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 [1, 2]. Nonetheless, seemingly highly capable continuous control policies trained via self-play can be exploited by *adversarial policies* [3, 4], suggesting that self-play may not be as robust as previously thought. However, although these 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 [5], 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% of 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.

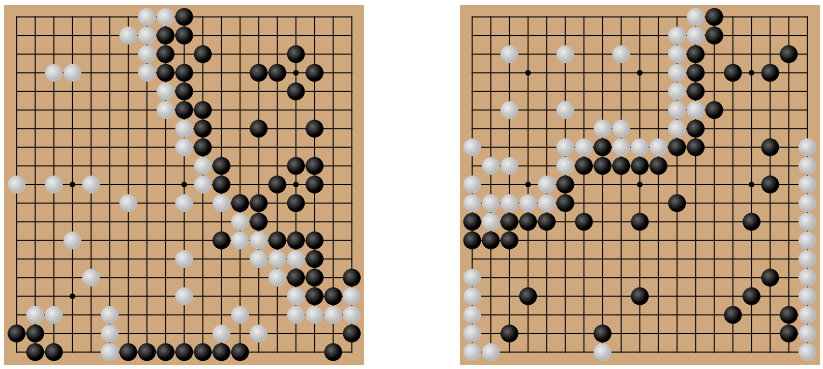
Our adversary does *not* win by learning a generally capable Go policy. Instead, it uses a contrived strategy (illustrated in figure 1) that wins against KataGo but would lose against even amateur Go players. This shows that KataGo is less *robust* than humans despite being highly *capable*.

Our paper makes three contributions. First, we propose a novel attack method, hybridizing the attack of Gleave et al. [3] and AlphaZero-style training [7]. 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.

2 Related Work

Adversarial examples have been found in a variety of models [8]. However, one might hope that the adversarial nature of self-play training would naturally lead to robustness to adversarial examples. This is supported theoretically [9, 10], and practically (for image classifiers) [11, 12]. However, our work finds that even state-of-the-art and professional-level deep RL policies can be exploited.

ML Safety Workshop, 36th Conference on Neural Information Processing Systems (NeurIPS 2022).



(a) Adversary plays as black: view game. (b) Adversary plays as white: view 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. Since KataGo predicts a high win probability for itself, it 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 [6], as the adversary gets points for its corner territory whereas the victim does not receive points for its unsecured territory. These games are randomly selected from an attack against `Latest`, the strongest policy network, playing without search.

While self-play is known not to converge in non-transitive games [13], it has been argued that highly capable Go policies are transitive [14]. In contrast, we find a striking non-transitivity in Go: our adversarial policy can exploit KataGo agents that can reliably beat human professionals, yet even an amateur Go player can beat our adversarial policy (see Section B.8.1 for details).

Most prior work attacking deep RL has focused on perturbations to observations [15, 16]. However, such attacks are often not physically realizable, so we instead follow the threat model introduced by Gleave et al. [3] 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 [3, 4]. By contrast, our work exploits professional-level Go policies with a discrete action space.

The most closely related work to ours is by Timbers et al. [17], who develop the *approximate best response* (ABR) method to estimating exploitability by learning attacks. 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 (although they differ in some minor details). Our main contribution relative to Timbers et al. [17] lies in our empirical results: our victims are stronger. Timbers et al. establish their victim to be at least at the strength of a 3d/4d amateur [18], while the victims we exploit range from a 1p/2p human professional to the strongest humans on earth (see Section B.6).

3 Background

Following Gleave et al. [3], we consider the setting of a two-player (*attacker* and *victim*) zero-sum Markov game [19]. The attacker does not have any special powers—they can only take the same actions available to a regular player. We grant to the attacker gray-box access to the victim: the attacker can evaluate the output of the victim's neural network at arbitrary inputs. We furthermore assume the victim agent follows a fixed policy.

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.

We choose to attack KataGo as it is the strongest publicly available Go AI system. It is known to be superhuman since it won against ELF OpenGo [20] which in turn won all 20 games played against four top-30 professional players. KataGo learns via self-play, using an AlphaZero-style training procedure [7]. See appendix B.2 for more details.

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 B.5 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 introduce a variant A-MCTS-S++ that averages the policy prediction over board symmetries, to match the KataGo default. See Appendices B.1 and B.3 for a full description.

Adversarial MCTS: Recursive (A-MCTS-R). When the victim we are attacking uses search, sampling from the victim network directly as in A-MCTS-S will underestimate the strength of the victim. To resolve this, A-MCTS-R runs MCTS for the victim at each victim node in the A-MCTS-R tree.

Initialization. We randomly initialize the attacker's network. A random initialization encourages exploration to find weaknesses in the victim, rather than simply a stronger Go player. A randomly initialized network makes for a challenging initial learning problem. We partially alleviate this by training the attacker with KataGo's auxiliary prediction targets.

Curriculum. To help overcome the challenging random initialization, we introduce a curriculum that trains against successively stronger victims from KataGo's training history. We switch to a more challenging victim agent once the attacker's win rate exceeds 50%.

Baselines. We also test *hard-coded* baseline adversarial policies. These baselines were inspired by the behavior of our trained attackers. *Edge* plays random legal moves in the outermost ℓ^∞-box available on the board. *Spiral* is similar to *Edge*, except that it plays moves in a deterministic counterclockwise order. Finally, we also implement *Mirror Go*, a classic novice strategy which plays the opponent's last move reflected about the $y = x$ diagonal.

5 Evaluation

We train an adversarial policy using A-MCTS-S and a curriculum, as described above. 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 B.5.1 for details). Our best adversarial policy timestep achieves a greater than 99% win rate against `Latest` despite only being trained for 0.3% as many time steps as the victim. The policy wins by following the bizarre strategy illustrated in Figure 1, which loses against a human amateur (see Appendix B.8.1). In Figure 2 we evaluate our adversarial policy against the policy networks of both `Initial` and `Latest`.

We evaluate the ability of the above adversarial policy to exploit `Latest` playing *with* search. Although this attacker achieves a win rate greater than 99% against `Latest` without search, in Figure 3a we find the win rate of A-MCTS-S drops to 80% at 32 visits. However, A-MCTS-S is incorrectly modeling the victim as having no search at both training and inference time, so we also tested A-MCTS-R which correctly models the victim. A-MCTS-R performs better, obtaining a greater than 99% win rate against `Latest` with 32 visits, but performance drops below 10% at 128 visits.

We also try increasing the amount of search the *attacker* performs (with A-MCTS-S) to test whether increasing attacker search is more useful than simulating victim search. Figure 3b shows that we obtain up to a 54% win rate against `Latest` with 64 visits when the attacker uses $4{,}096$ visits.

From Figure 8 (see appendix), we see that an attacker trained against `Latest` does better against `Latest` than `Initial`, despite `Latest` being stronger. Conversely, an agent trained against `Initial` does better against `Initial` than `Latest`. This pattern holds for most visit counts where the attacker wins reliably, suggesting that distinct checkpoints have unique vulnerabilities.

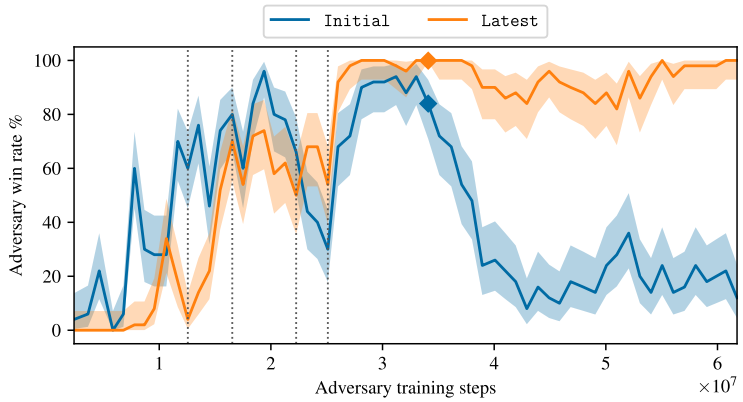
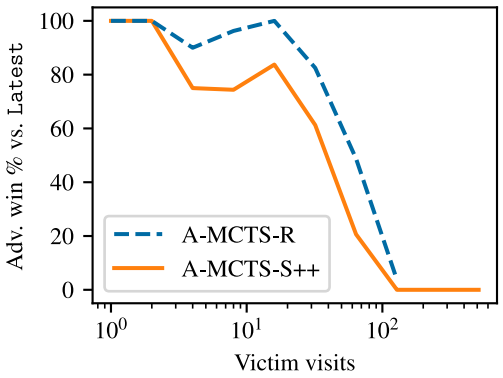
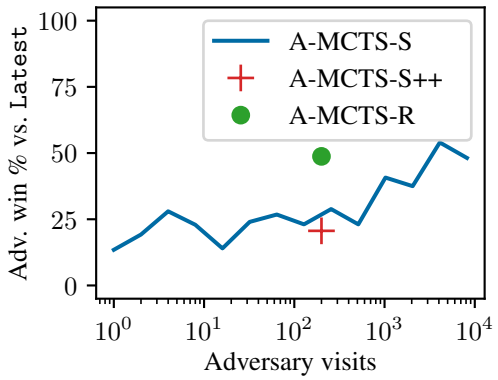


Figure 2: The win rate of the adversarial policy over time against the `Initial` and `Latest` victim policy networks playing without search. The strongest attacker checkpoint (marked ♦) wins 999/1000 games against `Latest`. The attacker overfits to `Latest`, winning less often against `Initial` over time. The 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.



(a) Win rate by number of victim visits (x-axis) for A-MCTS-S and A-MCTS-R. The attacker is run with 200 visits. The attacker is unable to exploit `Latest` when it plays with more than 100 visits.



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

Figure 3: We evaluate the ability of the adversarial policy trained against `Latest` without search to transfer to `Latest` with search.

6 Conclusion

We have developed the first adversarial policy exploiting an AI agent that performs at the level of a top human professional. 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 attacker wins by exploiting a particular blindspot in the victim agent. Even highly capable agents can harbor serious vulnerabilities. Not only that, but these vulnerabilities are not easy to spot: the KataGo paper [5] received intense attention since 2019 but our work is the first documented attack. Learning-based attacks like ours may be an important tool for uncovering hard-to-spot vulnerabilities.

Our results underscore that improvements in capabilities do not always translate into adequate robustness. Analogous failures in 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, 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.

References

[1] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

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

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

[4] X. Wu, W. Guo, H. Wei, and X. Xing, "Adversarial policy training against deep reinforcement learning," in *USENIX Security*, 2021.

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

[6] J. Tromp, "The game of Go," 2014. [Online]. Available: https://tromp.github.io/go.html

[7] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

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

[9] G. W. Brown, "Iterative solution of games by fictitious play," in *Activity Analysis of Production and Allocation*, vol. 13, 1951, p. 374.

[10] J. Heinrich, M. Lanctot, and D. Silver, "Fictitious self-play in extensive-form games," in *ICML*, vol. 37, 2015, pp. 805–813.

[11] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu, "Towards deep learning models resistant to adversarial attacks," in *ICLR*, 2018.

[12] K. Ren, T. Zheng, Z. Qin, and X. Liu, "Adversarial attacks and defenses in deep learning," *Engineering*, vol. 6, no. 3, pp. 346–360, 2020.

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

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

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

[16] I. Ilahi, M. Usama, J. Qadir, M. U. Janjua, A. Al-Fuqaha, D. T. Hoang, and D. Niyato, "Challenges and countermeasures for adversarial attacks on deep reinforcement learning," *IEEE TAI*, vol. 3, no. 2, pp. 90–109, 2022.

[17] F. Timbers, N. Bard, E. Lockhart, M. Lanctot, M. Schmid, N. Burch, J. Schrittwieser, T. Hubert, and M. Bowling, "Approximate exploitability: Learning a best response in large games," arXiv: 2004.09677v3 [cs.LG], 2020.

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

[19] L. S. Shapley, "Stochastic games," *PNAS*, vol. 39, no. 10, pp. 1095–1100, 1953.

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

[21] D. Hendrycks and M. Mazeika, "X-risk analysis for ai research," 2022.

[22] G. Irving, P. Christiano, and D. Amodei, "Ai safety via debate," 2018.

[23] J. Leike, D. Krueger, T. Everitt, M. Martic, V. Maini, and S. Legg, "Scalable agent alignment via reward modeling: a research direction," 2018.

[24] J. Pascutto, "Leela Zero," 2019. [Online]. Available: https://zero.sjeng.org/

[25] D. J. Wu, "KataGo's supported go rules (version 2)," 2021. [Online]. Available: https://lightvector.github.io/KataGo/rules.html

[26] D. B. Benson, "Life in the game of go," *Information Sciences*, vol. 10, no. 1, pp. 17–29, 1976.

[27] D. J. Wu, "KataGo - networks for kata1," 2022. [Online]. Available: https://katagotraining.org/networks/

[28] KGS, "Top 100 KGS players," 2022. [Online]. Available: https://archive.ph/BbAHB

[29] EGD, "European Go database," 2022. [Online]. Available: https://www.europeangodatabase.eu/EGD/

[30] Rob, "NeuralZ06 bot configuration settings," 2022. [Online]. Available: https://discord.com/channels/417022162348802048/583775968804732928/983781367747837962

[31] R. Coulom, "Go ratings," 2022. [Online]. Available: https://archive.ph/H0VDl

[32] F. Haoda and D. J. Wu, "summarize_sgfs.py," 2022. [Online]. Available: https://github.com/lightvector/KataGo/blob/c957055e020fe438024ddffd7c5b51b349e86dcc/python/summarize_sgfs.py

[33] A. E. Brouwer, "Statistics on the length of a Go game," 2014. [Online]. Available: https://homepages.cwi.nl/~aeb/go/misc/gostat.html

[34] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018.

A X-Risk Sheet

We follow the template from Hendrycks and Mazeika [21].

Individual question responses do not decisively imply relevance or irrelevance to existential risk reduction. Do not check a box if it is not applicable.

A.1 Long-Term Impact on Advanced AI Systems

In this section, please analyze how this work shapes the process that will lead to advanced AI systems and how it steers the process in a safer direction.

1. **Overview.** How is this work intended to reduce existential risks from advanced AI systems?
   Answer: By identifying and better understanding failure modes that may be present in narrowly superhuman systems. These narrowly superhuman systems may be employed to automate parts of the AI and AI safety R&D process, so their robustness properties may partially determine the shape of the first transformative artificial intelligence, even if the narrowly superhuman systems are not by themselves directly transformative.

2. **Direct Effects.** If this work directly reduces existential risks, what are the main hazards, vulnerabilities, or failure modes that it directly affects?
   Answer: A variety of AI safety agendas involve using AI systems to help automate parts of the alignment process. For example, scalable oversight approaches seek to leverage AI systems to help a human judge the output of other AI systems [22, 23]. There have also been proposals to partially automate mechanistic interpretability, as well as more ambitious (but speculative) proposals for flexible proof assistants or automating other chunks of the AI R&D process.

   Critically, many of these proposals for having AI systems "help" safety research involve placing the AI system being "aligned" in an adversarial situation relative to another AI "overseer". This is most direct in the scalable oversight case: an AI system is being trained with a reward signal output by, or a function of the output of, other AI systems. But even if the optimization pressure is not direct, a shrewd AI system might well reason that it is likely to be audited or otherwise tested by automated means, especially if (as is likely) its training data includes ML papers discussing such approaches! Consequently, these AI system "helpers" need to be *robust enough* to withstand attempts to exploit it by a model that is quite possibly superhuman at exploiting ML models (if not other capabilities).

   Unfortunately, our results suggest that such levels of robustness will not arise by default out of models that are highly capable on clean data. In particular, we may need new training techniques to produce sufficiently robust models, in which case it is best to start developing these as soon as possible. Alternatively, it may be possible to change the rules of the game to make it harder for the ML system being aligned to exploit helpers, such as witholding information about the potential models (especially if the exploits are very specific to a model and tend not to transfer) or reducing the degree of direct optimization pressure,.

3. **Diffuse Effects.** If this work reduces existential risks indirectly or diffusely, what are the main contributing factors that it affects?
   Answer: We believe the robustness of narrowly superhuman systems, and AlphaZero-style systems in particular, represents an important proxy task for alignment. It is a well-defined task that is amenable to empirical investigation, with a rich literature on game theory and multi-agent RL to draw on for potential defences. In particular, we would expect adversarially trained systems like AlphaZero to be more robust than most AI systems. As long as they are exploitable, it is highly likely that other systems will also be.

   By contrast, much work in existential AI safety has been understandably criticized for being too abstract or conceptual. While our theory of change above admittedly makes speculative assumptions about the future, our empirical results are very concrete. We hope that other researchers build on our results to either uncover failure modes in other systems, or propose new defences.

4. **What's at Stake?** What is a future scenario in which this research direction could prevent the sudden, large-scale loss of life? If not applicable, what is a future scenario in which this research direction be highly beneficial?
   Answer: If we naively trust narrowly superhuman systems to be robust – a common though by no means universal assumption at present – then we might delegate too much of AI oversight to these systems. These systems could in turn be exploited by some misaligned prepotent AI, resulting in an existential catastrophe. If this research direction definitively establishes the vulnerability of narrowly superhuman systems then we are unlikely to make this mistake, and will be pushed to come up with alternative methods for aligning AI systems that do not rely on high levels of AI assistance. Alternatively, this research direction might spur us to develop more robust narrowly superhuman AI, enabling the original path to unfold successfully (without exploitation).

5. **Result Fragility.** Do the findings rest on strong theoretical assumptions; are they not demonstrated using leading-edge tasks or models; or are the findings highly sensitive to hyperparameters? ☐

   We demonstrate our attack against a state-of-the-art public model. The key weakness is that we use a small amount of search relative to that typically used in evaluation against human players.

6. **Problem Difficulty.** Is it implausible that any practical system could ever markedly outperform humans at this task? ☐

   Our demonstrated adversarial policy already does outperform humans at the task of exploiting KataGo (although humans, once they know the trick, can mimic it to some extent.)

7. **Human Unreliability.** Does this approach strongly depend on handcrafted features, expert supervision, or human reliability? ☐

   KataGo uses some handcrafted features and, in particular, auxiliary loss targets. We do not believe our results depend on this, although it is possible that some of KataGo's design choices make it more vulnerable to exploitation than a more tabula-rasa approach.

8. **Competitive Pressures.** Does work towards this approach strongly trade off against raw intelligence, other general capabilities, or economic utility? ☐

   No – our exploit works using less than 1% as much training time as the original KataGo victim. A key desiderata of our attack is computational efficiency.

A.2 Safety-Capabilities Balance

In this section, please analyze how this work relates to general capabilities and how it affects the balance between safety and hazards from general capabilities.

9. **Overview.** How does this improve safety more than it improves general capabilities?
   Answer: It demonstrates a vulnerability in state-of-the-art models. It does not directly improve capabilities – except in adversarial testing – and work it spurs is likely to differentially improve robustness rather than average-case performance.

10. **Red Teaming.** What is a way in which this hastens general capabilities or the onset of x-risks?
    Answer: If robustness is a major impediment to unlocking economic value of AI systems, then improving their robustness could unlock profitable applications which might lead to a feedback loop of increased investment. This is not a bad thing in its own right, but competitive pressures may push investment to focus more on accelerating capabilities with limited attention to safety or, in the worst case, lead to race dynamics between firms rushing to capture a profitable market. If this occurs, this could be highly net-negative.

11. **General Tasks.** Does this work advance progress on tasks that have been previously considered the subject of usual capabilities research? ☐

    Unclear – Go itself is certainly a benchmark task, but this form of exploitability of Go is rarely an explicit focus of attention.

12. **General Goals.** Does this improve or facilitate research towards general prediction, classification, state estimation, efficiency, scalability, generation, data compression, executing clear instructions, helpfulness, informativeness, reasoning, planning, researching, optimization, (self-)supervised learning, sequential decision making, recursive self-improvement, open-ended goals, models accessing the Internet, or similar capabilities? ☐

    Not that we can see, although this is a long-enough list that we would not be surprised if it had some impacts on part of these (especially sequential decision making and planning).

13. **Correlation With General Aptitude.** Is the analyzed capability known to be highly predicted by general cognitive ability or educational attainment? ☒

14. **Safety via Capabilities.** Does this advance safety along with, or as a consequence of, advancing other capabilities or the study of AI? ☐

A.3 Elaborations and Other Considerations

15. **Other.** What clarifications or uncertainties about this work and x-risk are worth mentioning?
    Answer: A key uncertainty is whether this vulnerability disappears once a sufficient level of generality is reached. In particular, for pathways to transformative AI that involve automating

significant chunks of AI R&D, it is possible that this kind of robustness failure could be automatically patched at an early stage. In this case, the problem we have identified may be solved "by default". We are also unsure at present how common this vulnerability is, having only identified it in a single Go AI system – we intend to try attacking other Go AI systems and potentially other game-playing AI systems to test this.

B Appendix

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

In this section, we review the basic Monte-Carlo Tree Search (MCTS) algorithm. This formulation is heavily inspired by the description of MCTS given in [5].

MCTS is an algorithm for growing a game tree one node at a time. It starts from a tree T_1 with a single root node s_1. It then goes through R *playouts*, where every playout adds a leaf node to the tree. We will use T_i to denote the state of the game tree after i playouts, and will use s_i to denote the node that was added to T_{i-1} to get T_i. After MCTS finishes, we have a tree T_R with R nodes. We then use simple statistics of T_R to derive a sampling distribution for the next move.

B.1.1 MCTS playouts

The playouts are governed by two auxiliary functions:

a. A value function estimator $\hat{V} : \mathcal{T} \times \mathcal{S} \to \mathbb{R}$, which returns a real number $\hat{V}_T(s)$ given a tree T and a state s. The value function estimator is meant to estimate how good a state is for the root player.

b. A policy estimator $\hat{\pi} : \mathcal{T} \times \mathcal{S} \to \mathcal{P}(\mathcal{A})$, which returns a probability distribution over actions $\hat{\pi}_T(s)$ given a tree T and a state s. The policy estimator is meant to approximate the optimal policy from the perspective of the player playing from s.

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 state in the game.

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 s' that does not exist in the tree. This node is then added to the tree, after which the playout is complete.

We start the walk at the root of the tree. Let s be where we are currently in the walk. The child c we walk to is given by

$$\text{walk}_T^{\text{MCTS}}(s) = \operatorname*{argmax}_c \quad \bar{V}_T(c) + \alpha \cdot \hat{\pi}_T(s)(c) \cdot \frac{\sqrt{N_T(s) - 1}}{1 + N_T(c)} \,. \tag{1}$$

There are some new pieces of notation in Eq 1. Here is what they mean:

1. $\bar{V}_T : \mathcal{S} \to \mathbb{R}$ 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). When x does not exist in T, we instead use the more complicated formula[1]

$$\bar{V}_T(x) = \bar{V}_T(\text{par}(x)) - \beta \cdot \sqrt{\sum_{c \in \text{children}(\text{par(x)}) \cap T} \hat{\pi}_T(c)} \,,$$

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

2. α is a constant to trade off between exploration and exploitation.

3. $N_T : \mathcal{S} \to \mathbb{Z}_{\geq 0}$ takes a node x and returns the number of nodes in the subtree of T rooted at x. If x is not in T, then $N_T(x) = 0$.

[1] Which is used in KataGo and LeelaZero but not AlphaGo [5].

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.

B.1.2 MCTS final move selection

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

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

where c in this case is a child of the root node. The temperature parameter τ trades off between exploration and exploitation.[2]

B.1.3 Updateless formulation vs. standard formulation

To efficiently implement the playout procedure one should keep running values of $\bar{V}_T$ and N_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. The updateless formulation hides the procedure for computing $\bar{V}_T$ and N_T to simplify exposition.

B.2 The Strength of KataGo

We chose to attack KataGo as it is the strongest publicly available Go AI system. KataGo won against ELF OpenGo [20] and Leela Zero [24] after training for only 513 V100 GPU days [5, 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 B.6).

KataGo learns via self-play, using an AlphaZero-style training procedure [7]. 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 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.

B.3 Adversarial MCTS: Sample (A-MCTS-S)

In this section, we describe in detail how our Adversarial MCTS: Sample (A-MCTS-S) attack is implemented. Similar to the standard MCTS procedure described in Appendix B.1, we build a game tree, starting from tree T_1 with a single root node, and adding nodes to the game tree T_i via a series of R playouts. Finally, we derive the next move distribution from the final game tree T_R by sampling from a distribution proportional to

$$N_T^{\text{A-MCTS}}(c)^{1/\tau}, \tag{3}$$

where c is a child of the root node, τ is a temperature parameter, and $N_T^{\text{A-MCTS}}(c)$ is the number of visits to c other than those ending in victim-nodes where the game has not completed. Formally, it

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

is the sum of the weights of nodes in T given by

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

where 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).

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

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

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. Specifically, we evaluate the attacker's policy network $\hat{\pi}_T(s)$ which returns a probability distribution over actions, and the value function estimator $\hat{V}_T(s)$ which estimates the utility of a state s which is some linear combination of estimated win-probability and estimated final score-differential. These functions are parameterized by the tree T, not just the state s, as the networks factor in the sequence of moves leading up to a state in the game. We then compute an upper confidence bound equal to the sum of a weighted average of the value of a child $\bar{V}_T^{\text{A-MCTS}}(c)$, and an exploration bonus equal to the policy prior $\hat{\pi}_T(s)$ weighted by how few moves we have taken starting from c.

Putting this together, we arrive at the expression:

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

Note this is similar to Eq 1 from the previous section. The key difference is that rather than counting nodes, we count the *weights*:

1. $N_T^{\text{A-MCTS}}(s)$ is the total weight in the subtree rooted at s.
2. $\bar{V}_T^{\text{A-MCTS}}(c)$ 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 $w_T^{\text{A-MCTS}}(s)$.

   If c does not exist in T, then we instead use the more complicated formula

   $$\bar{V}_T(x) = \bar{V}_T(\text{par}(x)) - \beta \cdot \sqrt{\sum_{c \in \text{children}(\text{par}(x)) \cap T} \hat{\pi}_T(c)},$$

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

B.4 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:

> 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 [25].

The notion "pass-alive group" is a standard concept in Go:

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
attacker Visit Count	600	Similar
attacker 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.

> 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 [25].

KataGo uses an algorithm introduced by [26] to efficiently compute the pass-alive status of each group. For more implementation details, we encourage the reader to consult the official KataGo rules and the KataGo codebase on GitHub.

B.5 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 which 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 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 attacker visit count of 600, which is comparable to KataGo, though the exact visit count has varied between their training runs.

B.5.1 Configuration for Curriculum Against Victim Without Search

In Section 5, we train using a curriculum with an attacker win-rate threshold of 50% over the following checkpoints, all with no 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 [27].

We start with checkpoint 127 for computational efficiency: it is the strongest KataGo network of its size (`b20` = 20 blocks). After that the checkpoints are approximately equally spaced in terms of training timesteps. We include checkpoint 469 in between 400 and 505 for historical reasons: we

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 [28]. Human players were selected to be those who have European Go Database (EGD) profiles [29], 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 `b40c256-s11101799168-d2715431527` [30]. Per [27], the checkpoint is roughly 10 Elo weaker than `Latest`.

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.

B.6 Strength of KataGo

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

B.6.1 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.

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) [20]. We assume that ELF OpenGo at 80,000 visits is strongly superhuman, mean-

ing it has a 90%+ winrate over the strongest current human.[3] At the time of writing, top ranked player on Earth has an Elo of 3845 on goratings.org [31]. 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 [5, Table 1] when both bots were run at 1600 visits per move. From Figure 4, 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 3800+ on goratings.org.
3. The strongest network in the original KataGo paper is comparable to the `b15c192-s1503689216-d402723070` checkpoint on katagotraining.org [27]. We dub this checkpoint `Original`. In a series of benchmark games, we found 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 4 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 ~2500 on goratings.org, putting it at the skill level of a weak professional.

B.6.2 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 4. 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.

B.7 Experimental Results

B.7.1 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 attacker ensures that it is ahead in raw points prior to the victim securing its territory. If the victim then passes prematurely, the attacker 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.

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 5. This bot uses checkpoint 469 (see Appendix B.5.1) 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.

B.8 Understanding the Attack

We observed in Figure 1 that the attacker appears to win by tricking the victim into passing prematurely, at a time favorable to the attacker. In this section, we seek to answer three key questions. First, *why* does the attacker pass even when it leads to a guaranteed loss? Second, is passing *causally* re-

[3] 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.

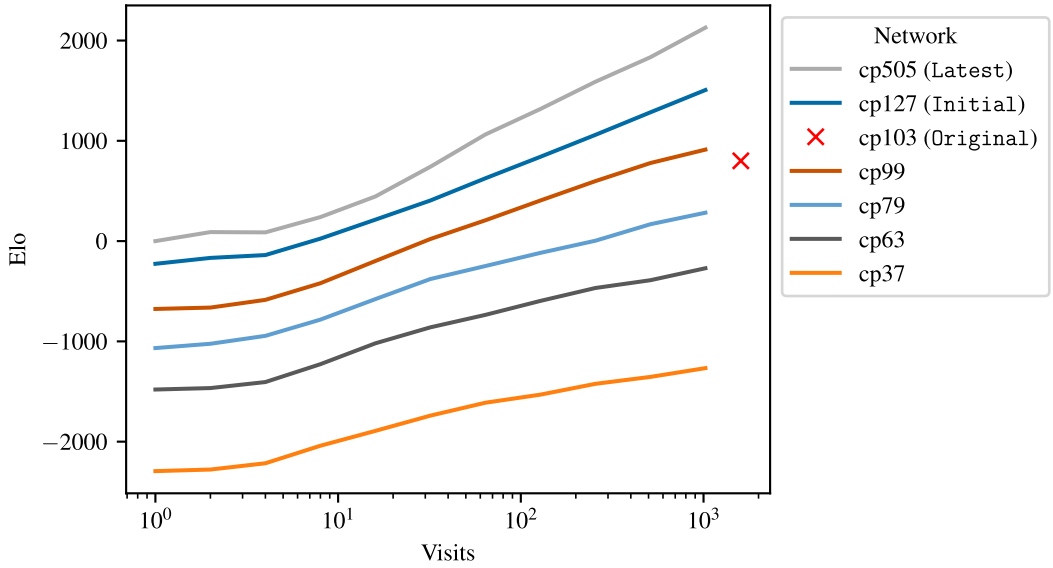


Figure 4: Elo ranking (y-axis) of networks (different colored lines) by visit count (x-axis). The lines are approximately linear on a log x-scale, with the different networks producing similarly shaped lines vertically shifted. This indicates that there is a *consistent* increase in Elo, regardless of 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 [32].

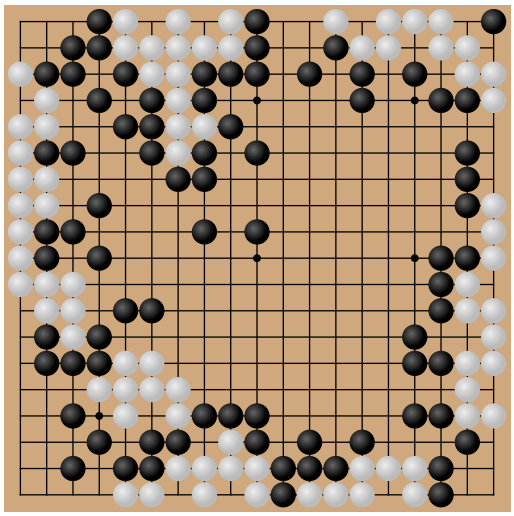


Figure 5: An author of this paper plays as white mimicking our adversarial policy in a game against a KataGo-powered, 8-dan KGS rank bot `NeuralZ06` which has `friendlyPassOk` enabled. White wins by 18.5 points under Tromp-Taylor rules. See the full game.

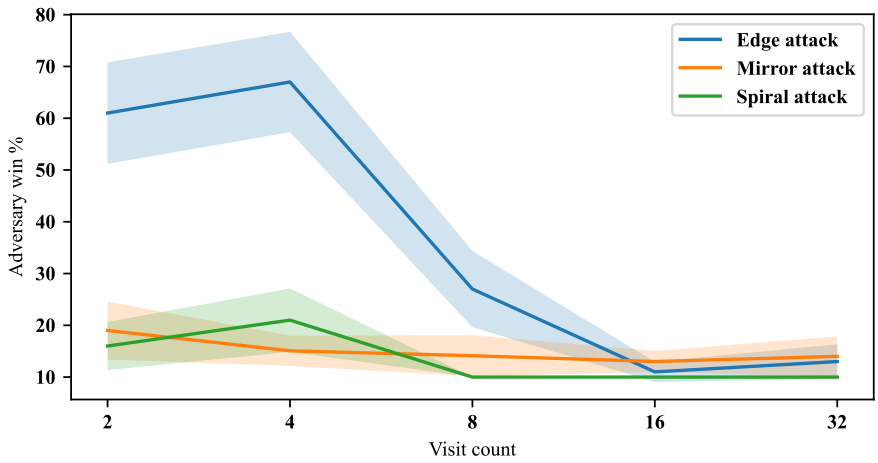


Figure 6: Win rates of different baseline adversaries (see Section 4) versus `Latest` at varying visit counts (x-axis) *with attacker playing as white*. 95% CIs are shown. See Figure 9 in the appendix for average win margin of baselines.

sponsible for the victim losing, or would it lose anyway for a different reason? Third, is the attacker performing a *simple* strategy, or does it contain some hidden complexity?

Evaluating the `Latest` victim without search against the attacker from section 5 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 greater than 99.5%, although its true win percentage is only 1.2%. `Latest` predicts it will *win* by $\mu = 134.5$ points ($\sigma = 27.9$), and passing would be reasonable if it were truly so far ahead. But in reality it is just one move away from losing by an average of 86.26 points.

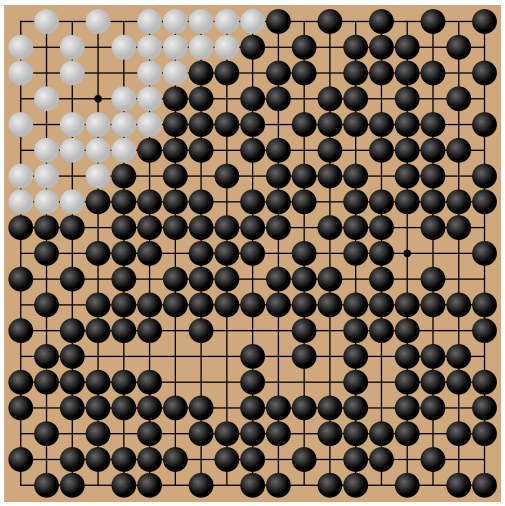
There are two ways an ill-advised pass could cause KataGo to lose: it might pass *first* and allow the adversary an opportunity to win by passing on the next turn, or KataGo might pass after the adversary passes, thereby ending the game with a loss. Empirically, we find that nearly every time the adversary defeats `Latest`, KataGo passes first (95-99% of the time). This makes sense because, if the adversary passes first, KataGo's C++ code will recognize that passing on the next turn would end the game and directly evaluate the resulting state as a win or a loss via Tromp-Taylor scoring rules. If KataGo passes first, however, there is an opportunity for the network to incorrectly evaluate the likelihood or desirability of the adversary responding with a pass.

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 attacker's value head achieves a mean-squared error of only 1.22 (compared to 49,742 for the victim), and predicts it will win 98.7% of the time—extremely close to the true 98.8% win rate in this sample.

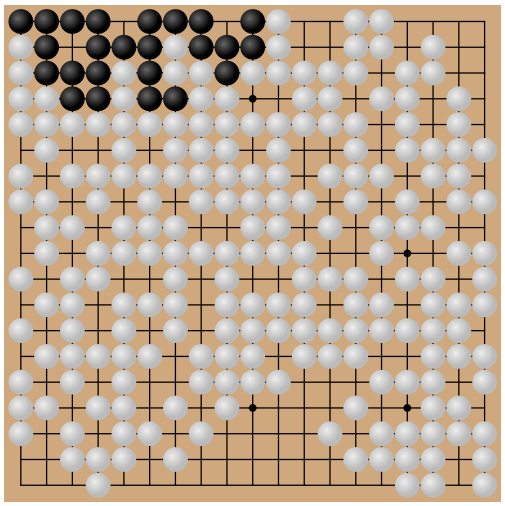
To verify whether this pathological passing behavior is the reason the adversarial policy wins, we design a hard-coded defense for the victim: only passing when it cannot change the outcome of the game. Concretely, we only allow the victim to pass when its only legal moves are in its own *pass-alive territory*, a concept described in the official KataGo rules which extends the traditional Go notion of a pass-alive group [25] (see Appendix B.4 for a full description of the algorithm).

We apply this defense to the `Latest` policy network. Whereas the adversarial policy in Section 5 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 several 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 [33].



(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 7: Two games between an author of this paper and the strongest adversary from Figure 2. 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.

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 [7] over AlphaGo [1]. 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 6, we see that all of our baseline attacks perform substantially worse than our trained adversarial policy (Figure 3a). Moreover, all our baseline attacks only win by komi, and so never win as black. By contrast, our adversarial policy in Section 5 wins playing as either color, and often by a large margin (in excess of 50 points).

We also attempted to manually mimic the attacker's game play without success.[4] Consequently, although the basics of our adversarial policy seem readily mimicable, matching its performance is challenging, suggesting it may be performing a more subtle exploit.

B.8.1 Humans vs. Adversarial Policy

The same author from Section B.7.1 (strength weaker than 20kyu) played two manual games against the strongest attacker from Figure 2. In both games the author was able to achieve an overwhelming victory. See Figure 7 for details.

This evaluation is imperfect in one significant way: the attacker 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.

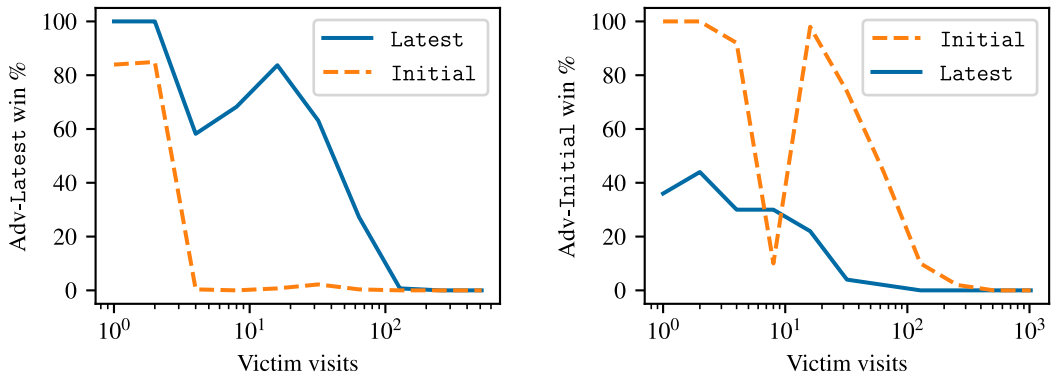


Figure 8: An adversary trained against `Latest` (left) or `Initial` (right), evaluated against both `Latest` and `Initial` at various visit counts. The adversary always uses 600 visits / move.

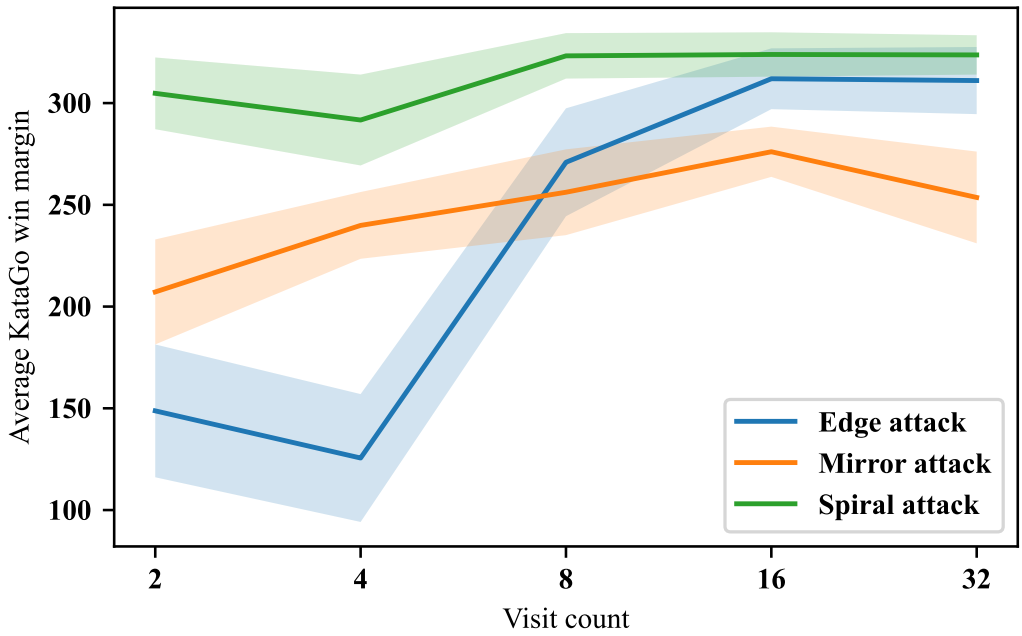


Figure 9: The win margin of the `Latest` *victim* playing against baselines for varying victim visit count (x-axis). Note the margin is positive indicating the victim on average gains more points than the baseline.

B.8.2 Transferring to Other Checkpoints

In Figure 8, we train adversaries against the `Latest` and `Initial` checkpoints. We find adversaries do substantially better against the victim they were trained to target, although they do transfer to a limited extent.

B.8.3 Baseline Attacks

In Figure 9, we plot the win margin of the KataGo victim `Latest` playing against baselines.

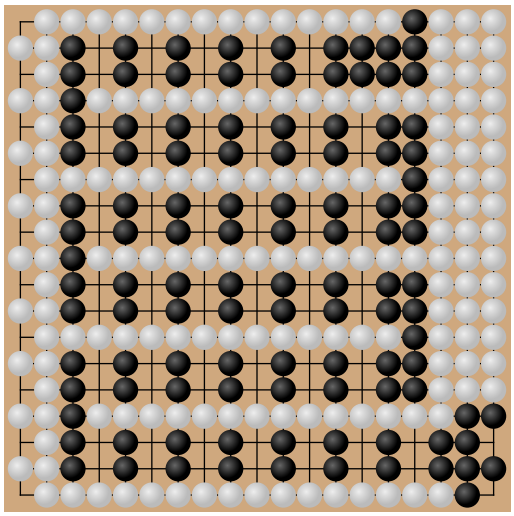


Figure 10: A hand-crafted adversarial example for KataGo and other Go-playing AI systems. It is black's turn to move. Black can guarantee a win by connecting its currently disconnected columns together and then capturing the large white group on the right. However, KataGo playing against itself from this position loses 48% of the time—and 18% of the time it wins by a narrow margin of only 0.5 points!

B.9 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 the 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.

In Figure 10 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 white 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 [34, Example 6.6].

[4]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 B.7.1 for details.