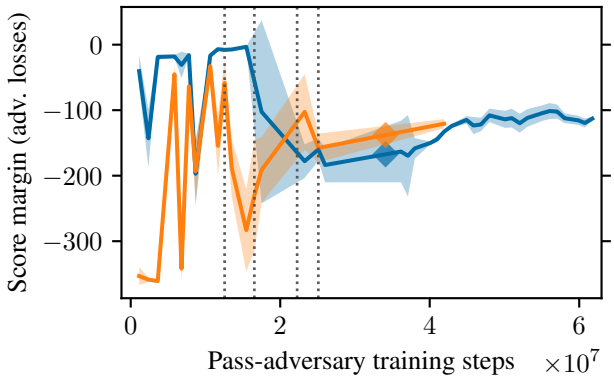
Figure F.1. The win rate (y-axis) of the pass-adversary from Section 5.1 over time (x-axis) against the `cp127` and `Latest` victim policy networks playing without search. The strongest adversary checkpoint (marked ♦) wins 1047/1048 games against `Latest`. The adversary overfits to `Latest`, winning 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.



(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.

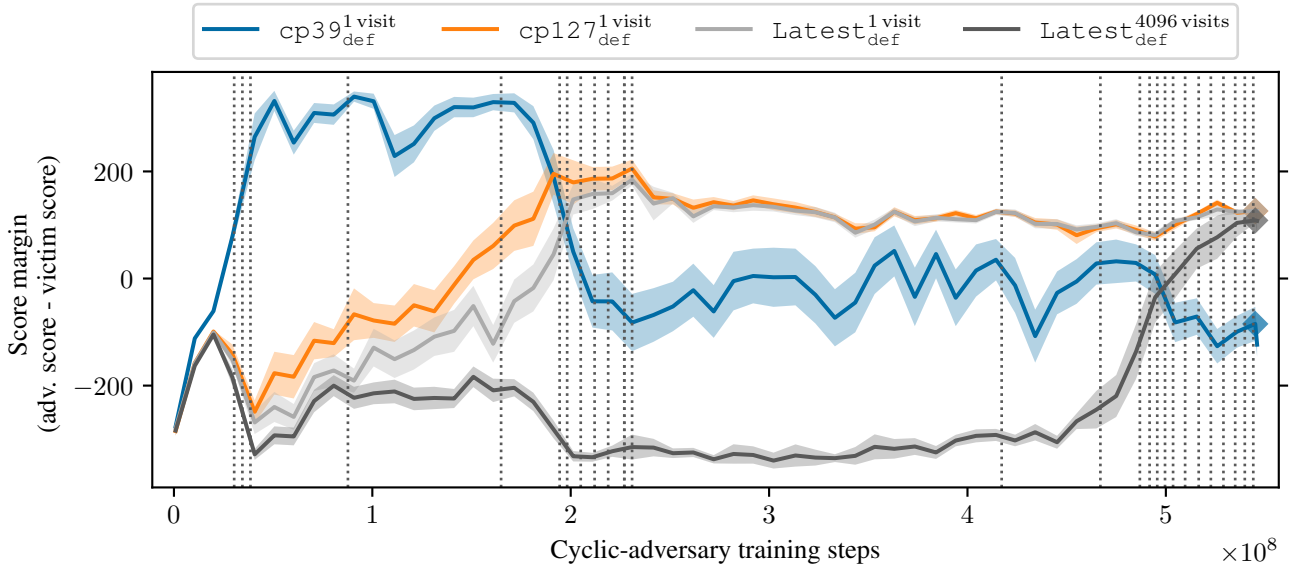


(c) Score margin, restricted to games adversary lost.

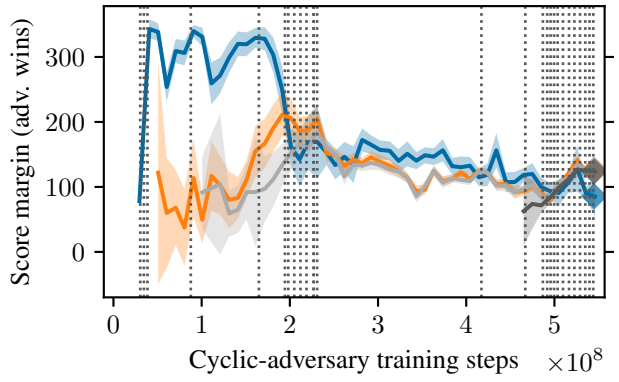
Figure F.2. We evaluate the average margin of victory for the pass-adversary from Section 5.1 against `Latest` without search as the training process progresses. Shaded regions are 95% T-intervals over $n = 50$ games per checkpoint.

F.2. Score Margin of the Cyclic-Adversary

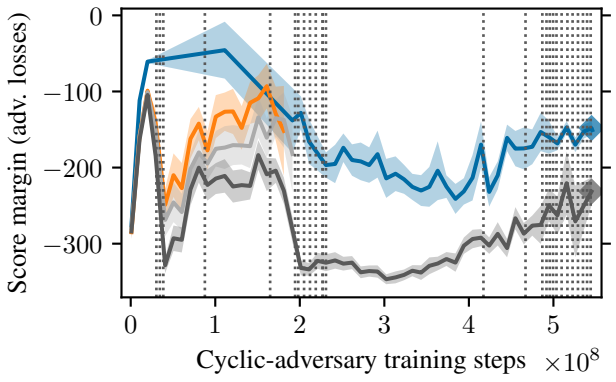
In Figure F.3, we show the margin of victory over the training process of the cyclic-adversary from Section 5.2 against victims with the pass-alive defense. The corresponding win rate is shown in Figure 5.1. Compared to Figure F.2, we see that the margin of victory is typically larger. This is likely 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 4096 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.



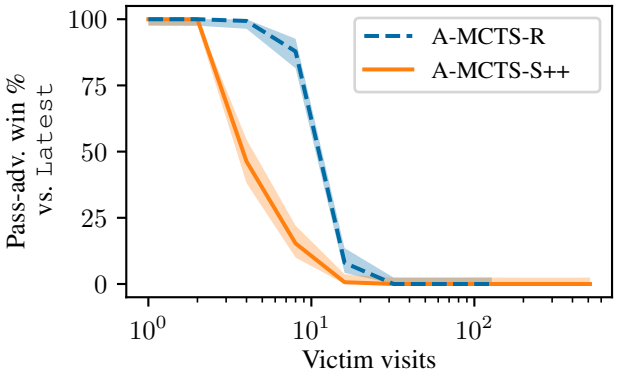
(c) Score margin, restricted to games adversary lost.

Figure F.3. We evaluate the average margin of victory for the cyclic-adversary from Section 5.2 against various victims as the training process progresses. Shaded regions are 95% T-intervals over $n = 50$ games per checkpoint.

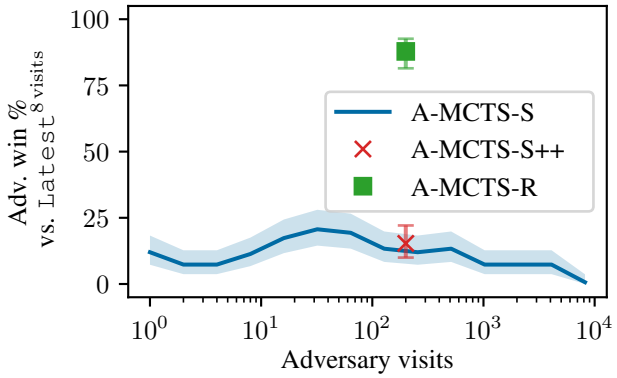
F.3. Pass-Adversary vs. Victims With Search

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 F.4a 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 F.4b, 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.



(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 F.4. We evaluate the ability of the pass-adversary from Section 5.1 trained against `Latest` without search to transfer to `Latest` with search.

F.4. Transferring Attacks Between Checkpoints

In Figure F.5, 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.

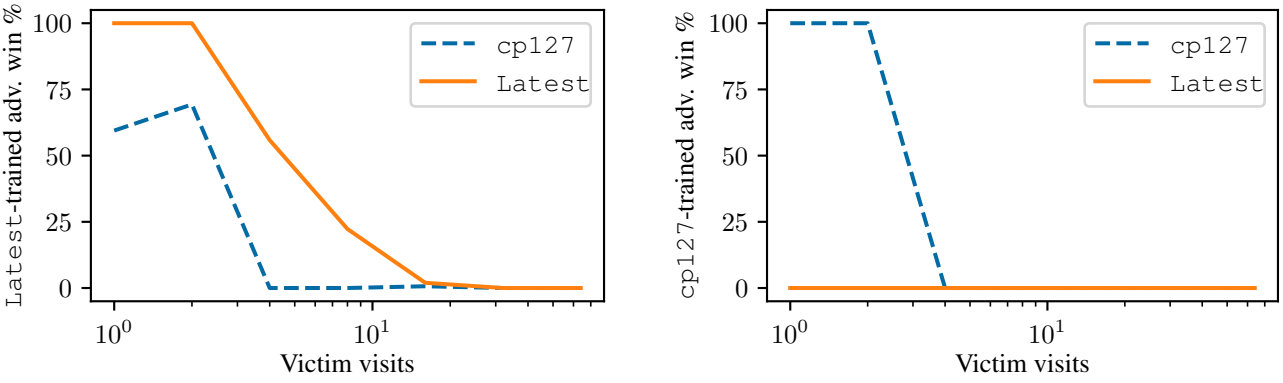


Figure F.5. An adversary trained against `Latest` (left) or `cp127` (right), evaluated against both `Latest` and `cp127` at various visit counts. The adversary always uses 600 visits/move.

F.5. Baseline Attacks

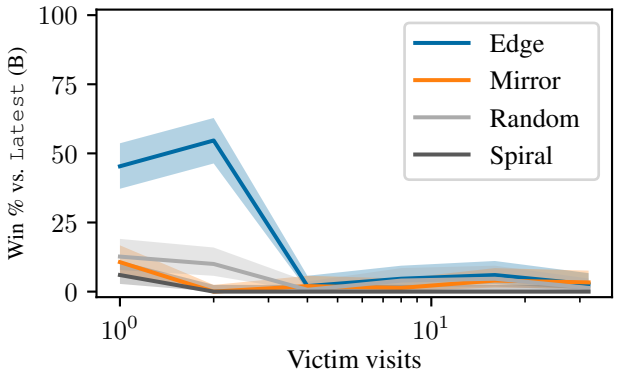
We also test *hard-coded* baseline adversarial policies. These baselines were inspired by the behavior of our trained adversary. The *Edge* attack plays random legal moves in the outermost ℓ^∞-box available on the board. The *Spiral* attack is similar to the *Edge* attack, except that it plays moves 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.

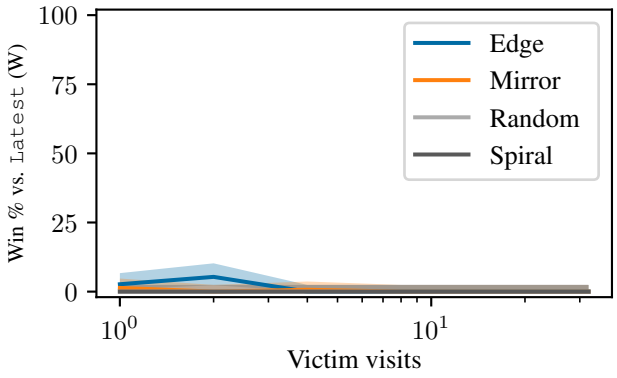
In Figure F.6, we plot the win rate and win margin of the baseline attacks against the KataGo victim `Latest`. The edge attack is the most successful, achieving a 45% win rate when `Latest` plays as black with no search. None of the attacks work well once `Latest` is playing with at least 4 visits.

In Figure F.7, 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.

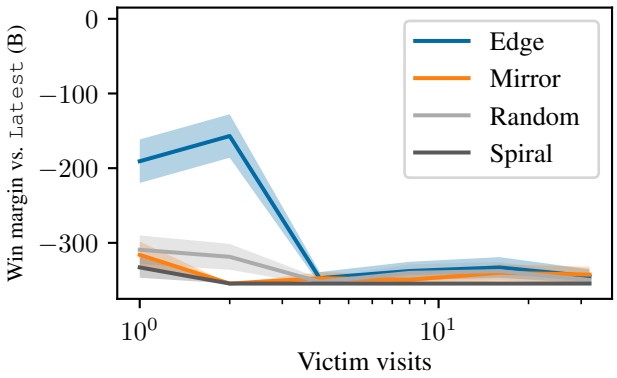
We also run the baseline attacks against the weaker `cp127`, with Figure F.8 plotting the win rate and win margin of the baseline attacks against `cp127` and Figure F.9 plotting the same statistics against $\texttt{cp127}_{\texttt{def}}$. `cp127` without search is shockingly vulnerable to simple attacks, 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.



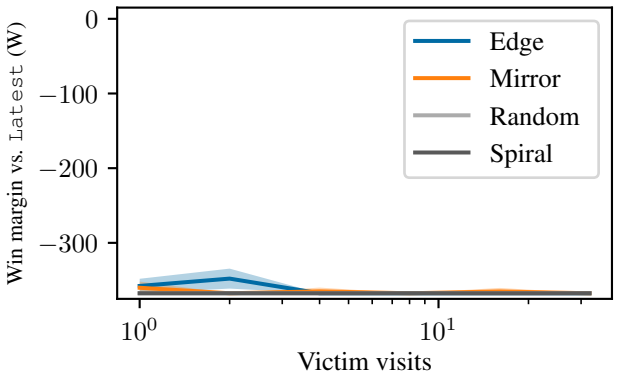
(a) Win rate as white against `Latest` as black.



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

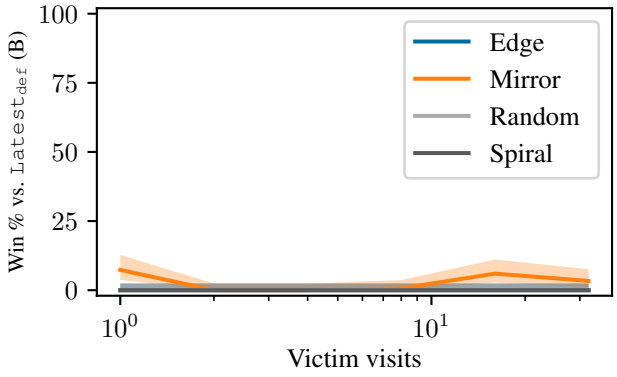


(c) Win margin as white against `Latest` as black.

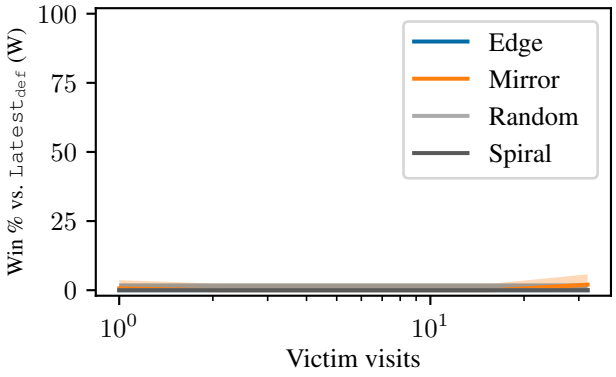


(d) Win margin as white against `Latest` as black.

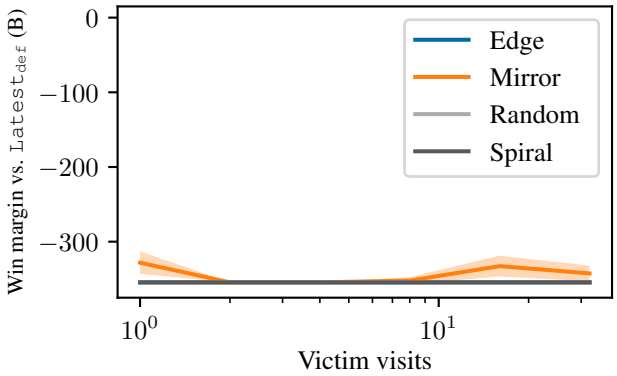
Figure F.6. Win rates and win margins of different baseline attacks versus `Latest` at varying visit counts (x-axis). 95% confidence intervals are shown. The win margins are negative, indicating that on average the victim gains more points than the attack does.



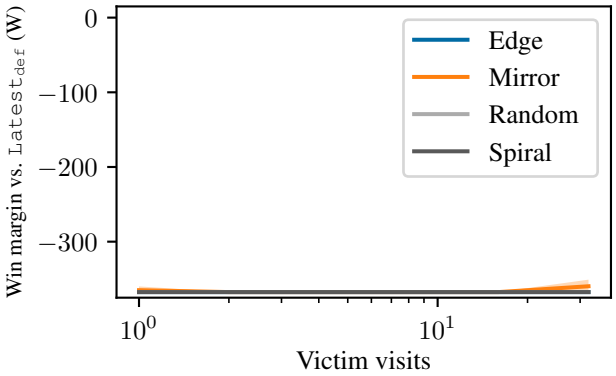
(a) Win rate as white against $\texttt{Latest}_{\texttt{def}}$ as black.



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

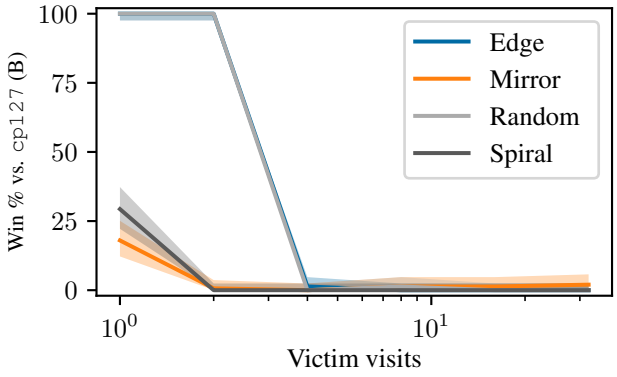


(c) Win margin as white against $\texttt{Latest}_{\texttt{def}}$ as black.

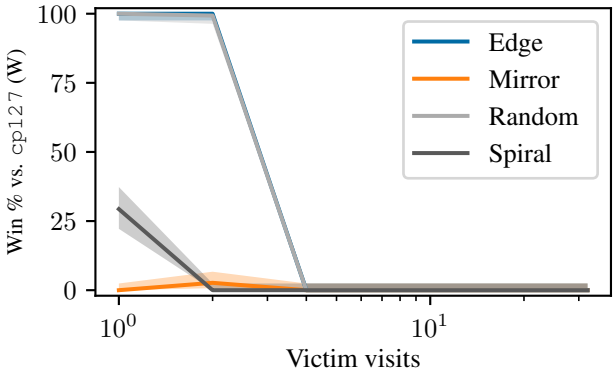


(d) Win margin as white against $\texttt{Latest}_{\texttt{def}}$ as black.

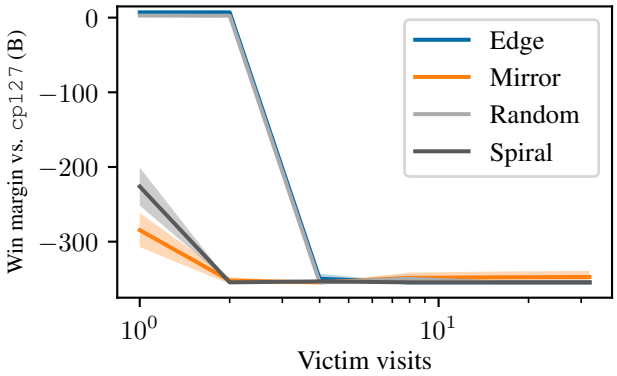
Figure F.7. Win rates and win margins of different baseline attacks versus $\texttt{Latest}_{\texttt{def}}$ at varying visit counts (x-axis). 95% confidence intervals are shown. None of the attacks see much success.



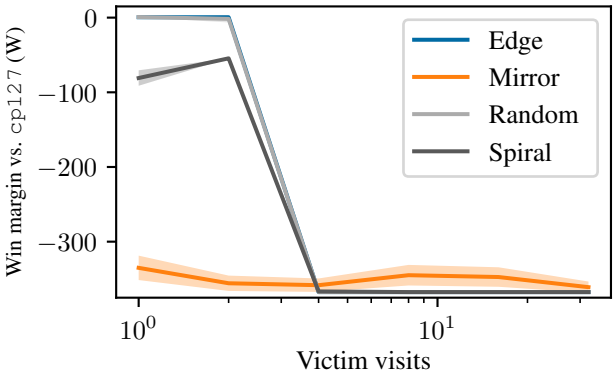
(a) Win rate as white against `cp127` as black.



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

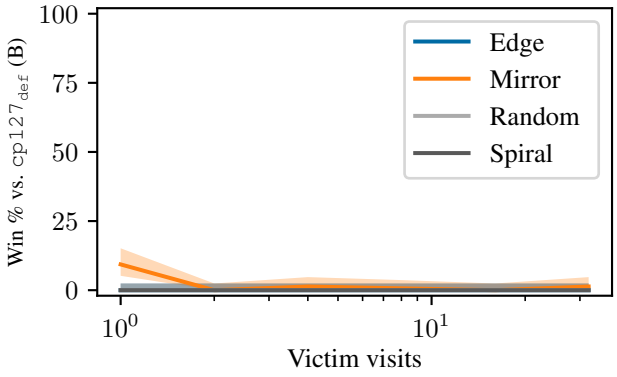


(c) Win margin as white against `cp127` as black.

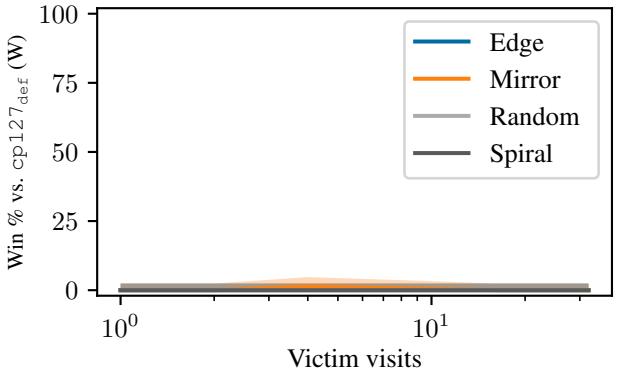


(d) Win margin as white against `cp127` as black.

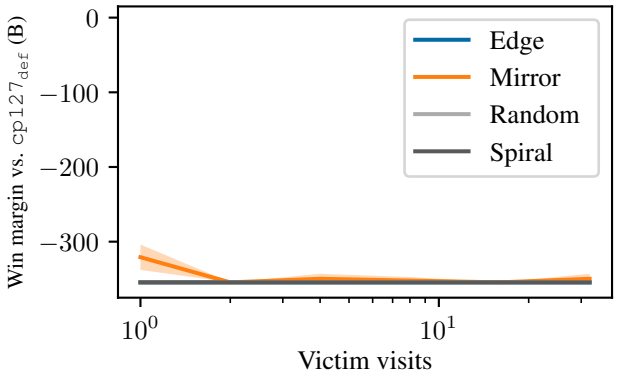
Figure F.8. Win rates and win margins of different baseline attacks versus `cp127` at varying visit counts (x-axis). 95% confidence intervals are shown.



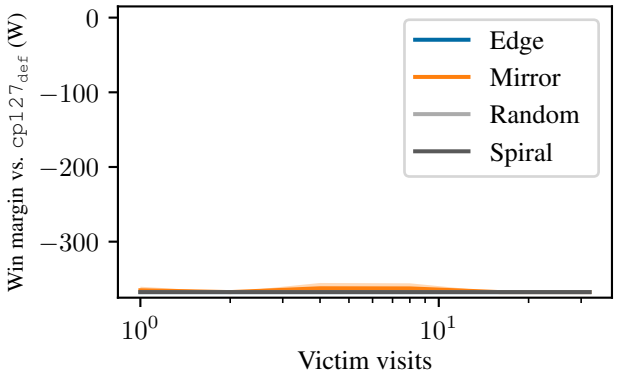
(a) Win rate as white against $\texttt{cp127}_{\texttt{def}}$ as black.



(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 F.9. 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.

F.6. Understanding the Pass-Adversary

We observed in Figure 1.1b 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 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 F.6, we see that all of our baseline attacks perform substantially worse than our pass-adversary (Figure F.4a). 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 F.1, so our adversaries are also able to play on smaller board sizes.

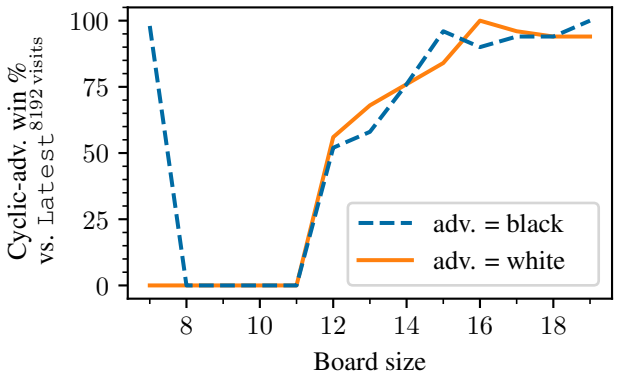
Board size ($n \times n$)	7	8	9	10	11	12	13	14	15	16	17	18	19
Training frequency (%)	1	1	4	2	3	4	10	6	7	8	9	10	35

Table F.1. Percentage of games played at each board size throughout the training of our adversaries. These percentages are the defaults for KataGo training.

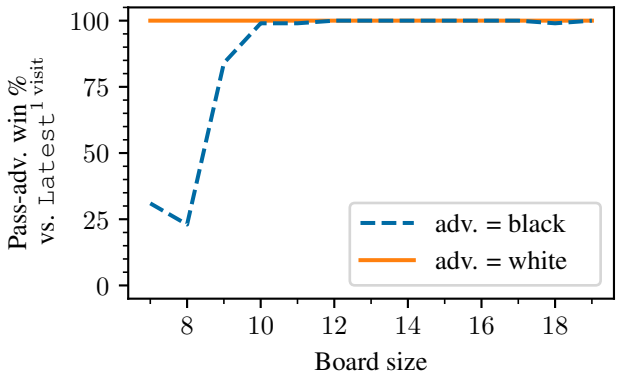
Figure F.10 plots the win rate across different board sizes for the cyclic-adversary against `Latest` playing with 8192 visits (Figure F.10a) and the pass-adversary against `Latest` playing without search (Figure F.10b). The komi is 8.5 for 7×7 boards, 9.5 for 8×8 boards, and 6.5 otherwise. These values were taken from analysis by David Wu, creator of KataGo, on fair komis for different board sizes under Chinese Go rules.[17] 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.

For comparison, Figure F.11 plots the win rate of `Latest` with 8192 visits playing against itself.



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



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

Figure F.10. Win rate of our adversaries playing as each color against `Latest` on different board sizes.

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

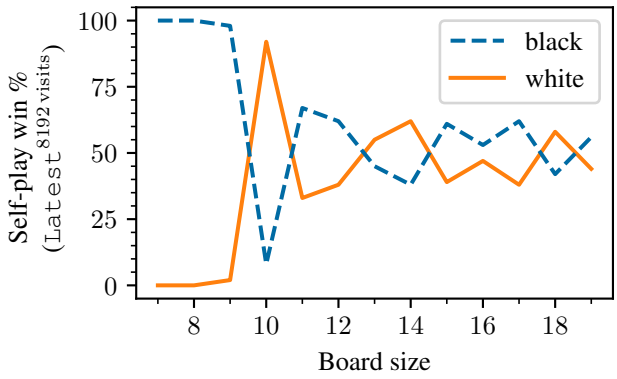


Figure F.11. Win rate of `Latest` with 8192 visits playing against itself on different board sizes.

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.

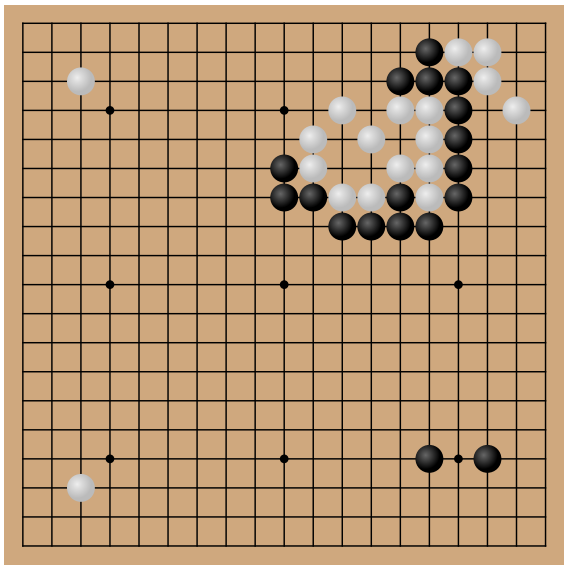
We ran ELF OpenGo with its final model and 80,000 rollouts. A weaker model with 80,000 rollouts was already strong enough to consistently defeat several top-30 Go players (Tian et al., 2019). We ran Leela Zero with its final model (released February 15, 2021), unlimited time, and a maximum of 40,000 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.

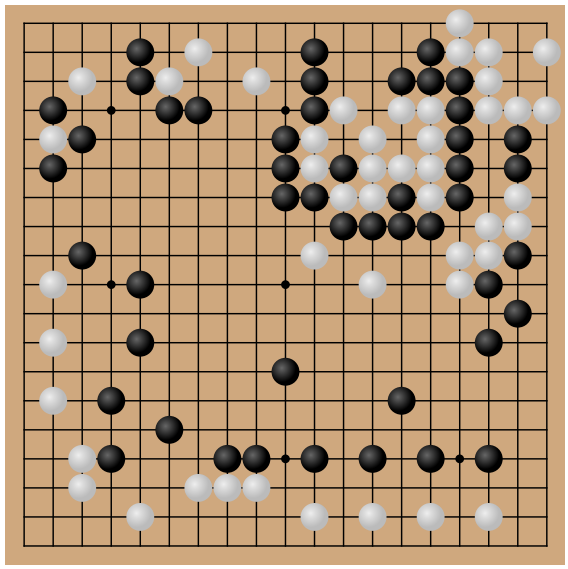
G.2. Human Transfer

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.

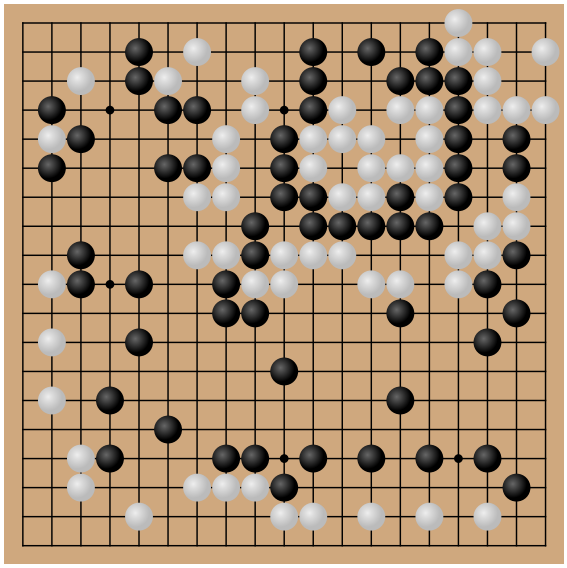
In the following figures we present selected positions from the games. The full games are available on our website. First, Figure G.1 shows key moments in a game against KataGo with 100k visits. Figure G.2 shows the same against LeelaZero with 100k visits. Figure G.3 shows a normal game against JBXKata005, while Figure G.4 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.



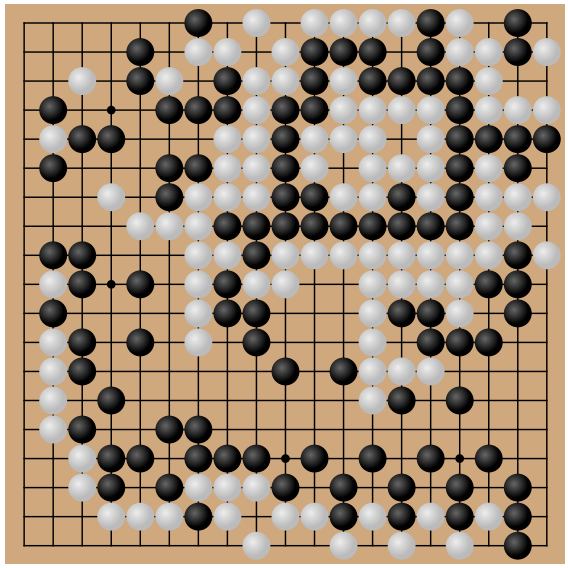
(a) Move 36: human has set up the inner group (top right middle) around which to lure the victim to create a cycle.



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

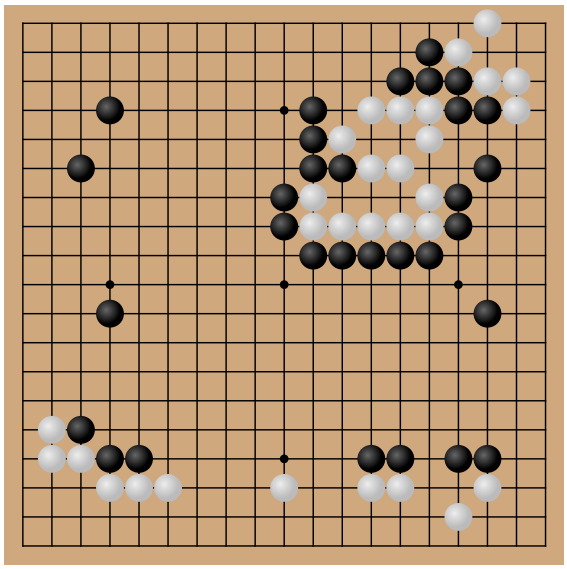


(c) Move 122: victim's cycle group is now surrounded. It remains to capture it before the victim catches on and defends.

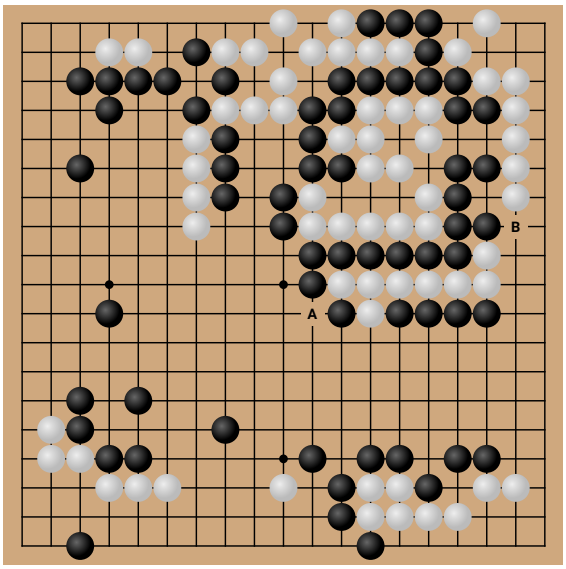


(d) Move 210: by now none of the victim's stones in the top right can avoid capture. The victim finally realizes and resigns.

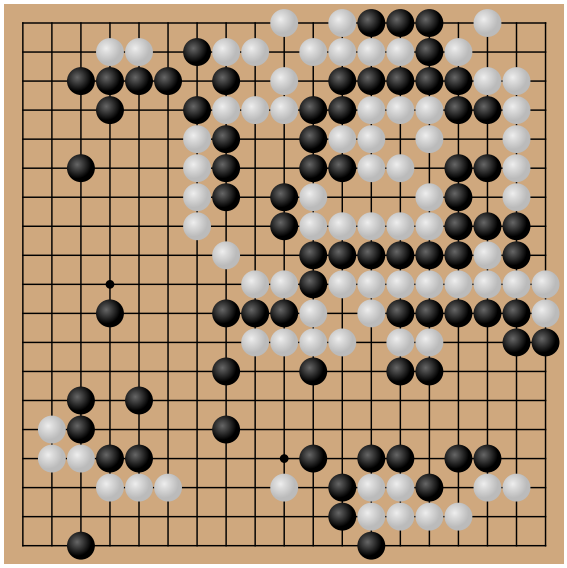
Figure G.1. Human (white) beats KataGo with 100k visits (black).



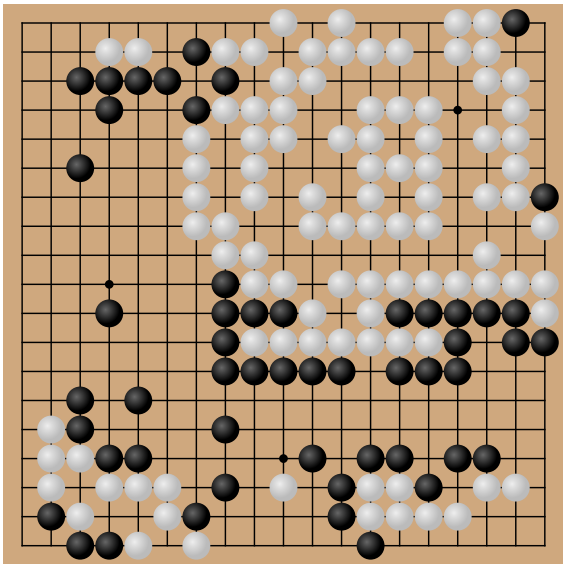
(a) Move 61: human has set up the inner group (top right middle) around which to lure the victim to create a cycle.



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

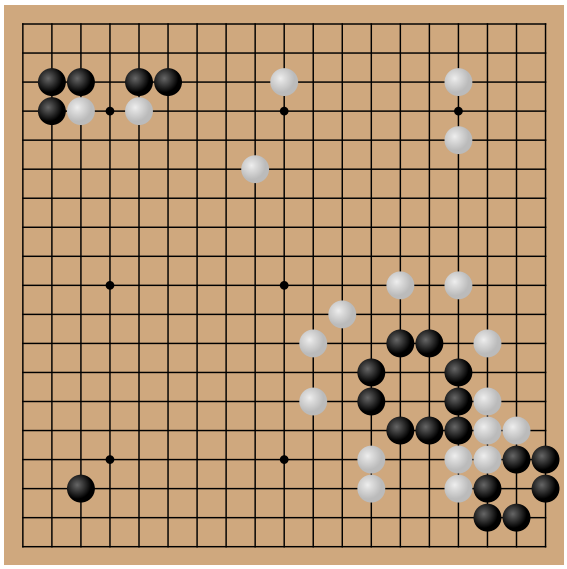


(c) Move 156: the encirclement is successful. Victim could survive by capturing one of the encircling groups, but will it?

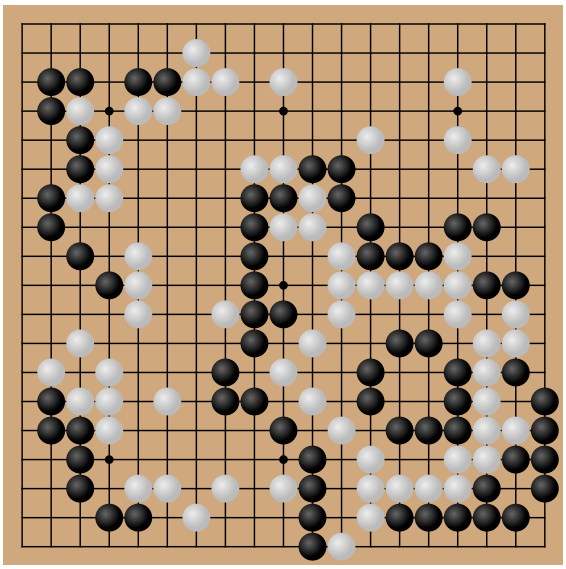


(d) Move 199: the victim failed to see the danger in time, was captured, and resigns.

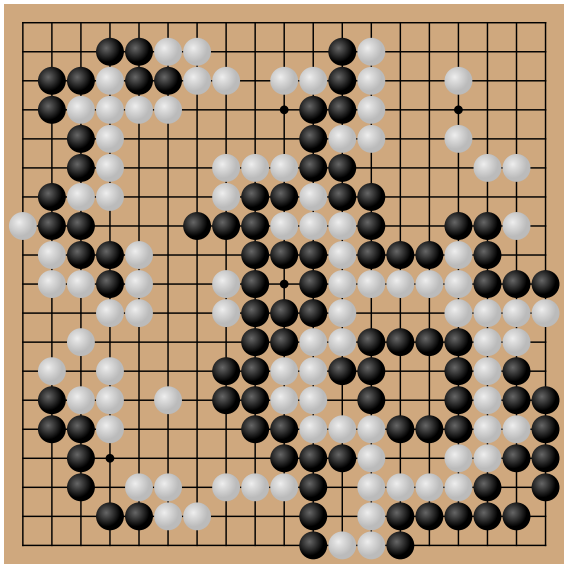
Figure G.2. Human (white) beats Leela Zero with 100k visits (black).



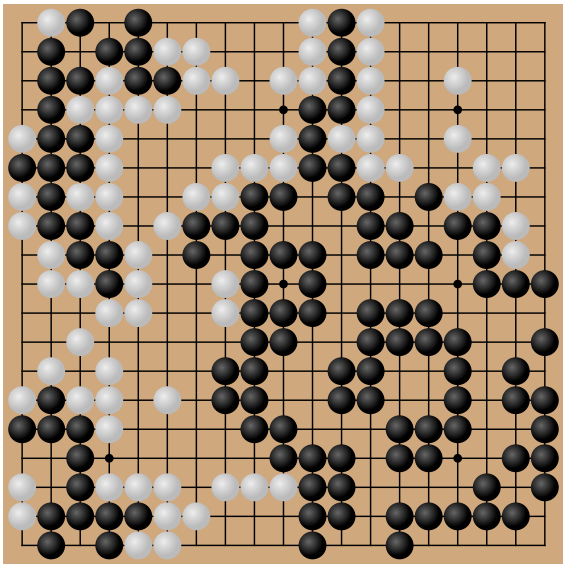
(a) Move 41: the frame of the cyclic group is set up (lower right middle)



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

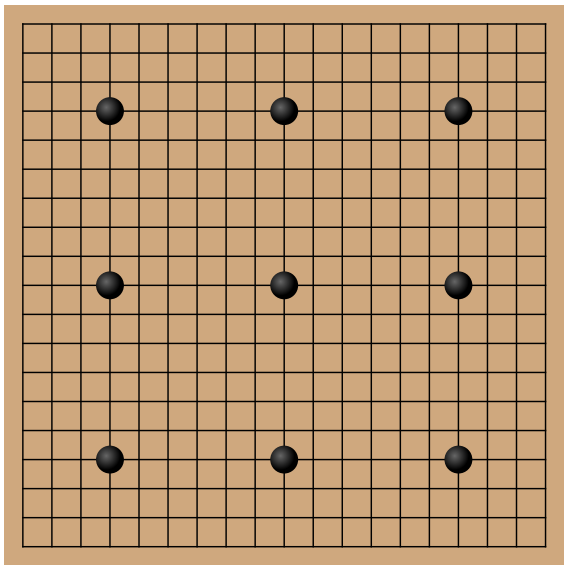


(c) Move 189: although victim's cyclic group has a number of liberties left, it can no longer avoid capture and the game is decided.

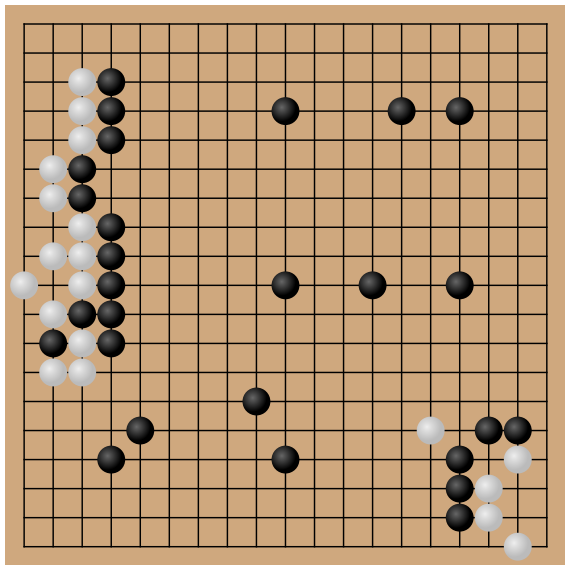


(d) Move 237: after nearly 40 more moves the cyclic group is captured. Victim realizes game is lost and resigns.

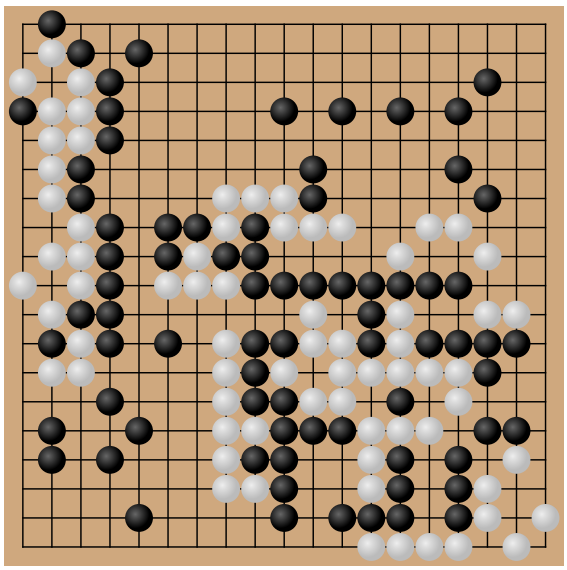
Figure G.3. Human (black) beats JBXKata005 (white).



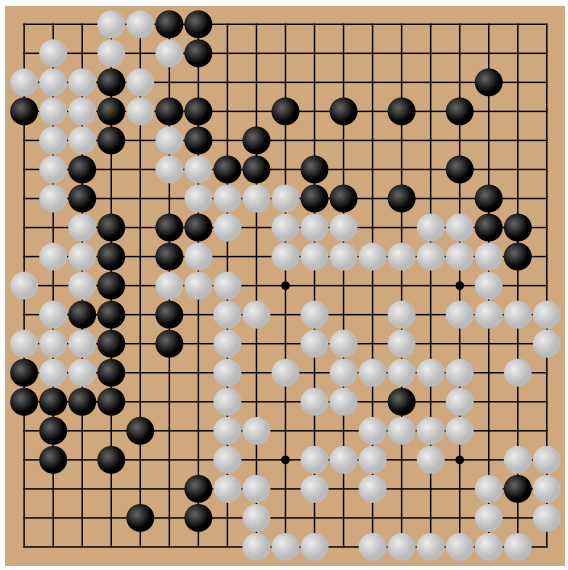
(a) Move 0: starting board position. In contrast to a normal game starting with an empty board, here the victim received 9 handicap stones, giving it an enormous initial advantage.



(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 G.4. Human (white) beats JBXKata005 (black), giving it 9 handicap stones.

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 5.2a). This provides evidence search increases robustness. However, there are potential confounders. First, the 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 H.1 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.

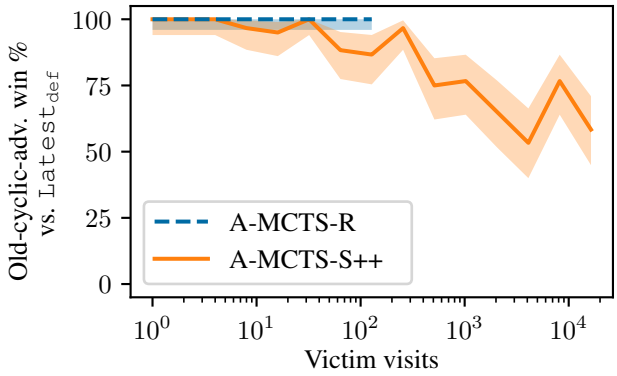


Figure H.1. We evaluate the win rate of an older version of our cyclic-adversary playing with 200 visits / move against $\texttt{Latest}_{\texttt{def}}$ playing with varying amounts of search. The cyclic-adversary was trained for 498 million steps and had a curriculum whose victims only went up to 256 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.

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 to asking "how much search is needed for the victim to play correctly in this position?" (ignoring stochasticity in the search and in the move choice, which depends on the chosen temperature hyperparameter).

Results are shown in Table H.1. "✓" 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 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

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 H.1. 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.

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 H.5 and H.6) further by varying the amount of search, as in the preceding experiment. Results are shown in Table H.2. 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.

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.

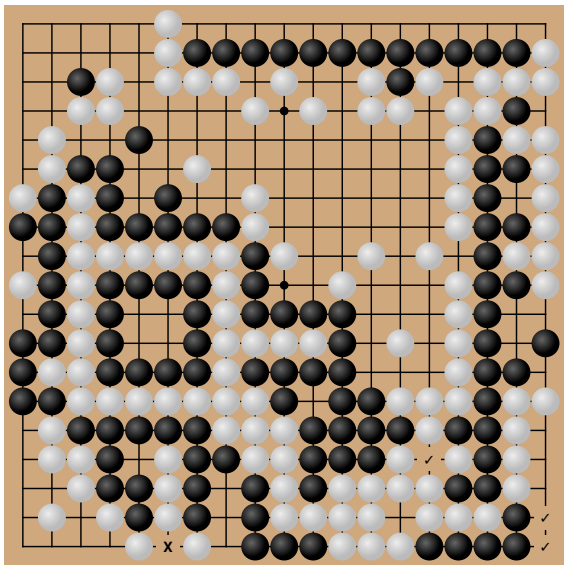
We show the board positions analyzed in Figures H.2, H.3, and H.4. Moves are marked according to the preceding table, though note the markings for wrong and inconclusive moves are non-exhaustive. Full game records are available on our website.

To further confirm that reasonable amounts of search are not sufficient for robustness, we examined 5 positions from random games where our adversary beat a KataGo victim with 1 million visits. We determined the last chance for victory as above, and gave KataGo 1 *billion* visits to find a correct move. The positions are shown in Figure H.7 and full game

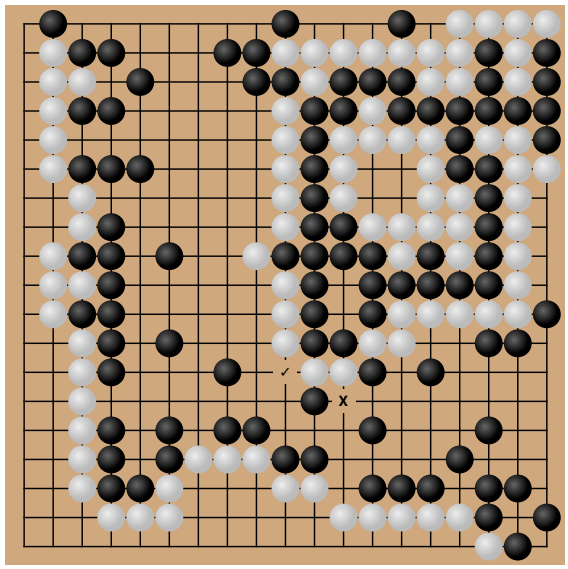
records are available on our website. In all 5 positions, a wrong game-losing move was still selected. This is around two orders of magnitude beyond the number of visits used in typical computer vs. computer tournaments, let alone normal games against humans. Consequently, short of revolutionary progress in hardware, we are unlikely to be able to solve this vulnerability through increasing search alone.

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

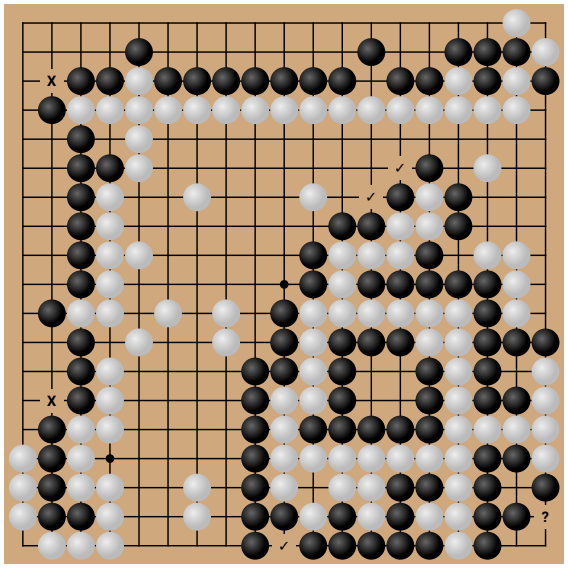
Table H.2. Examining how much search is needed to make the correct move in positions with a 3 move difference between the deciding move and capture. Similar to the preceding table, the original victim had 1.6k visits. Higher visits again leads to more correct moves and improved robustness.



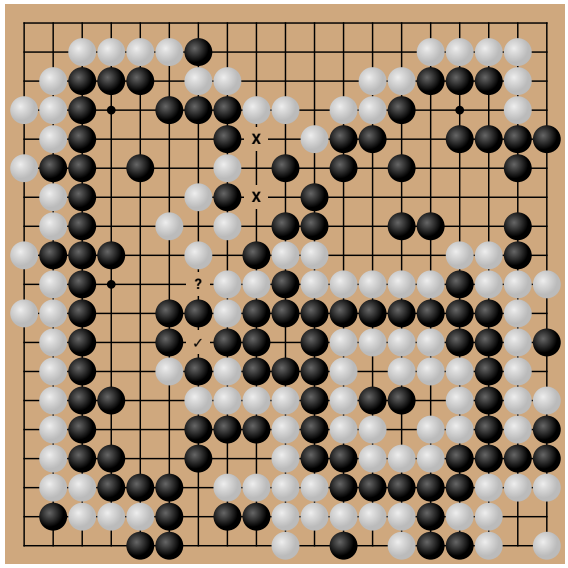
(a) White to play.



(b) Black to play.

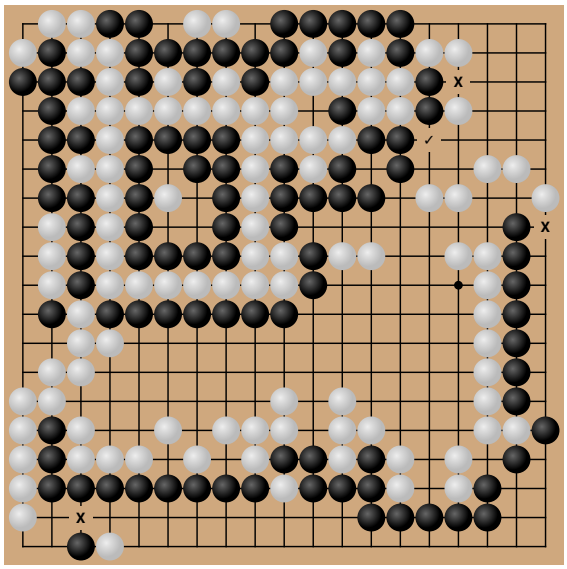


(c) White to play.

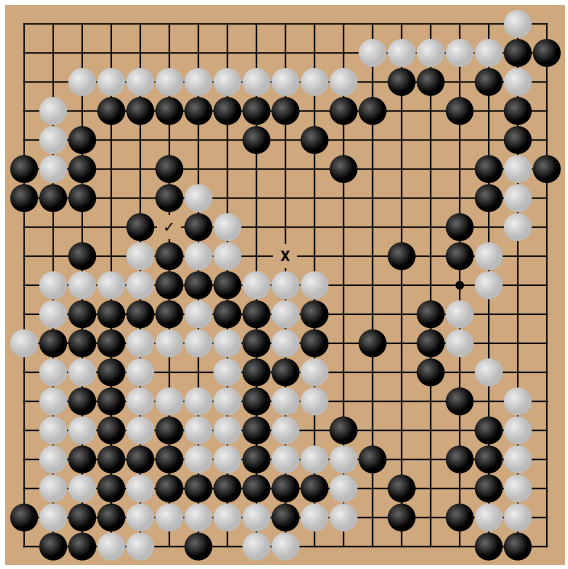


(d) Black to play.

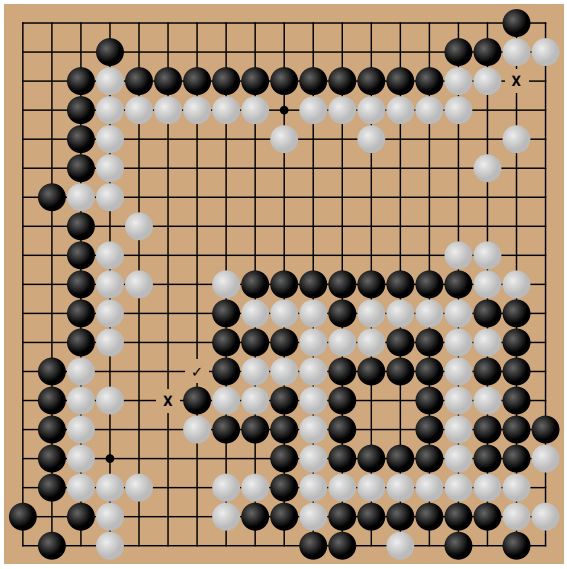
Figure H.2. Part 1 of positions analyzed with varying levels of search. Correct moves are marked "✓", and non-exhaustive examples of incorrect and inconclusive moves that the victim likes to play are marked with "✗" and "?" respectively.



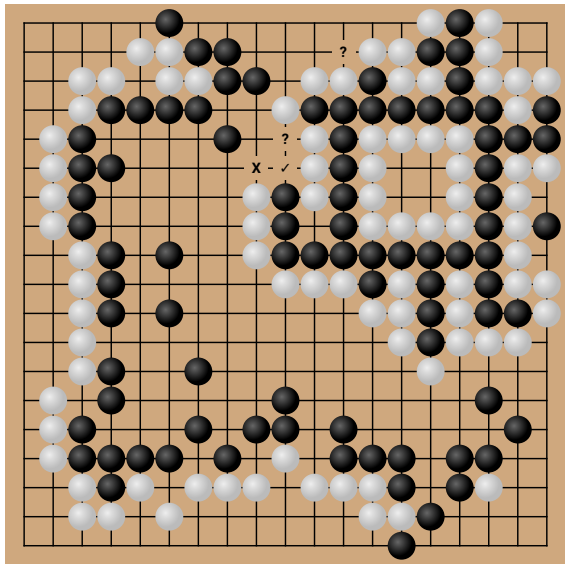
(a) White to play.



(b) Black to play.

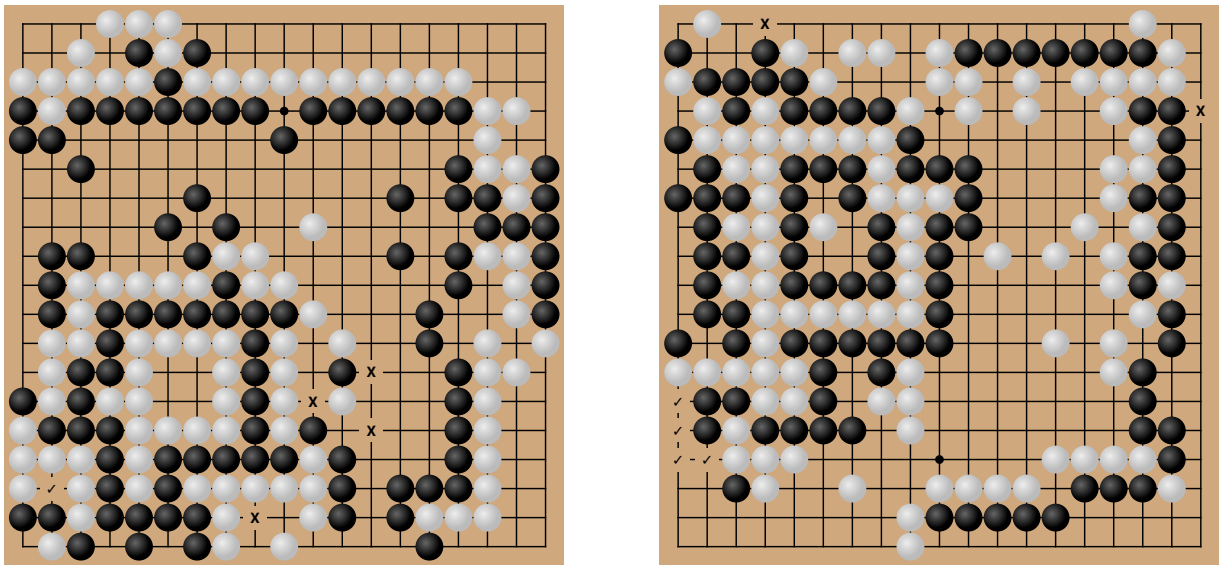


(c) White to play.



(d) Black to play.

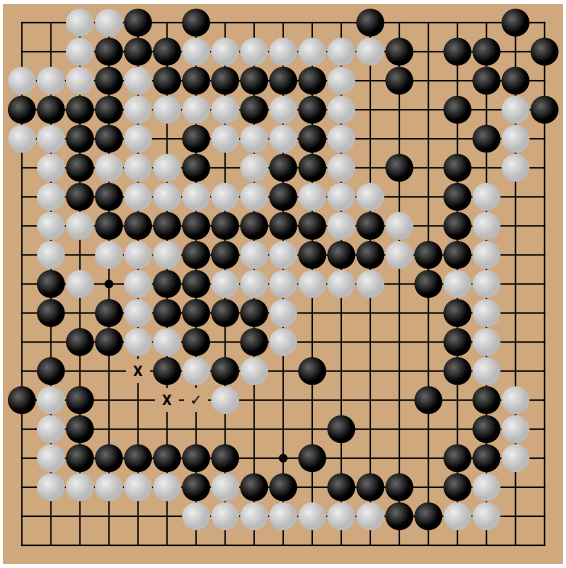
Figure H.3. Part 2 of positions analyzed with varying levels of search. Correct moves are marked "✓", and non-exhaustive examples of incorrect and inconclusive moves that the victim likes to play are marked with "✗" and "?" respectively.



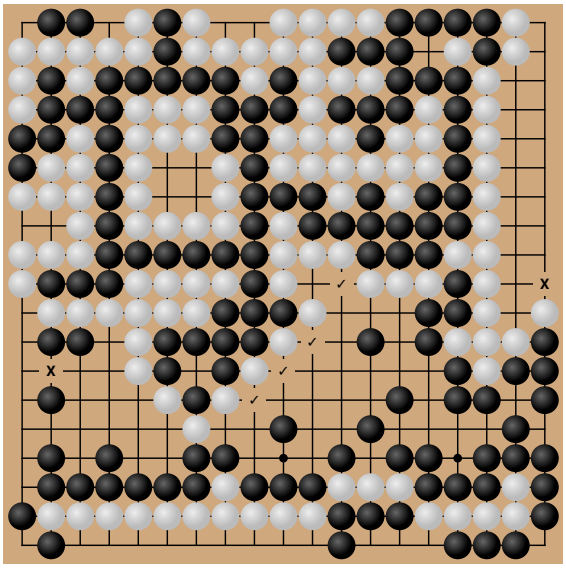
(a) Black to play.

(b) White to play.

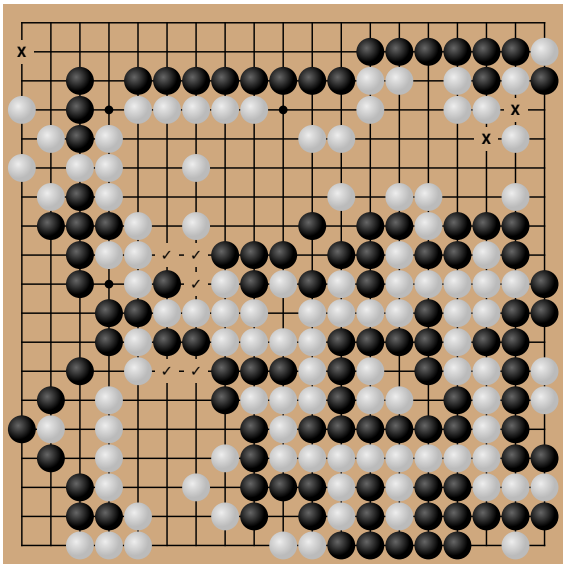
Figure H.4. Part 3 of positions analyzed with varying levels of search. Correct moves are marked "✓", and non-exhaustive examples of incorrect and inconclusive moves that the victim likes to play are marked with "✗" and "?" respectively.



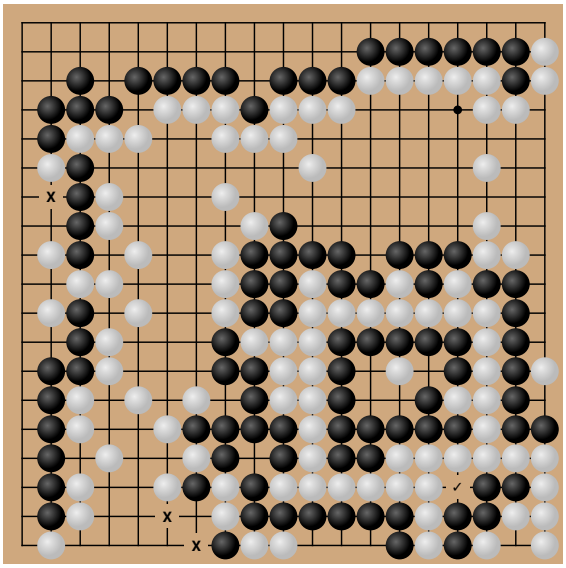
(a) Black to play.



(b) White to play.

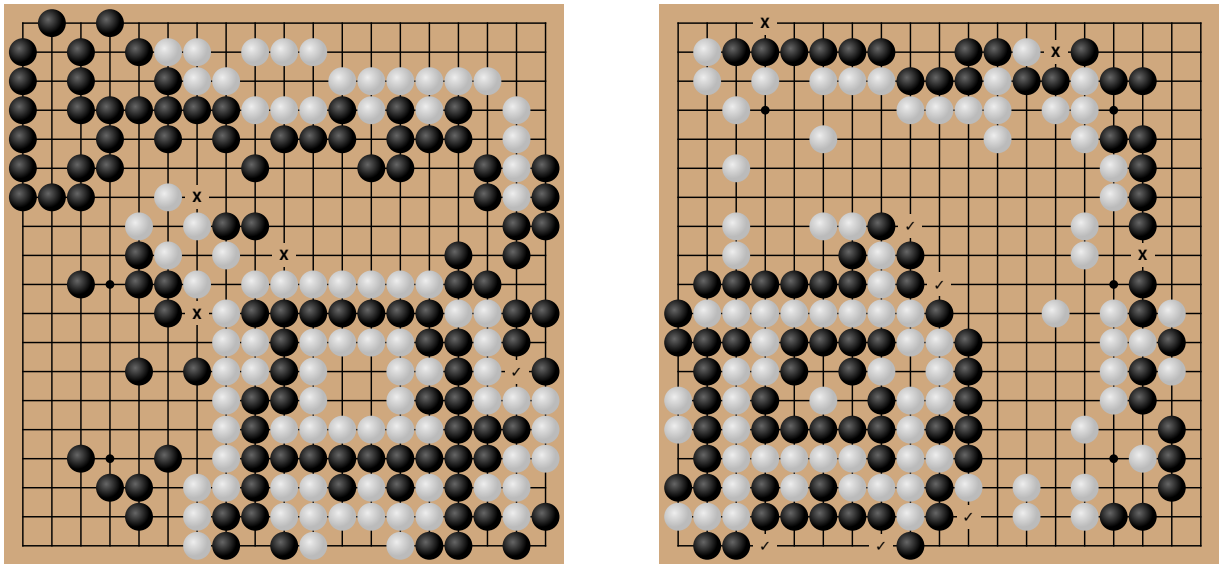


(c) White to play.



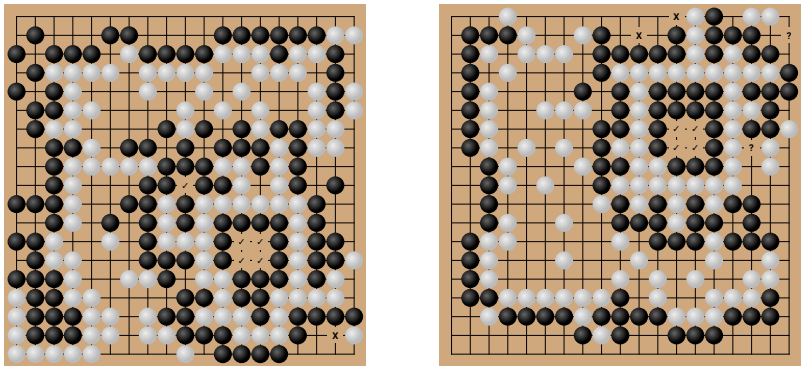
(d) White to play.

Figure H.5. 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.

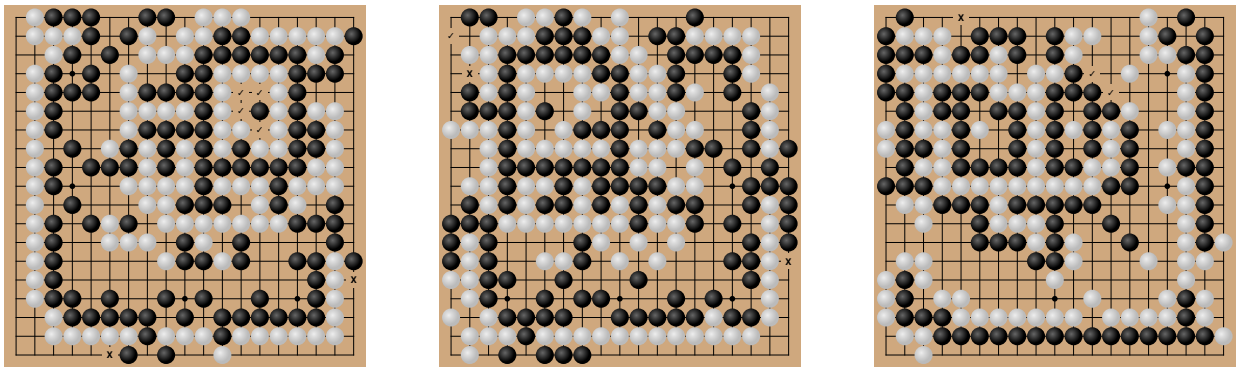


(a) Black to play. (b) White to play.

Figure H.6. 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.



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



(c) Black to play. (d) Black to play. (e) White to play.

Figure H.7. Positions that are not solved with even 1 billion visits.

I. A-MCTS-R Analysis

In this section we further examine the effect of A-MCTS-R to more accurately model the victim compared to A-MCTS-S++ by simulating the victim's search as well as its policy. We find A-MCTS-R yields modest but consistent improvements, even without simulating the entirety of victim search.

Position Analysis We examined 50 games the A-MCTS-S++ adversary played against a victim with 10^6 visits/move, looking for cases the adversary lost but had a winning advantage. Out of these 50 games, the adversary lost 9, and a Go expert (Kellin Pelrine) found in 5 of those it had a winning position (i.e. with optimal subsequent play from both sides, the adversary would be guaranteed to win). These are positions where a cyclic group was created and trapped, but then at some point the adversary made a mistake that let the victim break the encirclement and escape.

We took the last position where the adversary still had a guaranteed win and examined if A-MCTS-R would secure this win where A-MCTS-S++ had failed to. The positions are shown in Figure I.2. We simulated the victim with $1, 10, 10^2, 10^3, 10^4, 10^5$ visits in A-MCTS-R and checked if each level of simulated victim search would lead to a correct move played by the adversary. In all cases, including the original games, the adversary uses 600 visits.

Results are shown in Table I.1. The 10^0 case means the adversary simulated the victim without search by sampling directly from the policy head, i.e. using A-MCTS-S. In these cases the adversary makes a game-losing move, replicating the results of the original games. By contrast, the adversary playing with A-MCTS-R eventually find the correct move in each position. The number of simulated victim visits needed varies from only 10 to 10^5. The results here are monotonic, i.e., if the correct move is found with one number of simulated victim visits then it is maintained with a higher number.

Considering these games represent 5 out of the 9 total losses in 50 games against this victim, these results suggest A-MCTS-R can produce a solid increase in win rate. While fully simulating a high-search victim is prohibitively expensive computationally, the examples here show a benefit can be gained against a high-search victim even with a low-search simulated one. In such regimes, this low-search simulated victim does not substantially increase the computation needed – for example, against a victim with 10^6 visits and an adversary with 600 visits, simulating 100 victim visits would lead to approximately $600 * 100 = 60,000$ additional visits per move, which is small compared to the 10^6 victim visits. Consequently, this might improve adversary training, and even in evaluation alone can provide a stronger challenge for robustness. We therefore recommend this strategy for future work with high-visit victim systems.

Visits	Game 0	Game 1	Game 2	Game 3	Game 4
1	✗	✗	✗	✗	✗
10	✓	✗	✓	✗	✗
100	✓	✗	✓	✓	✗
1,000	✓	✗	✓	✓	✗
10,000	✓	✗	✓	✓	✓
100,000	✓	✓	✓	✓	✓

Table I.1. Varying the number of victim visits simulated in the recursive part of A-MCTS-R, in 5 positions where the adversary blundered (✗). In each case, with enough simulated visits the blunder is avoided (✓), and in several cases even a small number is sufficient.

Match Analysis A potential limitation of the analysis above is that it only considers specific positions rather than full games. In Figure I.1, we show results of our adversary with 503 million training steps and 128 visits against a victim with 8192 visits. This is a weaker version of our adversary compared to the 545 million training steps with 600 visits that we use in the main experiments. The adversary here has an A-MCTS-S++ win rate well below 100%, so we can look at the impact of varying the A-MCTS-R simulated victim visits. Each data point is the result of 48 games.

We see that performance trends upwards, providing additional evidence that a low amount of recursive victim simulations can improve the winrate against a significantly higher visit real victim. We note that the results here are not perfectly monotonic, but this is likely due to the limited sample size. In future work, we plan to run a similar experiment with more samples against a 10^6 visit victim to further confirm these results.

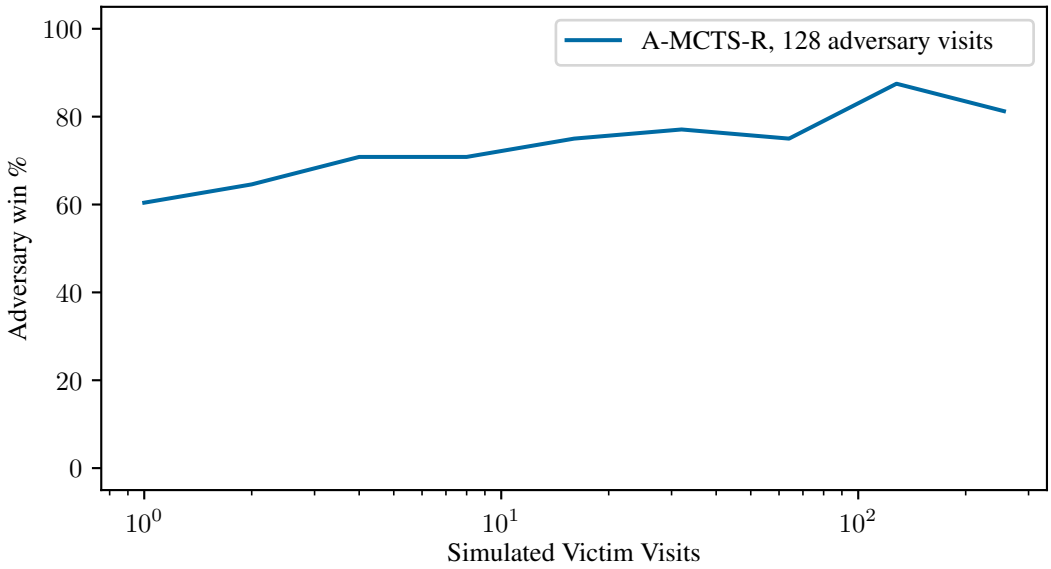
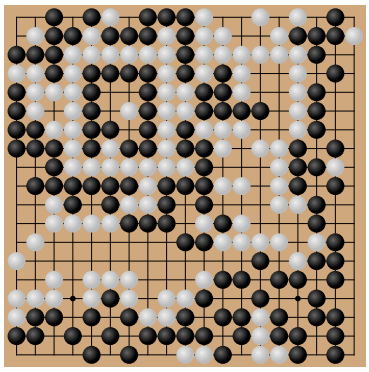
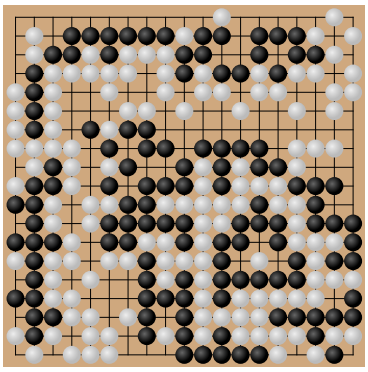


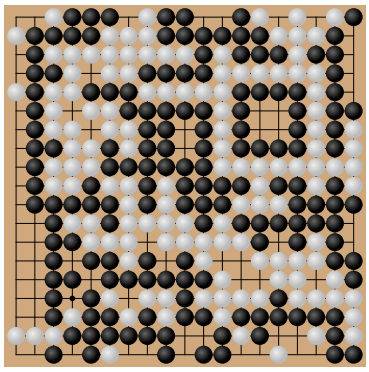
Figure I.1. Testing adversary win rate with varying levels of A-MCTS-R simulated victim visits. Even while well below the actual victim's 8K visits, A-MCTS-R can provide some improvement.



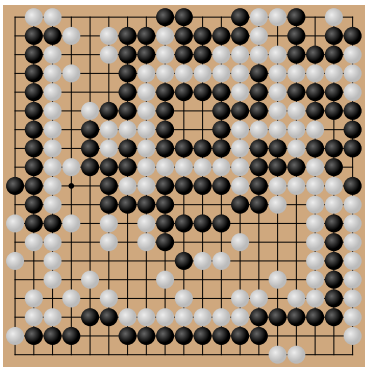
(a) Black to play.



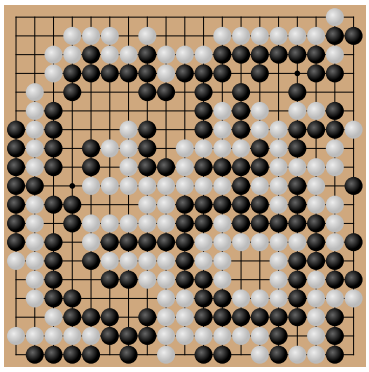
(b) Black to play.



(c) Black to play.



(d) Black to play.



(e) White to play.

Figure I.2. Positions where the A-MCTS-S++ adversary blundered. We found A-MCTS-R does better.

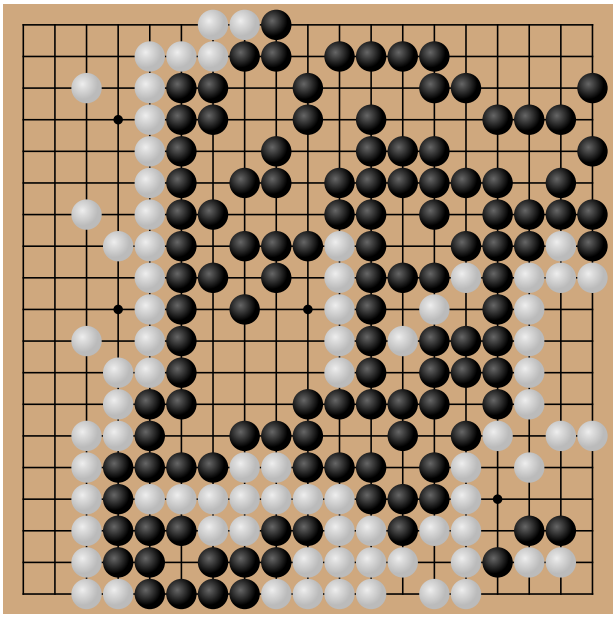
J. Human Experiments and Analysis

J.1. Humans vs. Adversarial Policies

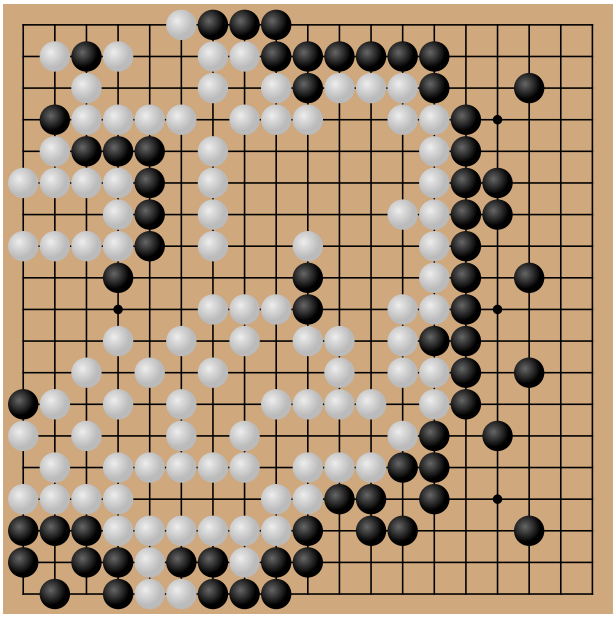
An author who is a Go novice played manual games against both the strongest cyclic-adversary from Figure 5.1 and the strongest pass-adversary from Figure F.1. In the games against the pass-adversary, the author was able to achieve an overwhelming victory. In the games against the cyclic-adversary, the author won but with a much smaller margin. See Figure J.1 for details.

We also set up a bot running our strongest cyclic-adversary on the KGS Go server under the username `Adversary0`. This bot was available for the public to play for a period of a month. It played over 2300 games and was ranked around 17kyu. This is further evidence that our cyclic-adversary plays at the level of a novice Go player.

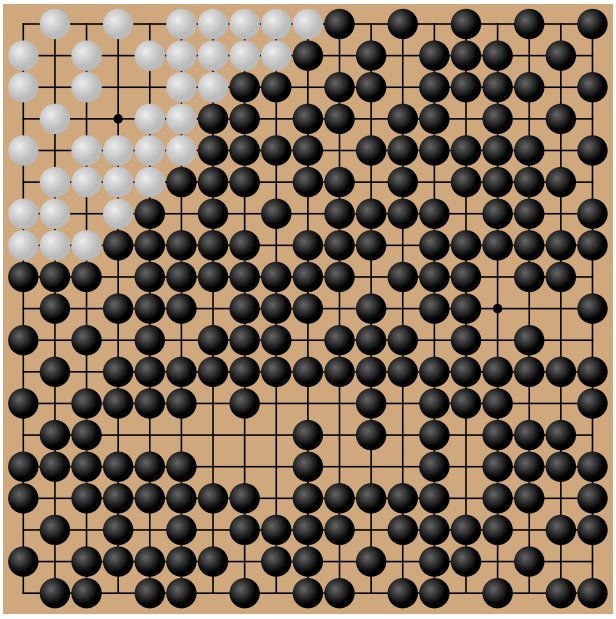
Our evaluation is imperfect in one significant way: the adversaries did not play with an accurate model of their human opponents (rather they modeled their opponents as `Latest` with 1 visit). However, given the limited transferability of our adversaries to different KataGo checkpoints (see Figure 5.1, Figure F.1, Appendix F.4, and Appendix L.1), we conjecture that our adversaries would not win significantly more even if they had access to an accurate model of their human opponents.



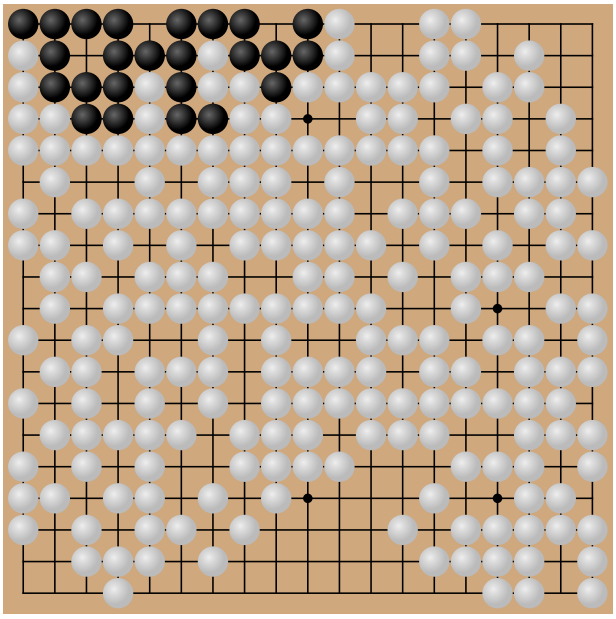
(a) An author (B) defeats the strongest cyclic-adversary from Figure 5.1 by 36.5 points. Explore the game.



(b) An author (W) defeats the strongest cyclic-adversary from Figure 5.1 by 65.5 points. Explore the game.



(c) An author (B) defeats the strongest pass-adversary from Figure F.1. Explore the game.



(d) An author (W) defeats the strongest pass-adversary from Figure F.1 using A-MCTS-S++. Explore the game.

Figure J.1. Games between an author of this paper (who is a Go amateur) and the strongest adversaries from Figure 5.1 and Figure F.1. 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.

J.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 1.1a) 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 J.2 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 J.2a 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 J.2b), 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 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.

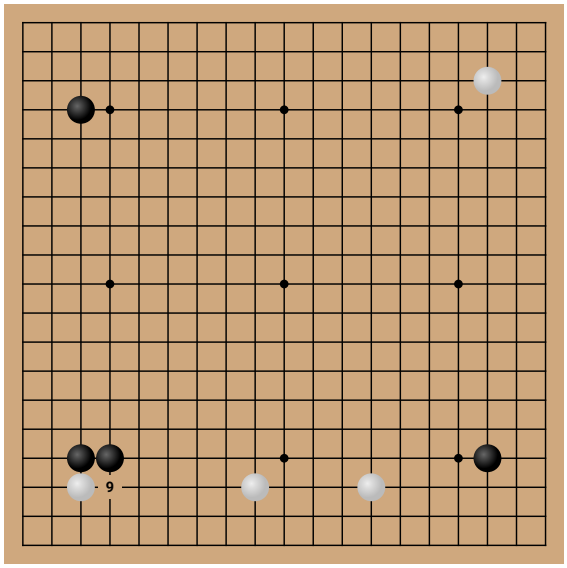
On move 74 (Figure J.2c), the adversary begins mobilizing its "dead" stones to set up an encirclement. Over the next 100+ moves, it gradually surrounds the victim in the top left. A key pattern here is that it leads the victim into forming an isolated group that loops around and connects to itself (a 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 J.2d), 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.

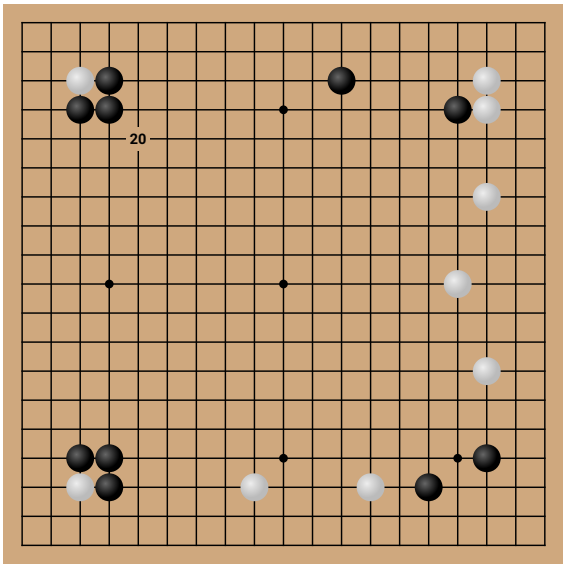
Adversary loss analysis In all cases examined where the adversary lost, it did set up a cycle group, or a cycle group with one stone missing, which is likely still a cycle as perceived by the neural net of the victim (see Figure J.2d for an example where it misjudges such a position).

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 J.3. 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.

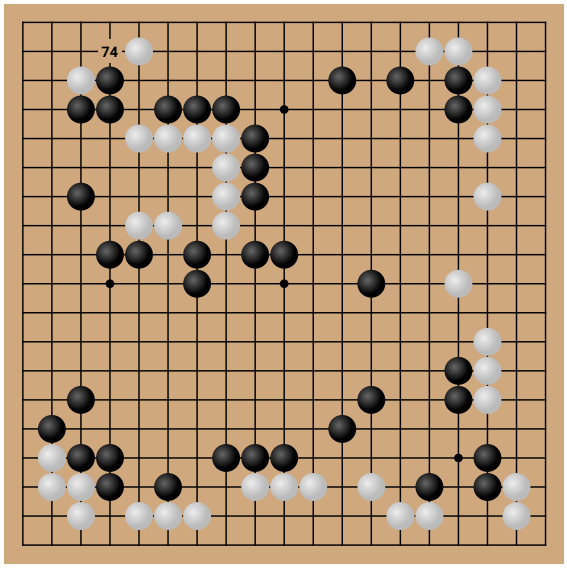
In the other six cases the adversary never has any clear opportunity to capture the cycle group. This is because the victim breaks through the attempted encirclement in some fashion, either by capturing some surrounding stones or simply connecting to one of its other groups. Although this could indicate 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.



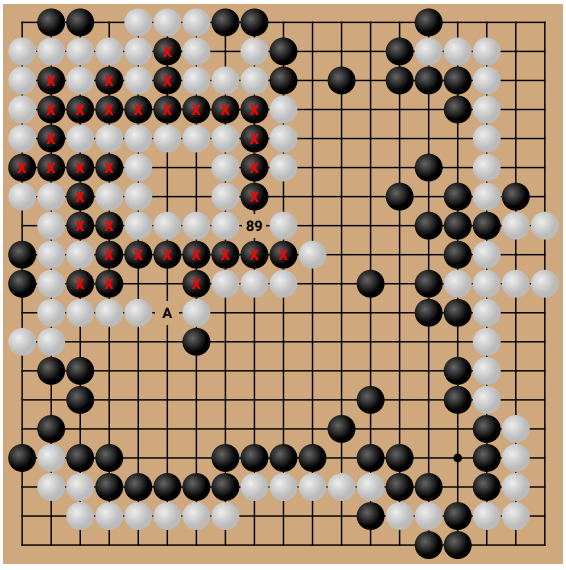
(a) Move 9: after this move victim already has the advantage, if it were robust.



(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 J.2. 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.

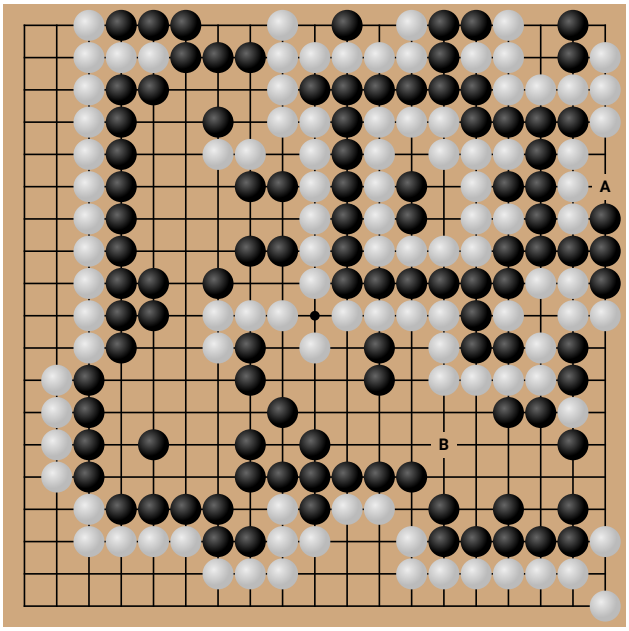


Figure J.3. A game the cyclic-adversary (white) lost. The adversary could take a decisive lead by capturing at A, but instead plays B and lets the victim (black) save their group.

J.3. Human Analysis of the Adversary's Training Progression

To understand the evolution of our adversary over the training process, we randomly sampled 5 games against each of 4 victims at approximately 10% training step intervals up to 545 million steps. The 4 victims were $\texttt{cp39}_{\texttt{def}}$ with no search, $\texttt{cp127}_{\texttt{def}}$ with no search, $\texttt{Latest}_{\texttt{def}}$ with no search, and $\texttt{Latest}_{\texttt{def}}$ with 4096 visits. These correspond to the victims in Figure 5.1. The full sampled games are available on our website. An expert Go player on our team analyzed the games, looking for patterns in how the adversary won (or lost).

We first note that in all cases against a defended victim, the adversary wins as a result of a key capture which its opponent somehow misjudged. In no game that we analyzed did the victim simply play too conservatively and lose without a fight.

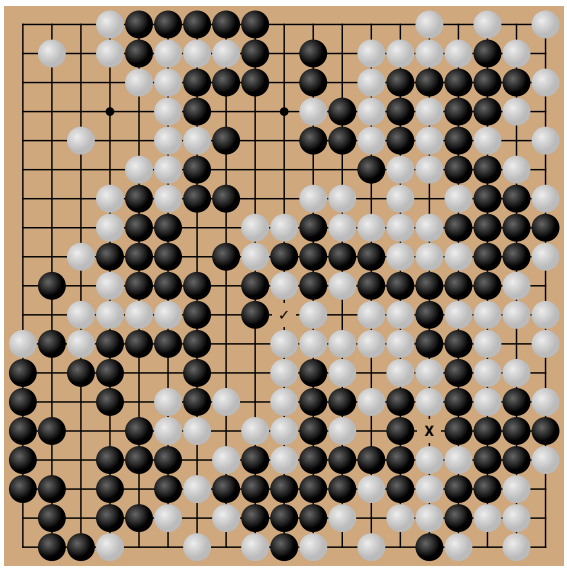
Early in the training process, the adversary quickly achieves a good winrate against $\texttt{cp39}_{\texttt{def}}$, but is only gradually improving against the other victims. Here the attack is not very consistent. When the adversary is successful, the exploit typically has low moves-to-capture (defined in Appendix H). Our analysis did not reveal a consistent strategy understandable by humans. Overall, these early training games suggest that without search, KataGo's early checkpoints have significant trouble accurately judging liberties in a variety of positions.

Around 220 million training steps, the adversary is winning consistently against $\texttt{cp127}_{\texttt{def}}$ and $\texttt{Latest}_{\texttt{def}}$ with no search. We now see consistent patterns. The adversary lures the victim into creating a large group with few liberties. It sets up numerous kos and many other single stones in atari (i.e., could be captured by the victim on the next move) around that group. The victim does not realize it is about to run out of liberties, and the adversary captures the large group, leading to a win. The finishing blow is often a move where the victim fills one of their own last two liberties in order to make sure two of their groups stay connected, but this leads to losing everything. An example is shown in Figure J.4.

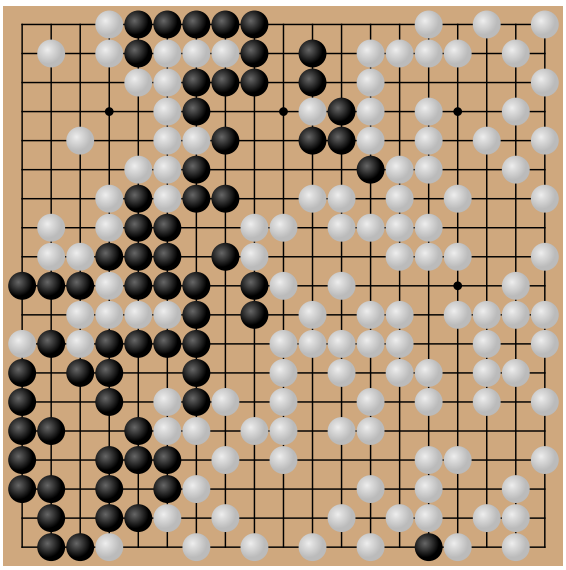
This attack has similarities to the cycle attack, in that it attacks a large group that the victim seems unable to judge accurately. However, there is a far stronger emphasis on kos and stones in atari, and the moves-to-capture is generally minimal (i.e., up until the actual capture the victim could save its group, or the key portion of it). The targeted group is sometimes a cyclic group but more often not. Furthermore, unlike later versions of the cycle attack, this attack seems inadequate to beat a very moderate amount of search – the adversary still has near 0% winrate against $\texttt{Latest}_{\texttt{def}}$ with 4096 visits.

At the same time, the win percentage against $\texttt{cp39}_{\texttt{def}}$ begins to fall. Analyzing the games, we hypothesize that this is not due to $\texttt{cp39}_{\texttt{def}}$ judging these positions more accurately, but that it more frequently plays defensive moves when far ahead. Compared to earlier attacks, this one requires giving the opponent an enormous point lead, because to set up the large target group the adversary gives up most of the board. In addition, leaving many stones in atari provides numerous straightforward defensive captures that happen to save the large group. We observe in practice that $\texttt{cp39}_{\texttt{def}}$ makes many of these captures, as well as other straightforward defensive moves, even when they are not mandatory and there are other places on the board that would give more points. At the same time, although the adversary's win percentage falls against this victim, it never goes to 0; the adversary still wins a non-trivial number of games. In combination, this suggests that $\texttt{cp39}_{\texttt{def}}$ does not clearly see the trap being created or intentionally defend against it, but rather just plays many defensive moves that often happen to save it. This has similarities to the behavior of human beginners, who often make many random defensive moves because they cannot tell clearly if a defense is needed. As in $\texttt{cp39}_{\texttt{def}}$'s games, in many cases they are unnecessary, but in some cases they avert a disaster.

Around 270 million steps and beyond, the adversary is mostly using the cycle attack, only occasionally making non-cycle attacks. It is doing very well against $\texttt{cp127}_{\texttt{def}}$ and $\texttt{Latest}_{\texttt{def}}$ without search. However, until nearly 500 million steps the adversary still struggles against opponents with search. We hypothesize the main factor is that the moves-to-capture is too low to fool a victim with this level of search - successful attacks against this victim seem to have a moves-to-capture of at least 3, while the attacks produced at this stage in the training still often have fewer (frequently due to the many kos and stones in atari, which seemed helpful for the previous attack but not always helpful here). Towards the end of the training, the adversary starts producing cycles with higher moves-to-capture, and starts to win consistently against stronger victims with more search.



(a) Black should have played at the location marked "✓", giving up some of their stones on the bottom in order to secure the main group. But instead black played the move marked "✗", which makes their entire right side capturable on white's next move.



(b) Both players exchange a few moves on the left side, giving the victim more chances to escape, but it doesn't see the danger. At this point White captures everything on the right, leading to victory by a large margin.

Figure J.4. The losing move and decisive capture in a game between $\texttt{Latest}_{\texttt{def}}$ (black) and the adversary with 220 million training steps (white).

K. Activation Analysis

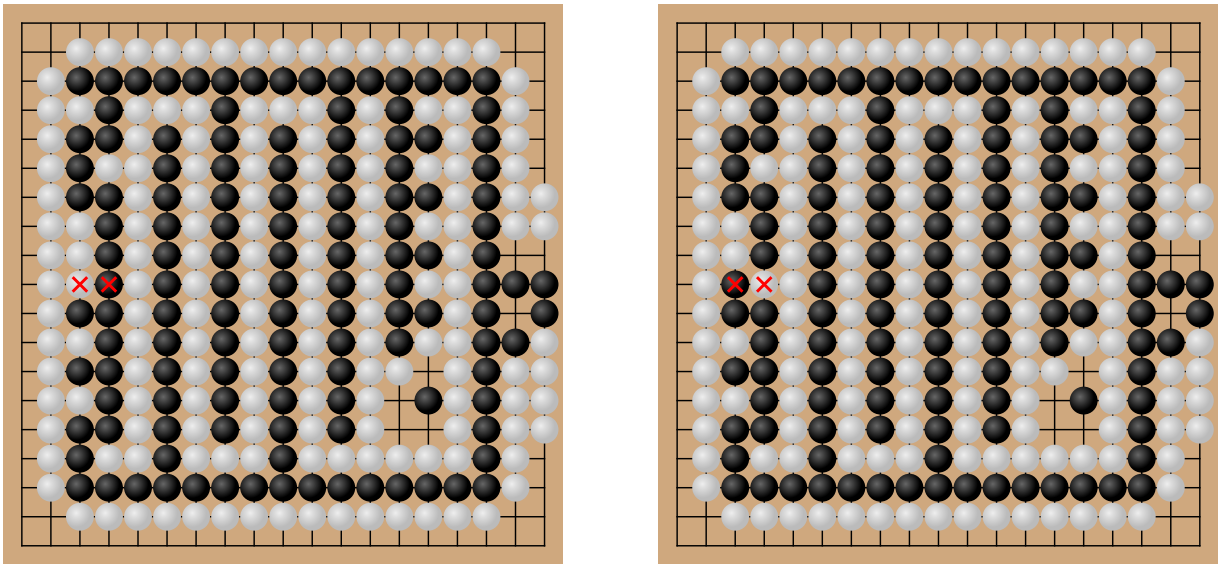
We examine activations in different positions and with different versions of KataGo to try to identify where and ultimately why the system misjudges cyclic positions.

Setup We examine the activations of each main convolutional layer in the network. Specifically, using the TensorFlow version of KataGo (before KataGo switched to using PyTorch in KataGo version 1.12) we look at "conv1" for the input layer, then the equivalent "rconvK" in subsequent layers (where K ranges from 2 to 41), then the final "trunk" layer. These layers are described in detail in Appendix A of the KataGo paper (Wu, 2019).

We examine positions with a cyclic group, and minimal perturbations of such a position where the cycle is broken or incomplete. We consider a manually constructed position which has few moves available and low complexity (Figure K.1), an example from a real game played by an author against KataGo (Figure K.2), and an example based on a game played by our adversary against KataGo (Figure K.3). In each case a position and perturbation is chosen such that the game state is mostly unchanged (connected groups, status of all groups, best move, player in the lead, etc.) but the cycle is broken. The perturbation between the two positions is also made minimal by either swapping the color of a single stone or a pair of adjacent stones, or moving a single stone one space over.

In the first two cases the cyclic group is dead and the victim is doomed, and in the last case it is alive and the victim currently has a winning position. In the last case, the two positions themselves are a slight modification of the real game played by KataGo and our adversary. Specifically, in the real game the victim had one of the marked stones already and played the other one that completes the cycle. Here, to avoid playing moves in the board locations we want to perturb, we have it play a nearby connecting move that does not change the game state.

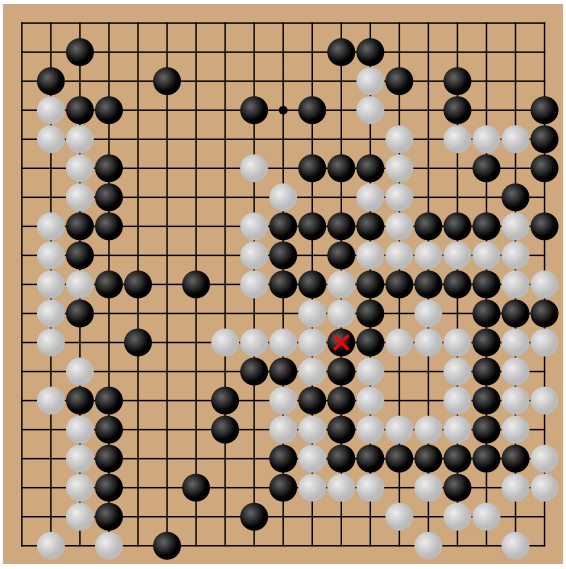
Finally, we also consider a situation where the cyclic group is already broken and the perturbation is irrelevant to both the game state and the cyclic-ness. This case is derived from the preceding one (Figure K.3), and shown in Figure K.4.



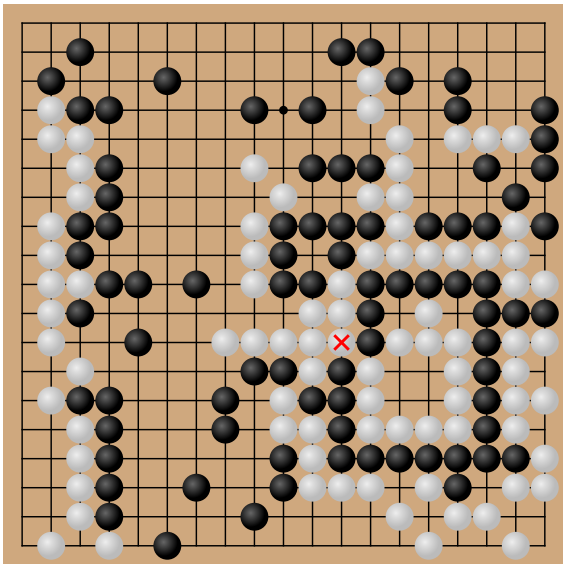
(a) Here the cycle is intact and complete.

(b) Here the cycle is broken.

Figure K.1. In this manually constructed position, by swapping the colors of the stones marked X, we can complete or break the cyclic group. The impact on the actual game state is minimal—the score does not change, nor the life and death status or liberty counts of any stones, nor the best subsequent moves. But the two boards are evaluated dramatically differently by many KataGo checkpoints.

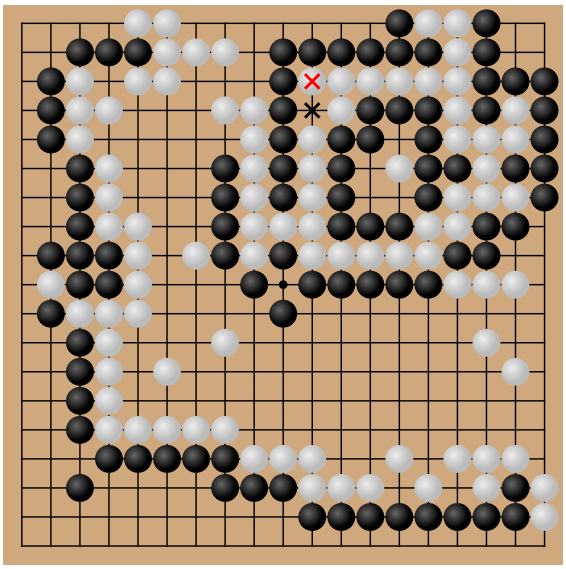


(a) Here the cycle is intact and complete.

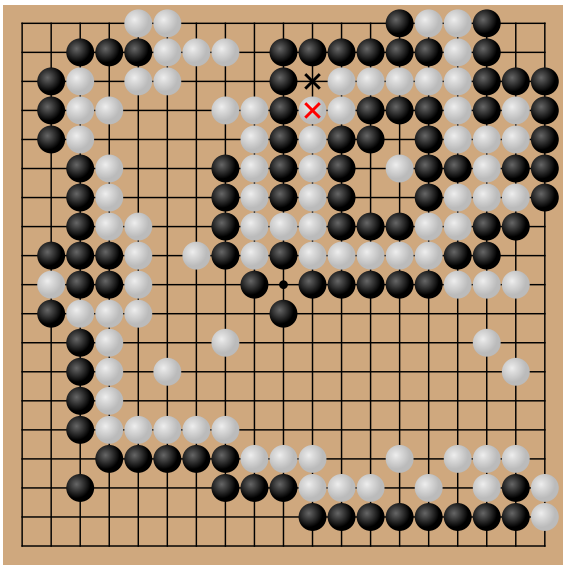


(b) Here the cycle is broken.

Figure K.2. In this real game position, by swapping the color of the stone marked X, we can complete or break the cyclic group. The impact on the actual game state is minimal—the score is changed by at most one point which does not affect who is winning, it does not affect the life and death status or liberty counts of any stones, and it does not change the best subsequent moves. But it dramatically changes the evaluation of many KataGo checkpoints.



(a) Cycle is complete.



(b) Cycle is incomplete, but still could be completed.

Figure K.3. In this position, by swapping the position of the stone marked X, we can complete or leave open the option to complete the cycle group. As in preceding positions, the impact on the actual game state is minimal—the score does not change, it does not affect the life and death status or liberty counts of any stones, and it does not change the best subsequent moves.

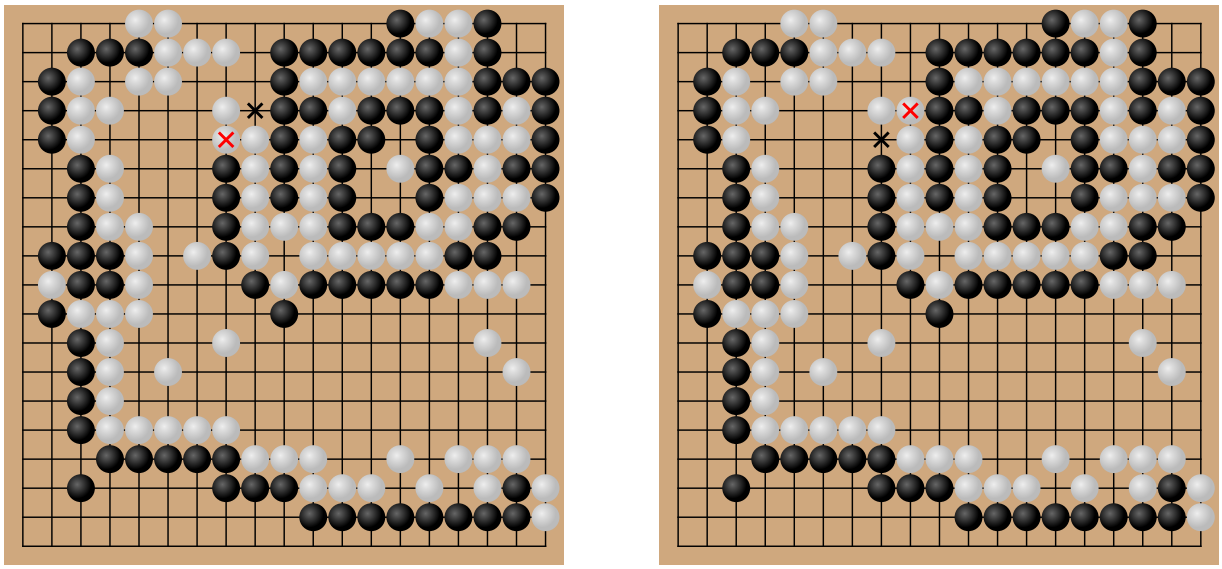


Figure K.4. These positions, derived from the same game as the preceding pair, have a near-cycle that is already broken. Swapping the most recent move X impacts neither the cyclic-ness of the position nor the game state.

Progression of models We examine a progression of model checkpoints before and after adversarial training began, to attempt to disentangle evolution due to normal training from changes specific to adversarial training. Specifically, we look at 9 checkpoints centered on `Latest`, resulting in four before it and four after. Of these, the two most recent ones, `cp559` and `cp580`, have been adversarially trained on cyclic positions and demonstrate improved performance in some of them. The specific models examined are:

1. Checkpoint 455: `b40c256-s10823908608-d2638763986`.
2. Checkpoint 468: `b40c256-s11078294784-d2707780120`.
3. Checkpoint 478: `b40c256-s11290411776-d2760978415`.
4. Checkpoint 492: `b40c256-s11574569216-d2829125899`.
5. **Checkpoint 505**: `b40c256-s11840935168-d2898845681` (`Latest`).
6. Checkpoint 522: `b40c256-s12096598272-d2984620981`.
7. Checkpoint 535: `b40c256-s12350780416-d3055274313`.
8. Checkpoint 559: `b40c256-s12604774912-d3126339815` (`cp559`).
9. Checkpoint 580: `b40c256-s12860905472-d3197353276` (`cp580`).

These were selected to approximate equal training steps, taking into account the inexact schedule of checkpoint releases. In each case we use the network to directly evaluate the positions without tree search.

Results We begin by analyzing the progression of activation magnitudes in the position of Figure K.2a. Figure K.5 shows the maximum activation per layer. We see that the maximum grows as we progress through the network. The two most recent checkpoints (and particularly the most recent one), which have adversarial training and are shown with dashed lines, exhibit some differences from the other checkpoints.

In Figure K.6 we instead look at the maximum per layer of the *difference* between the activations of `Latest` and every other checkpoint. This highlights that the two most recent networks are substantially more different from `Latest` than previous models. This suggests that adversarial training results in larger differences in the evaluation of these positions than a comparable number of time steps of usual training.

To more easily see which differences represent a substantial change, in Figure K.7 we normalize the differences by dividing by the maximum activation of `Latest` at that layer. We see there is little difference between the models up to around layer 11. After that, the most recent model gradually begins to break away, but this does not become pronounced until around layer 16 and especially 21–22. These results suggest that adversarial training may have little effect on early layers, implying that KataGo's misjudgment takes place in later layers, perhaps especially around layer 21.

As a robustness check, in addition to analysing the activations of individual neurons we also analyse the activations of entire channels. We follow a similar process to the above, but after taking the difference in activations we then average the activations within each channel. Then we examine the max over all channels of the channel-mean activation, dividing by the maximum channel-mean activation of `Latest`. In Figure K.8 we see that the checkpoints have a similar pattern up to layer 21, after which the two adversarially-trained checkpoints break away from the other models.

So far our analysis has involved per-layer summary statistics. To gain a more granular understanding, we visualized the activations within each layer directly. In Figure K.9, we plot the difference in activations between `Latest` and the most recent network (`cp580`) in layer 26. There are 256 channels each containing 19x19 activations, corresponding to the Go board. We pad to 20x20 to separate the channels slightly for clearer visualization. Brighter colors in the plot indicate a larger difference in activations.

We see 2-3 channels (especially the M4 and K15 channels) at layer 26 have a difference in activations across the entire board, shown in the visualization by the whole square representing that channel being brightly colored. This suggests these channels may be misjudging cyclic positions. Earlier layers do not exhibit comparable full-channel differences (there are

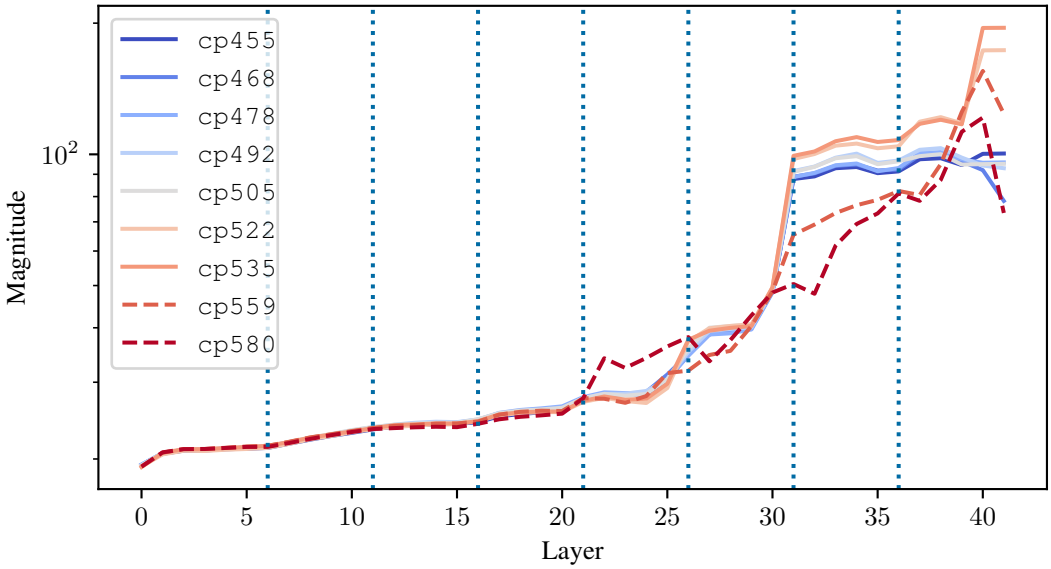


Figure K.5. The magnitude of the maximum activation per layer across 9 checkpoints. The two most recent checkpoints, which have been adversarially trained, have a markedly different curve.

some early layers with channels whose activations show whole board differences, but these are layers where the preceding analysis shows the overall magnitude difference is small).

Moreover, we find a similar anomaly in layer 26 when holding a model fixed and comparing a cyclic position to the same position with the cycle broken. In Figure K.10, we compare `Latest`'s activations on the the (a) and (b) positions from Figure K.2, finding the same channels differ between positions as between networks. We see a similar pattern in Figure K.11 comparing the two manually constructed board states from Figure K.1. Note that the colors are reversed—in our testing, the sign of the activations corresponds to whose turn it is to play. However, the channels that differ the most remain the same.

We likewise see standout differences in M4 and K15 channels in the position where the cyclic group is alive (the activations shown in Figure K.12, corresponding to the positions in Figure K.3). In this case, while still clearly visible, the channels are fainter indicating a smaller relative magnitude difference, which we hypothesize is because the group is alive independent of the illusion created by the cycle. Finally, as an additional validity check, we consider the case of Figure K.4, where the cycle has been broken in both cases. In Figure K.13 we see that M4 and K15 no longer show unusual behavior, further confirming it was due to cyclic-ness and not arbitrary perturbations.

Layer 26, where we see the unusual behavior discussed above, is later than the first layers where activation magnitude substantially differs in the preceding aggregate analysis. Accordingly, the channels at layer 26 may not be the root cause of the problem, but they do appear to be most strongly affected by it. This suggests these stand-out channels at layer 26 are a good area in the network to examine further for a better understanding of why this vulnerability occurs, as well as potentially a place for targeted interventions aimed at improving robustness.

We include interactive plots for these positions and models, which show the activation differences of all the layers, on our website.

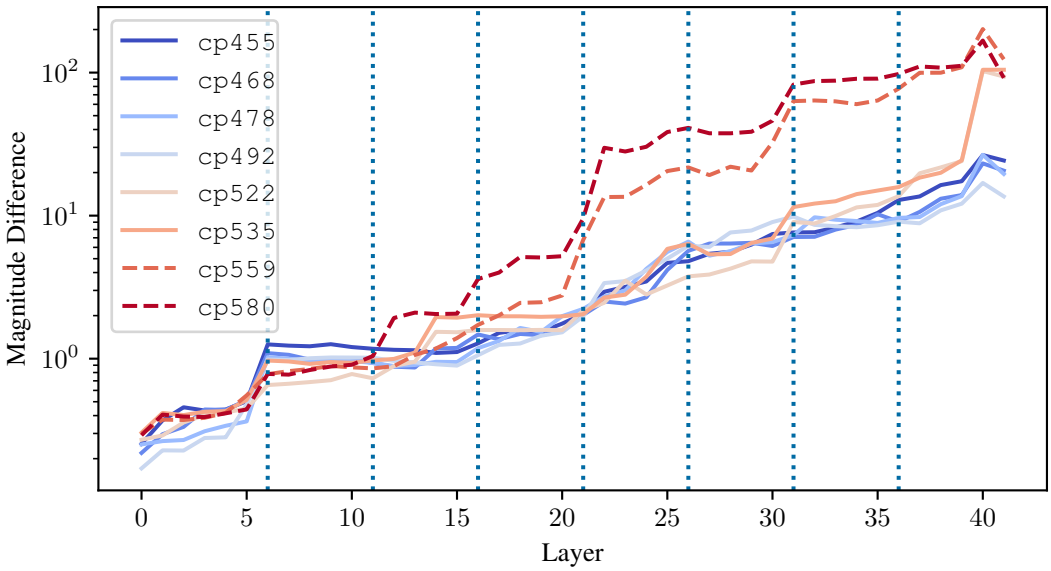


Figure K.6. The magnitude of the maximum difference from `Latest` in activations per layer across the 8 models (excluding `Latest`). Again, the two checkpoints with adversarial training differ from the rest.

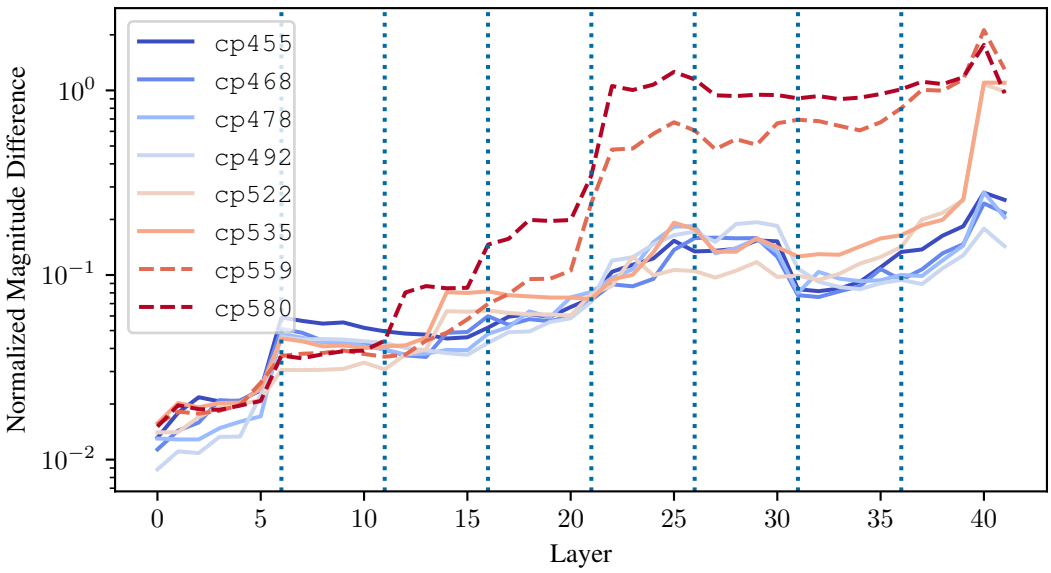


Figure K.7. The magnitude of the maximum difference from `Latest` in activations per layer, normalized by dividing by the maximum activation per layer of `Latest`. We see an especially strong difference between the checkpoints with adversarial training and the rest starting around layer 21.

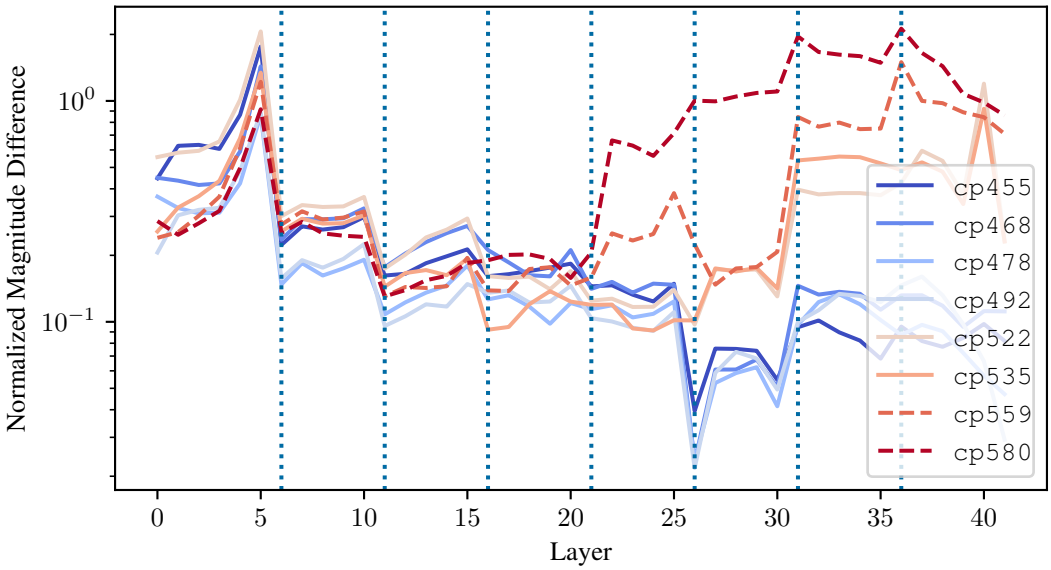


Figure K.8. The magnitude of the maximum difference from `Latest` in *channel-averaged* activations per layer, normalized by dividing by the maximum per layer of `Latest`. Similar to previous figures, sharp differences begin around layer 21.

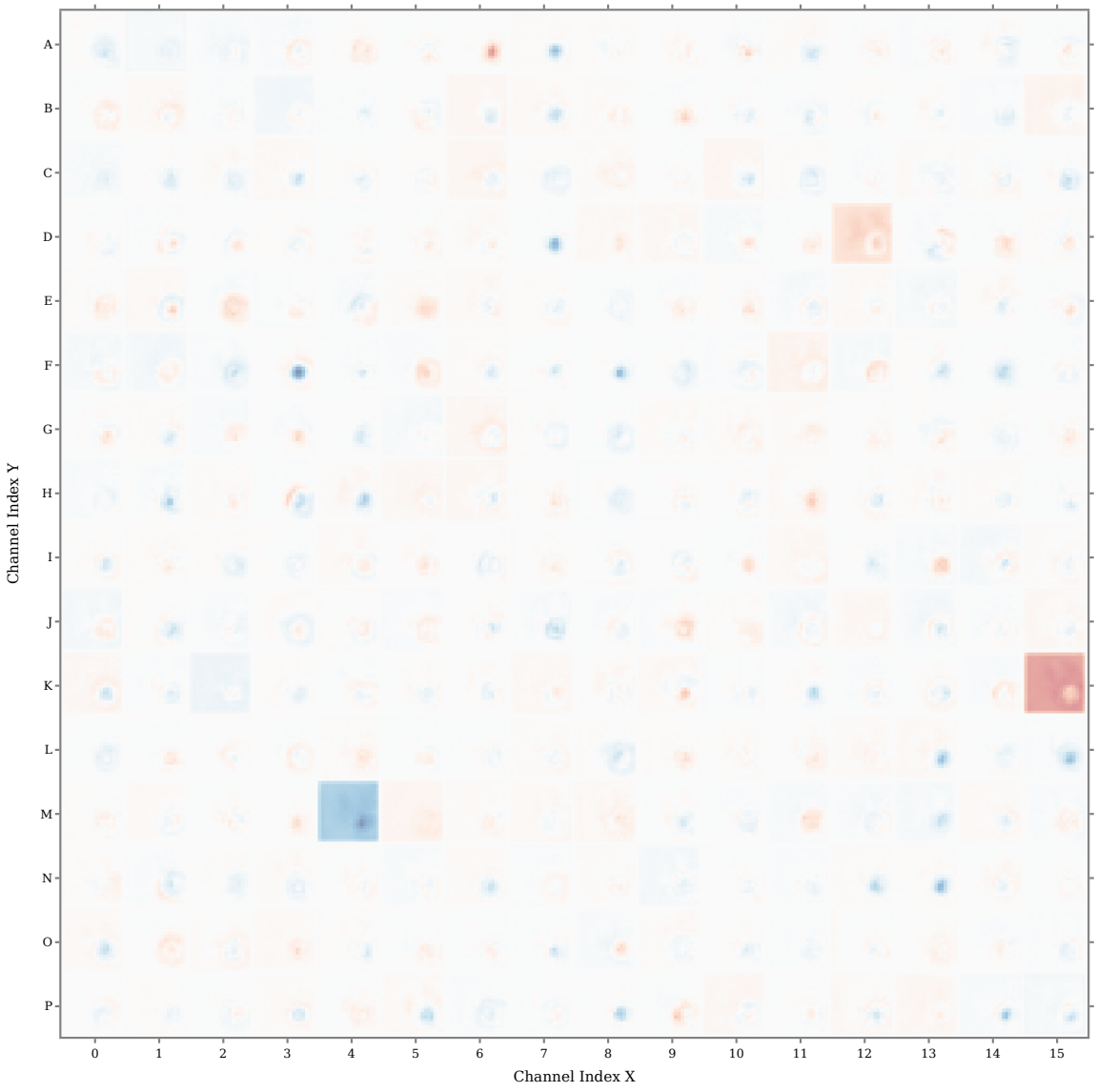


Figure K.9. The difference in activations in layer 26 between `Latest` and `cp580` in a real game position (Figure K.2).

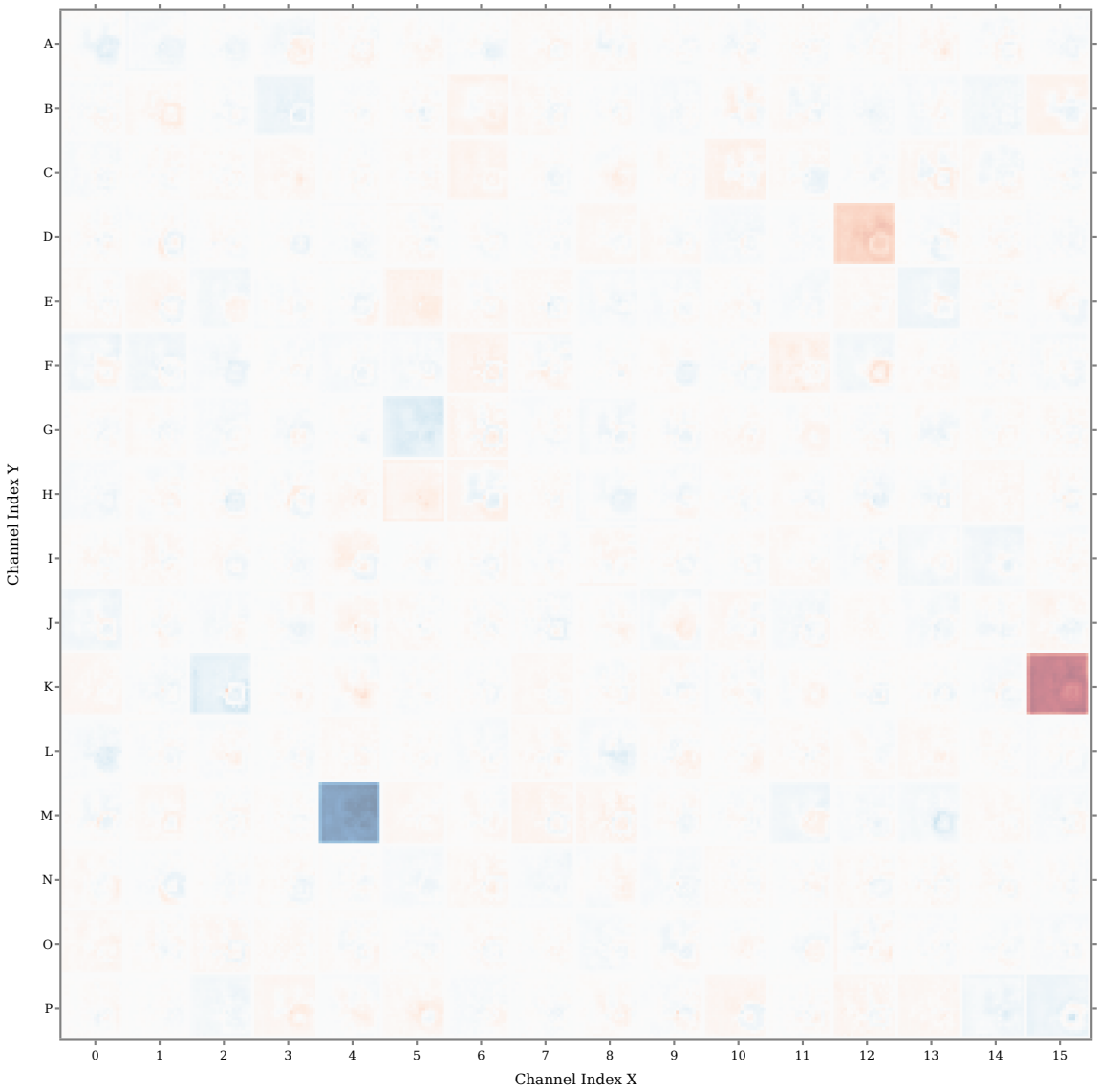


Figure K.10. The difference in activations in layer 26 of `Latest` between a real-game cyclic position (Figure K.2; the same position as the preceding Figure K.9) and a minimally perturbed version of it that breaks the cycle.

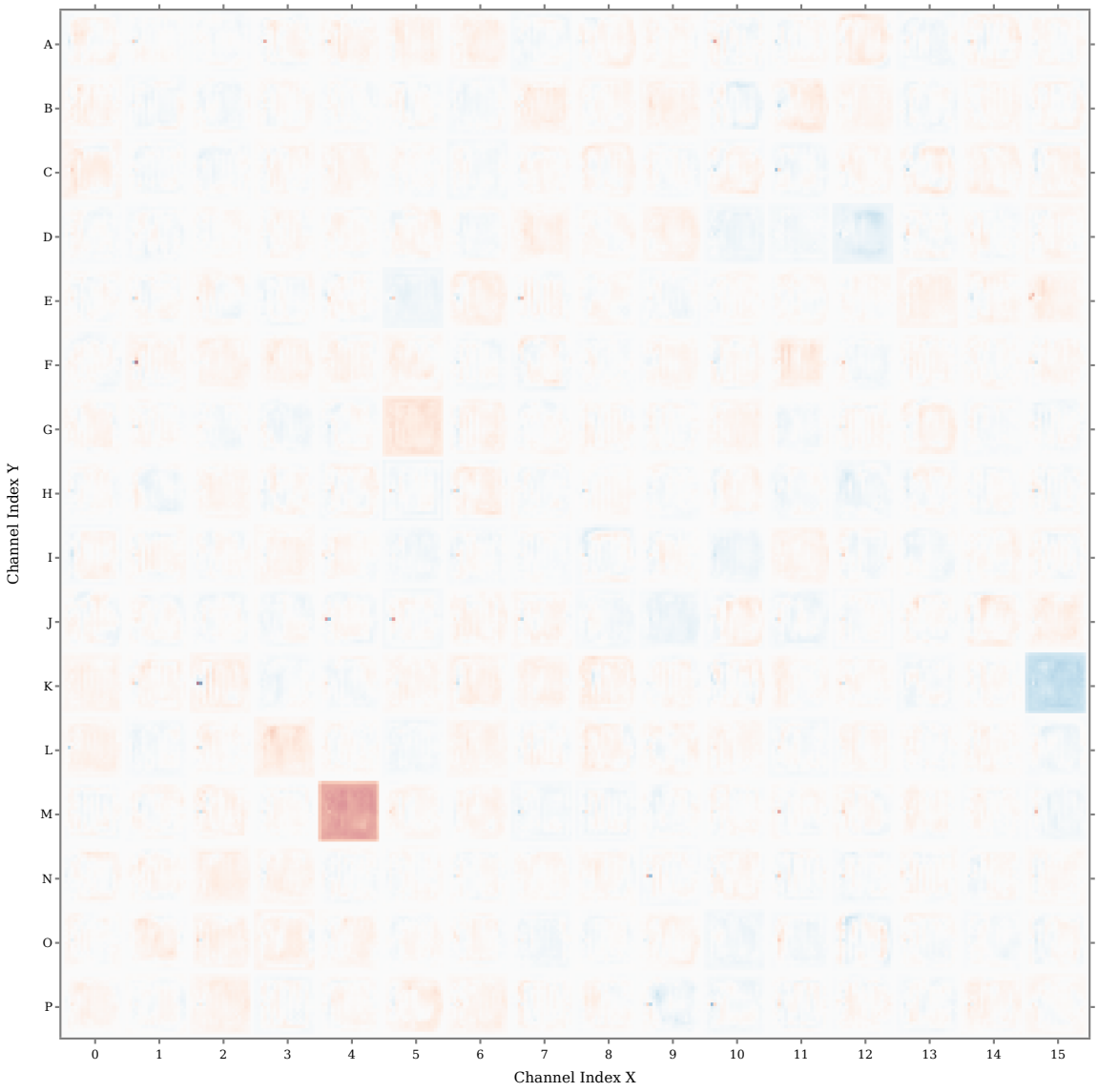


Figure K.11. The difference in activations in layer 26 between `Latest` in manually constructed cyclic vs. non-cyclic positions (Figure K.1).

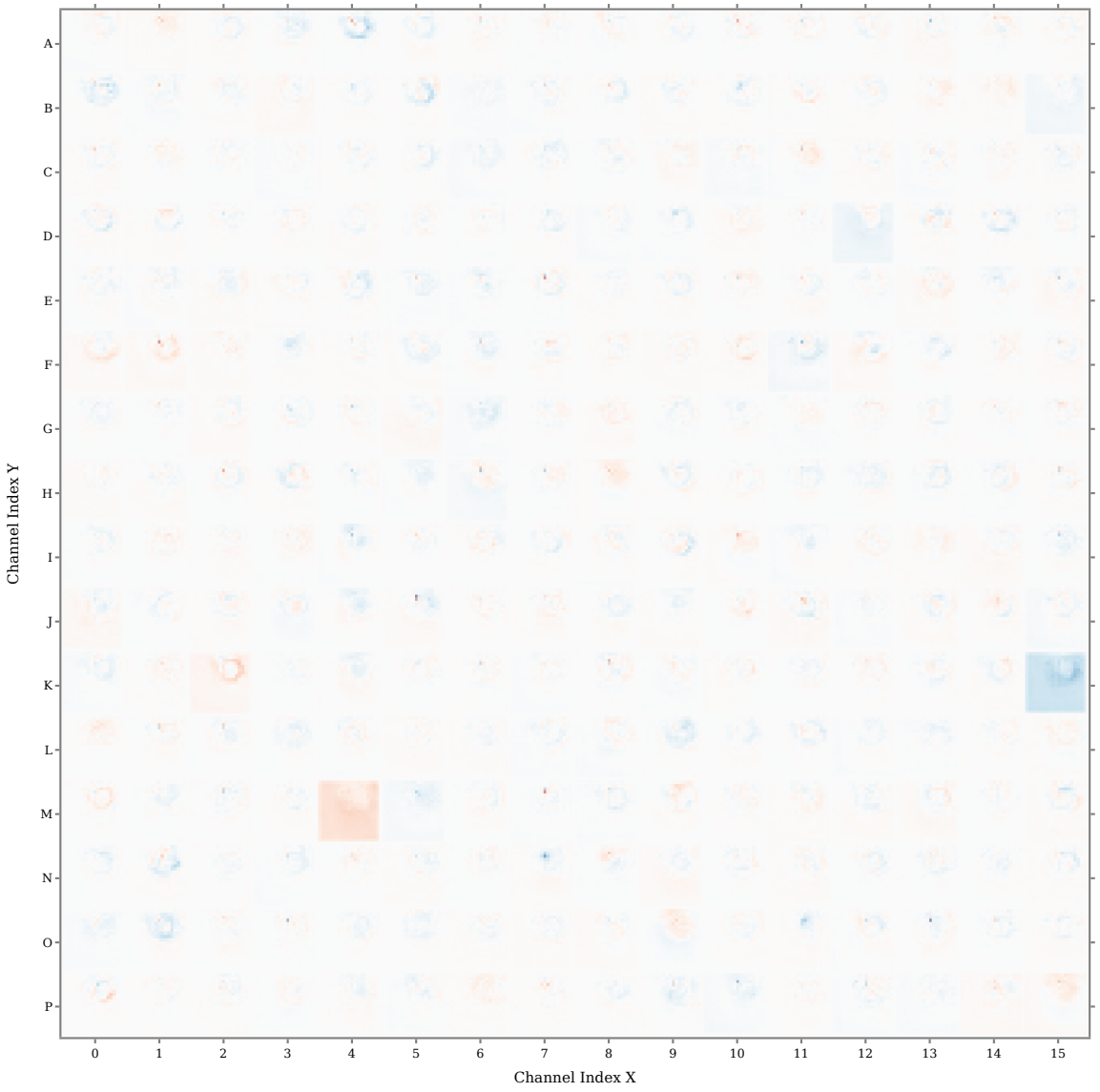


Figure K.12. The difference in activations in layer 26 of `Latest` between a near real-game cyclic and non-cyclic position, where the cycle group is safe (Figure K.3).

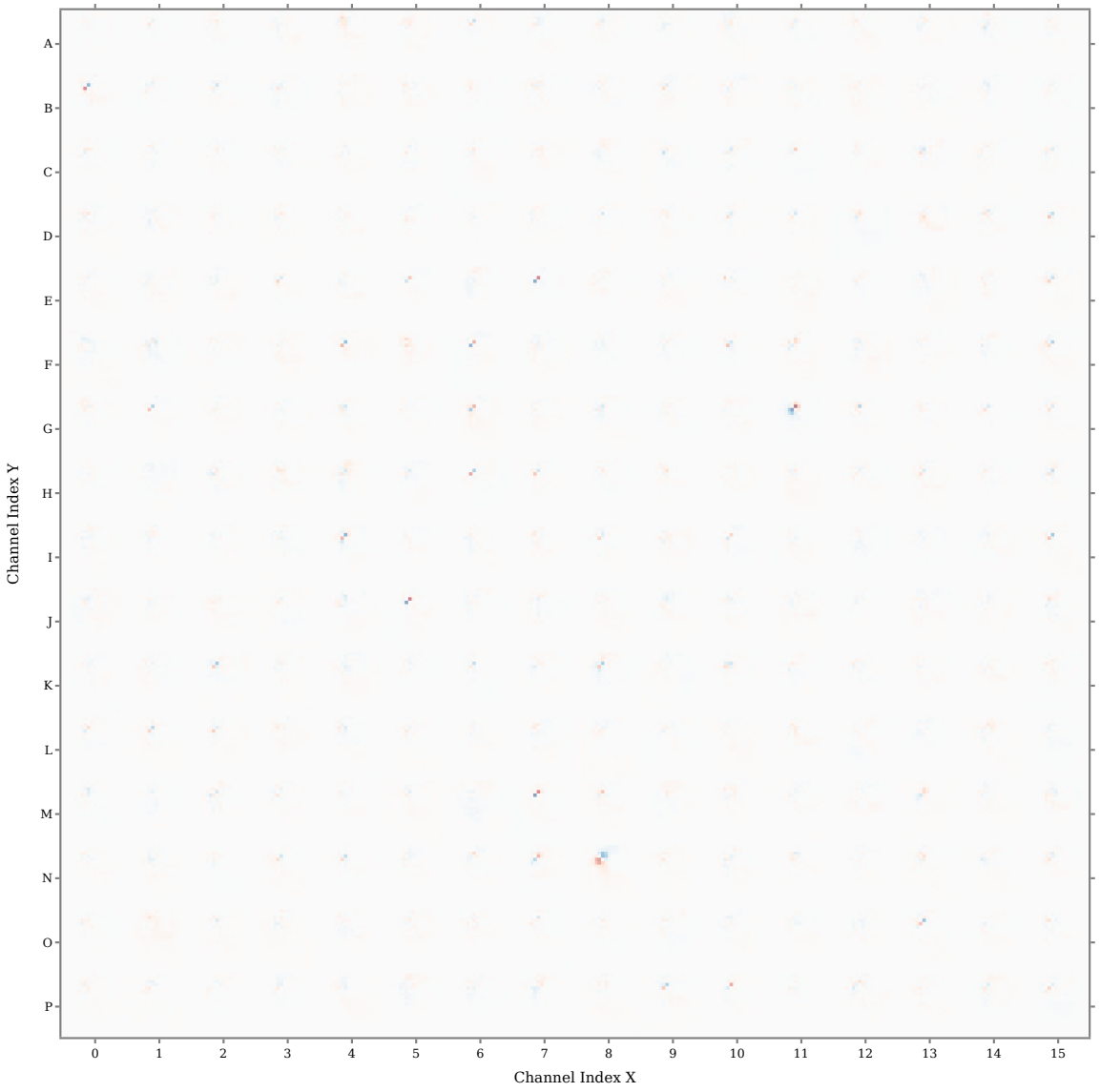


Figure K.13. The difference in activations in layer 26 of `Latest` between two near real-game positions where the cycle is already broken and does not change (Figure K.4).

L. KataGo Adversarial Training

Adversarial training against the cyclic exploit was incorporated into the official distributed training run of KataGo (Wu, 2022b) in mid-December 2022.[18] This adversarial training consists of starting a small fraction of self-play games in positions where the cyclic exploit is being executed, with the remainder of games being regular self-play games. These games were hand-selected by David Wu (the creator and primary developer of KataGo) from a collection of training and evaluation games taken from our cyclic-adversary training run (Figure 5.1):

> I uploaded some sgfposes and hintposes about cyclic topology groups. I spent multiple hours hand-adding a bunch [of] contrastive examples, where you would have the same group with different numbers of liberties, or with different numbers of "false" (but actually real) eyes. Roughly 0.08% of games should involve them now, we'll see if that tiny rate has any effect on learning them over the next weeks.
>
> (David Wu, Computer Go Community Discord, December 15 2022)

Figure L.2 shows that over the course of 6 months, adversarial training caused the cyclic-adversary's win rate to decrease significantly across several KataGo architectures. Figure L.3 shows that the different architectures all improved at a similar rate with respect to the total amount of data they were trained with.

In numbers:

- Prior to adversarial training, our cyclic-adversary$^{600\text{ visits}}$ wins 1048 / 1048 games against `Latest`$^{1\text{ visit}}$, 973 / 1000 games against `Latest`$^{4096\text{ visits}}$, and 50 / 50 games against `b60-s6729m`$^{200\text{ visits}}$.
- After adversarial training, our cyclic-adversary$^{600\text{ visits}}$ wins 118 / 2000 games against `b60-s7702m`$^{1\text{ visit}}$ and 0 / 400 games against `b60-s7702m`$^{1600\text{ visits}}$. (`b60-s7702m` refers to KataGo network `b60c320-s7701878528-d3323518127`, released on May 17, 2023, the latest 60-block KataGo network available at the time we ran this experiment. We round the number of training steps (7,701,878,528) to the nearest million to shorten the name.)

However, we are able to fine-tune our original adversary to defeat these updated networks (discussed below in Appendix L.1). This suggests that it is non-trivial to defend against the cyclic exploit, unlike the pass exploit which we were able to manually patch. Thus, developing techniques to train agents that are immune to the cyclic-exploit while maintaining high Go strength remains an interesting open problem.

Figure L.1 displays one game of the fine-tuned cyclic-adversary against `b60-s7702m`$^{100{,}000\text{ visits}}$. The attack is still a cyclic attack, though the placement of the cyclic group has moved from the corner of the board to the center of one side of the board.

L.1. Adversary Fine-Tuning

While KataGo's adversarial training was effective at defending against our original cyclic-adversary, we were able to fine-tune our cyclic-adversary to defeat the adversarially trained KataGo networks (Figures L.4 and L.5).

In numbers:

- Prior to fine-tuning, our original cyclic-adversary cyclic-adversary$^{600\text{ visits}}$ wins 118 / 2000 = 5.9% of games against `b60-s7702m`$^{1\text{ visit}}$ and 0 / 400 games against `b60-s7702m`$^{1600\text{ visits}}$.
- After fine-tuning, our improved cyclic-adversary$^{600\text{ visits}}$ wins 188 / 400 = 47% of games against `b60-s7702m`$^{4096\text{ visits}}$ and 7 / 40 = 17.5% of games against `b60-s7702m`$^{100000\text{ visits}}$.
- Our fine-tuning also transfers (albeit in a weaker form) to the `b18-s5832m` network, winning 51 / 400 = 12.75% of games against `b18-s5832m`$^{4096\text{ visits}}$.

[18] The latest net before the adversarial training was introduced was the b60c320 network at 6,729,327,872 steps of training, released on December 16 2022. Source: https://discord.com/channels/417022162348802048/583775968804732928/1056607918545457252.

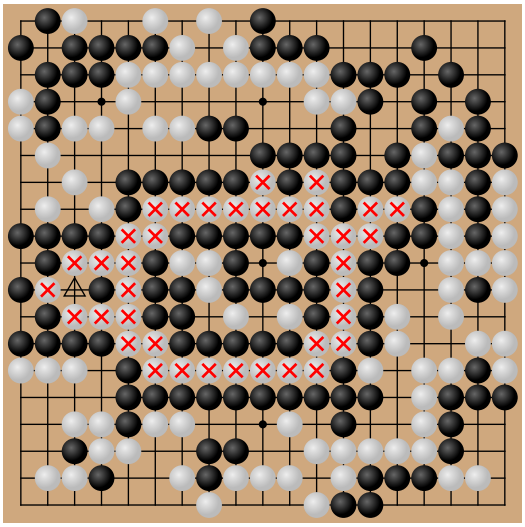


Figure L.1. Our fine-tuned cyclic-adversary, playing as black, still wins by capturing a cyclic group (×) that the victim (`b60-s770m`, 10^5 visits, 10 search threads) leaves vulnerable. The adversary plays at the square marked Δ to capture the group. Explore the game.

- Finally, fine-tuning causes our improved cyclic-adversary to do worse against `Latest`, which it only wins 212 / 380 = 55.79% of games against (compared to 973 / 1000 = 97.3% prior to fine-tuning).

In total, we spent 1154.9 V100 GPU-days on fine-tuning our cyclic-adversary, which underwent 1.68×10^8 steps of gradient descent. During training, we had two crashes that stalled the progress of training, so the amount of compute we used is somewhat higher than was necessary to achieve these win rates.

We stopped our fine-tuning run due to conference deadline time constraints, but trends in the training curve suggest that we could have improved our win rate against `b60-s7702m` with more compute (Figure L.5). However, the win-rate-compute scaling against adversarially trained networks is much worse than against undefended networks (compare Figure L.5 with Figure D.3).

Our cyclic-adversary fine-tuning was performed using the following curriculum:

1. `b60c320-s7047906048-d3140270330`$^{32 \text{ visits}}$
2. `b60c320-s7047906048-d3140270330`$^{128 \text{ visits}}$
3. `b60c320-s7047906048-d3140270330`$^{512 \text{ visits}}$
4. `b60c320-s7047906048-d3140270330`$^{1600 \text{ visits}}$
5. `b60c320-s7701878528-d3323518127`$^{32 \text{ visits}}$
6. `b60c320-s7701878528-d3323518127`$^{64 \text{ visits}}$

7–10. ...

11. `b60c320-s7701878528-d3323518127`$^{2048 \text{ visits}}$

For the final network in the curriculum, `b60-s7702m` (curriculum steps #6–11), its last 9.73×10^8 (12.63% of its total 77.02×10^8) steps of training included cyclic positions discovered by our cyclic-adversary.

The curriculum was advanced whenever the cyclic-adversary's win rate reached 75%, except we trained for about two million extra steps on `b60c320-s7047906048-d3140270330`$^{1600 \text{ visits}}$ (`b60-s7048m`$^{1600 \text{ visits}}$) since that part of the fine-tuning run was performed at a time when `b60-s7048m` was the latest 60-block KataGo network. We found that to attack `b60-s7702m`, we achieved a stronger win rate by continuing from that existing fine-tuning run rather than redoing fine-tuning with a curriculum that started at `b60c320-s7701878528-d3323518127`$^{1 \text{ visit}}$.

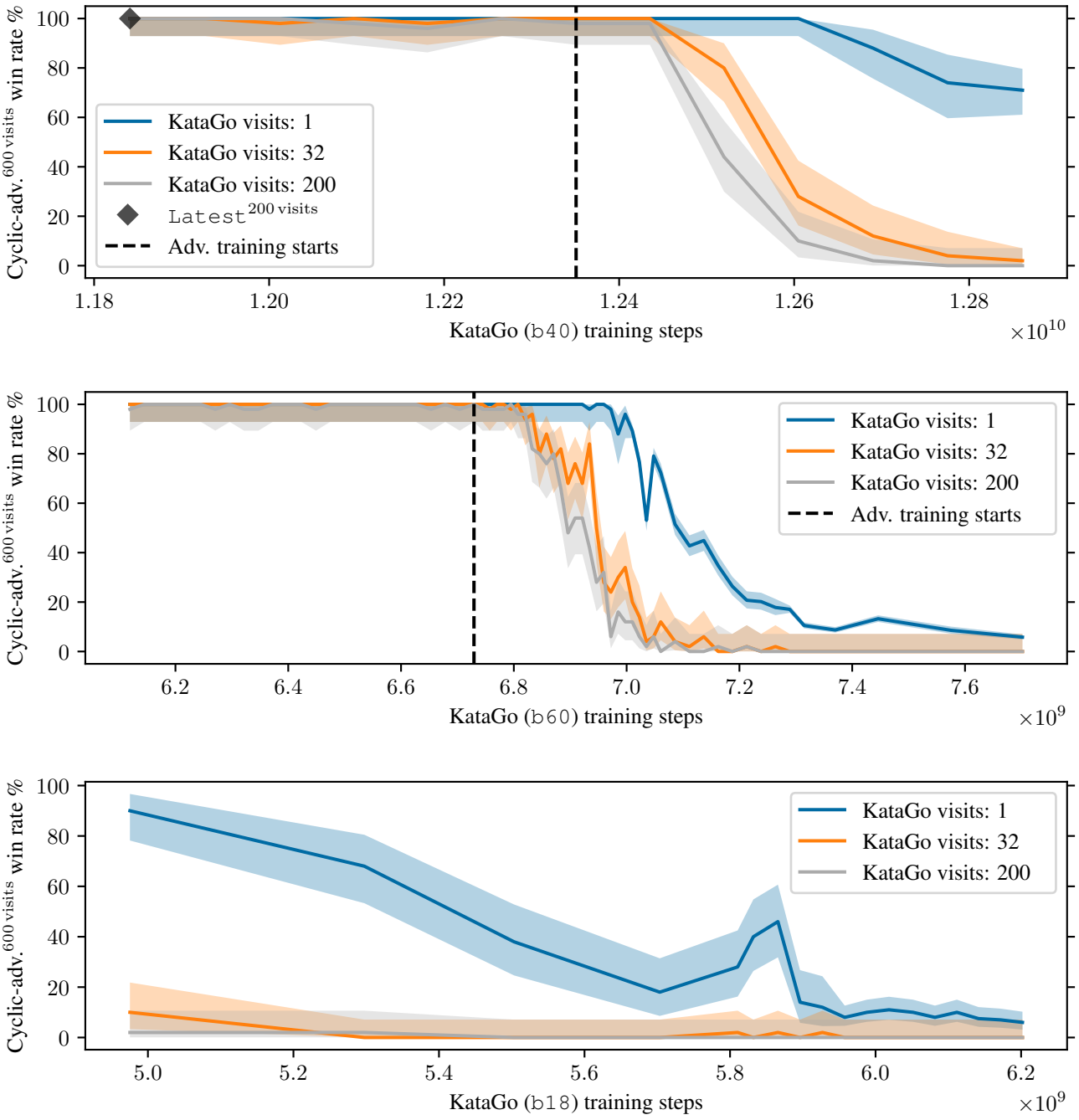


Figure L.2. Win rate of the cyclic-adversary against `b40c256` (40-block, 256-channel), `b60c320` (60-block, 320-channel), and `b18c384nbt` (18-block, 384-channel) KataGo networks, which are convolutional+residual networks. The `b18c384nbt` networks have "nested residual bottleneck blocks", where "each block has a linear bottleneck [He et al. (2016)] from 384 trunk channels [to] 192 channels, followed by two regular 192 × 192 residual blocks, followed by a linear recovery from 192 channels [to] 384 trunk channels" (Wu, 2022a). Each subfigure displays win rate (vertical axis) against KataGo networks of a particular architecture with increasing amounts of self-play training (horizontal axis). The dotted black line marks when the adversarial cyclic positions were introduced into KataGo's distributed training run. The earliest `b18` network was released several months after the adversarial training positions were introduced.

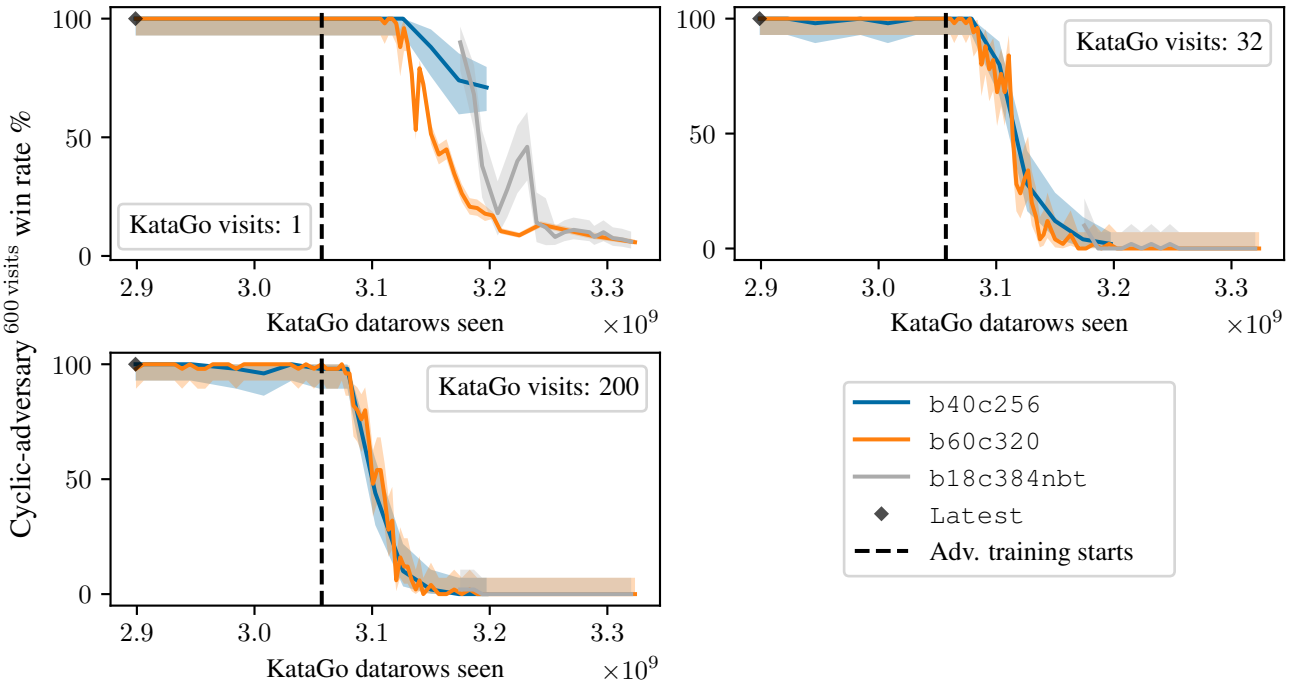
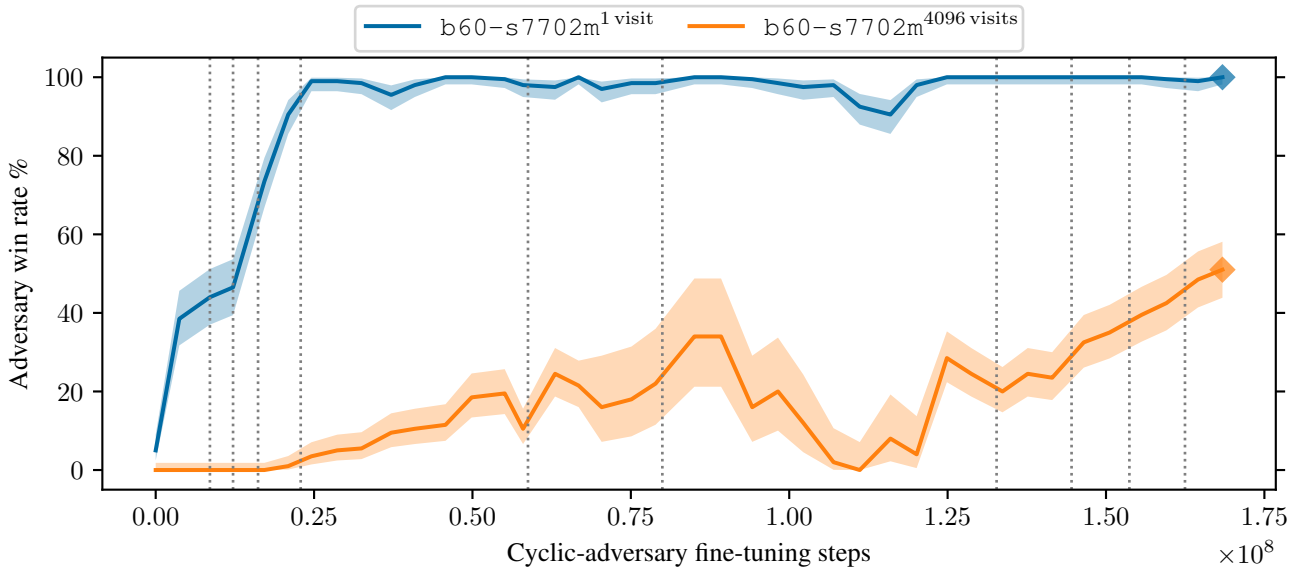
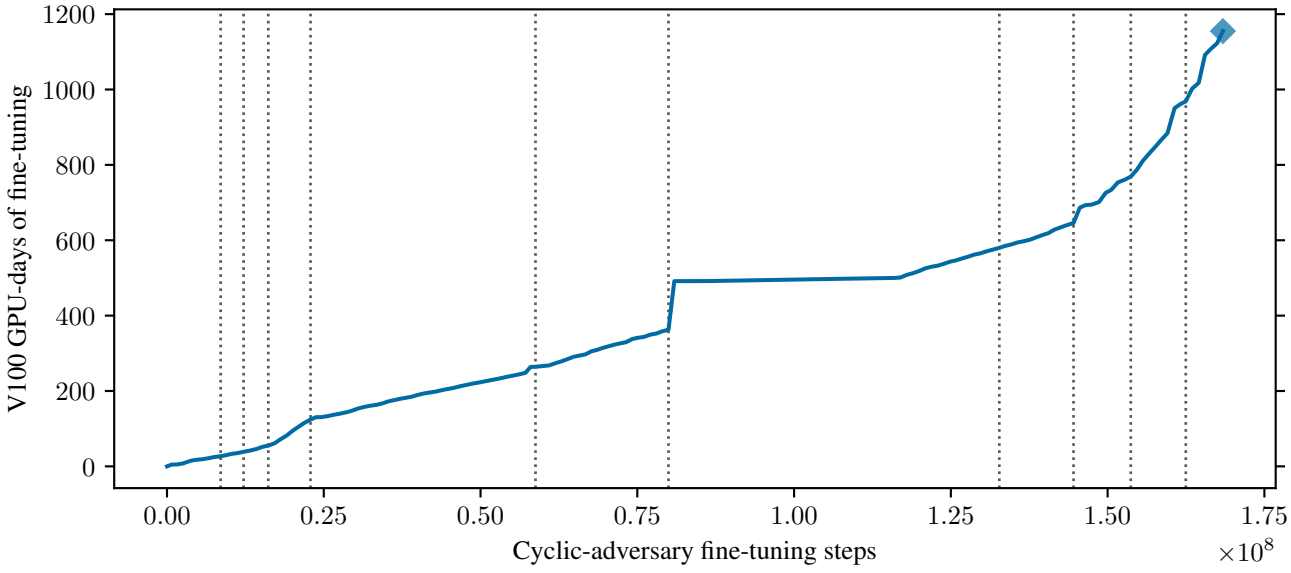


Figure L.3. Win rate of the cyclic-adversary against different KataGo networks. This is the same data as in Figure L.2, except with the horizontal axis as number of datarows seen instead of number of SGD steps taken. In KataGo's distributed training run, all networks are trained on the same self-play data (which can be shared among different network types), so the training progress of different network architectures can be compared on the same x-axis scale by plotting against the number of datarows seen. We see that adversarial training improves performance against the cyclic-adversary across network architectures at roughly the same rate with respect to the number of datarows seen.

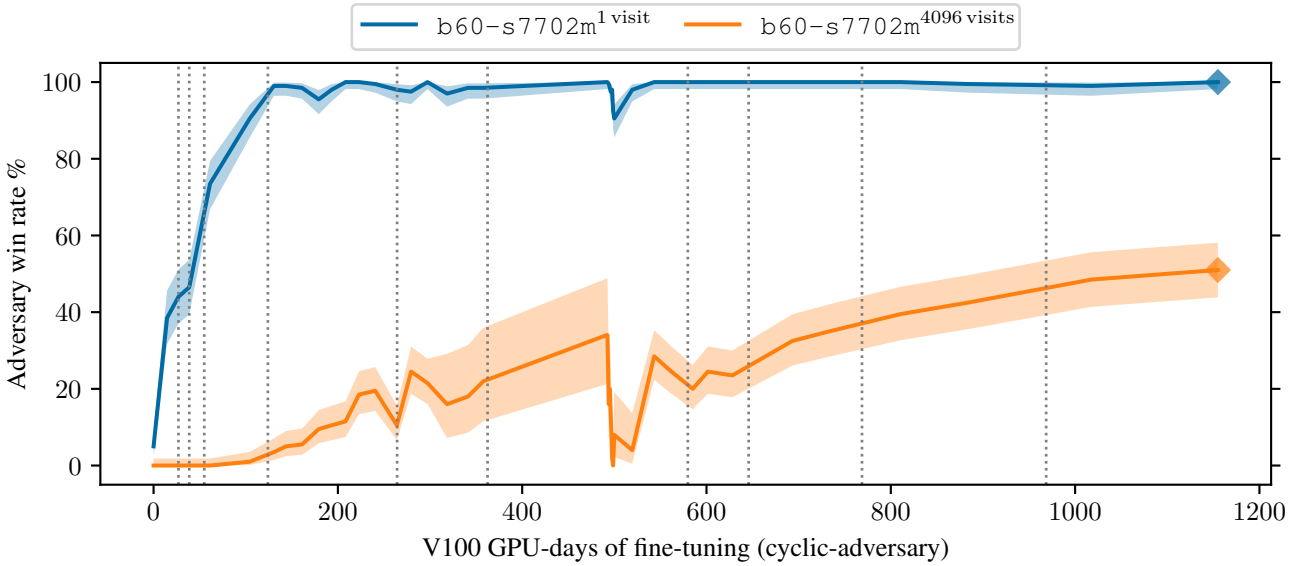


(a) Win rate (y-axis) of the cyclic-adversary$^{600 \text{ visits}}$ against an adversarially trained KataGo network (with varying visits) over the course of fine-tuning. The x-axis is the number of steps of gradient descent taken during fine-tuning. The shaded interval is a 95% Clopper-Pearson interval over evaluation games (some checkpoints have more evaluations than others).

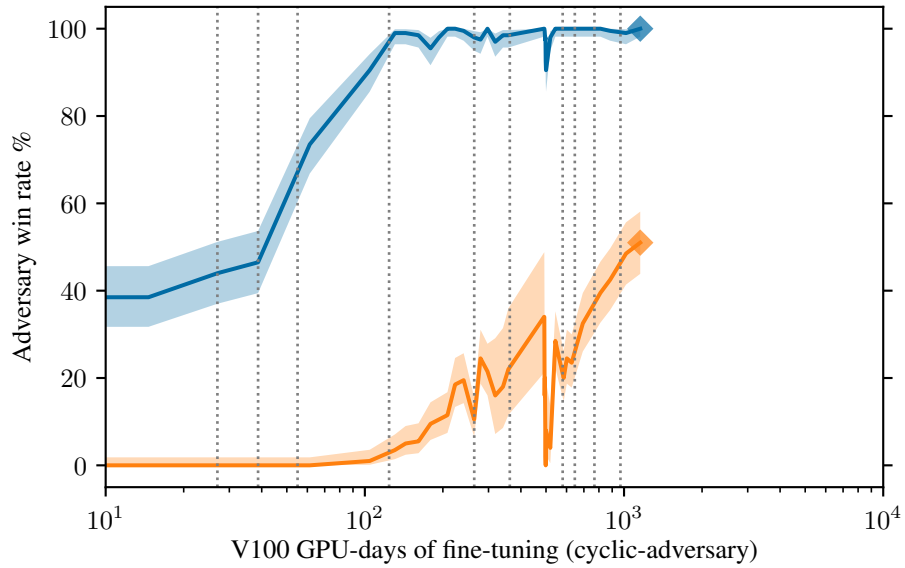


(b) Fine-tuning steps (x-axis) vs. GPU days (y-axis) of the cyclic-adversary.

Figure L.4. Plots showing the fine-tuning progression of our cyclic-adversary against adversarially trained KataGo networks. Vertical dotted lines denote switches in the curriculum (as detailed in Appendix L.1). The strongest cyclic-adversary checkpoint is marked with a diamond (♦). Our data-shuffler crashed at 0.81×10^8 steps of fine-tuning, indicated by the sharp discontinuity in compute. This resulted in us subsequently taking many training steps with the same data, resulting in win-rate instability lasting until around 1.2×10^8 steps of fine-tuning. Similarly, the dip in win rate at 0.57×10^8 steps may be due to restarting the training run at that point due to a crash.



(a) Win rate (y-axis) of the cyclic-adversary$^{600\text{ visits}}$ against an adversarially trained KataGo network (with varying visits) over the course of fine-tuning. The x-axis is the amount of compute spent fine-tuning. The shaded interval is a 95% Clopper-Pearson interval over evaluation games (some checkpoints have more evaluations than others).



(b) The same plot as above, but with a log-scaled x-axis.

Figure L.5. These plots are the same as as Figure L.4a, but in terms of compute. The effects of the data-shuffler crash can be seen in 400-500 V100 GPU days region, where the win rate sharply drops and then recovers. On the right end of the plots, we see that the win-rate of the cyclic-adversary against b60-s7702m$^{4096\text{ visits}}$ steadily increases at roughly a log-linear rate. Rough eyeballing suggests that our cyclic-adversary may be able to reach a 90%+ win rate against b60-s7702m$^{4096\text{ visits}}$ in about 10,000 V100 GPU days. This win-rate-compute scaling is much worse than the scaling of the original cyclic adversary (compare with Figure D.3).

M. 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 M.1 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).

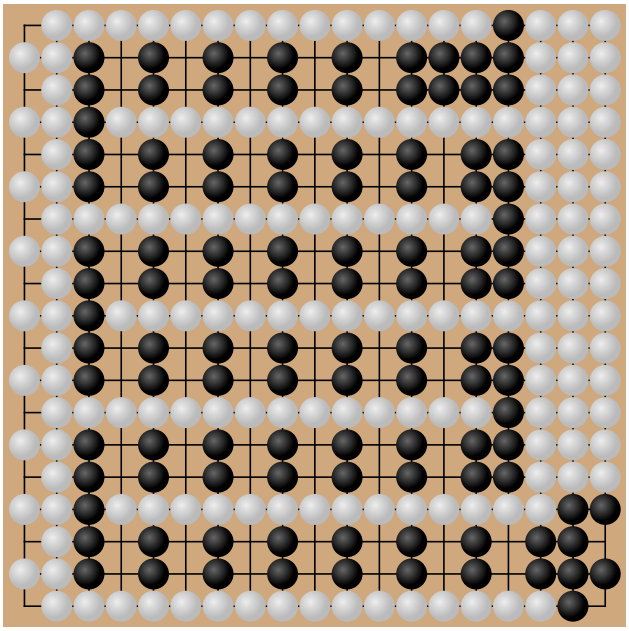


Figure M.1. A hand-crafted adversarial example for KataGo and other Go-playing AI systems. It is black's turn to move. Black can guarantee a win by connecting its currently disconnected columns together and then capturing the large white group on the right. However, KataGo playing against itself from this position loses 40% of the time as black.

N. Known Failures of Go-playing Agents

The cyclic vulnerability our adversary finds is unique in the confluence of 3 key factors. First, it affects top Go-playing agents, even when they have a very large amount of search. Second, it consistently produces a game-winning advantage. Third, this consistency does not require exact sequences or board positions. Along with other factors like how it is non-trivial to defend against (see Appendix L), that makes this vulnerability particularly significant.

Nonetheless, there are other known vulnerabilities of Go AIs, some of which exhibit a subset of those significant characteristics and can be noteworthy of their own right. The following overview 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 N.1. 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 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.

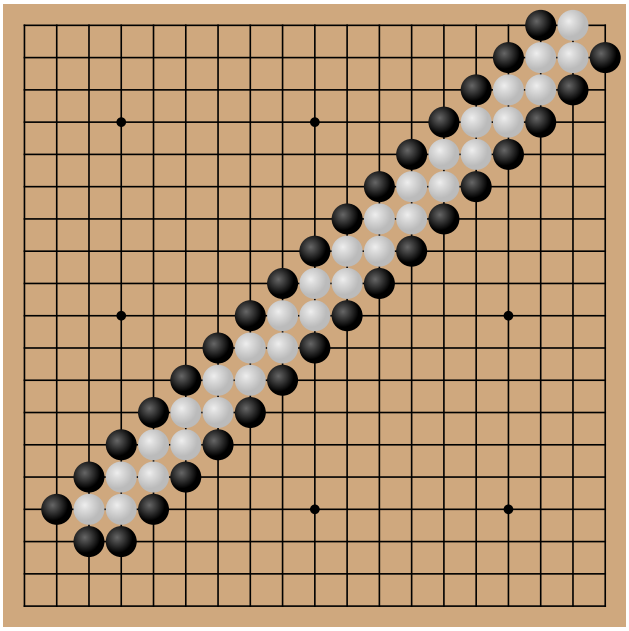


Figure N.1. Illustration of a ladder. White ran across the board, but has hit the top edge and has nowhere left to run. Black can capture on the next move by playing in the top right corner.

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 1.1a, and the weakness our cyclic-adversary exploits. Such situations are 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.

We, the authors of this paper, were not aware of this weakness of Go bots until after we had trained the cyclic-adversary. It was also not known that this weakness could be consistently exploited.

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.

O. Reproducibility Statement

We take the following steps to promote and ensure reproducibility:

- Our code is available at GitHub. The code is containerized and includes instructions for running it.
- We make many game records available through our website. We will make more game records available to researchers upon request and have already provided game records to David Wu, the creator and primary developer of KataGo, for use in KataGo's training process.
- We set up a bot running the most recent checkpoint of our cyclic-adversary on the KGS Go server, under the username `Adversary0`. This bot was available for the public to play for a period of a month. See Appendix J.1 for more details.

A number of our key results have already been reproduced:

- The vulnerability to the passing attack has been independently confirmed by David Wu.
- The vulnerability to the cyclic attack has been independently confirmed by David Wu, as well as many others in the computer Go community.
- The cyclic vulnerability and the adversary's ability to use it has been replicated through normal bot play against the KGS bot we made available, as has the result that novice human play beats the adversary.
- Human ability to use the cyclic attack has been independently reproduced against KataGo, as well as in transfer settings against ELF OpenGo, FineArt, Leela Zero, and Sai.

P. Acknowledgements

Thanks to David Wu and the Computer Go Community Discord for sharing their knowledge of computer Go with us and for their helpful advice on how to work with KataGo, to Adrià Garriga-Alonso for feedback and assistance setting up activation analysis and infrastructure, to Lawrence Chan, Euan McLean, and Niki Howe for their feedback on earlier drafts of the paper, to ChengCheng Tan and Alyse Spiehler for assistance preparing illustrations, to David Fontaine for help with debugging KataGo deadlocks, to Matthew Harwit for help with Chinese communication and feedback especially on Go analysis, to Daniel Filan for Go game analysis and feedback on project direction, and to Nir Shavit for his support of and high level feedback on the project.

Tony Wang was supported by funding from the Eric and Wendy Schmidt Center at the Broad Institute of MIT and Harvard.

Q. Author Contributions

Tony Wang invented and implemented the A-MCTS-S algorithm, made several other code contributions, and ran and analyzed many of the experiments. Adam Gleave managed the project, wrote the core of the paper, suggested the curriculum approach, helped manage the cluster experiments were run on, and implemented some minor features. Tom Tseng implemented and ran transfer experiments, trained and ran experiments with the pass-hardening defense enabled, and ran many of the evaluations. Kellin Pelrine (our resident Go expert) provided analysis of our adversary's strategy, search vs. robustness, activation analysis, and manually reproduced the cyclic-attack against different Go AIs. Nora Belrose implemented and ran the experiments for baseline adversarial policies, and our pass-hardening defense. Joseph Miller developed the website showcasing the games, and an experimental dashboard for internal use. Michael Dennis developed an adversarial board state for KataGo that inspired us to pursue this project, and contributed a variety of high-level ideas and guidance such as adaptations to MCTS. Yawen Duan ran some of the initial experiments and investigated the adversarial board state. Viktor Pogrebniak implemented the curriculum functionality and improved the KataGo configuration system. Sergey Levine and Stuart Russell provided guidance and general feedback.