ADVERSARIAL POLICIES BEAT SUPERHUMAN GO AIS

Tony Tong Wang[*1] **Adam Gleave**[*23] **Nora Belrose**[3] **Tom Tseng**[3]
Joseph Miller[3] **Kellin Pelrine**[34] **Michael D Dennis**[2] **Yawen Duan**[2]
Viktor Pogrebniak **Sergey Levine**[2] **Stuart Russell**[2]
[1]MIT [2]UC Berkeley [3]FAR AI [4]McGill University; Mila [*]Equal contribution.

ABSTRACT

We attack the state-of-the-art Go-playing AI system, KataGo, by training adversarial policies that play against frozen KataGo victims. Our attack achieves a >99% win rate when KataGo uses no tree-search, and a >77% win rate when KataGo uses enough search to be superhuman. Notably, our adversaries do not win by learning to play Go better than KataGo—in fact, our adversaries are easily beaten by human amateurs. Instead, our adversaries win by tricking KataGo into making serious blunders. Our results demonstrate that even superhuman AI systems may harbor surprising failure modes. Example games are available at goattack.far.ai.

1 INTRODUCTION

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

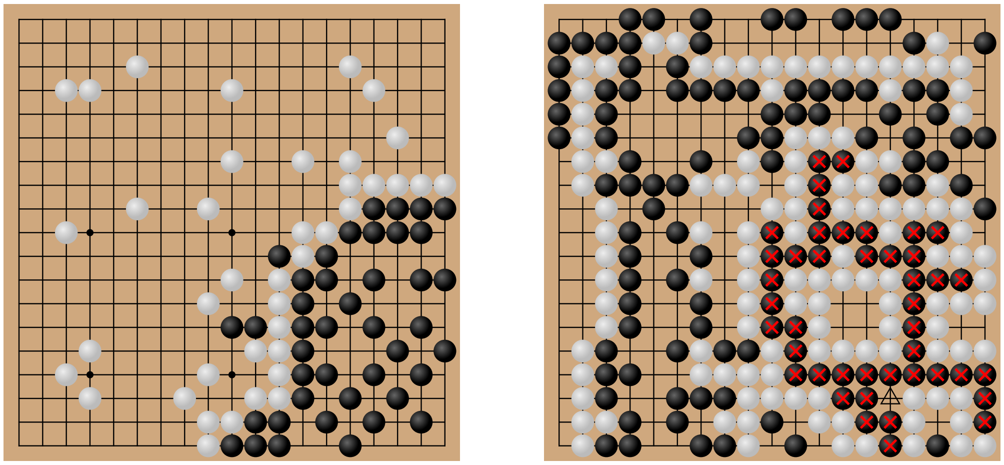
Nonetheless, prior work has found that seemingly highly capable continuous control policies trained via self-play can be exploited by *adversarial policies* (Gleave et al., 2020; Wu et al., 2021). This suggests that self-play may not be as robust as previously thought. However, although the victim agents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

To answer this, we study a domain where self-play has achieved very strong performance: Go. Specifically, we train adversarial policies end-to-end to attack KataGo (Wu, 2019), the strongest publicly available Go-playing AI system. Using less than 5% of the compute used to train KataGo, we obtain adversarial policies that win >99% of the time against KataGo with no search, and >77% of the time against KataGo with enough search to be superhuman.

Critically, our adversaries do *not* win by learning a generally capable Go policy. Instead, the adversaries trick KataGo into making serious blunders that result in KataGo losing the game (Figure 1). Despite being able to beat KataGo, our adversarial policies lose against even amateur Go players (see Appendix F.2). So KataGo is in this instance less *robust* than human amateurs, despite having superhuman *capabilities*. This is a striking example of non-transitivity, illustrated in Figure 2.

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

KataGo can play at a strongly superhuman level, winning (Wu, 2019, Section 5.1) against ELF OpenGo (Tian et al., 2019) and Leela Zero (Pascutto, 2019) that are themselves superhuman. In Appendix D, we estimate that KataGo without search plays at the level of a top 100 European player, and that KataGo with 512 visits per move of search is superhuman. Our attack scales far beyond this, achieving a 72% win rate against KataGo playing with 10^7 visits per move of search.



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

(b) Our *cyclic-adversary* wins as white by capturing a cyclic group (X) that the victim ($\texttt{Latest}_{\texttt{def}}$, 2048 visits) leaves vulnerable. Explore the game.

Figure 1: Games between the strongest KataGo network (which we refer to as Latest) and two different types of adversaries we trained. (a) Our *pass-adversary* beats 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. (b) Our *cyclic-adversary* beats a superhuman version of KataGo immunized against the "passing trick". The adversary lures the victim into letting a large group of cyclic victim stones (X) get captured by the adversary's next move (Δ). Appendix F.3 has a detailed description of this adversary's behavior.

Our paper makes three contributions. First, we propose a novel attack method, hybridizing the attack of Gleave et al. (2020) and 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, including a qualitative analysis of their game play. 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, many image classifiers reach or surpass human performance (Ho-Phuoc, 2018; Russakovsky et al., 2015; Shankar et al., 2020; Pham et al., 2021). Yet even these state-of-the-art image classifiers are vulnerable to adversarial examples (Carlini et al., 2019; Ren et al., 2020). This raises the question: could highly capable deep RL policies be similarly vulnerable?

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

It is known that self-play may not converge in non-transitive games (Balduzzi et al., 2019). However, Czarnecki et al. (2020) has argued that real-world games like Go grow increasingly transitive as skill increases. This would imply that while self-play may struggle with non-transitivity early on during training, comparisons involving highly capable policies such as KataGo should be mostly transitive. By contrast, we find a striking non-transitivity: our adversaries exploit KataGo agents that beat human professionals, yet even an amateur Go player can beat our adversaries (Appendix F.2).

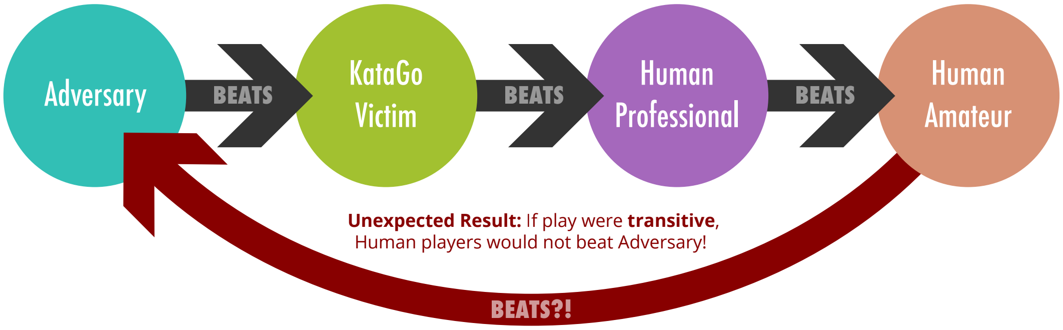

Adversary
BEATS
KataGo Victim
BEATS
Human Professional
BEATS
Human Amateur
Unexpected Result: If play were **transitive**,
Human players would not beat Adversary!
BEATS?!

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

Most prior work attacking deep RL has focused on perturbing observations (Huang et al., 2017; Ilahi et al., 2022). Concurrent work by Lan et al. (2022) shows that KataGo with $\leq$50 visits can be induced to play poorly by adding two adversarially chosen moves to the move history input of the KataGo network, even though these moves do not substantially change the win rate estimated by KataGo with 800 visits. However, the perturbed input is highly off-distribution, as the inputs tell KataGo 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 professional-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 which has approximately 10^{172} possible states (Allis, 1994, Section 6.3.12). Prior work has lower bounded the exploitability in some poker variants using search (Lisý & Bowling, 2017), but the method relies on domain-specific heuristics that are not applicable to Go.

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

Our attack methodology is similar to Timbers et al.: we both use an AlphaZero-style training procedure that is adapted to use the *opponent's* policy during search. However, unlike Timbers et al. we attempt to model the victim's search process inside our adversary via A-MCTS-R (see Section 4), and our curriculum uses checkpoints as well as search. Our main contribution lies in our experiments: we find our attack scales to victims with up to 10^7 visits whereas Timbers et al. evaluates only up to 800 visits. Additionally, we propose and evaluate a defense; show the victim's predicted win rate is miscalibrated; qualitatively analyse games played by the adversary; and investigate transfer of adversarial policies to other Go programs.

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 against some *victim* agent. Critically, the attacker does not have any special powers—they can only take the same actions available to a regular player.

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, such as a commercially available or open-source Go AI system.

This is a challenging setting even with gray-box access. Although finding an exact Nash equilibrium in a game as complex as Go is intractable, a priori it seems plausible that a professional-level Go system might have reached a *near-Nash* or *ϵ-equilibrium*. In this case, the victim could only be exploited by an ϵ margin. Moreover, even if there *exists* a policy that can exploit the victim, it might be computationally expensive to find given that self-play training did not discover the vulnerability.

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

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

3.2 KataGo

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

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

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

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

4 Attack methodology

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

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

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

Adversarial MCTS: Recursive (A-MCTS-R). A-MCTS-S underestimates the strength of victims that use search since it models the victim as sampling directly from the policy head. To resolve this, A-MCTS-R (Appendix B.3) runs MCTS for the victim at each victim node in the A-MCTS-R tree. Unfortunately, this change increases the computational complexity of both adversary training and inference by a factor equal to the victim search budget. We include A-MCTS-R primarily as an upper bound to establish how much benefit can be gained by resolving this misspecification.

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

Curriculum. To help overcome the challenging random initialization, we introduce a curriculum that trains against successively stronger versions of the victim. 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 gradually increase the amount of search that the victim performs during victim-play.

5 Evaluation

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

In Section 5.2, we then add a *pass-alive defense* to the victim to defend against the aforementioned attack. The defended victim $\texttt{Latest}_{\texttt{def}}$ is provably unable to lose via accidentally passing and is roughly the same strength as `Latest`.[1] However, we are still able train a *cyclic-adversary* that achieves a 99.8% win rate against $\texttt{Latest}_{\texttt{def}}$ playing without search. The cyclic-adversary succeeds against victims playing with search as well, with its best result in Section 5.3 being a 79.6% win rate against $\texttt{Latest}_{\texttt{def}}$ with 2048 visits.

The cyclic-adversary is qualitatively very different from the pass-adversary as it does not use the pass-trick (Figure 1b). To check that our defense did not introduce any unintended weaknesses,

[1] $\texttt{Latest}_{\texttt{def}}$ won 456 / 1000 games against `Latest` when both agents used no tree-search and 461 / 1000 games when both agents used 2048 search visits per move.

we evaluate our cyclic-adversary against the unmodified Latest. The cyclic-adversary achieves a 100% win rate (over 500 games) against Latest without search, and a 77.6% win rate against Latest with 2048 visits. Moreover, a version of the cyclic-adversary that was trained for longer achieved a 72% win rate against Latest with 10^7 visits. In Appendix D.2, we estimate that Latest with 2048 visits is already much stronger than the best human Go players, and Latest with 10^7 visits far surpasses human players.

5.1 Attacking a victim without search

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).

Our best pass-adversary checkpoint playing with 600 visits against Latest playing with no search achieves a 99.9% win rate. Notably, this high win rate is achieved despite our pass-adversary being trained for only 3.4×10^7 time steps—just 0.3% as many time steps as the victim it is exploiting. Critically, the pass-adversary does not win by playing a stronger game of Go than the victim. Instead, it follows a bizarre strategy illustrated in Figure 1a that loses even against human amateurs (see Appendix F.2). The strategy tricks the KataGo policy network into passing prematurely at a point where the adversary has more points.

Appendix E 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 Defending against the pass-adversary

We design a hard-coded defense for the victim against the attack found in Section 5.1: only passing when it cannot change the outcome of the game. Concretely, we only allow the victim to pass when its only legal moves are in its own *pass-alive territory*, a concept described in the official KataGo rules (Wu, 2021) and which extends the traditional Go notion of a pass-alive group (see Appendix B.5 for a full description of the defense). Given a victim V, we let $\mathtt{V}_{\mathtt{def}}$ denote the victim with this defense applied. The defense completely thwarts the pass-adversary from Section 5.1; the pass-adversary loses every game out of 1000 against $\mathtt{Latest}_{\mathtt{def}}$.

With the defense always enabled for the victim, we again train using A-MCTS-S to obtain our cyclic-adversary. The curriculum starts from an earlier checkpoint $\mathtt{cp39}_{\mathtt{def}}$. After the curriculum reaches $\mathtt{Latest}_{\mathtt{def}}$ playing with no search, the curriculum starts increasing the number of visits used by $\mathtt{Latest}_{\mathtt{def}}$. See Appendix C.2 for curriculum details.

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

Our best cyclic-adversary checkpoint playing with 600 visits against $\mathtt{Latest}_{\mathtt{def}}$ playing with no search achieves a 99.8% win rate. The cyclic-adversary also still works against Latest with the defense disabled, achieving a 100.0% win rate over 500 games. The cyclic-adversary is trained for 4.98×10^8 time steps—just 4.2% as many time steps as the Latest, but 15× more than the pass-adversary from Section 5.1. The cyclic-adversary still loses against human amateurs (see Appendix F.2).

5.3 Attacking a victim with search

Here, we evaluate the ability of our cyclic-adversary to exploit $\mathtt{Latest}_{\mathtt{def}}$ playing *with* search. Against $\mathtt{Latest}_{\mathtt{def}}$ with 2048 visits, the cyclic-adversary achieves a win rate of 72%–79.6% depending on how much search our adversary uses (Figure 4b). With 600 visits, our cyclic-adversary achieves a 77.6% win rate (over 500 games) against a non-defended Latest with 2048 visits, verifying that our adversary is not exploiting anomalous behavior introduced by the defense.

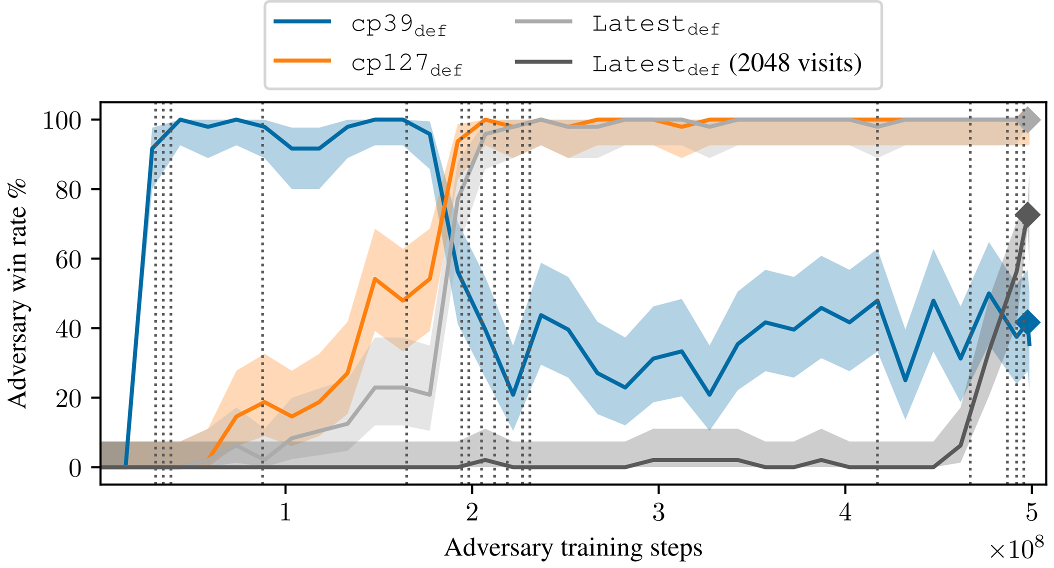
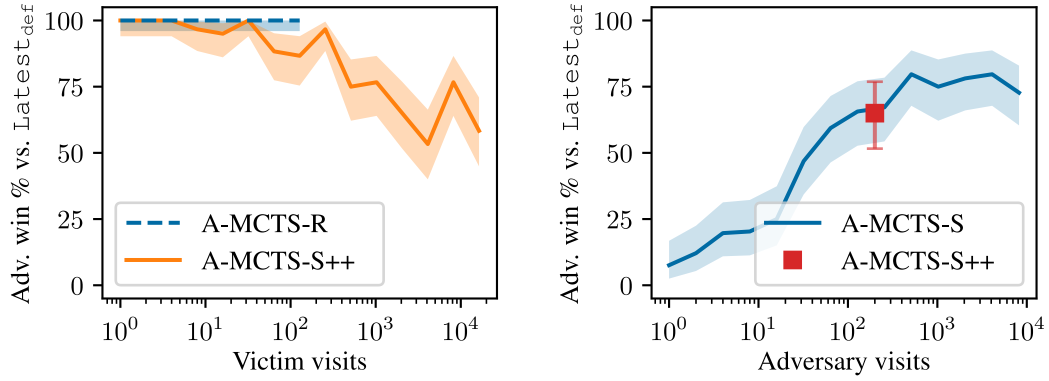


Figure 3: The win rate (y-axis) of the cyclic-adversary over time (x-axis) playing with 600 visits against the $\texttt{cp39}_{\texttt{def}}$, $\texttt{cp127}_{\texttt{def}}$, and $\texttt{Latest}_{\texttt{def}}$ victim policy networks playing without search, as well as $\texttt{Latest}_{\texttt{def}}$ playing with 2048 visits. The strongest cyclic-adversary checkpoint (marked ♦) wins 547/548 = 99.8% games against $\texttt{Latest}_{\texttt{def}}$ without search and 398/548 = 72.6% games against $\texttt{Latest}_{\texttt{def}}$ with 2048 visits. The shaded interval is a 95% Clopper-Pearson interval over 48 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 256 visits. Vertical dashed lines denote switches to stronger victim networks or to an increase in $\texttt{Latest}_{\texttt{def}}$'s search budget. See Appendix C.2 for the exact curriculum specification.



(a) The cyclic-adversary's win rate vs. $\texttt{Latest}_{\texttt{def}}$ varying the amount of victim search (x-axis). The adversary plays with a fixed 200 visits / move. Victims with more search are harder to exploit.

(b) The cyclic-adversary's win rate varying the amount of adversary search (x-axis). The victim $\texttt{Latest}_{\texttt{def}}$ plays with a fixed 2048 visits / move. Higher adversary visits moderately improve win rates.

Figure 4: We evaluate the ability of the cyclic-adversary to win against the $\texttt{Latest}_{\texttt{def}}$ victim with varying amounts of victim and adversary search. Shaded regions and error bars denote 95% Clopper-Pearson confidence intervals over ~64 games.

Even when its victims are allowed to perform tree-search, the cyclic-adversary is able to trick its victims into making severe mistakes a human would avoid (see Appendix F.3 for further analysis).

In Figure 4, we study the effects of varying both adversary and victim search. We find that for a fixed adversary search budget, victims with more search are harder to exploit; and for a fixed victim search budget, more adversary search results in a stronger attack. For both the victim and adversary, increasing search seems to yield diminishing returns.

From Figure 4a, we see that A-MCTS-R (which correctly models the victim) outperforms A-MCTS-S (which incorrectly models the victim as having no search). The cyclic-adversary with A-MCTS-R achieves an impressive 100% win rate against $\texttt{Latest}_{\texttt{def}}$ up to 128 victim visits over 90 games at each data point. However, this comes at the cost of drastically more compute—so much that we were unable to evaluate A-MCTS-R at higher visit counts.

Finally, we also scaled up the cyclic-adversary run from Figure 3 so that the curriculum reached all the way to $2^{18} \approx 250$k victim visits. This scaled up run took roughly 11.6% of the compute Latest was trained with. Our scaled-up cyclic-adversary (with 600 visits / move) was able to achieve an 82% win rate (over 50 games) against Latest with 10^6 visits / move, and a 72% win rate (over 25 games) against Latest with 10^7 visits / move.[2] 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.

5.4 Understanding the cyclic-adversary

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

To better understand the attack, we examined the win rate predictions produced by both the adversary's and the victim's value networks at each turn of a game. As depicted in Figure 5a, the victim's win rate predictions are strikingly uncalibrated in the games where it loses (the majority), with a high cross-entropy loss throughout the bulk of the game. The predictions only improve in accuracy close to the end game.

Typically the victim predicts that it will win with over 99% confidence[3] for most of the game, then suddenly realizes it will lose with high probability, often just *one move* before its circular group is captured. In some games, we observe that the victim's win rate prediction oscillates wildly before finally converging on certainty that it will lose; see Figure 5b for an example. This is in stark contrast to the adversary's own predictions, which change much more slowly and are less confident.

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

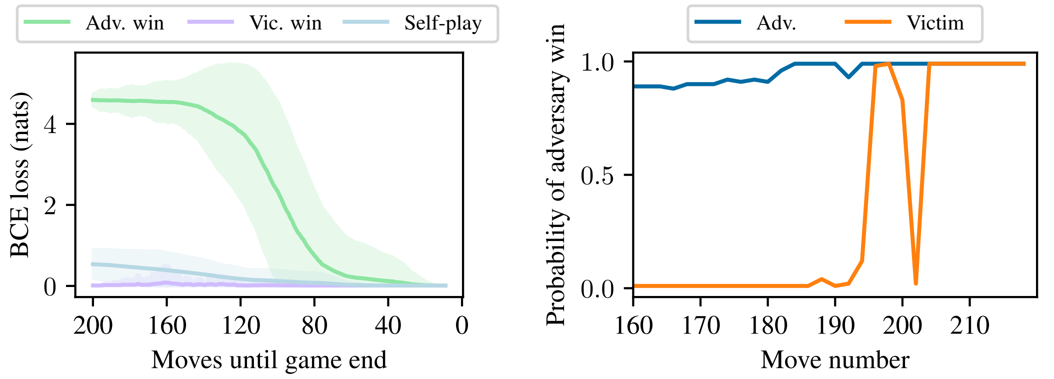
6 Limitations and future work

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

If superhuman policies can be vulnerable to attack, it is natural to ask how we can *defend* against this possibility. Fortunately, there are a number of promising multi-agent RL techniques. An important

[2] In order to speed up evaluations, we used 10 and 40 search threads for the 10^6 and 10^7 visit evaluations respectively (in contrast to the single search thread we used in all other evaluations). For the same number of visits, more search threads generally decreases the strength of the adversary.

[3] Win rate predictions are rounded to the nearest two decimal places before being logged in KataGo.



(a) Binary cross-entropy loss of the predicted win rates for `Latest` at 1,600 visits, playing against an adversary at 200 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 standard deviation.

(b) Probability of adversary victory according to the adversary and victim value networks, for a portion of a randomly selected game. Note the sudden changes in win rate prediction between moves 196 and 204 during a ko fight. Explore the game.

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

direction for future work is to evaluate policies trained with these approaches to determine if they are also exploitable, or if this is an issue limited to self-play.

One promising technique is counterfactual regret minimization (Zinkevich et al., 2007, CFR) that can beat professional human poker players (Brown & Sandholm, 2018). CFR has difficulty scaling to high-dimensional state spaces, but regularization methods (Perolat et al., 2021) can natively scale to games such as Stratego with a game tree 10^{175} times larger than Go (Perolat et al., 2022). Alternatively, methods using populations of agents such as policy-space response oracles (Lanctot et al., 2017), AlphaStar's Nash league (Vinyals et al., 2019) or vanilla PBT (Czempin & Gleave, 2022) may be more robust than self-play, although are substantially more computationally expensive.

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 try and find more effective and compute-efficient methods for attacking agents that use large amounts of search, such as modeling the victim (Appendix B.4).

7 Conclusion

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

KataGo was published in 2019 and has since been used by many Go enthusiasts and professional players as a playing partner and analysis engine (Wu, 2019). However, despite the large amount of attention placed on KataGo, to our knowledge the vulnerabilities discussed in this paper were previously unknown. This suggests that learning-based attacks like the one developed in this paper may be an important tool for uncovering hard-to-find vulnerabilities in AI systems.

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

Author contributions

Tony Wang invented and implemented the A-MCTS-S algorithm, made several other code contributions, and ran and analysed many of the experiments. Adam Gleave managed the project, wrote the majority of the paper, suggested the curriculum approach, helped manage the cluster experiments were run on, and implemented some minor features. Nora Belrose implemented and ran the experiments for baseline adversarial policies, and our pass-hardening defense. Tom Tseng implemented and ran transfer experiments, and trained and ran experiments with the pass-hardening defense enabled. Kellin Pelrine was our resident Go expert and provided analysis of our adversary's strategy. 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.

Acknowledgments

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 Lawrence Chan and Euan McLean for their feedback on earlier drafts of the paper, to ChengCheng Tan and Alyse Spiehler for assistance preparing illustrations, and to Daniel Filan for Go game analysis and feedback on the project direction.

References

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

David Balduzzi, Marta Garnelo, Yoram Bachrach, Wojciech Czarnecki, Julien Pérolat, Max Jaderberg, and Thore Graepel. Open-ended learning in symmetric zero-sum games. In *ICML*, 2019.

Petr Baudiš and Jean-loup Gailly. PACHI: State of the art open source Go program. In *Advances in Computer Games*, 2012.

Petr Baudiš and Jean-loup Gailly. PACHI readme, 2020. URL https://github.com/pasky/pachi/blob/a7c60ec10e1a071a8ac7fc51f7ccd62f006fff21/README.md.

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

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

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

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

Nicholas Carlini, Anish Athalye, Nicolas Papernot, Wieland Brendel, Jonas Rauber, Dimitris Tsipras, Ian Goodfellow, Aleksander Madry, and Alexey Kurakin. On evaluating adversarial robustness. arXiv:1902.06705v2 [cs.LG], 2019.

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

Wojciech M. Czarnecki, Gauthier Gidel, Brendan Tracey, Karl Tuyls, Shayegan Omidshafiei, David Balduzzi, and Max Jaderberg. Real world games look like spinning tops. In *NeurIPS*, 2020.

Pavel Czempin and Adam Gleave. 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/.

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

Adam Gleave, Michael Dennis, Cody Wild, Neel Kant, Sergey Levine, and Stuart Russell. Adversarial policies: Attacking deep reinforcement learning. In *ICLR*, 2020.

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

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

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

Sandy H. Huang, Nicolas Papernot, Ian J. Goodfellow, Yan Duan, and Pieter Abbeel. Adversarial attacks on neural network policies. arXiv:1702.02284v1 [cs.LG], 2017.

Inaam Ilahi, Muhammad Usama, Junaid Qadir, Muhammad Umar Janjua, Ala Al-Fuqaha, Dinh Thai Hoang, and Dusit Niyato. Challenges and countermeasures for adversarial attacks on deep reinforcement learning. *IEEE TAI*, 3(2):90–109, 2022.

Michael Johanson, Kevin Waugh, Michael H. Bowling, and Martin Zinkevich. 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.

Li-Cheng Lan, Huan Zhang, Ti-Rong Wu, Meng-Yu Tsai, I-Chen Wu, and Cho-Jui Hsieh. Are alphazero-like agents robust to adversarial perturbations? In *NeurIPS*, 2022.

Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Perolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. In *NeurIPS*, pp. 4190–4203, 2017.

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

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *ICLR*, 2018.

OpenAI, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. arXiv:1912.06680v1 [cs.LG], 2019.

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

Julien Perolat, Remi Munos, Jean-Baptiste Lespiau, Shayegan Omidshafiei, Mark Rowland, Pedro Ortega, Neil Burch, Thomas Anthony, David Balduzzi, Bart De Vylder, Georgios Piliouras, Marc Lanctot, and Karl Tuyls. From poincaré recurrence to convergence in imperfect information games: Finding equilibrium via regularization. In *ICML*, volume 139, pp. 8525–8535, 2021.

Julien Perolat, Bart de Vylder, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer, Paul Muller, Jerome T. Connor, Neil Burch, Thomas Anthony, Stephen McAleer, Romuald Elie, Sarah H. Cen, Zhe Wang, Audrunas Gruslys, Aleksandra Malysheva, Mina Khan, Sherjil Ozair, Finbarr Timbers, Toby Pohlen, Tom Eccles, Mark Rowland, Marc Lanctot, Jean-Baptiste Lespiau, Bilal Piot, Shayegan Omidshafiei, Edward Lockhart, Laurent Sifre, Nathalie Beauguerlange, Remi Munos, David Silver, Satinder Singh, Demis Hassabis, and Karl Tuyls. Mastering the game of Stratego with model-free multiagent reinforcement learning. arXiv: 2206.15378v1 [cs.AI], 2022.

Hieu Pham, Zihang Dai, Qizhe Xie, and Quoc V. Le. Meta pseudo labels. In *CVPR*, June 2021.

Kui Ren, Tianhang Zheng, Zhan Qin, and Xue Liu. 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.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet large scale visual recognition challenge. *IJCV*, 115(3):211–252, dec 2015.

Martin Schmid, Matej Moravcik, Neil Burch, Rudolf Kadlec, Josh Davidson, Kevin Waugh, Nolan Bard, Finbarr Timbers, Marc Lanctot, Zach Holland, Elnaz Davoodi, Alden Christianson, and Michael Bowling. Player of games. arXiv: 2112.03178v1 [cs.LG], 2021.

Vaishaal Shankar, Rebecca Roelofs, Horia Mania, Alex Fang, Benjamin Recht, and Ludwig Schmidt. Evaluating machine accuracy on ImageNet. In *ICML*, 2020.

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

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

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.

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

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *ICLR*, 2014.

Yuandong Tian, Jerry Ma, Qucheng Gong, Shubho Sengupta, Zhuoyuan Chen, James Pinkerton, and Larry Zitnick. ELF OpenGo: an analysis and open reimplementation of AlphaZero. In *ICML*, 2019.

Finbarr Timbers, Nolan Bard, Edward Lockhart, Marc Lanctot, Martin Schmid, Neil Burch, Julian Schrittwieser, Thomas Hubert, and Michael Bowling. Approximate exploitability: Learning a best response in large games. arXiv: 2004.09677v5 [cs.LG], 2022.

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

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, Laurent Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander S. Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom L. Paine, Caglar Gulcehre, Ziyu Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 2019.

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

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

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

Xian Wu, Wenbo Guo, Hua Wei, and Xinyu Xing. Adversarial policy training against deep reinforcement learning. In *USENIX Security*, 2021.

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

Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. 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.5). 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, 2021). 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.[4]

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 (2021). 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 the players either continue playing the game until the situation is clarified, or a referee arbitrates the outcome of the game.

[4] 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".

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 include 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 (left) 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 evaluating KataGo against other AI systems or human players. In those experiments, we configured `gtp` to follow the same Tromp-Taylor rules described above, with any human-like extensions disabled.

B Search algorithms

B.1 A Review of Monte-Carlo Tree Search (MCTS)

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

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

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. 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.[5] 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).

[5] 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.

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

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

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 are 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[6]

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

   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.
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.[7]

B.1.3 EFFICIENTLY IMPLEMENTING MCTS

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

B.2 ADVERSARIAL MCTS: SAMPLE (A-MCTS-S)

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

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

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

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

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

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:

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

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

B.3 Adversarial MCTS: 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.

In practice, simulating the victim with its full visit count at *every* victim node in the adversary's search tree can be prohibitively expensive. We therefore sometimes simulate the victim using a recursive MCTS search with fewer visits than the victim would actually use. Although this is no longer a perfect model of the victim, it will tend to be more accurate than A-MCTS-S, which wrongly assumes the victim performs *no* search.

B.4 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) uses 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.5 Pass-alive defense

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

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

> 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, 2021):

> 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.

C Hyperparameter settings

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

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

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

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`).

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

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

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

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

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

16–18. ...

19. Checkpoint 505: `b40c256-s11840935168-d2898845681` ($\texttt{Latest}_{\texttt{def}}$), 256 visits.

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

We move on to the next checkpoint when the adversary's win rate exceeds 50% until we reach $\texttt{Latest}_{\texttt{def}}$ without search, at which point we increase the win rate threshold to 75%.

D Strength of Go AI systems

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

D.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 2, on the online Go platform KGS, a slightly earlier (and weaker) checkpoint than `Latest` playing without search is roughly at the level of a top-100 European player. However, some caution is needed in relying on KGS rankings:

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

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

1. Our anchor is ELF OpenGo at 80,000 visits per move, which won all 20 games played against four top-30 professional players, including five games against the now world number one (Tian et al., 2019). We assume that ELF OpenGo at 80,000 visits is strongly su-

perhuman, meaning it has a 90%+ win rate over the strongest current human.[8] 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.

2. The strongest network in the original KataGo paper was shown to be slightly stronger than ELF OpenGo (Wu, 2019, Table 1) when both bots were run at 1600 visits per move. From Figure 6, we see that the relative strengths of KataGo networks is maintained across different amounts of search. We thus extrapolate that strongest network in the original KataGo paper with 80,000 visits would also have an Elo of 4245+ on goratings.org.
3. The strongest network in the original KataGo paper is comparable to the `b15c192-s1503689216-d402723070` checkpoint on katagotraining.org (Wu, 2022). We dub this checkpoint `Original`. In a series of benchmark games, we found that `Latest` without search won 27/3200 games against `Original` with 1600 visits. This puts `Original` with 1600 visits ~823 Elo points ahead of `Latest` without search.
4. Finally, log-linearly extrapolating the performance of `Original` from 1600 to 80,000 visits using Figure 6 yields an Elo difference of ~834 between the two visit counts.
5. Combining our work, we get that `Latest` without search is roughly $823 + 834$ = ~1657 Elo points weaker than ELF OpenGo with 80,000 visits. This would give `Latest` without search an Elo rating of $4245 - 1657$ = ~2588 on goratings.org, putting it at the skill level of a weak professional.

As a final sanity check on these calculations, the raw AlphaGo Zero neural network was reported to have an Elo rating of 3,055, comparable to AlphaGo Fan's 3,144 Elo.[9] 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.

D.2 Strength of KataGo with search

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

Assuming Elo transitivity, we can estimate the strength of `Latest` by utilizing Figure 6. Our evaluation results tell us that `Latest` with 8 playouts/move is roughly 325 Elo stronger than `Latest` with no search. This puts `Latest` with 8 playouts/move at an Elo of ~2913 on goratings.org—within the top 750 in the world. Beyond 128 playouts/move, `Latest` plays at a superhuman level. `Latest` with 512 playouts/move, for instance, is roughly 1762 Elo stronger than `Latest` with no search, giving an Elo of ~4350, over 500 points higher than the top player on goratings.org.

D.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 3. 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

[8] 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.

[9] 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.

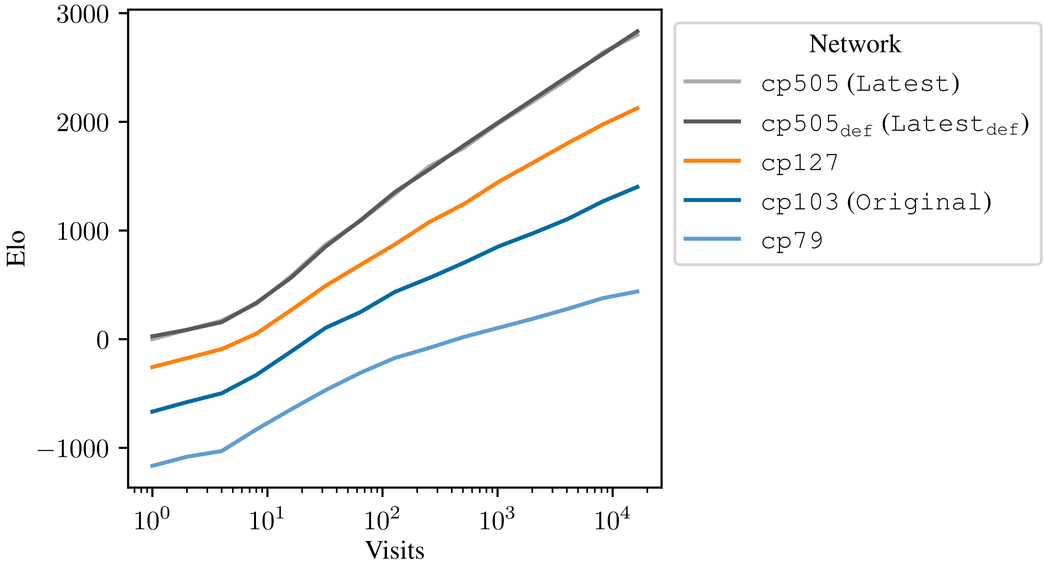


Figure 6: Elo ranking (y-axis) of networks (different colored lines) by visit count (x-axis). The lines are approximately linear on a log x-scale, with the different networks producing similarly shaped lines vertically shifted. This indicates that there is a *consistent* increase in Elo, regardless of network strength, that is logarithmic in visit count. Elo ratings were computed from self-play games among the networks using a Bayesian Elo estimation algorithm (Haoda & Wu, 2022).

Agent	Victim?	Elo (rel GnuGo)	Elo (rel victim)
AlphaZero(s=16k, t=800k)		+3139	+1040
AG0 3-day(s=16k)		+3069	+970
AlphaGo Lee(time=1s)		+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 3: 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.

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), giving Pachi(s=10k) an EGF rating of 2200 GoR.[10] The victim AlphaZero(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.[11] 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.[12] 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

[10] 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 $1/(1+\left(\frac{3300-G_A}{3300-G_B}\right)^7)$.

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

[12] 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.

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 3. However, from Figure 6 we would expect the *relative* Elo to be comparable between the two systems at different visit counts, so we extrapolate AG0 3-day at 16k visits as being an Elo of $3139 - 70 = 3069$ relative to GnuGo.

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

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

E More evaluations of adversaries against KataGo

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

E.1 Evolution of pass-adversary over training

In Figure 7 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 8, 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.

E.2 Score margin of the cyclic-adversary

In Figure 9, we show the margin of victory over the training process of the cyclic-adversary from Section 5.2 against victims with the pass-alive defense. The corresponding win rate is shown in Figure 3. Compared to Figure 8, we see that the margin of victory is typically larger. This is likely because the cyclic-adversary either captures a large group or gives up almost everything in a failed attempt. After approximately 250 million training steps, the margins are relatively stable, but we do see a gradual reduction in the loss margin against $\texttt{Latest}_{\texttt{def}}$ with 2048 visits (preceding the eventual spike in win rate against that victim).

E.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 10a 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

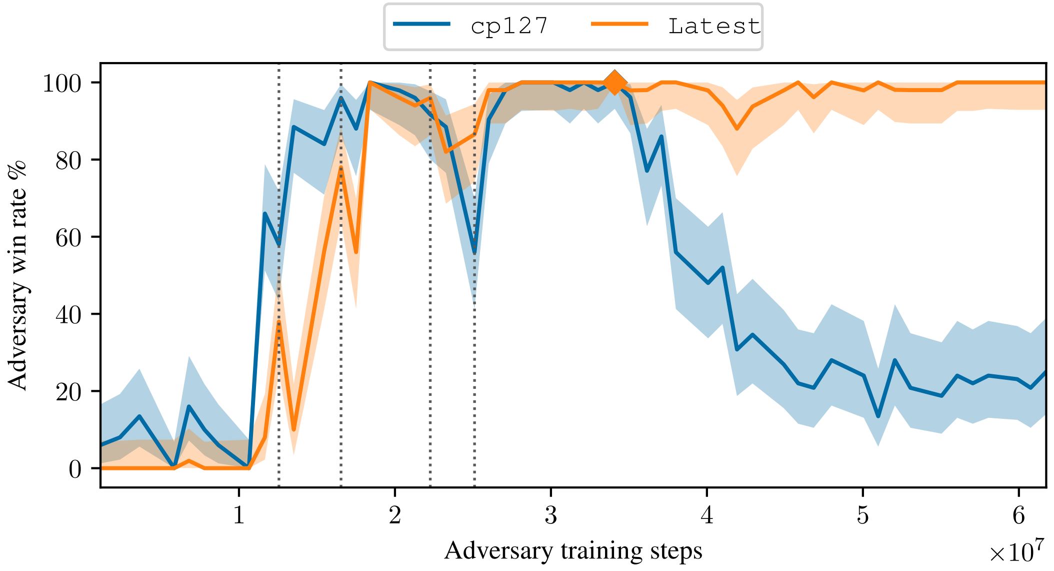


Figure 7: The win rate (y-axis) of the pass-adversary from Section 5.1 over time (x-axis) against the cp127 and Latest victim policy networks playing without search. The strongest adversary checkpoint (marked ⧫) wins 1047/1048 games against Latest. The adversary overfits to Latest, winning less often 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.

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 10b, 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.

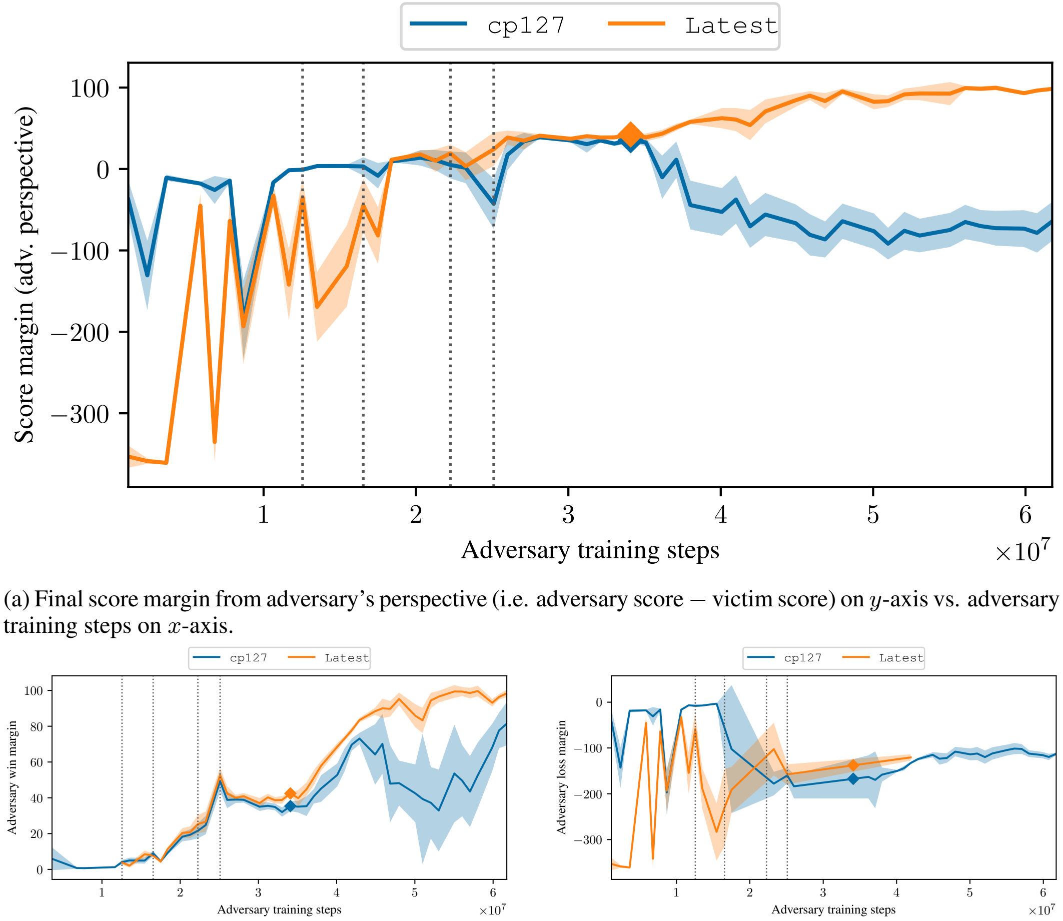
E.4 Transferring attacks between checkpoints

In Figure 11, 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.

E.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. Finally, we also implement *Mirror Go*, a classic novice 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.

In Figure 12, 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 55% win rate when Latest plays as

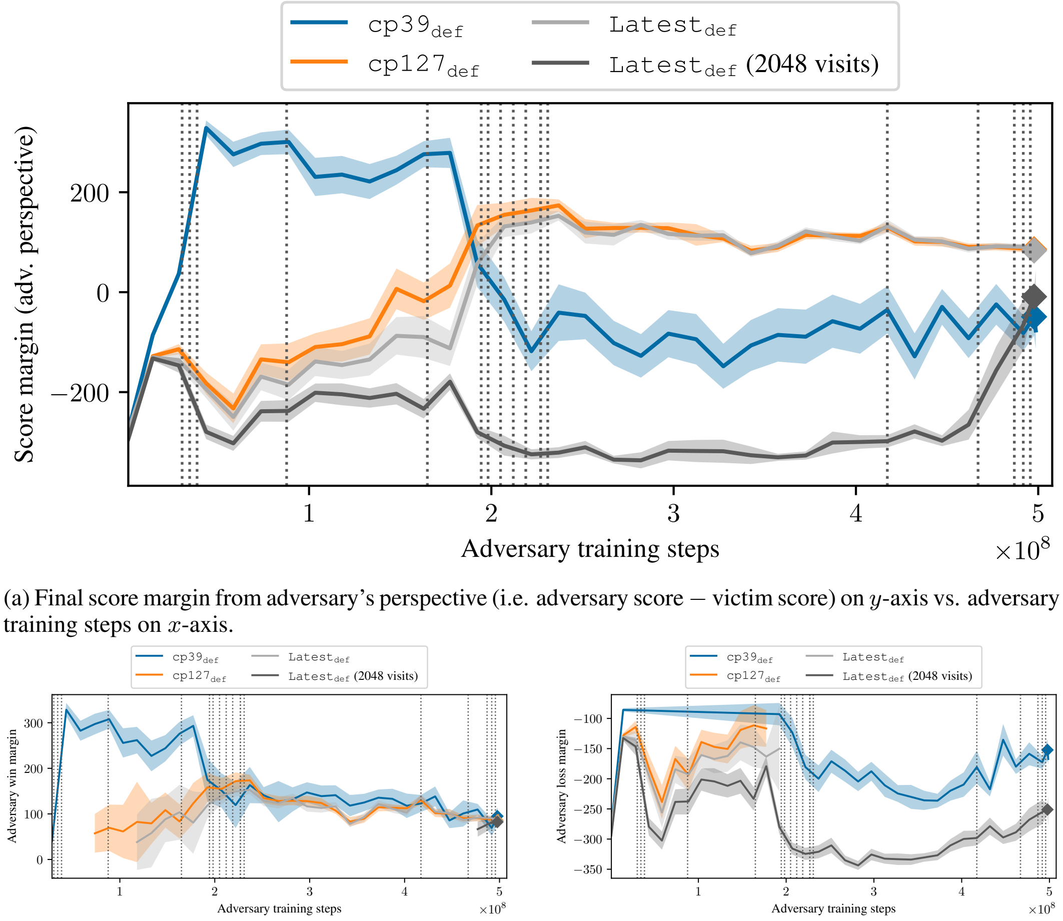


(a) Final score margin from adversary's perspective (i.e. adversary score − victim score) on y-axis vs. adversary training steps on x-axis.

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

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

Figure 8: We evaluate the average margin of victory for the pass-adversary from Section 5.1 as the training process progresses. Shaded regions are 95% T-intervals over $n = 50$ games per checkpoint.

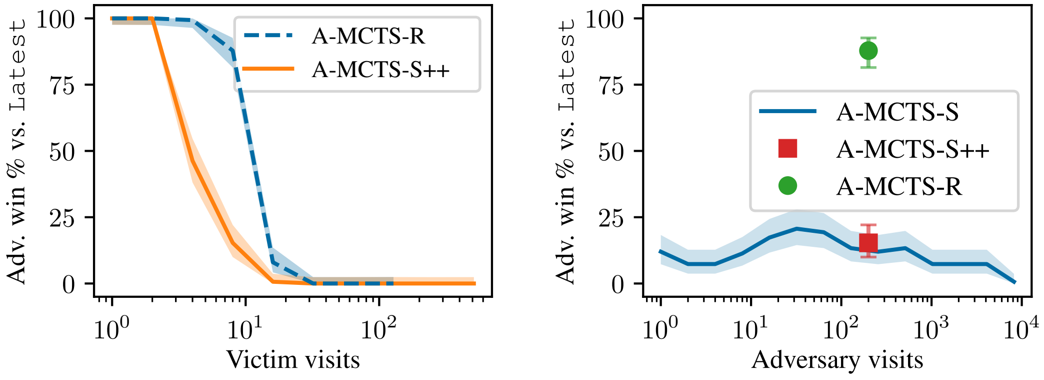


(a) Final score margin from adversary's perspective (i.e. adversary score − victim score) on y-axis vs. adversary training steps on x-axis.

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

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

Figure 9: We evaluate the average margin of victory for the cyclic-adversary from Section 5.2 as the training process progresses. Shaded regions are 95% T-intervals over $n = 50$ games per checkpoint.



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

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

Figure 10: We evaluate the ability of the pass-adversary from Section 5.1 trained against Latest without search to transfer to Latest with search.

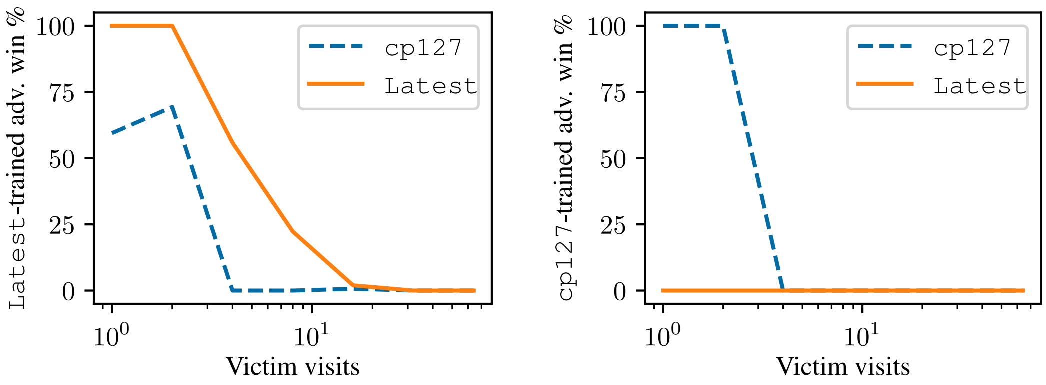
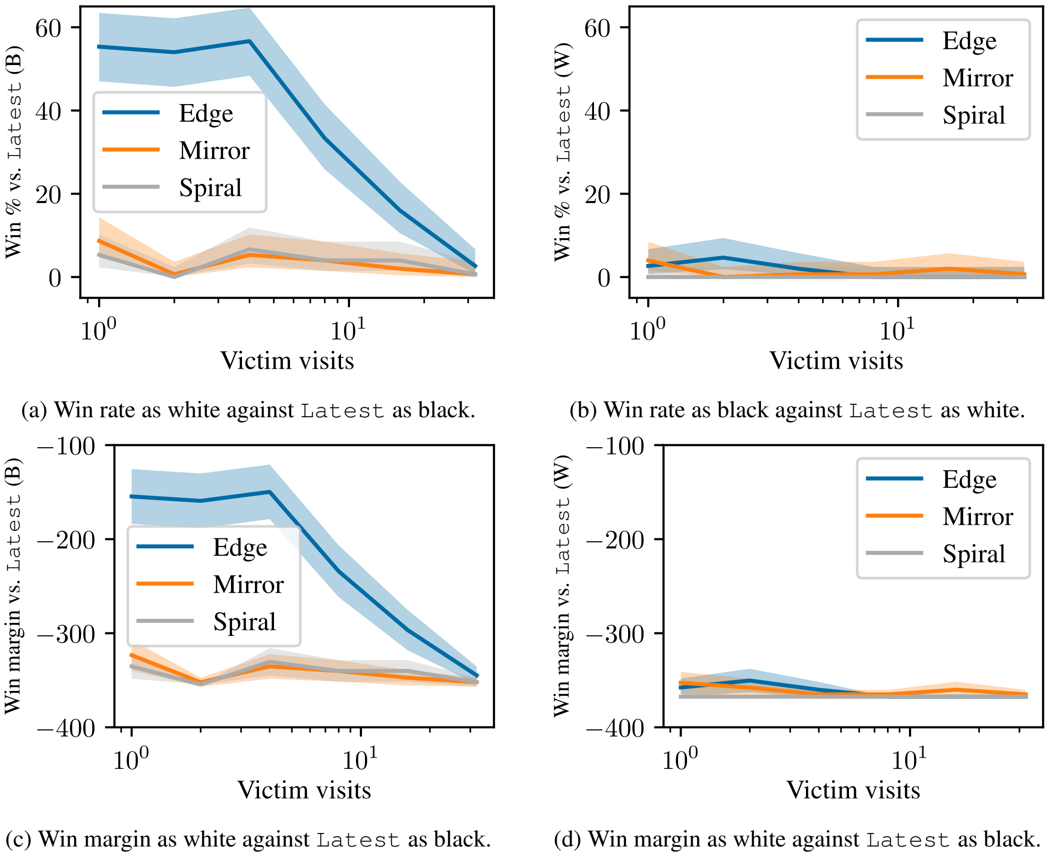


Figure 11: An adversary trained against `Latest` (left) or `cp127` (right), evaluated against both `Latest` and `cp127` at various visit counts. The adversary always uses 600 visits / move.



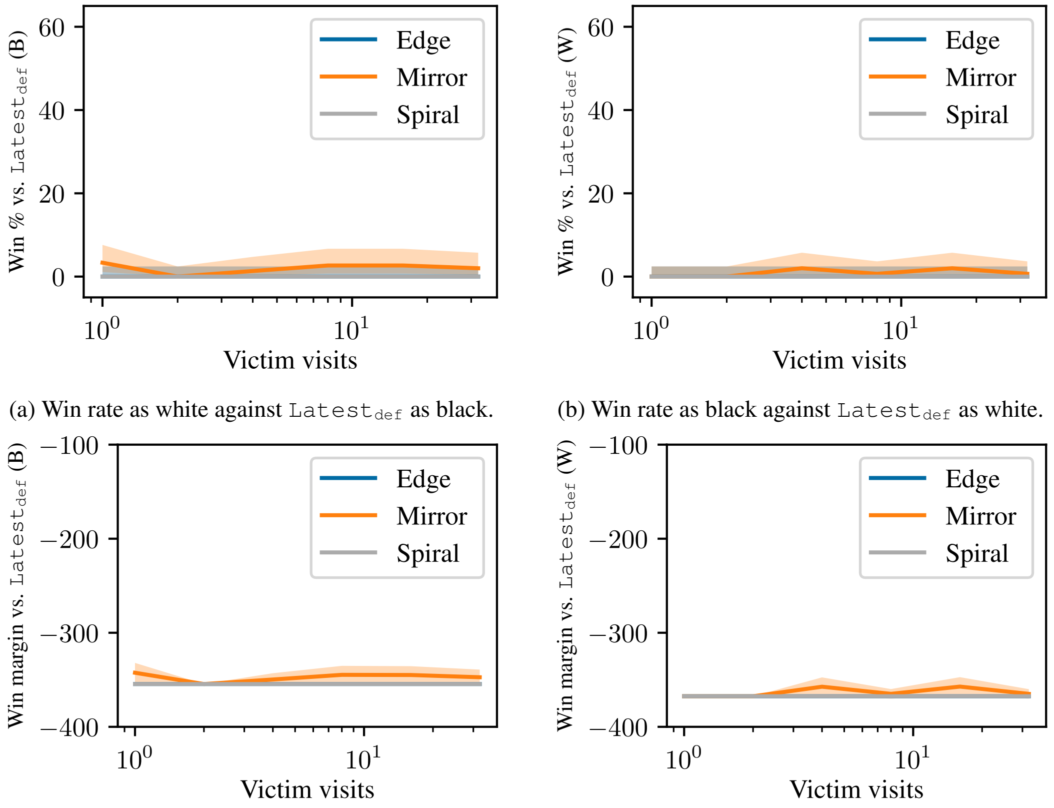
(a) Win rate as white against `Latest` as black.

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

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

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

Figure 12: 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 13: 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.

black with at most 4 visits. None of the attacks work well once `Latest` is playing with at least 32 visits.

In Figure 13, 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.

E.6 Understanding the pass-adversary

We observed in Figure 1a 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 a new adversary that can beat it: the cyclic-adversary). 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 E.5. In Figure 12, we see that all of our baseline attacks perform substantially worse than our pass-adversary (Figure 10a). 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).

We also attempted to manually mimic the adversary's gameplay with limited success in Appendix F.1. Although the basics of our adversarial policy seem easy to mimic, matching its performance is challenging, suggesting it may be performing a more subtle exploit.

F Human experiments and analysis

F.1 Mimicking the pass-adversary

Our passing-based adversarial policy appears to follow a very simple strategy. It plays in the corners and edges, staking out a small region of territory while allowing the victim to amass a larger territory. However, the pass-adversary ensures that it is ahead in raw points prior to the victim securing its territory. If the victim then passes prematurely, the adversary wins.

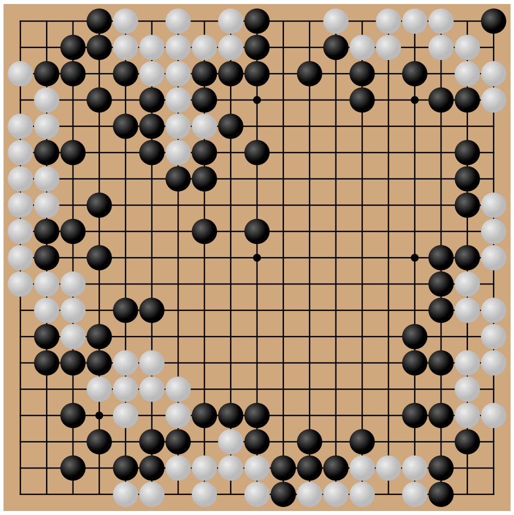


Figure 14: An author of this paper plays as white mimicking our adversarial policy in a game against a KataGo-powered, 8-dan KGS rank bot `NeuralZ06` which has `friendlyPassOk` enabled. White wins by 18.5 points under Tromp-Taylor rules. See the full game.

However, it is possible that this seemingly simple policy hides a more nuanced exploit. For example, perhaps the pattern of stones it plays form an adversarial example for the victim's network. To test this, one of the authors attempted to mimic the adversarial policy after observing some of its games.

The author was unable to replicate this attack when KataGo was configured in the same manner as for the training and evaluation runs in this paper. However, when the `friendlyPassOk` flag in KataGo was turned on, the author was able successfully replicate this attack against the `NeuralZ06` bot on KGS, as illustrated in Figure 14. This bot uses checkpoint 469 (see Appendix C.1) with no search. The author has limited experience in Go and is certainly weaker than 20 kyu, so they did not win due to any skill in Go.

F.2 Humans vs. adversarial policies

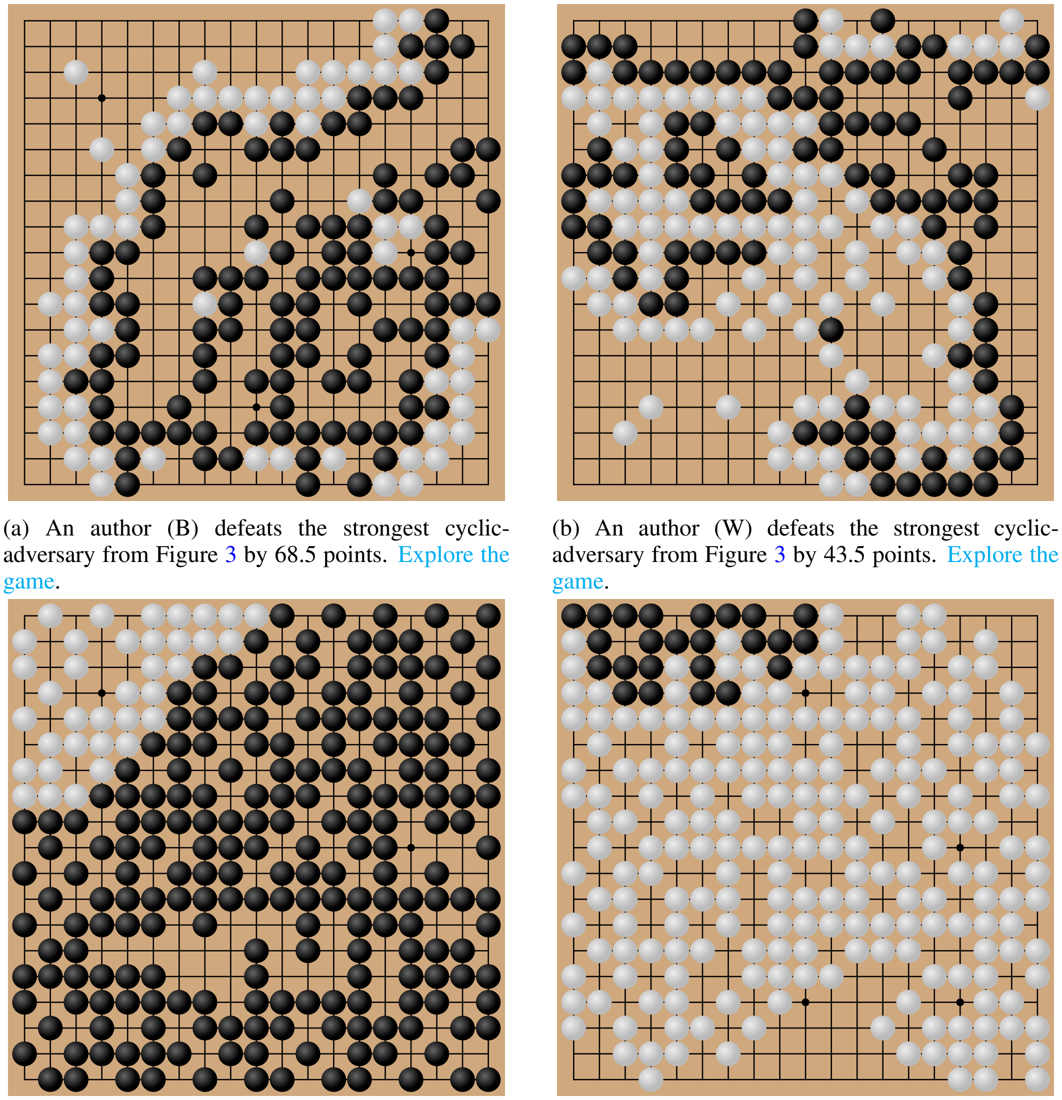
The same author from Appendix F.1 (strength weaker than 20kyu) played manual games against both the strongest cyclic-adversary from Figure 3 and the strongest pass-adversary from Figure 7. In the games against the pass-adversary, the author was able to achieve an overwhelming victory. In the game against the cyclic-adversary, the author won but with a much smaller margin. See Figure 15 for details.

Our evaluation is imperfect in one significant way: the adversaries did not play with an accurate model of the author (rather they modeled the author as `Latest` with 1 visit). However, given the limited transferability of our adversaries to different KataGo checkpoints (see Figure 3, Figure 7, and Appendix E.4), we conjecture that our adversaries would not win even if they had access to an accurate model of the author.

F.3 Human analysis of the cyclic-adversary

In the following we present human analysis of games with the cyclic-adversary (the type shown in Figure 1b) 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 This game 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.



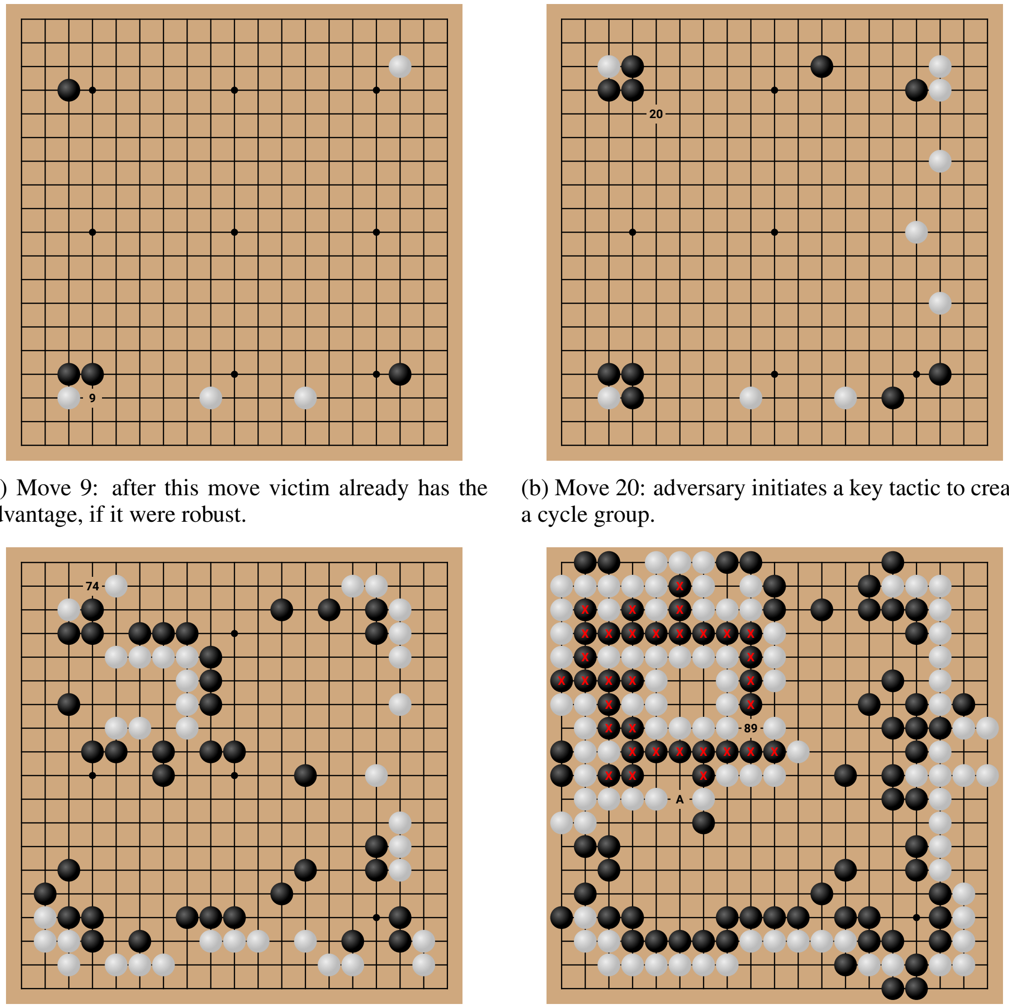
(a) An author (B) defeats the strongest cyclic-adversary from Figure 3 by 68.5 points. Explore the game.

(b) An author (W) defeats the strongest cyclic-adversary from Figure 3 by 43.5 points. Explore the game.

(c) An author (B) defeats the strongest pass-adversary from Figure 7. Explore the game.

(d) An author (W) defeats the strongest pass-adversary from Figure 7 using A-MCTS-S++. Explore the game.

Figure 15: Games between an author of this paper and the strongest adversaries from Figure 3 and Figure 7. In all games, the author achieves a victory. The adversaries used 600 playouts / move and used `Latest` as the model of its human opponent. The adversaries used A-MCTS-S for all games except the one marked otherwise.



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

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

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

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

Figure 16: 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.

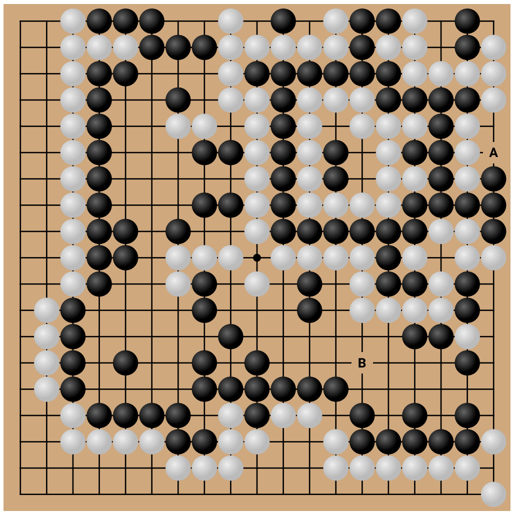


Figure 17: A game the cyclic-adversary (white) lost. The adversary could take a decisive lead by capturing at A, but instead plays B and lets the victim (black) save their group.

In this game the victim plays black and the adversary white. The full game is available on our website. We see in Figure 16a 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 16b), 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 16c), 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 16d), 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 16d 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 17. But instead it allowed the victim to save the group and win the game. This might suggest the adversary itself is not completely immune to misjudging cycle groups. However, it could also be that the adversary is simply not that skilled at Go in general and makes many mistakes.

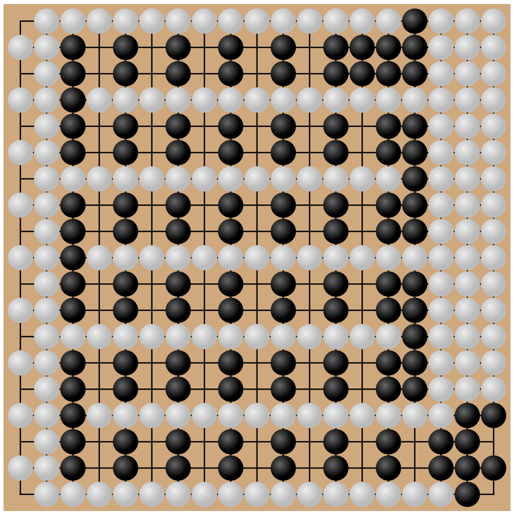


Figure 18: A hand-crafted adversarial example for KataGo and other Go-playing AI systems. It is black's turn to move. Black can guarantee a win by connecting its currently disconnected columns together and then capturing the large white group on the right. However, KataGo playing against itself from this position loses 40% of the time as black.

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.

G 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 18 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).

H Known failures of Go-playing agents

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

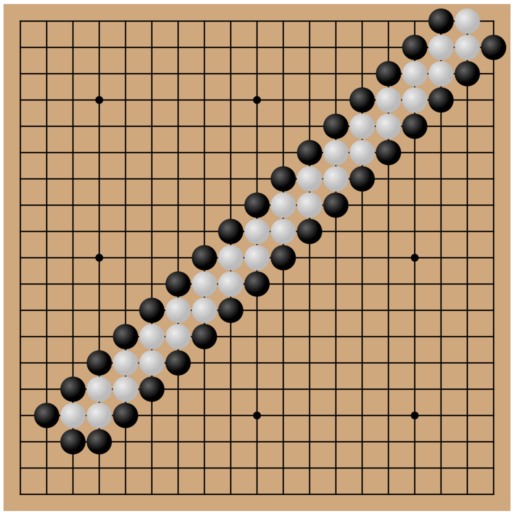


Figure 19: Illustration of a ladder. White ran across the board, but has hit the top edge and has nowhere left to run. Black can capture on the next move by playing in the top right corner.

Ladders A "ladder" is often the first tactic a beginner learns. An example is shown in Figure 19. 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.

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

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

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

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

I CHANGELOG

`arxiv.org/abs/2211.00241v1` → `arxiv.org/abs/2211.00241v2` changes:

- We train, evaluate, and analyze a new *cyclic-adversary* (Figure 1b) which behaves in a qualitatively different manner from our `v1` adversary (now called the *pass-adversary*). Our cyclic-adversary can beat KataGo playing with up to 10^7 visits per move of search.
- We add a detailed description of the Go rules used in our evaluations (Appendix A).
- We add an estimate of the strength of the AlphaZero agent that was successfully adversarially attacked by Timbers et al. (2022) using methods similar to our own (Appendix D.3).
- We redo the evaluations of our `v1` pass-adversary with configurations more similar to those used in actual match-play (Appendix E).
- We add a summary of known failures of Go AIs (Appendix H).
- We make small improvements to figures and text across the whole paper.
- We update our paper website (`goattack.far.ai`) to reflect the `v2` paper changes.