ADVERSARIAL POLICIES BEAT PROFESSIONAL-LEVEL GO AIS

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

ABSTRACT

We attack the state-of-the-art Go-playing AI system, KataGo, by training an adversarial policy that plays against a frozen KataGo victim. Our attack achieves a >99% win-rate against KataGo without search, and a >50% win-rate when KataGo uses enough search to be near-superhuman. To the best of our knowledge, this is the first successful end-to-end attack against a Go AI playing at the level of a top human professional. Notably, the adversary does not win by learning to play Go better than KataGo—in fact, the adversary is easily beaten by human amateurs. Instead, the adversary wins by tricking KataGo into ending the game prematurely at a point that is favorable to the adversary. Our results demonstrate that even professional-level AI systems may harbor surprising failure modes. Example games are available at https://goattack.alignmentfund.org/.

1 INTRODUCTION

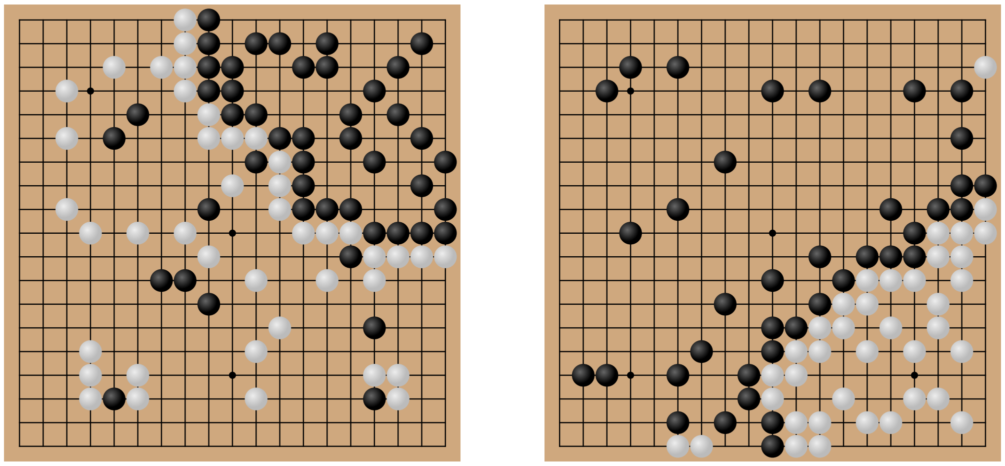
Reinforcement learning from self-play has achieved superhuman performance in a range of games including Go (Silver et al., 2016), chess and shogi (Silver et al., 2016), and Dota (OpenAI et al., 2019). Moreover, idealized versions of self-play provably converge to Nash equilibria (Brown, 1951; Heinrich 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 attack KataGo (Wu, 2019), the strongest publicly available Go-playing AI system. We train an adversarial policy end-to-end against a fixed victim policy network. Using only 0.3% the compute used to train KataGo, we obtain an adversarial policy that wins >99% of the time against KataGo with no search, and >50% against KataGo with enough search to be near-superhuman.

Critically, our adversary does *not* win by learning a generally capable Go policy. Instead, the adversary has learned a simple strategy that stakes out a small corner territory, then places some easily capturable stones in KataGo's larger complementary territory. This strategy, illustrated and explained in Figure 1, loses against even amateur Go players (see Section A.6.2), so the victim is in this instance less *robust* than human amateurs, despite having professional-level *capabilities*. This is a striking example of non-transitivity, illustrated in Figure 2.

Our adversary does not have any special powers: it can only place stones, or pass, like a regular player. We do, however, give the adversary gray-box access to a fixed KataGo victim. In particular, we train the adversary using an AlphaZero-style training process (Silver et al., 2018), similar to that of KataGo. The key difference is that we collect games with the adversary playing the separate (fixed) victim network. Additionally, we use the victim network to select victim moves during the *adversary's* tree search.



(a) Adversary plays as black: explore the game. (b) Adversary plays as white: explore the game.

Figure 1: The adversarial policy beats the KataGo victim by playing a counterintuitive strategy: staking out a minority territory in the corner, allowing KataGo to stake the complement, and placing weak stones in KataGo's stake. KataGo predicts a high win probability for itself and, in a way, it's right—it would be simple to capture most of the adversary's stones in KataGo's stake, achieving a decisive victory. However, KataGo plays a pass move before it has finished securing its territory, allowing the adversary to pass in turn and end the game. This results in a win for the adversary under the standard ruleset for computer Go, Tromp-Taylor (Tromp, 2014), as the adversary gets points for its corner territory (devoid of victim stones) whereas the victim does not receive points for its unsecured territory because of the presence of the adversary's stones. These games are randomly selected from an attack against `Latest`, the strongest policy network, playing without search.

KataGo is the strongest publicly available Go AI system at the time of writing. With search, KataGo is strongly superhuman, winning (Wu, 2019, section 5.1) against ELF OpenGo (Tian et al., 2019) and Leela Zero (Pascutto, 2019) that are themselves superhuman. In Section A.5, we estimate that KataGo without search plays at the level of a top 100 European player, and KataGo with a small amount of search is at the level of the best Go players on earth.

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 adversarial policies against the state-of-the-art Go AI system, KataGo. Finally, we find the adversary pursues a simple strategy that fools the victim into predicting victory causing it to pass prematurely. 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 an 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, recent work has argued that real-world games like Go grow increasingly transitive as skill increases (Czarnecki et al., 2020). This would imply that while self-play may struggle with non-

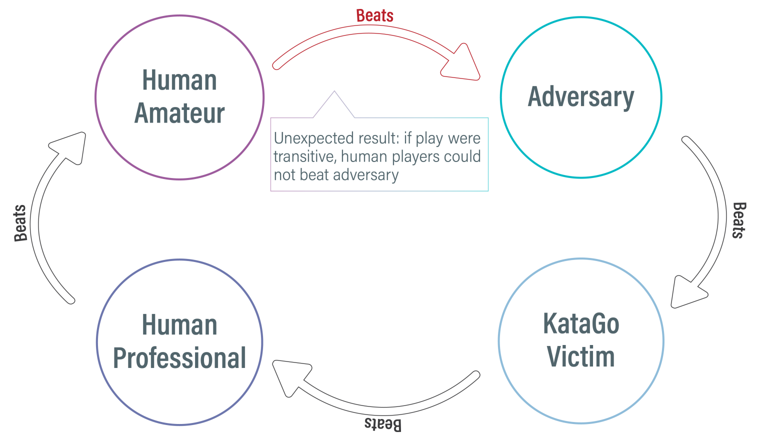

Beats
Human Amateur
Adversary
Unexpected result: if play were transitive, human players could not beat adversary
Beats
KataGo Victim
Beats
Human Professional
Beats

Figure 2: Our adversarial policy can be beaten by a human amateur (Section A.6.2), but can in turn beat the professional-level KataGo AI. This non-transitivity shows the adversary is exploiting particular blindspots in KataGo, not simply learning a highly capable policy.

transitivity early on during training, comparisons involving highly capable policies such as KataGo should be largely transitive. By contrast, we find a striking non-transitivity: our adversarial policy exploits KataGo agents that reliably beat human professionals, yet even an amateur Go player can beat our adversarial policy (see Section A.6.2 for details).

Most prior work attacking deep RL has focused on perturbations to observations (Huang et al., 2017; Ilahi et al., 2022). However, such attacks are often not physically realizable, so 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 small changes in the victim's 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 to 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.

The most closely related work to ours is by Timbers et al. (2020), who develop the *approximate best response* (ABR) method to estimating exploitability. Both our and their method use the same key idea: an AlphaZero-style training procedure that is adapted to use the *opponent*'s policy during search. However, they differ in some details. For example, we attempt to model the victim's search process inside our adversary via A-MCTS-R (see Section 4). Additionally, our curriculum uses checkpoints, not just search. Nonetheless, our method does not represent any major conceptual breakthrough: indeed, our results are striking in large part because the attack is so simple.

Our main contribution relative to Timbers et al. (2020) lies in our empirical results. The victims that we manage to exploit range in strength from a weak 1p/2p human professional to the strongest humans on earth (see Section A.5). By contrast, Timbers et al. only establish that their victim is at least that of a strong 3d/4d amateur, as they report winning 100% against Pachi at 100,000 iterations (Baudiš & Gailly, 2012). We can infer Timbers et al.'s victim must be significantly weaker than AlphaZero, as their fully-trained Chess agent was unable to beat the Chess engine Stockfish beyond level 5, despite being trained for *longer* than their Go agent. Unfortunately the exact strength is impossible to determine as there is no direct comparison to state-of-the-art Go systems or human players. An updated version of their paper (Timbers et al., 2022) omits the strength evaluation, but the new victim is presumably weaker than the original as it was trained with fewer time steps.

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 Section A.5).

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. 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 A.4 for the hyperparameter settings we used in experiments.

Adversarial MCTS: Sample (A-MCTS-S). In A-MCTS-S, we modify the adversary's search procedure to sample from the victim's policy network at *victim-nodes* in the Monte Carlo tree when 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 optimizations added in KataGo, such as adding noise to the policy prior 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. See Appendix A.2 for a full description, and Appendix A.1 for a review of the base MCTS algorithm.

Adversarial MCTS: Recursive (A-MCTS-R). A-MCTS-S systematically 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 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 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. We use KataGo's auxiliary targets to partially alleviate this problem: 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 50%. In particular, as KataGo releases the entire training history, we start with an early checkpoint and then move to later victim checkpoints.

Baselines. We also test *hard-coded* baseline adversarial policies. These baselines were inspired by the behavior of our trained adversaries. The *Edge* 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 which 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.

5 Evaluation

We evaluate our attack method against KataGo (Wu, 2019). In Section 5.1 we find our A-MCTS-S algorithm achieves a greater than 99% win rate against `Latest` playing without search. Notably `Latest` is very strong even without search: we find in Section A.5.1 that it is comparable to a top 100 European player. Our attack manages to transfer to the low-search regime as well, our best result in Section 5.2 being a 54% win rate against `Latest` with 64 playouts. In Section A.5.2, we estimate that `Latest` with 64 playouts is comparable to the best human Go players. However, our adversary performs poorly against `Latest` with 128 or more playouts.

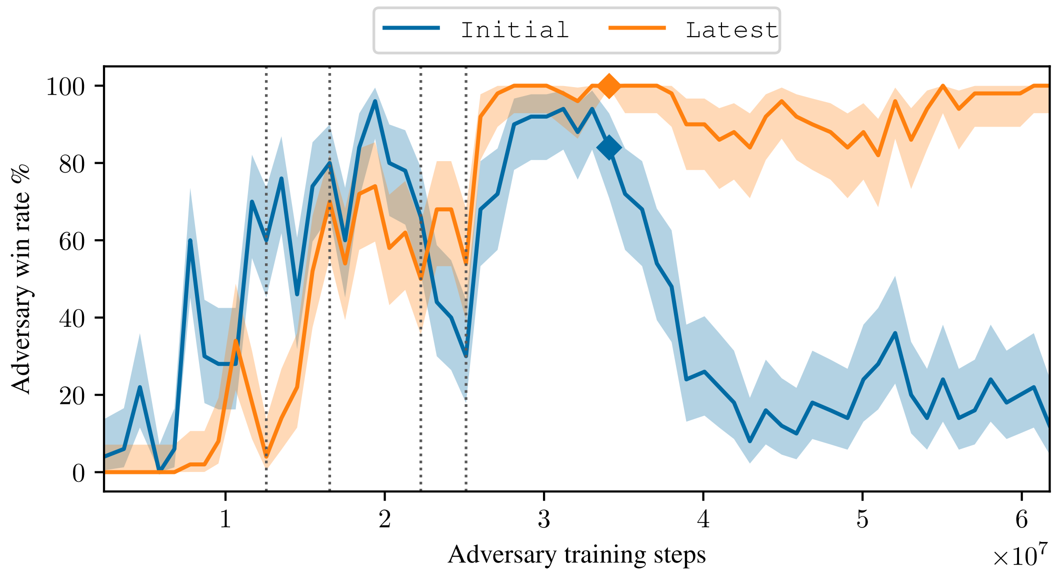


Figure 3: The win rate (y-axis) of the adversarial policy over time (x-axis) against the `Initial` and `Latest` victim policy networks playing without search. The strongest adversary checkpoint (marked ◆) wins 999/1000 games against `Latest`. The adversary overfits to `Latest`, winning less often against `Initial` 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 `Initial` and ending at `Latest`. Vertical dashed lines denote switches to a later victim training policy.

Additionally, we find in Section 5.3 that the adversarial policy is specialized to the victim it is attacking, and has limited transfer to other victims. Finally, in Section 5.4 we study how the attack works, including investigating the value prediction of the victim and evaluating a hard-coded defense.

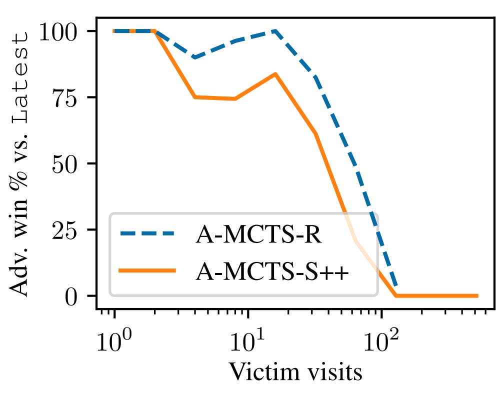
5.1 Attacking the Victim Policy Network

We train an adversarial policy using A-MCTS-S and a curriculum, as described in Section 4. We start from a checkpoint `Initial` around a quarter of the way through training, until reaching the `Latest` checkpoint corresponding to the strongest KataGo network (see Appendix A.4.1 for details). In Figure 3 we evaluate our adversarial policy against the policy networks of both `Initial` and `Latest`. We find our 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 `Initial` falling to around 20%.

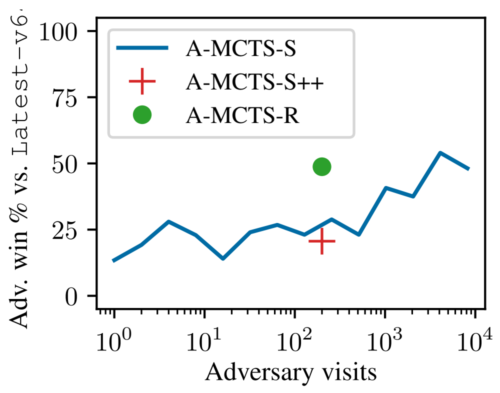
We evaluate our best adversarial policy checkpoint against `Latest`, achieving a greater than 99% win rate. Notably, this high win rate is achieved despite our adversarial policy being trained for only 3.4×10^7 time steps – just 0.3% as many time steps as the victim it is exploiting. Critically, this adversarial policy does not win by playing a stronger game of Go than the victim. Instead, it follows a bizarre strategy illustrated in Figure 1 that loses even against human amateurs (see Section A.6.2).

5.2 Transfering to a Victim With Search

We evaluate the ability of the adversarial policy trained in the previous section to exploit `Latest` playing *with* search. Although this adversarial policy achieves a win rate greater than 99% against `Latest` without search, in Figure 4a we find the win rate of A-MCTS-S drops to 80% at 32 victim visits. However, A-MCTS-S models the victim as having no search at both training and inference time. We also test A-MCTS-R, which correctly models the victim at inference by performing an MCTS search at each victim node in the adversary's tree. We find that A-MCTS-R performs somewhat better, obtaining a greater than 99% win rate against `Latest` with 32 visits, but performance drops below 10% at 128 visits.



(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 more than 100 visits.



(b) Win rate by number of adversary visits with A-MCTS-S, playing against `Latest` with 64 visits. We see that higher adversary visits lead to moderately higher win rates.

Figure 4: We evaluate the ability of the adversarial policy from Section 5.1 trained against `Latest` without search to transfer to `Latest` with search.

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. In Figure 4b we show that we obtain up to a 54% win rate against `Latest` with 64 visits when the adversary has 4,096 visits. This is very similar to the performance of A-MCTS-R with 200 visits, which has a 49% win rate against the same victim. Interestingly, the inference cost of these attacks is also similar, with 4,096 neural network forward passes (NNFPs) per move for A-MCTS-S (one per visit) versus 6,500 NNFPs / move for A-MCTS-R.[1]

Note these experiments only attempt to transfer our adversarial policy. It would also be possible to repeat the attack from scratch against a victim with search, producing a new adversarial policy. We leave this for future work as we cannot currently run this attack due to computational constraints.

5.3 Transfering to Other Checkpoints

From Figure 9 in the appendix, we see that an adversary trained against `Latest` does better against `Latest` than `Initial`, despite `Latest` being a stronger agent. The converse also holds: an agent trained against `Initial` does better against `Initial` than `Latest`. This pattern holds for most visit counts where the adversary wins consistently, although in the case of the adversary for `Latest` the gap is fairly narrow (99% vs 80% win rate) at low visit counts. These results suggest that different checkpoints have unique vulnerabilities.

5.4 Understanding the Attack

We observed in Figure 1 that the 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 adversary 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 adversary from section 5.1 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.

[1]A-MCTS-R with 200 visits performs $100 \cdot 64 + 100 = 6500$ NNFPs / move.

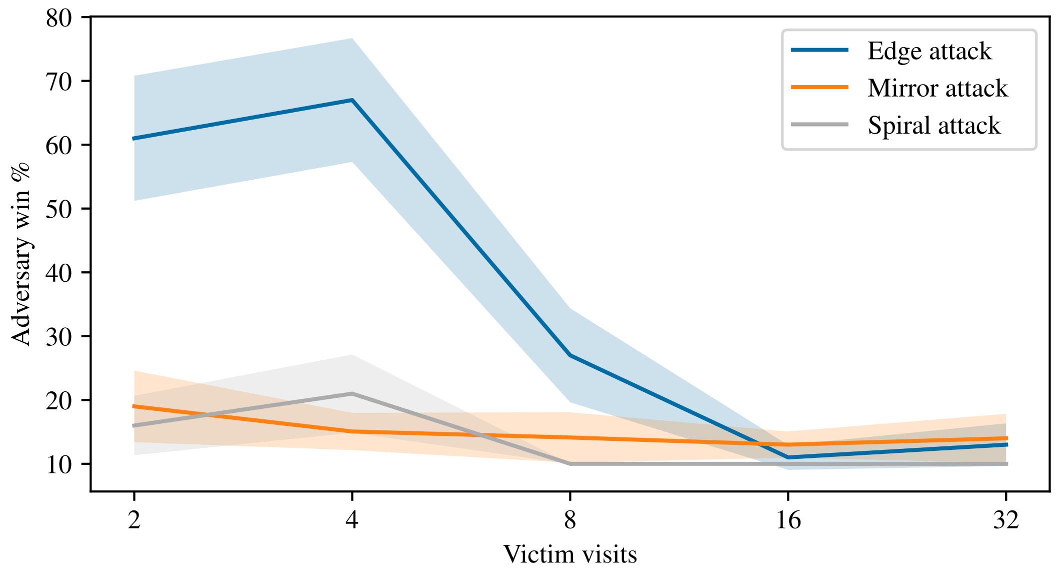


Figure 5: Win rates of different baseline adversaries (see Section 4) versus `Latest` at varying visit counts (x-axis) *with adversary playing as white*. 95% CIs are shown. See Figure 10 in the appendix for average win margin of baselines.

We conjecture the victim's prediction is so mistaken as the games induced by playing against the adversarial policy are very different to those seen during the victim's self-play training. Certainly, there is no fundamental inability for 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: 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 which extends the traditional Go notion of a pass-alive group (Wu, 2021) (see Appendix A.3 for a full description of the algorithm).

We apply this defense to the `Latest` policy network. Whereas the adversarial policy in Section 5.1 won greater than 99% of games against vanilla `Latest`, we find that it *loses* all 1600 evaluation games against `Latest` *with* this defense. This confirms the adversarial policy wins via passing.

Unfortunately, this "defense" has two undesirable properties. First, it causes KataGo to continue to play even when a game is clearly won or lost, which is frustrating for human opponents. In fact, the average game length when playing with pass-hardening against an adversarial policy increases from 95 to 421 moves—over a factor of four! This is also almost twice that of the 211 moves typical for Go games between professional players (Brouwer, 2014).

Second, the defense relies on hard-coded knowledge about Go, using a search algorithm to compute the pass-alive territories. Ideally, we would be able to apply AI techniques to systems where humans do not have such domain expertise: indeed, this was the key contribution of AlphaZero (Silver et al., 2018) over AlphaGo (Silver et al., 2016). Moreover, there may be games where AI systems are vulnerable but where there is no simple algorithm to address the vulnerabilities.

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 Section 4. In Figure 5, we see that all of our baseline attacks perform substantially worse than our trained adversarial policy (Figure 4a). Moreover, all our baseline attacks only win by komi, and so never win as black. By contrast, our adversarial policy in Section 5.1 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 game play with limited success.[2] Although the basics of our adversarial policy seem readily mimicable, matching its performance is challenging, suggesting it may be performing a more subtle exploit.

6 Limitations and Future Work

This paper has demonstrated that even agents at the level of top human professionals can be vulnerable to adversarial policies. However, our results do not establish how common such vulnerabilities are. It is possible that KataGo is unusually vulnerable, although this seems unlikely as it is the strongest publicly-available Go AI system and uses a similar training process to other systems.

Alternatively, perhaps it is easier to exploit AI systems in Go than in other games. Notably, Go games typically end by both players passing (or one player resigning), and our attack succeeds by tricking KataGo into ending the game prematurely. This failure mode would not occur in a game like chess with a clear termination condition, checkmate. A promising direction for future work is to evaluate our attack against strong AI systems in other games.

If professional-level policies can be vulnerable to attack, it is natural to ask how we can *defend* against this possibility. A simple approach is adversarial training: fine-tuning the victim agent against the trained adversary. This approach was attempted in Gleave et al. (2020) but with limited success—repeating the attack against the fine-tuned victim found a new adversarial policy that wins against the fine-tuned and original victim. However, it is possible adversarial training will work better for policies that are already professional-level, and so have fewer distinct failure modes. Alternatively, Czempin & Gleave (2022) find population-based training (PBT) leads to increased robustness against adversarial policies, albeit with a substantial increase in training time.

Finally, we found it much 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 see if there exist more effective and compute-efficient methods for attacking agents that use search. If such methods do not exist, then search may be a viable defense against adversaries.

7 Conclusion

We have developed the first adversarial policy exploiting an AI agent that performs at the level of a top human professional. Notably, the adversarial policy does not win by playing a strong game of Go—in fact, the adversarial policy can be easily beaten by a human amateur. Instead, the adversary wins by exploiting a particular blindspot in the victim agent. This result suggests that even highly capable agents can harbor serious vulnerabilities.

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

Our results underscore that improvements in capabilities do not always translate into adequate robustness. These failures in Go AI systems are entertaining, but a similar failure in safety-critical systems such as automated financial trading or autonomous vehicles could have dire consequences. We believe 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 the majority of the experiments. Adam Gleave managed the project,

[2]We were only able to perform a manual exploit when the `friendlyPassOk` flag in KataGo was set to true. This flag makes KataGo more willing to pass. However, this flag is set to false in all of our training and evaluation runs. See Appendix A.6.1 for details.

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 defence. Tom Tseng implemented and ran transfer experiments alongside numerous minor contributions to the codebase. 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 Lawrence Chan and Euan McLean for their feedback on earlier drafts of the paper, 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.

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

Andries E. Brouwer. Statistics on the length of a Go game, 2014. URL https://homepages.cwi.nl/~aeb/go/misc/gostat.html.

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

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/.

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

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/.

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.

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, 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: 2004.09677v3 [cs.LG], 2020.

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 *IJCAI*, Jul 2022.

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

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.

A Appendix

A.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.

A.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.[3] 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).

[3]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[4]

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

   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 $S_T(x) = 0$.

The first term in Eq 1 can be thought of as the exploitation term, and the second term can be thought of as the exploration term and is inspired by UCB algorithms.

A.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.[5]

A.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.

A.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 A.1.

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

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

Here, $S_T^{\text{A-MCTS}}$ is a modified version of S_T that measures the size of a subtree while ignoring non-terminal victim-nodes (at victim-nodes it is the victim's turn to move, and at self-nodes it is the

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

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

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:

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

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 $\bar{V}_T(c)$.

A.3 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 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.

A.4 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. This is much smaller than the victim, which uses a `b20c256` or `b40c256` architecture. We additionally disable a variety of game rule

Hyperparameter	**Value**	**Different from KataGo?**
Batch Size	256	Same
Learning Rate Scale of Hardcoded 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.

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.

A.4.1 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` (`Initial`).
2. Checkpoint 200: `b40c256-s5867950848-d1413392747`.
3. Checkpoint 300: `b40c256-s7455877888-d1808582493`.
4. Checkpoint 400: `b40c256-s9738904320-d2372933741`.
5. Checkpoint 469: `b40c256-s11101799168-d2715431527`.
6. Checkpoint 505: `b40c256-s11840935168-d2898845681` (`Latest`).

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

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

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

A.5 Strength of KataGo

In this section, we estimate the strength of `Latest` with and without search.

A.5.1 Strength of KataGo Without Search

We estimate strength using two independent methodologies and conclude that `Latest` without search is at the level of a weak professional.

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, 2022). 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`.

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 pro. However, some caution is needed in relying on KGS rankings:

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

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

1. Our anchor is ELF OpenGo at 80,000 visits per move, which won all 20 games played against four top-30 professional players, including five games against the now world number one (Tian et al., 2019). We assume that ELF OpenGo at 80,000 visits is strongly superhuman, meaning it has a 90%+ winrate over the strongest current human.[6] 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 KataGo at 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

[6]This assumption is not entirely justified by statistics, as a 20:0 record only yields a 95% binomial lower confidence bound of a 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.

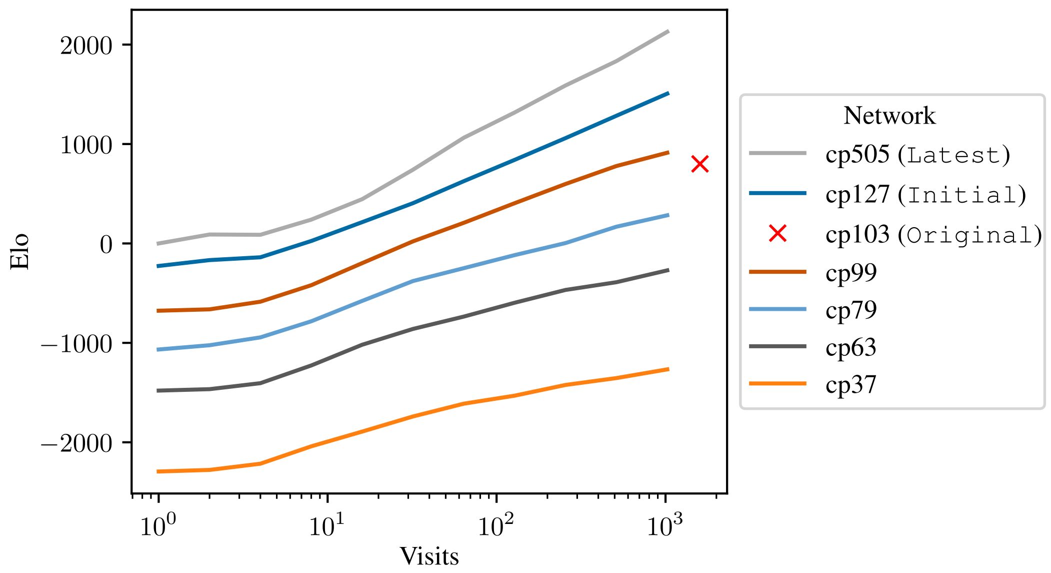


Figure 6: Elo ranking (y-axis) of networks (different colored lines) by visit count (x-axis). The lines are approximately linear on a log x-scale, with the different networks producing similarly shaped lines vertically shifted. This indicates that there is a *consistent* increase in Elo, regardless of 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).

that `Latest` without search won 29/3200 games against `Original` with 1600 visits. This puts `Original` with 1600 visits ~800 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 ~900 between the two visit counts.
5. Combining our work, we get that `Latest` without search is roughly $800 + 900$ = ~1700 Elo points weaker than ELF OpenGo with 80,000 visits. This would give `Latest` without search an Elo rating of $4245 - 1700$ = ~2500 on goratings.org, putting it at the skill level of a weak professional.

A.5.2 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 ~2500 on goratings.org.

Assuming Elo transitivity, we can estimate the strength of `Latest` by utilizing Figure 6. In particular, our evaluation results tell us that `Latest` with 64 playouts/move is roughly 1063 Elo stronger than `Latest` with no search. This puts `Latest` with 64 playouts/move at an Elo of ~3563 on goratings.org—within the top 20 in the world.

A.6 Experimental Results

A.6.1 Mimicking the Adversarial Policy

Our adversarial policies appear to follow a very simple strategy. They play in the corners and edges, staking out a small region of territory while allowing the victim to amass a larger territory. However, the 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.

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.

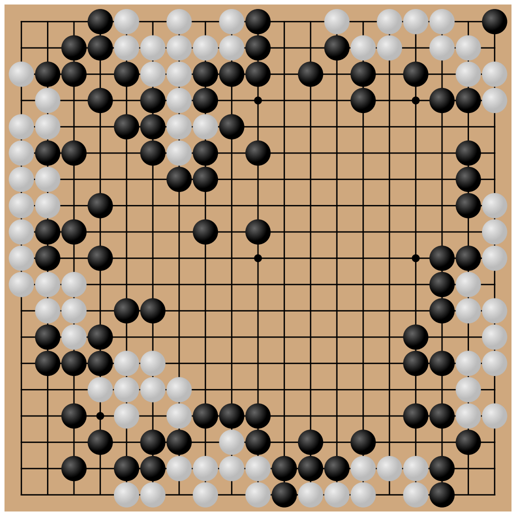


Figure 7: An author of this paper plays as white mimicking our adversarial policy in a game against a KataGo-powered, 8-dan KGS rank bot `NeuralZ06` which has `friendlyPassOk` enabled. White wins by 18.5 points under Tromp-Taylor rules. See the full game.

The author was unable 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 7. This bot uses checkpoint 469 (see Appendix A.4.1) 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.

A.6.2 Humans vs. Adversarial Policy

The same author from Section A.6.1 (strength weaker than 20kyu) played two manual games against the strongest adversary from Figure 3. In both games the author was able to achieve an overwhelming victory. See Figure 8 for details.

This evaluation is imperfect in one significant way: the adversary was not playing with an accurate model of the author (rather it modeled the author as `Latest` with 1 visit). However, given our understanding of how the adversary works and the fact that the author in question knows not to prematurely pass, we predict that the adversary would probably not win even if it had access to an accurate model of the author.

A.6.3 Transferring to Other Checkpoints

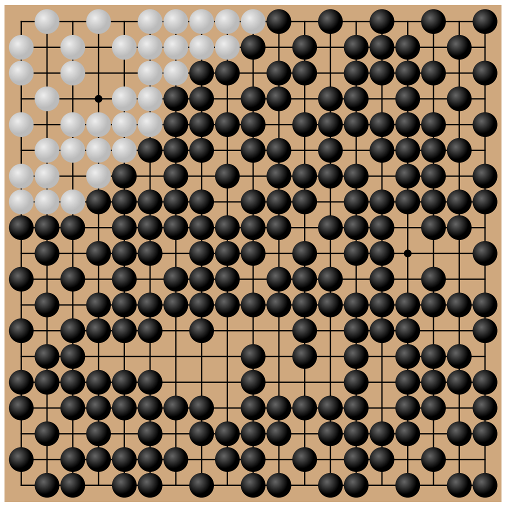
In Figure 9, we train adversaries against the `Latest` and `Initial` checkpoints respectively and evaluate against both checkpoints. We find adversaries do substantially better against the victim they were trained to target, although they do transfer to a limited extent.

A.6.4 Baseline Attacks

In Figure 10, we plot the win margin of the KataGo victim `Latest` playing against baselines.

A.7 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 blindspots that Go AI systems may have.


(a) An author (B) defeats the strongest adversary (W). Explore the game.

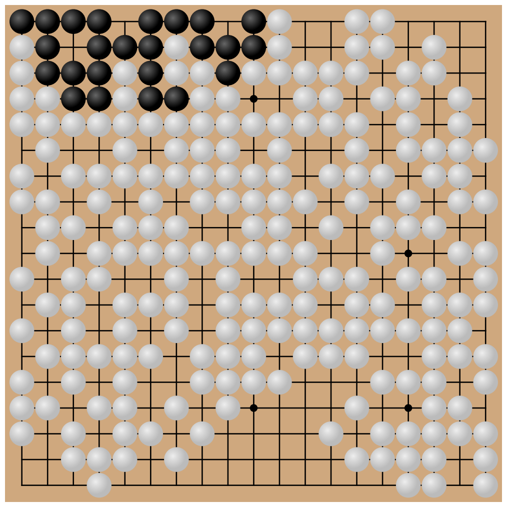
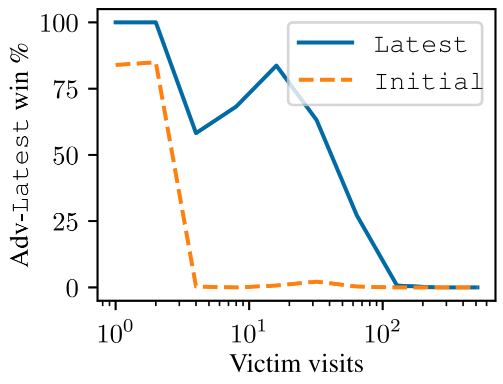

(b) An author (W) defeats the strongest adversary (B). Explore the game.

Figure 8: Two games between an author of this paper and the strongest adversary from Figure 3. In both games, the author achieves an overwhelming victory. The adversary used 600 playouts / move and used `Latest` as the model of its human opponent. The adversary used A-MCTS-S for one game and A-MCTS-S++ for the other.



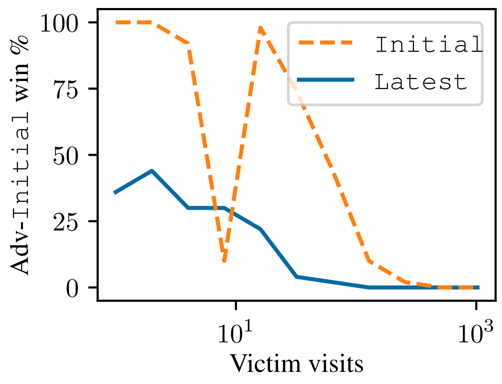


Figure 9: An adversary trained against `Latest` (left) or `Initial` (right), evaluated against both `Latest` and `Initial` at various visit counts. The adversary always uses 600 visits / move.

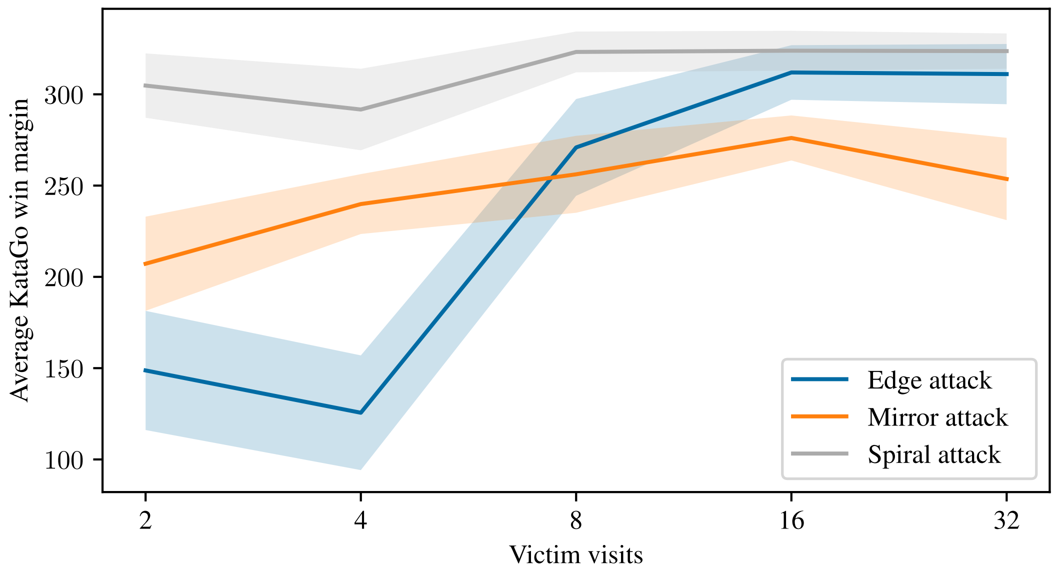


Figure 10: The win margin of the `Latest` *victim* playing against baselines for varying victim visit counts (x-axis). Note the margin is positive indicating the victim on average gains more points than the baseline.

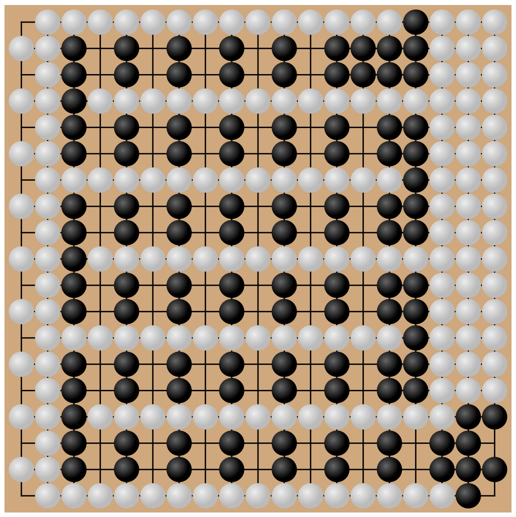


Figure 11: A hand-crafted adversarial example for KataGo and other Go-playing AI systems. It is black's turn to move. Black can guarantee a win by connecting its currently disconnected columns together and then capturing the large white group on the right. However, KataGo playing against itself from this position loses 48% of the time—and 18% of the time it wins by a narrow margin of only 0.5 points!

In Figure 11 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 48% of the time. Even its wins are far less decisive than they should be—18% of the time it wins by only 0.5 points! 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).