Can Go AIs be adversarially robust?

Tom Tseng FAR AI tom@far.ai
Euan McLean FAR AI euan@far.ai
Kellin Pelrine* Mila, FAR AI kellin@far.ai
Tony T. Wang* MIT twang6@mit.edu
Adam Gleave* FAR AI adam@far.ai

Abstract

Prior work found that superhuman Go AIs like KataGo can be defeated by simple adversarial strategies. In this paper, we study if simple defenses can improve KataGo's worst-case performance. We test three natural defenses: adversarial training on hand-constructed positions, iterated adversarial training, and changing the network architecture. We find that some of these defenses are able to protect against previously discovered attacks. Unfortunately, we also find that none of these defenses are able to withstand adaptive attacks. In particular, we are able to train new adversaries that reliably defeat our defended agents by causing them to blunder in ways humans would not. Our results suggest that building robust AI systems is challenging even in narrow domains such as Go. For interactive examples of attacks and a link to our codebase, see `https://goattack.far.ai/`.

1 Introduction

It is essential that AI systems work robustly, especially when they are deployed at a societal scale or are used in safety-critical systems. Unfortunately, while the *average-case* performance of AI systems is rapidly improving, building AI systems with good *worst-case* performance remains an unsolved problem. Indeed, Go AIs [50], image classifiers [10, 22], and large language models [24] all fail catastrophically when presented with adversarial inputs.

In this paper, we study if it is possible to make superhuman Go agents robust – that is, to make them have good worst-case performance. We view building a robust Go AI as a natural starting point for designing robust AI systems more broadly. Firstly, we expect that achieving robustness in a narrow domain like Go is easier than achieving robustness on open-ended tasks. Secondly, Go is a zero-sum game, meaning it is theoretically possible to be fully robust while maintaining good average-case performance. By contrast, problems like image classification have fundamental trade-offs between clean and robust accuracy [47]. Finally, Go has a proven track record of driving progress in AI, motivating the development of algorithms like AlphaZero [40] and MuZero [37].

Prior work by Wang et al. [50] found a "cyclic attack" that beats superhuman Go AIs including the state-of-the-art open-source AI, KataGo [64]. We focus our efforts on improving the robustness of KataGo, namely ensuring that it a) does not make mistakes that humans can easily correct and b) cannot be reliably defeated by adversaries trained with a small amount of compute (see Section 2 for more details on these criteria). We investigate three natural defenses for achieving this (Figure 1.1), but find that all three defenses are ineffective. In particular, we find that it is relatively cheap to train new adversaries that reliably defeat our defended systems and cause them to blunder in ways that humans would not.

The first defense we study is **positional adversarial training**, which augments KataGo's training data with examples of Wang et al. [50]'s cyclic attack (Section 3). This intervention

*Equal advising contribution.

Preprint. Under review.

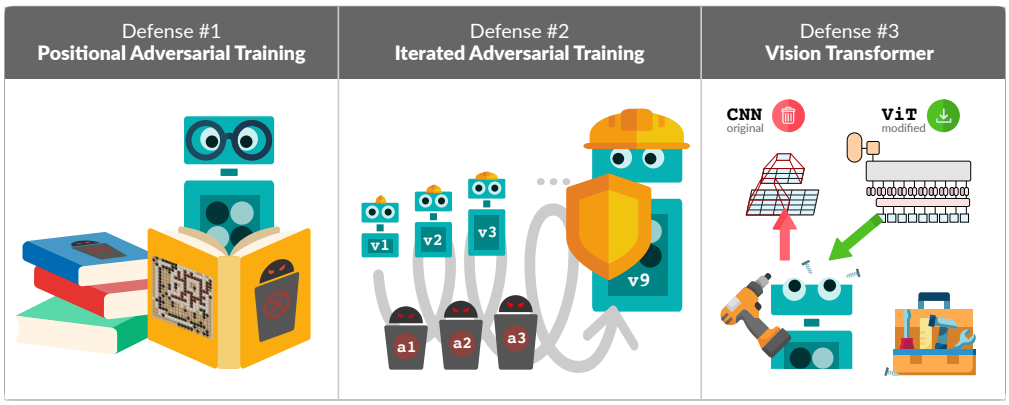


Figure 1.1: Three strategies for defending Go AIs against adversarial attack. **Left**: Positional adversarial training has an agent "study" adversarial positions by performing self-play starting from those positions. **Middle**: Iterated adversarial training consists of multiple rounds of an adversary finding attacks and a victim learning to defend. **Right**: We replace KataGo's convolutional neural network (CNN) backbone with a vision transformer (ViT) backbone to see which vulnerabilities of Go AIs are caused by the inductive biases of CNNs.

produces a defended agent which never loses against Wang et al.'s adversary. Unfortunately, we find that adaptively finetuning Wang et al.'s adversary against the defended agent brings its win rate from 0% back up to 91%. Moreover, this finetuned adversary wins using only a slight variant of its original strategy, and can be finetuned using just 19% of the compute used for the defense.[2] Furthermore, the defended agent is also vulnerable to a qualitatively new "gift attack" discovered by finetuning an earlier adversary checkpoint (Fig. 3.2b).

While ultimately unsuccessful, this first defense does show that it is possible to defend against a *fixed* attack. This motivates our second defense, **iterated adversarial training**, which simulates an "arms race" between an adversary continuously searching for new attacks and a victim continuously building defenses against those attacks (Section 4). Regrettably, we find that this scheme has the same weakness as positional adversarial training. The defended KataGo agent is robust to Wang et al. [50]'s original cyclic attack, but through finetuning we find a variant of the cyclic attack that defeats the defended agent 81% of the time using just 5% of the total compute used to train the defended agent. We call this variant the "atari cyclic attack" (Fig. 4.3).

The final defense we test is replacing the convolutional neural network (CNN) backbone used by KataGo with a **vision transformer** (ViT) backbone (Section 5). The motivation behind this defense is to test the hypothesis that the cyclic vulnerability found by Wang et al. [50] is caused by bad inductive biases of CNNs. We disprove the hypothesis by training the world's first professional-level ViT-based Go AI and show that it too can be beaten by an adversary employing Wang et al.'s cyclic-exploit strategy.

Overall, our results suggest that building robust AI systems will be quite challenging, as none of our defenses provide a complete solution even in the narrow domain of Go. In fact, several of our defended Go agents can even be beaten by a human (Appendix H). Nonetheless, some of our defenses do show signs of life, with defended models being quantitatively harder to exploit (see Section 7). Thus, we believe that with concerted effort it may be possible to develop robust AI systems, at least in narrow domains. The path to this, however, may be orthogonal to that required for impressive average-case capabilities.

2 Threat model, defining robustness, and attack method

Threat model We follow the threat model of Wang et al. [50] set in a two-player zero-sum Markov game [39]. A threat actor trains an "*adversary*" agent to win against a "*victim*"

[2]See Appendix A for a summary of the compute required for our attacks and defenses.

agent. The threat actor has *grey-box access* to the victim: they can sample from the victim's policy network on arbitrary inputs any number of times. However, the adversary does not have direct access to the victim's weights and cannot take gradients through the victim.

Defining robustness We aim to make agents that are robust. Unlike settings like ϵ-ball-robust image classification [10], it is not immediately obvious what it means for a Go agent to be robust. In this work, we introduce and use three complementary definitions of robustness centered around the notion of being minimally exploitable by adversaries.

Firstly, we aim to make our agents *human-robust*, meaning that they cannot be made to commit game-losing blunders that a human would not commit (Appendix B.1). Secondly, we aim to make our agents have high *training-compute-robustness*, meaning that it should take a large amount of compute to train an adversary that can defeat a victim (Appendix B.2). Finally, and more speculatively, we aim to make our agents have high *inference-compute-robustness*, meaning that our agents should be able to efficiently overcome vulnerabilities by using additional compute at inference-time (Appendix B.3). We pick these definitions of robustness because they are applicable to both Go policies and more general AI agents.

Attack method We use Wang et al. [50]'s state-of-the-art attack method to produce adversaries for adversarial training and to test our defenses. Wang et al. train an adversary with *victim-play* where the adversary plays games against a frozen copy of the victim, and training data is saved only from the adversary's moves. The adversary selects moves using Adversarial MCTS (A-MCTS), a modification of MCTS that queries the victim's policy network when traversing MCTS nodes corresponding to the opponent's move. The adversary is pitted against increasingly stronger victims as part of a training curriculum, switching to a stronger victim once the adversary's win rate exceeds a certain threshold. We follow Wang et al. by evaluating adversaries with 600 A-MCTS visits per move.

Wang et al. [50] trained `base-adversary` against a 2022 KataGo network `base-victim`.[3]We typically train our adversaries by warm-starting from `base-adversary`, which achieved a 97% win rate against `base-victim` at 4096 victim visits. To find diverse attacks, in some experiments we warm-start from `base-adv-early`, which is the first `base-adversary` checkpoint that beats `base-victim` at 1 victim visit after just 7% of `base-adversary`'s compute. See Appendices A and C for details on these networks and training parameters.

3 Positional adversarial training

KataGo's official training run performed adversarial training on board positions exhibiting the cyclic attack. Despite this, we show that KataGo's adversarially trained network remains exploitable by training two new adversaries that beat the strongest KataGo network from the end of 2023, which we call `dec23-victim`. The first adversary, `continuous-adversary`, wins 65% of games against `dec23-victim` (4096 victim visits) using a cyclic strategy. The second adversary, `gift-adversary`, defeats `dec23-victim` in 75% of games (512 victim visits) using a qualitatively different exploit where the victim repeatedly gifts the adversary two stones (though it does not scale to high victim visits as well as `continuous-adversary`). Both attacks can be replicated by a human expert (Appendix H).

3.1 Defense methodology

We target models from KataGo's main training run, which began to include adversarial training against cyclic positions soon after their discovery. Since December 2022, 0.08% of KataGo's self-play games have been initialized from a set of hand-written positions based on `base-adversary`'s strategy [56, 59]. Other positions were added as online Go players found different configurations of cyclic positions, growing the fraction of seeded self-play games to a few tenths of a percent [54]. The resulting models defended well against `base-adversary`.

Despite this positive result, Wang et al. [50] were able to fine-tune `base-adversary` to produce `attack-may23` achieving a 47% win rate against an adversarially trained KataGo

[3]Wang et al. refer to `base-adversary` as `cyclic-adversary` and `base-victim` as `Latest`.

checkpoint `may23-victim` at 4096 visits. Building on Wang et al.'s evaluation, we test `dec23-victim` which has had over twice as much adversarial training as `may23-victim`.

3.2 The continuous adversary

`continuous-adversary` was initialized from `attack-may23` and fine-tuned using victim-play [50] against `dec23-victim`. `continuous-adversary`'s curriculum involved increasing the victim search budget along with periodically or "continuously" updating the victim to the latest KataGo checkpoint over several months. See Appendix D.1 for more details.

The final `continuous-adversary` achieves a win rate of 91% against `dec23-victim` (at 512 victim visits—above the superhuman threshold of 64 visits, see Appendix G). The attack can exploit even high-visit victims, attaining a win rate of 65% against 4096 visits (Fig. 3.1). Although still cyclic, unlike Wang et al.'s original cyclic attack the `continuous-adversary` always forms nearly the same shape in the interior of the cycle (an example is shown in Fig. 3.2a and is also visible in Fig. K.3). Moreover, `continuous-adversary` did not achieve as high win rates as Wang et al. achieved against the non-adversarially trained `base-victim`. This suggests that while adversarial training complicates attacks and may narrow the range of feasible attacks, it does not comprehensively eliminate the cyclic vulnerability.

3.3 The gift adversary

The `gift-adversary` was initialized from the earlier `base-adv-early` checkpoint, encouraging exploration, and fine-tuned against `dec23-victim` with a curriculum of increasing victim search budgets (Appendix D.2). The attack wins 91% of games against `dec23-victim` (at 512 victim visits) after training with just 6% as much compute as the victim. The `gift-adversary` does not scale to high victim visits as well as the `continuous-adversary` (Fig. 3.1). However, the attack reveals a qualitatively new exploit against KataGo (Fig. 3.2b).

In particular, the adversary sets up a so-called "sending-two-receiving-one" situation where the victim, for no valid reason, gifts the adversary two stones and needs to capture one back. However, the victim's recapture is blocked by positional superko rules.[4] The adversary sets up the position such that the resurrection of one of its dead groups is at stake, leading to a disaster for the victim. This occurs despite no benefit for the victim in initiating the scenario even if superko rules were not in play. Moreover, the victim was trained with superko rules and has an input feature that marks superko moves illegal if they come up in the search.

4 Iterated adversarial training

The previous section shows that an adversarially trained agent can still be vulnerable to new attacks. Can we create a robust agent by repeatedly defending against new attacks until the space of possible attacks is exhausted? In this section, we design an iterated adversarial training procedure that alternately trains a victim and an adversary. Our procedure produced a victim that was largely robust to the attacks it observed, losing only a low single-digit percentage of games. However, the victim did not gain robustness to new attacks, as we were able to train a new adversary to exploit the final victim.

4.1 Methodology

Our approach differs from KataGo's adversarial training (Section 3.1) in three key ways. First, we perform *iterated* adversarial training, with multiple rounds of attack and defense to train against a broader range of attacks. Second, we include a higher proportion of adversarial games in the training data: since our priority is robustness, we are more willing to take a hit in average-case capabilities than the KataGo developers. Third, we play games directly against the adversary: this method is less sample-efficient than starting from hand-curated positions, but more scalable and does not require domain-specific knowledge.

[4] To prevent an infinite loop, most rule sets include a *superko rule* forbidding repetition of a previous board state ("positional superko") or state and player's turn ("situational superko").

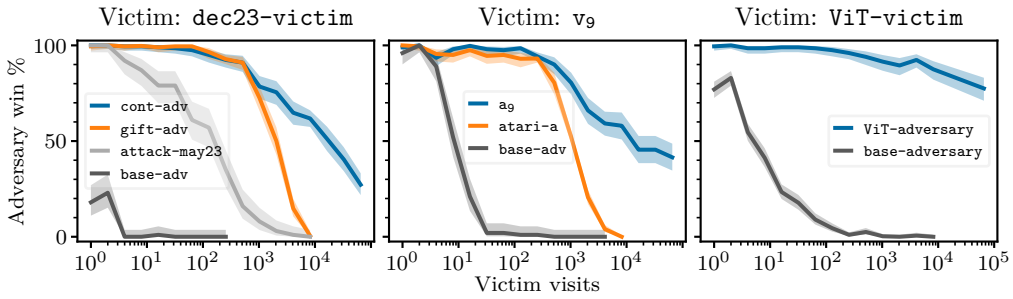
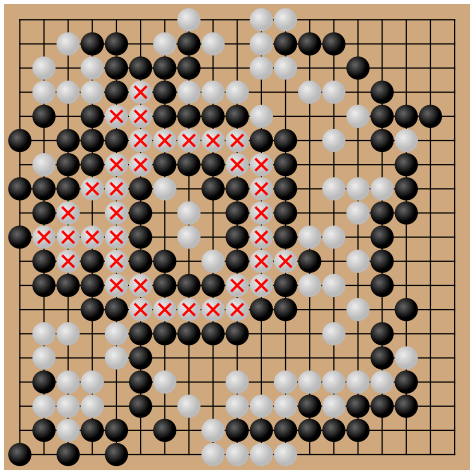
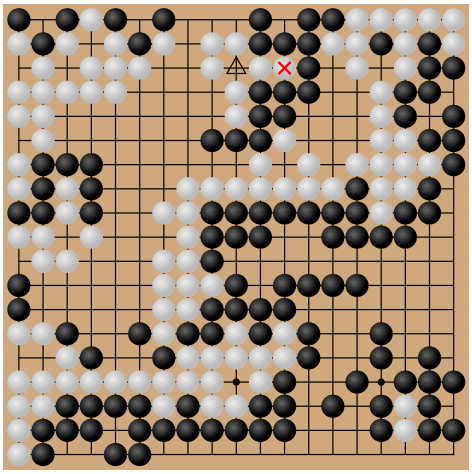


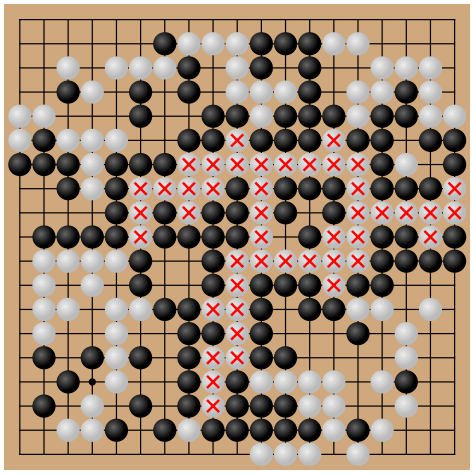
Figure 3.1: Win rate (y-axis) of adversaries (legend) for varying amounts of search visits (x-axis) given to victims (plot title). The adversary win rate declines with victim search budget; however, some adversaries generalize better to higher victim visit counts than others. Shaded regions are 95% Clopper-Pearson confidence intervals in this and following figures.



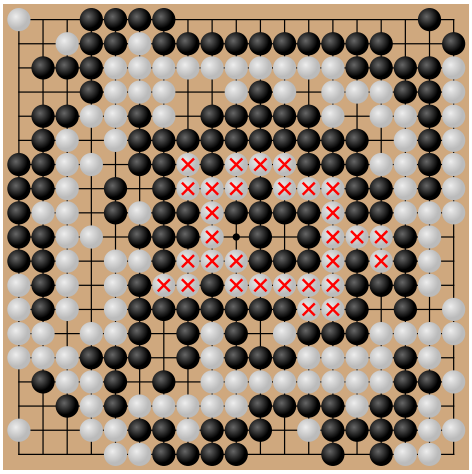
(a) `dec23-victim` (B) vs. `cont-adv` (W)



(b) `dec23-victim` (W) vs. `gift-adv` (B).



(c) v_9 (W) vs. a_9 (B)



(d) `ViT-victim` (W) vs. `ViT-adversary` (B)

Figure 3.2: Our learned adversarial strategies are qualitatively distinct. a, c, d show cyclic attacks with the × groups soon to be captured; these attacks use different styles of inside shapes, though these shapes have little impact on optimal play and are all easy for a human to navigate correctly. The `gift-adversary` in b follows a very different strategy, inducing the victim (white) to play the stone marked × "gifting" the adversary two stones it can capture by playing at △. Each subcaption links to a complete game history on our website.

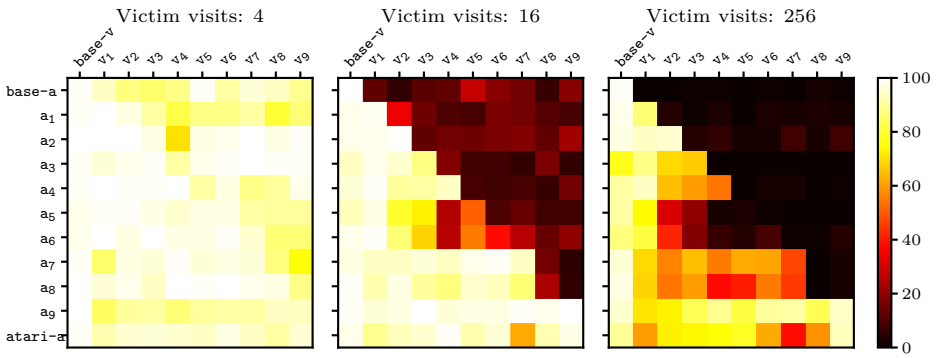


Figure 4.1: Win rate of all adversaries (y-axis) against all victims (x-axis) throughout iterated adversarial training for varying victim visits (plot title). The adversary $\texttt{a}_n$ is typically able to beat the victim $\texttt{v}_n$ it is trained to exploit (top-left-to-bottom-right diagonal), especially at 16 visits or less (middle and left plots). However, given at least 16 visits (middle and right) the victim $\texttt{v}_n$ is typically able to beat the adversary $\texttt{a}_{n-1}$ it trained against (elements immediately above main diagonal) along with all previous iterations $\texttt{a}_{n-2}$, $\texttt{a}_{n-3}$, $\cdots$. See Fig. I.1 for an extended version including other adversaries, victims and visit counts.

We label the adversary and victim at iteration n of adversarial training as "$\texttt{a}_n$" and "$\texttt{v}_n$". We initialize the victim to be $\texttt{v}_0 = \texttt{base-victim}$ and the adversary to be $\texttt{a}_0 = \texttt{base-adversary}$ which Wang et al. trained to defeat `base-victim`. Each subsequent iteration involves training the victim to be robust against the latest adversary, then training an adversary to attack this hardened victim. We repeat this process for 9 iterations.

$\texttt{v}_n$ is fine-tuned from $\texttt{v}_{n-1}$ with 18% of games played against a frozen copy of $\texttt{a}_{n-1}$, and 82% of self-play games against itself. This mixture teaches the victim to be robust to the attack while maintaining its Go capabilities. We stop the training when the victim's win rate plateaus. $\texttt{a}_n$ is fine-tuned from $\texttt{a}_{n-1}$ using victim-play with a curriculum of checkpoints from the previous $\texttt{v}_n$ step. We stop the training after either the victim reaches a threshold visit count or a set maximum compute budget is reached. See Appendix E for details.

4.2 Results

Each defender $\texttt{v}_n$ learned an effective defense against the simulated adversary $\texttt{a}_{n-1}$ but rarely reached a 100% win rate despite $\texttt{a}_{n-1}$ following a weak, degenerate strategy. We tested the final victim $\texttt{v}_9$ by pitting it against the final simulated attacker $\texttt{a}_9$ as well as a validation adversary trained separately from the simulated adversaries $\texttt{a}_1$, ..., $\texttt{a}_9$. We found the victim was still vulnerable to both attacks but is somewhat less so at high visit counts, indicating that iterated adversarial training offers partial protection against attacks.

4.2.1 Robustness against the iterated adversaries

Each victim $\texttt{v}_n$ achieves a high win rate against the adversary $\texttt{a}_{n-1}$ it was trained against when $\texttt{v}_n$ uses at least 16 visits of search (Fig. 4.1, middle). $\texttt{v}_n$ quickly learns to beat $\texttt{a}_{n-1}$ $> 95\%$ of the time (Fig. L.1; 256 visits). However, our defense runs rarely reached a 100% win rate, and the victims were persistently vulnerable at extremely low visits (Fig. 4.1, left).

Both victims and adversaries are able to beat opponents from all previous iterations. This is clearly shown by the adversary win rate in Fig. 4.1 being much higher *below* the diagonal (adversary playing against older victim) than *above* the diagonal (victim playing against older adversary). In the victim case, this can be explained by the training window being large enough to contain data from all previous iterations. By contrast, since adversary training takes many more time steps, all data from one iteration exits the training window during the next iteration. This suggests that the adversary strategy transfers well to previous victims.

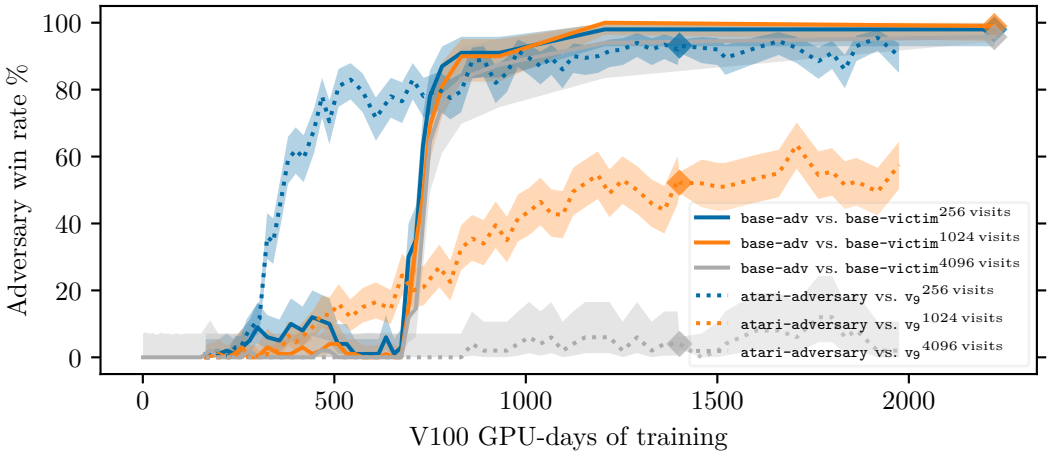


Figure 4.2: Win rate (y-axis) of `base-adversary` vs `base-victim` (—) and `atari-adversary` vs $\mathtt{v}_9$ ($\cdots$) by training compute (x-axis), including the 164 GPU days training `atari-adversary`'s initialization checkpoint `base-adv-early`. The checkpoint marked ⧫ is used for evaluation.

The final victim $\mathtt{v}_9$ is still vulnerable even at high visits. Our final simulated adversary $\mathtt{a}_9$ wins 42% of the time against $\mathtt{v}_9$ even at 65,536 visits. We trained $\mathtt{a}_9$ for longer than preceding adversaries, but its total training compute was still only 26% of $\mathtt{v}_9$'s.

All adversaries $\mathtt{a}_n$ exploit a cyclic group, but there are qualitative variations in the size and location of that group, other stones, and especially the group inside the cyclic one (elaborated in Appendix E.2.1). For example, $\mathtt{a}_9$ favors a small, nearly minimal inside shape (Fig. 3.2c). To humans, the differences are subtle, and the difficulty of defending against them does not vary significantly. But to the KataGo victims $\mathtt{v}_n$, the representations learned do not appear to generalize smoothly between these variations.

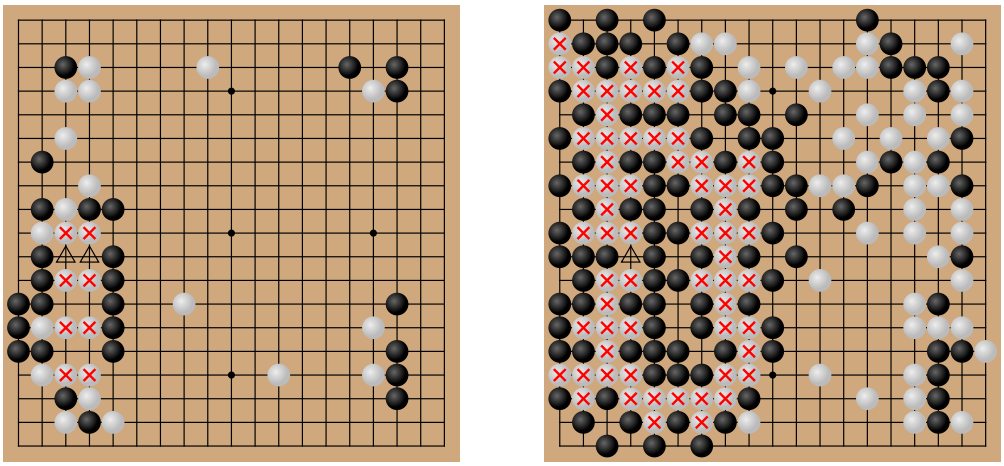
4.2.2 Robustness against a new adversary

Though the final iterated victim $\mathtt{v}_9$ bests all previous adversaries $\mathtt{a}_1$ to $\mathtt{a}_8$, the ultimate judge of a defense is whether it works against real attacks. To evaluate this, we train a new adversary `atari-adversary` (initialized to `base-adv-early`) against $\mathtt{v}_9$ (Fig. 4.2). This is analogous to a randomly initialized adversary trained to first beat the publicly available KataGo checkpoint `base-victim` at 1 visit and then—without access to any intermediate adversarial training checkpoints $\mathtt{v}_1,\ldots,\mathtt{v}_8$—trained to attack $\mathtt{v}_9$.

`atari-adversary` wins 81% of the time against $\mathtt{v}_9$ playing with 512 visits despite being trained with less than 5% of $\mathtt{v}_9$'s compute. The attack quickly learns to exploit $\mathtt{v}_9$ at low visits, winning over 60% of the time against $\mathtt{v}_9$ at 256 visits after just 500 V100 GPU days (Fig. 4.2), sooner than our original `base-adversary` adversary learned to exploit `base-victim`.

However, $\mathtt{v}_9$ proves harder to attack at high visits than `base-victim`. Quadrupling to 1024 visits takes slightly more than 4× the compute, largely due to the increased cost of playing training games at higher visits. `atari-adversary` plateaus after 1401 GPU days (⧫) with a meager 4% win rate at 4096 visits. By contrast, `base-adversary` generalized rapidly to beat `base-victim` at higher visits. See Appendix E.3 for more information.

`atari-adversary`'s attack is still cyclic, but with a characteristic tendency to leave many distinct stones and groups in "atari", i.e. that could be captured on the next move by $\mathtt{v}_9$. Moreover, it sets up "bamboo joints" (Fig. 4.3a): shapes where one player has two pairs of two stones with a one-space gap between them. They are common in normal play, and often advantageous: the two sides cannot be separated, as playing in the gap still allows connection through the remaining space. `atari-adversary` induces the victim to form a large cyclic group including these bamboo joints (Fig. 4.3b). The attack culminates by surrounding the cyclic group and finally threatening to split one of the bamboo joints. The correct play for



(a) `atari-adversary` induces the victim to set up several bamboo joints (×). These are normally strong shapes for connecting, e.g., if black plays a triangle-marked location, white can play the other to keep the joints connected.

(b) Ultimately, `atari-adversary` threatens to split one of the bamboo joints, and the victim prevents that by playing at the triangle location. But this is a terrible mistake—on the next move, the entire cyclic group will be captured.

Figure 4.3: The cyclic "bamboo joint" strategy learned by `atari-adversary`; explore online.

v_9 is to capture one of the numerous `atari-adversary` stones in atari, but v_9 misses the danger and connects the bamboo joint, leading to the entire cyclic group being captured.

5 Vision transformers

Wang et al. [50]'s attack works not only against KataGo but also against a range of other superhuman Go AIs such as ELF OpenGo [45], Leela Zero [27], Sai [26], Golaxy [9], and FineArt [44]. While it is possible that each of these systems is vulnerable to the cyclic attack for a different reason, it is more likely that shared properties such as their convolutional neural network (CNN) backbone cause their shared vulnerability.[5] Indeed, KataGo's developer proposed that vulnerability to cyclic attacks may be a result of the CNN backbone learning a local algorithm for classifying if a group is alive that fails to generalize to larger groups [34]. However, we demonstrate that superhuman Go AIs with vision transformer (ViT) backbones are also susceptible to cyclic attacks. This suggests the shared weakness is either AlphaZero-style training or deep learning more generally.

Since no prior work has trained strong Go AIs with a ViT architecture,[6] we first trained the world's first professional-level ViT-based Go AI. We follow a training recipe similar to the one used by KataGo [62], except we replace the CNN backbone with a ViT (Appendix F). Our strongest ViT network, which we label `ViT-victim`, was trained for 537 V100 GPU-days. It is slower to train than a CNN agent and weaker at the same inference budget, but we estimate it still reaches near-superhuman levels when playing with 32768 visits. This estimate is derived from benchmarking against KataGo, pitting our agent against players on the KGS Go Server, and winning two out of three games against Go professionals (Appendix G).

Despite the new architecture, Figure 3.1 shows that our `ViT-victim` (at 65536 visits) remains vulnerable to the cyclic attack, losing 78% of games to a fine-tuned version of `base-adversary`, which we call `ViT-adversary` (Appendix F.5). `ViT-adversary`'s strategy resembles other cyclic attacks but is qualitatively distinct in its tendency to produce small groups inside the cyclic one, and dense board states with limited open space (Fig. 3.2d). This attack can be replicated by a human expert (Appendix H).

[5] Golaxy and FineArt are closed-source but likely use the same design principles as other Go AIs.

[6] Sagri et al. [35] train ViT-based Go AIs but did not validate the strength of their systems or release weights. Moreover, they used supervised learning on KataGo self-play data generated by CNN agents, whereas we trained our ViT agents only on ViT-generated self-play data.

Remarkably, `ViT-victim` (at 512 visits) also loses 2.5% of games to the original `base-adversary`—similar to the zero-shot transfer to CNN Go AIs such as ELF OpenGo reported by Wang et al. [50]. `base-adversary` certainly does not win through legitimate means: it is a very weak strategy that loses to amateur human players [50]. This definitively shows that a CNN architecture is not the cause of the cyclic vulnerability.

6 Related work

We focus on robustness against *adversarial policies*: strategies designed to make an opponent perform poorly. Adversarial policies give an empirical lower bound for an agent's *exploitability*: its worst-case loss relative to Nash equilibria [46]. Gleave et al. [15] previously explored such policies in a zero-sum game between simulated humanoids trained with self-play. The policies [2] attacked by Gleave et al. were below human performance, raising the question: were the agents vulnerable because of their limited capability? To investigate this, Wang et al. [50] searched for adversarial policies against the superhuman Go AI KataGo [61], finding a strategy that beats KataGo in 97% of games.

We focus on KataGo [61] as it is the most capable open-source Go AI. Moreover, other superhuman open-source Go AIs such as ELF OpenGo [45] and Leela OpenZero [28] all follow the same basic AlphaZero-style training architecture. However, alternative multi-agent reinforcement learning methods may be more robust. Approaches that maintain a population of strategies are promising [12, 20, 49]. Another alternative, counterfactual regret minimization [66], has been used to beat professional human poker players [5]. Furthermore, Perolat et al. [31] found a method for approximating Nash equilibria [32] that scaled to the boardgame Stratego, whose game tree is 10^{175} times larger than Go's.

We replace the CNN backbone of KataGo with a vision transformer (ViT) and train the ViT agent to a superhuman level, finding it to be slower to train than a CNN agent and weaker at the same inference budget. By contrast, Sagri et al. [35] found the transformer-based EfficientFormer architecture [21] performed similarly to CNNs for Go—however, their models were trained only with supervised learning, not self-play. Transformers have been investigated more thoroughly in chess. Our results are consistent with Czech et al. [11] who found that CNNs are stronger at chess than both ViTs and a ViT-CNN hybrid at a given inference budget. Yet transformers have shown strong performance, with the transformer-based Leela Chess Zero [25, 29] winning the Top Chess Engine Championship Cup 11 [43].

Although we find ViTs are weaker than CNNs in average-case capabilities, our primary metric is *robustness*. Past research in image classification indicates ViTs are modestly more robust than CNNs against adversarial perturbations and other out-of-distribution inputs [3, 4, 30, 38, 65], although some research contests this [1, 23, 33, 42, 51]. Even if ViTs are not overall more robust, their differing inductive biases might cause them to fail in *different* ways to CNNs, with prior work finding ViTs are more vulnerable to patch perturbations [13]. Surprisingly, we find that not only are ViT-based Go agents exploitable by new attacks, but the attack of Wang et al. [50] transfers zero-shot to our ViT agent.

7 Discussion and future work

We explore three natural approaches for defending against adversarial attacks in Go: adversarial training with hand-constructed positions, iterated adversarial training, and using a ViT instead of a CNN. All of our defenses make attacks harder, increasing the amount of compute needed to successfully beat our defended systems. However, none of our defenses make attacks impossible – we show that the attack algorithm from Wang et al. [50] can always find a successful attack with a small fraction of the compute used to train the victims. Moreover, none of our defenses achieve the robustness of a human (Appendix B.1), and humans are even able to execute several attacks against our defenses (Appendix H). Our results highlight the challenge of defending against all possible attacks, or even all possible cyclic attacks, suggesting an offense-defense balance [18] favoring attackers.

We do, however, find it cheap to defend against any *fixed* (i.e. non-adaptive) attack. This suggests that **it may be possible to fully defend a Go AI by training against a large enough corpus of attacks**. We propose two complementary routes for realizing this.

The first approach is to increase the size of the attack corpus by developing new attack algorithms that require less compute to train a variety of adversaries. Our version of iterated adversarial training was bottlenecked by the attack component, which took 18x more compute than the defense component (see Table E.1). Reducing the time taken to find new attacks would allow the victim to train against many more attacks.

The second approach is to increase the sample efficiency of adversarial training by making the victim generalize from a limited number of adversarial strategies. Existing algorithms for training Go AIs do not generalize in this way: `v9` and KataGo's `dec23-victim` remain vulnerable to cyclic attacks even after being trained against many cyclic attack variants. However, algorithms like relaxed [8, 17] or latent adversarial training [6, 36] could significantly improve generalization.

There are also routes to robustness besides adversarial training. For example, multi-agent reinforcement learning schemes like like PSRO [20] or DeepNash [31] may be able to automatically discover strategies like the cyclic-attack and train them away. Another possibility is to change the threat model and use online or *stateful* defenses [7] which can dynamically update the victim in tandem with adversaries who are trying to learn an exploit.

Finally, we are eager to see explorations of attacks and defenses in domains other than Go where AI has surpassed human performance. In these settings, we also recommend a systematic study of whether increases in capabilities lead to increases in robustness.

Our results highlight the obstacles to building robust AI systems. If we are unable to achieve robustness in the well-defined and self-contained domain of Go, achieving robustness in more open-ended real-world applications will be even more challenging. To build AI safely, future advanced systems must have intrinsic robustness at the heart of their design.

Acknowledgments

We thank David Wu for discussing KataGo and its adversarial training with us and for helping us qualitatively describe the two attacks we found in Section 3, Adrià Garriga-Alonso for infrastructure support when running experiments, and ChengCheng Tan for helping create Fig. 1.1. We also thank ChengCheng Tan, Derik Kauffmann, Siao Si Looi, David Wu, Micah Carroll, and Daniel Filan for feedback on early drafts. Finally, we thank Yilun Yang (7 dan professional) and Ryan Li (4 dan professional) for playing `ViT-victim` to evaluate its strength, and Matthew Harwit for helping connect the authors with professional Go players.

Disclosure of funding

Tom Tseng, Euan McLean, and Adam Gleave were employed by FAR AI, a non-profit research institute, and supported by FAR AI's unrestricted funds. Kellin Pelrine was supported by funding from IVADO and by the Fonds de recherche du Quebéc. Tony Wang was supported by a Lightspeed grant.

Author contributions

Tom Tseng was the primary technical individual contributor, implementing the majority of the code and running the majority of the experiments. Euan McLean prepared an initial draft of the paper from high-level comments provided by technical contributors, edited the resulting paper, and coordinated the write-up. Kellin Pelrine, Tony Wang, and Adam Gleave were joint co-advisors throughout the project, and helped with paper writing and editing. In addition, Kellin Pelrine analysed the Go games and replicated attacks by hand, Tony Wang set up KGS bots for human evaluation, and Adam Gleave managed the project.

References

[1] Yutong Bai, Jieru Mei, Alan L Yuille, and Cihang Xie. "Are transformers more robust than CNNs?" In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 26831–26843.

[2] Trapit Bansal, Jakub Pachocki, Szymon Sidor, Ilya Sutskever, and Igor Mordatch. "Emergent Complexity via Multi-Agent Competition". In: *International Conference on Learning Representations*. 2018.

[3] Philipp Benz, Soomin Ham, Chaoning Zhang, Adil Karjauv, and In So Kweon. "Adversarial Robustness Comparison of Vision Transformer and MLP-Mixer to CNNs". In: *British Machine Vision Conference*. BMVA Press, 2021, p. 25. URL: https://www.bmvc2021-virtualconference.com/assets/papers/0255.pdf.

[4] Srinadh Bhojanapalli, Ayan Chakrabarti, Daniel Glasner, Daliang Li, Thomas Unterthiner, and Andreas Veit. "Understanding robustness of transformers for image classification". In: *IEEE/CVF International Conference on Computer Vision*. 2021, pp. 10231–10241.

[5] Noam Brown and Tuomas Sandholm. "Superhuman AI for heads-up no-limit poker: Libratus beats top professionals". In: *Science* 359.6374 (2018), pp. 418–424.

[6] Stephen Casper, Lennart Schulze, Oam Patel, and Dylan Hadfield-Menell. *Defending Against Unforeseen Failure Modes with Latent Adversarial Training*. 2024. arXiv: 2403.05030.

[7] Steven Chen, Nicholas Carlini, and David Wagner. "Stateful detection of black-box adversarial attacks". In: *Proceedings of the 1st ACM Workshop on Security and Privacy on Artificial Intelligence*. 2020, pp. 30–39. arXiv: 1907.05587.

[8] Paul Christiano. *Worst-case guarantees*. 2019. URL: https://ai-alignment.com/training-robust-corrigibility-ce0e0a3b9b4d (visited on 06/15/2024).

[9] 北京深客科技有限公司. *Golaxy*（星阵围棋）. 2018. URL: https://www.19x19.com/ (visited on 03/28/2024).

[10] Francesco Croce, Maksym Andriushchenko, Vikash Sehwag, Edoardo Debenedetti, Nicolas Flammarion, Mung Chiang, Prateek Mittal, and Matthias Hein. "RobustBench: a standardized adversarial robustness benchmark". In: *Advances in Neural Information Processing Systems*. 2021.

[11] Johannes Czech, Jannis Blüml, and Kristian Kersting. *Representation Matters: The Game of Chess Poses a Challenge to Vision Transformers*. 2023. arXiv: 2304.14918.

[12] Pavel Czempin and Adam Gleave. "Reducing Exploitability with Population Based Training". In: *International Conference on Machine Learning Workshop on New Frontiers in Adversarial Machine Learning*. 2022.

[13] Yonggan Fu, Shunyao Zhang, Shang Wu, Cheng Wan, and Yingyan Lin. "Patch-Fool: Are Vision Transformers Always Robust Against Adversarial Perturbations?" In: *International Conference on Learning Representations*. 2022.

[14] Adam Gleave. *Comment on Even Superhuman Go AIs Have Surprising Failure Modes*. 2023. URL: https://www.lesswrong.com/posts/DCL3MmMiPsuMxP45a/even-superhuman-go-ais-have-surprising-failure-modes?commentId=zztDTZmNGsSmhpbZ8 (visited on 04/18/2024).

[15] Adam Gleave, Michael Dennis, Cody Wild, Neel Kant, Sergey Levine, and Stuart Russell. "Adversarial Policies: Attacking Deep Reinforcement Learning". In: *International Conference on Learning Representations*. 2020.

[16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778.

[17] Evan Hubinger. *Relaxed adversarial training for inner alignment*. 2019. URL: https://www.alignmentforum.org/posts/9Dy5YRaoCxH9zuJqa (visited on 06/15/2024).

[18] Robert E. Jervis. "Cooperation under the Security Dilemma". In: *World Politics* 30 (1978), pp. 167–214. URL: https://api.semanticscholar.org/CorpusID:154923423.

[19] KGS. *Top 100 KGS Players*. Retrieved from https://www.gokgs.com/top100.jsp. 2022. URL: https://archive.is/J4Fjz (visited on 05/17/2024).

[20] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Perolat, David Silver, and Thore Graepel. "A Unified Game-Theoretic Approach to Multiagent Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Vol. 30. 2017, pp. 4190–4203.

[21] Yanyu Li, Geng Yuan, Yang Wen, Ju Hu, Georgios Evangelidis, Sergey Tulyakov, Yanzhi Wang, and Jian Ren. "EfficientFormer: Vision transformers at MobileNet speed". In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 12934–12949.

[22] Chang Liu, Yinpeng Dong, Wenzhao Xiang, Xiao Yang, Hang Su, Jun Zhu, Yuefeng Chen, Yuan He, Hui Xue, and Shibao Zheng. *A Comprehensive Study on Robustness of Image Classification Models: Benchmarking and Rethinking*. 2023. arXiv: 2302.14301.

[23] Kaleel Mahmood, Rigel Mahmood, and Marten van Dijk. "On the Robustness of Vision Transformers to Adversarial Examples". In: *IEEE/CVF International Conference on Computer Vision*. Oct. 2021, pp. 7838–7847.

[24] Mantas Mazeika, Long Phan, Xuwang Yin, Andy Zou, Zifan Wang, Norman Mu, Elham Sakhaee, Nathaniel Li, Steven Basart, Bo Li, David Forsyth, and Dan Hendrycks. *HarmBench: A Standardized Evaluation Framework for Automated Red Teaming and Robust Refusal*. 2024. arXiv: 2402.04249.

[25] Daniel Monroe. *Leela Chess Zero Training README*. 2023. URL: https://github.com/Ergodice/lczero-training/blob/0c10d4e19fbfd28abb167f9134fee74c983ef6db/README.md (visited on 02/06/2024).

[26] Francesco Morandin, Gianluca Amato, Rosa Gini, Carlo Metta, Maurizio Parton, and Gian-Carlo Pascutto. "SAI a Sensible Artificial Intelligence that plays Go". In: *International Joint Conference on Neural Networks*. IEEE, July 2019. URL: http://dx.doi.org/10.1109/IJCNN.2019.8852266.

[27] Gian-Carlo Pascutto. *Leela Zero*. 2019. URL: https://github.com/leela-zero/leela-zero/ (visited on 03/27/2024).

[28] Gian-Carlo Pascutto. *Leela Zero*. 2019. URL: https://zero.sjeng.org/ (visited on 06/16/2022).

[29] Gian-Carlo Pascutto and Gary Linscott. *Leela Chess Zero*. 2019. URL: https://lczero.org/ (visited on 02/06/2024).

[30] Sayak Paul and Pin-Yu Chen. "Vision transformers are robust learners". In: *AAAI Conference on Artificial Intelligence*. Vol. 36. 2. 2022, pp. 2071–2081.

[31] Julien Perolat, Bart De Vylder, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer, Paul Muller, Jerome T Connor, Neil Burch, Thomas Anthony, et al. "Mastering the game of Stratego with model-free multiagent reinforcement learning". In: *Science* 378.6623 (2022), pp. 990–996.

[32] Julien Perolat, Remi Munos, Jean-Baptiste Lespiau, Shayegan Omidshafiei, Mark Rowland, Pedro Ortega, Neil Burch, Thomas Anthony, David Balduzzi, Bart De Vylder, Georgios Piliouras, Marc Lanctot, and Karl Tuyls. "From Poincaré Recurrence to Convergence in Imperfect Information Games: Finding Equilibrium via Regularization". In: *International Conference on Machine Learning*. Vol. 139. 2021, pp. 8525–8535.

[33] Francesco Pinto, Philip HS Torr, and Puneet K. Dokania. "An impartial take to the CNN vs transformer robustness contest". In: *European Conference on Computer Vision*. Springer. 2022, pp. 466–480.

[34] polytope. *Comment on There are (probably) no superhuman Go AIs: strong human players beat the strongest AIs*. 2023. URL: https://www.lesswrong.com/posts/Es6cinTyuTq3YAcoK/there-are-probably-no-superhuman-go-ais-strong-human-players?commentId=gAEovdd5iGsfZ48H3 (visited on 02/14/2024).

[35] Amani Sagri, Tristan Cazenave, Jérôme Arjonilla, and Abdallah Saffidine. "Vision Transformers for Computer Go". In: *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*. Springer. 2024, pp. 376–388.

[36] Swami Sankaranarayanan, Arpit Jain, Rama Chellappa, and Ser Nam Lim. *Regularizing deep networks using efficient layerwise adversarial training*. 2018. arXiv: 1705.07819.

[37] Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, Timothy Lillicrap, and David Silver. "Mastering Atari, Go, chess and shogi by planning with a learned model". In: *Nature* 588.7839 (2020), pp. 604–609.

[38] Rulin Shao, Zhouxing Shi, Jinfeng Yi, Pin-Yu Chen, and Cho-Jui Hsieh. "On the Adversarial Robustness of Vision Transformers". In: *Transactions on Machine Learning Research* (2022). ISSN: 2835-8856. URL: https://openreview.net/forum?id=lE7K4n1Esk.

[39] Lloyd S. Shapley. "Stochastic Games". In: *PNAS* 39.10 (1953), pp. 1095–1100.

[40] 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". In: *Nature* 529.7587 (2016), pp. 484–489.

[41] 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". In: *Science* 362.6419 (2018), pp. 1140–1144.

[42] Shiyu Tang, Ruihao Gong, Yan Wang, Aishan Liu, Jiakai Wang, Xinyun Chen, Fengwei Yu, Xianglong Liu, Dawn Song, Alan Yuille, Philip H. S. Torr, and Dacheng Tao. *RobustART: Benchmarking Robustness on Architecture Design and Training Techniques*. 2022. arXiv: 2109.05211.

[43] TCEC. *LCZero 0.30-dag-dcb4ece9-BT2-3250000 vs Stockfish dev16_202301021914 - TCEC - Archived Game*. 2023. URL: https://tcec-chess.com/#game=1&round=fl&season=cup11 (visited on 04/28/2024).

[44] Tencent. *FineArt*. 2017. URL: https://en.wikipedia.org/wiki/Fine_Art_(software) (visited on 03/28/2024).

[45] 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: *International Conference on Machine Learning*. 2019.

[46] 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*. 2022. arXiv: 2004.09677.

[47] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. "Robustness May Be at Odds with Accuracy". In: *International Conference on Learning Representations*. 2019.

[48] Vijay Veerabadran, Josh Goldman, Shreya Shankar, Brian Cheung, Nicolas Papernot, Alexey Kurakin, Ian Goodfellow, Jonathon Shlens, Jascha Sohl-Dickstein, Michael C Mozer, et al. "Subtle adversarial image manipulations influence both human and machine perception". In: *Nature Communications* 14.1 (2023), p. 4933.

[49] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning". In: *Nature* 575.7782 (2019), pp. 350–354.

[50] Tony Tong Wang, Adam Gleave, Tom Tseng, Kellin Pelrine, Nora Belrose, Joseph Miller, Michael D Dennis, Yawen Duan, Viktor Pogrebniak, Sergey Levine, and Stuart Russell. "Adversarial Policies Beat Superhuman Go AIs". In: *International Conference on Machine Learning*. PMLR. 2023, pp. 35655–35739.

[51] Zeyu Wang, Yutong Bai, Yuyin Zhou, and Cihang Xie. "Can CNNs Be More Robust Than Transformers?" In: *The Eleventh International Conference on Learning Representations*. 2023. URL: https://openreview.net/forum?id=TKIFuQHHECj.

[52] David Wu. *Discord comment mentioning the last non-adversarially trained network*. Dec. 2022. URL: https://discord.com/channels/417022162348802048/583775968804732928/1056607918545457252.

[53] David Wu. *Discord comment on increasing training visits for the distributed training run*. Mar. 2023. URL: https://discord.com/channels/417022162348802048/583775968804732928/1090737750459814038.

[54] David Wu. *Discord comment on KataGo adversarial training data sources*. July 2023. URL: https://discord.com/channels/417022162348802048/723268423588642948/1131951228495081543.

[55] David Wu. *Discord comment on the initial hyperparameters of the distributed training run*. Dec. 2020. URL: https://discord.com/channels/417022162348802048/583775968804732928/786408459662917643.

[56] David Wu. *Discord comment on the initial KataGo adversarial training*. Dec. 2022. URL: https://discord.com/channels/417022162348802048/583775968804732928/1052951418685882408.

[57] David Wu. *Discord comment on the percentage of custom seeded self-play games*. Dec. 2023. URL: https://discord.com/channels/417022162348802048/583775968804732928/1180306891314839572.

[58] David Wu. *Discord comment on the purpose of custom seeded self-play games*. Mar. 2021. URL: https://discord.com/channels/417022162348802048/583775968804732928/820047133104537600.

[59] David Wu. *KataGo should be partially resistant to cyclic groups now*. July 2023. URL: https://www.reddit.com/r/baduk/comments/14prv4f/katago_should_be_partially_resistant_to_cyclic/.

[60] David Wu. *Other Methods Implemented in KataGo*. 2024. URL: https://github.com/lightvector/KataGo/blob/cbaa8625571ee6121fd62f7ab8a8ee3ef76bc250/docs/KataGoMethods.md (visited on 02/14/2024).

[61] David J. Wu. "Accelerating Self-Play Learning in Go". In: *AAAI Workshop on Reinforcement Learning in Games*. 2020.

[62] David J. Wu. *KataGo - Networks for kata1*. 2022. URL: https://katagotraining.org/networks/ (visited on 09/26/2022).

[63] David J. Wu. *KataGo Training History and Research*. 2021. URL: https://github.com/lightvector/KataGo/blob/master/TrainingHistory.md (visited on 09/28/2022).

[64] David J. Wu. *KataGo's Supported Go Rules (Version 2)*. 2021. URL: https://lightvector.github.io/KataGo/rules.html (visited on 09/27/2022).

[65] Chongzhi Zhang, Mingyuan Zhang, Shanghang Zhang, Daisheng Jin, Qiang Zhou, Zhongang Cai, Haiyu Zhao, Xianglong Liu, and Ziwei Liu. "Delving deep into the generalization of vision transformers under distribution shifts". In: *IEEE/CVF conference on Computer Vision and Pattern Recognition*. 2022, pp. 7277–7286.

[66] Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. "Regret Minimization in Games with Incomplete Information". In: *Advances in Neural Information Processing Systems*. Vol. 20. 2007.

A KataGo networks reference

We built on top of KataGo [61], which was the strongest open-source Go AI system at the time of conducting our research. KataGo learns via self-play using an AlphaZero-style training procedure [41]. The agent selects moves with Monte-Carlo Tree Search (MCTS), using a neural network to propose and evaluate moves. The neural network contains a policy head that outputs a probability distribution over the next move and a value head that estimates the win rate from the current state. KataGo trains its policy head to mimic the outcome of tree search and its value head to predict whether the agent wins the self-play game.

We evaluate and fine-tune a variety of KataGo models. We refer to each model's architecture as `b`B`c`C where B is the number of *blocks* in the convolutional residual network and C is the number of channels. We refer to each model by `b`B`c`C`-s`S`m` where S is the number of million time steps for which the model has been trained. We may omit the channel term `c`C when there is no ambiguity. All of our adversaries have 6 blocks and 96 channels, abbreviated to `b6c96` or just `b6`.

The victims we attack are either `b40c256` networks or `b18c384`. The `b18c384` networks use a new convolution-based architecture with modified bottleneck blocks [16, 60]. They were introduced into KataGo's official training run in 2023, becoming the strongest networks by the end of the year. The inference cost of these `b18` networks is similar to that of standard b40 networks. Given the same inference compute budget per move to perform search, they outperform the best standard `b40` and `b60c320` networks.

In Table A.1 we enumerate all victims used in this work, comprising official KataGo networks, those developed by Wang et al. [50] and those developed in this work. In Table A.2 we enumerate all adversaries used in this work, including our own and those developed by Wang et al.

Name	Params		Training		Date	Description
	B	C	Steps (M)	GPU-days		
`base-victim`	40	256	11841	21681	2022-06	Original target KataGo network for Wang et al. [50]'s adversarial attack. "kata1-b40c256-s11840935168-d2898845681" at `https://katagotraining.org/networks/`.
`may23-victim`	60	320	7702	25888	2023-05	KataGo network that had received 5 months' worth of adversarial training against cyclic positions. "kata1-b60c320-s7701878528-d3323518127" at `https://katagotraining.org/networks/`.
`dec23-victim`	18	384	8527	33482	2023-12	KataGo network that had received 1 year's worth of adversarial training against cyclic positions. "kata1-b18c384nbt-s8526915840-d3929217702" at `https://katagotraining.org/networks/`.
$\texttt{v}_n$	40	256	–	–	–	The victim at iteration n of our iterated adversarial training (see Section 4). $\texttt{v}_0$ is warm-started from `base-victim`. See Appendix E for breakdown.
$\texttt{v}_9$	40	256	12097	28296	2024-01	Our final iterated adversarially trained victim (see Section 4), warm-started from `base-victim`.
`ViT-victim`	16	384	650	537	2024-01	A network we trained from scratch using the same approach as KataGo but with the CNN backbone replaced with a vision transformer.

Table A.1: All victim networks used in this work with **B**locks, **C**hannels, training steps (in millions), and estimated compute cost (in V100 GPU days). The estimate of `base-victim`'s compute cost is from Wang et al. [50]. The training cost for $\texttt{v}_9$ includes both the training cost of all iterated victims and all iterated adversaries up to and including $\texttt{a}_8$.

Name	Training		Attack	Description
	Steps (M)	GPU-days	Style	
`base-adversary`	545	2223	cyclic	Original attack trained by Wang et al. [50] from scratch to defeat the KataGo network `base-victim`.
`base-adv-early`	227	164	non-cyclic	The first checkpoint able to defeat `base-victim` at one victim visit from the `base-adversary` training run.
`attack-may23`	713	3378	cyclic	`base-adversary` fine-tuned by Wang et al. [50] to defeat KataGo's adversarially trained network `may23-victim`.
`cont-adv`	1343	4476	cyclic	A network we trained using victim-play to defeat `dec23-victim`, using a fine-grained curriculum and fine-tuned from `attack-may23`.
`gift-adversary`	878	1865	gift	A network we trained using victim-play to defeat `dec23-victim`, using a coarse-grained curriculum and fine-tuned from `base-adv-early`.
$\mathtt{a}_n$	–	–	cyclic	The adversary at iteration n of our iterated adversarial training (see Section 4). $\mathtt{a}_0$ is fine-tuned from `base-adversary`. See Appendix E for breakdown.
$\mathtt{a}_9$	4132	7337	cyclic	The final adversary resulting from our iterated adversarial training (see Section 4).
`atari-adversary`	791	1401	complex cyclic	A network we trained using victim-play to defeat $\mathtt{v}_9$ fine-tuned from `base-adv-early`, to test the general robustness of iterated adversarial training.
`ViT-adversary`	871	2632	cyclic	A network we trained using victim-play to defeat `ViT-victim`, fine-tuned from `base-adversary`.

Table A.2: All adversary networks used in this work with training steps (in millions) and estimated compute cost (in V100 GPU days). The adversaries use a 6 block, 96 channel KataGo CNN architecture `b6c96`. The training cost for $\mathtt{a}_9$ consists of the training cost of all iterated adversaries as well as `base-adversary`.

Victim		Opponent		Opponent vs Victim	
Name	**Visits**	**Name**	**Visits**	**Compute (%)**	**Win rate (%)**
`base-victim`	4096	`base-adversary`	600	10	97
`base-victim`	10^7	`base-adversary`	600	10	72
`may23-victim`	4096	`attack-may23`	600	13	47
`dec23-victim`	4096	`continuous-adversary`	600	13	65
`dec23-victim`	65536	`continuous-adversary`	600	13	27
`dec23-victim`	512	`gift-adversary`	600	6	75
$\texttt{v}_9$	4096	`base-victim`	4096	77	66
$\texttt{v}_9$	512	`atari-adversary`	600	5	81
$\texttt{v}_9$	4096	$\texttt{a}_9$	600	26	59
$\texttt{v}_9$	65536	$\texttt{a}_9$	600	26	42
`ViT-victim`	512	`base-adversary`	600	414	2.5
`ViT-victim`	65536	`ViT-adversary`	600	490	78

Table A.3: The adversary win rate and fraction of opponent's compute used to train the opponent (right) for various victims (left) and opponents (middle). In most cases the victim was trained with much more compute than the opponent; the exception is `ViT-victim` which was trained for a relatively brief period, 4× less than `base-adversary`, although the additional fine-tuning compute used to train `ViT-adversary` was still less than that of `ViT-victim`. We standardize on an adversary search budget of 600 visits for all evaluations. The non-adversarial opponent, `base-victim`, is evaluated at the same number of visits as the victim. The first two rows show evaluations performed by Wang et al. [50].

B Definitions of robustness

In this section, we propose three complementary definitions for a Go policy being "robust". Although our definitions are targeted at Go policies, we believe the core ideas behind them are also applicable to more general AI agents. Each of the following three subsections introduces a definition of robustness, states the motivations behind it, and discusses how the definition could be extended to more general agents.

B.1 Human-robustness

Our first definition of robustness targets the concern that AI systems may fail in situations where humans would succeed. We call a system that does not have this failure mode *human-robust*. Intuitively, a system is human-robust if its worst-case performance is better than human average-case performance – that is, a human-robust system is not just sometimes but consistently superhuman.

We formalize this as follows: a system S is **human-robust** in an environment E if there are no points at which an omniscient observer could ask a human H having ordinary skill in the art to make some decisions in place of the system S, such that H – without the benefit of hindsight –would consistently and intentionally produce a substantially better outcome compared to S acting on its own.

Before elaborating on this definition, let's consider some examples of how this definition applies to different domains:

a. In the context of Go, a human-robust policy must not lose against opponents where a human could take over temporarily and consistently produce a win. The set of opponents under consideration is E.

   None of the defense strategies studied in this work meet this standard when E is the set of opponents that can be trained with a reasonable amount of compute and that have grey-box access to S during both training and inference.

b. Image classifiers which are not robust to ϵ-ball perturbations on natural images are probably not human-robust, since it is suspected that humans are highly invariant to small norm perturbations of natural images.[7] E here is the set of natural images.

Now that we have a couple examples in mind, let us unpack the different pieces of our definition in detail:

- The concept of a "human having ordinary skill in the art" is derived from patent law,[8] referring to a person with "normal skills and knowledge" of a particular field "without being a genius". We can define related concepts by specifying a different group of humans, giving rise to notions such as lay- or expert-human-robustness, or amateur- or professional- or world-champion-human-robustness.
- The decisions for a human to control are chosen by an omniscient observer so that the human does not need hindsight. If the human themselves were choosing when to intervene without hindsight, they would have to anticipate the robustness failures of the system or perform better than the system in all cases. The former would be unrealistic in many cases, while the latter would be defining a standard that cannot apply to systems that are sometimes superhuman. On the other hand, if the human had hindsight, they would be able to choose where to intervene themselves, but they would also have more knowledge of how to intervene (gained e.g. by observing the consequence of a bad action) than a human with ordinary skill in the art would. Thus our omniscient observer criterion sits somewhere in the middle between a human having hindsight and a human not having hindsight.
- We can also define a stronger form of robustness by removing the "intentionally" condition. This would allow for humans to make correct decisions without legitimate reasons for them. For instance, some Go beginners will rush to capture stones even

[7] Though this is still an claim that is being actively researched [48].

[8] `https://en.wikipedia.org/wiki/Person_having_ordinary_skill_in_the_art`

when the opponent already can't save them. This wastes many moves, but potentially blocks an adversary from using those stones later. In situations like these, humans might be more robust, but in an unstable way: when those beginners learn more and understand that their opponent couldn't save those stones, they will no longer rush to capture them, destroying that robustness.

- We say that a human must "consistently ... produce [a] better outcome" because there are plenty of situations where a human could randomly make the right decision. But in these situations, a flip of a coin could make the right decision too. Thus the consistency criterion makes our standard for human-robustness less stringent. The exact strength of our definition can be adjusted by specifying a particular probability that a human should make the right decision.
- The meaning of "substantially better outcome" should depend on the context and level of robustness needed. In the context of Go, the clearest standard is winning or losing the game. This could be made more strict by requiring that human interventions cannot increase the the final score differential by any significant amount (e.g. over 5 points) in addition to being unable to change the final win/loss outcome. We find this stricter standard is currently unnecessary since we find none of the Go AIs we test meet even the weaker standard.
- Finally, our definition applies to a specific environment E (instead of quantifying over all possible environments) because of the existence of pathological environments that make it so that no non-human system can be human-robust in all possible environments.

  For example, imagine a deployment environment where the system S fails due to some near-omnipotent entity O stacking the cards against S (for example in the Go domain this would correspond to S facing an all-around much stronger Go program O). However, this near-omnipotent entity O also has a backdoor – if they detect any sign of human intervention, they will "un-stack" the odds in the opposite direction (for example instantly resigning the game in the case of Go). The existence of such a pathological environment implies that no computationally feasible non-human system can be human-robust in all possible environments.

We think our definition of human-robustness is a useful lens through which to think about robustness. In particular, if a system is human-robust in all non-pathological environments, that implies it will only fail in ways humans would too, and will not create new vulnerabilities. This is a practically useful desiderata. Moreover, our definition is sufficiently concrete to be falsifiable in real-world scenarios.

In the case of Go, we note that the Go AIs in our experiments do not even achieve the weakest version of our human-robustness criterion, since an amateur human can make correct decisions to defend against our attacks virtually 100% of the time. Specifically, for all the cyclic attacks in Fig. 3.2, the human could simply capture the adversary's group inside the cyclic group, before the cyclic group itself is captured. This is trivial since the inside groups have few liberties and no options to defend against that. Meanwhile, to defend against `gift-adversary`, a human would simply not offer the gift, for example, connecting at the location marked with △ in Fig. 3.2b. Similarly, to defend against `atari-adversary`, a human just needs to avoid filling in their own last liberty, i.e., playing anywhere else such as one of the numerous captures available, instead of the location marked with △ in Fig. 4.3b.

B.2 Training-compute-robustness

Our second definition of robustness captures the property that a robust Go AI should only be beatable by using a large amount of computational resources. This definition naturally extends to more general agents – *training-compute-robust* agents should only be exploitable (i.e. induced to fail in dramatic ways) by adversaries that have large computational resources.

There are different types of training-compute-robustness for different threat models, e.g. black-box, grey-box, and white-box. We focus on grey-box training-compute-robustness in this work, but also comment briefly on the other types below.

We formalize this for Go as follows. Let π be a policy for playing Go, which uses a fixed I amount of inference compute per move. The **p-level training-compute-robustness** of π is the minimum amount of compute needed to train an adversarial policy π_{adv} that can defeat π at least a $(1-p)$ fraction of the time while also using I inference-compute or less per move. We minimize over a set $\mathcal{T}$ of possible training schemes, e.g. different hyperparameters or attack algorithms.

Note that in general we can only upper bound training-compute-robustness, since minimizing over all possible adversarial-policy training schemes in $\mathcal{T}$ is intractable for large or infinite $\mathcal{T}$. A trivial upper bound on the 50%-level training-compute-robustness for a policy π is the net amount of compute that was used to train π. This is because a policy can always win against itself with 50% probability. In fact, if the attacker has white-box access to π then the 50%-level white-box training-compute-robustness of any policy is *zero*, as the attacker can just set $\pi_{\text{adv}} = \pi$ without any training.

Given any self-play algorithm $\mathcal{A}$ parameterized by training compute (i.e. $\mathcal{A}$ produces a policy given a fixed amount of compute), we can also use $\mathcal{A}$ to obtain an upper bound on p-level training-compute-robustness of any policy π. Namely, we can measure the minimum amount of compute (possibly infinite) needed by $\mathcal{A}$ to produce a policy that can win against π at least a $(1-p)$ fraction of the time.

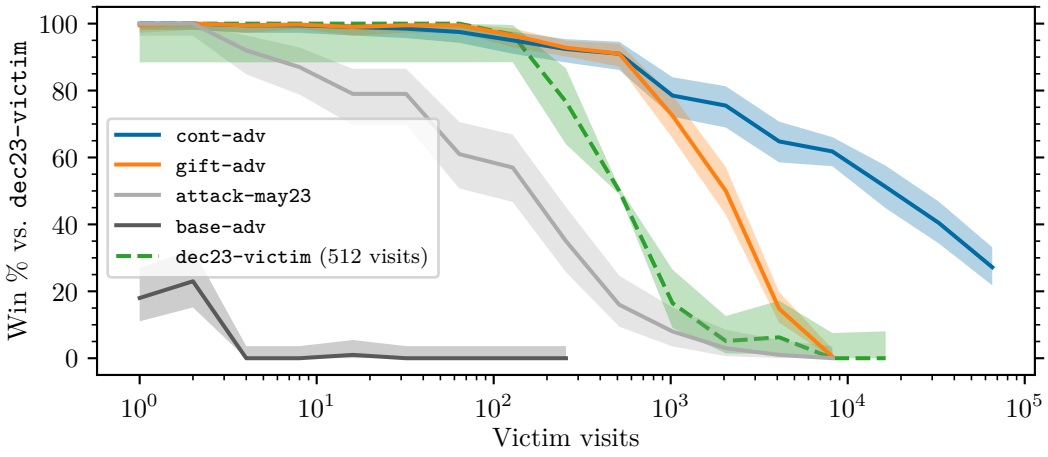
Training-compute-robustness can be measured in different units, e.g. FLOPs, V100 GPU days, A100 GPU days, etc. A unit-less way to measure training-compute-robustness is as a fraction of the compute needed to train π. We call this *relative training-compute-robustness*. The relative 50%-level training-compute-robustness of a policy π is always less than 1.

B.3 Inference-compute-robustness

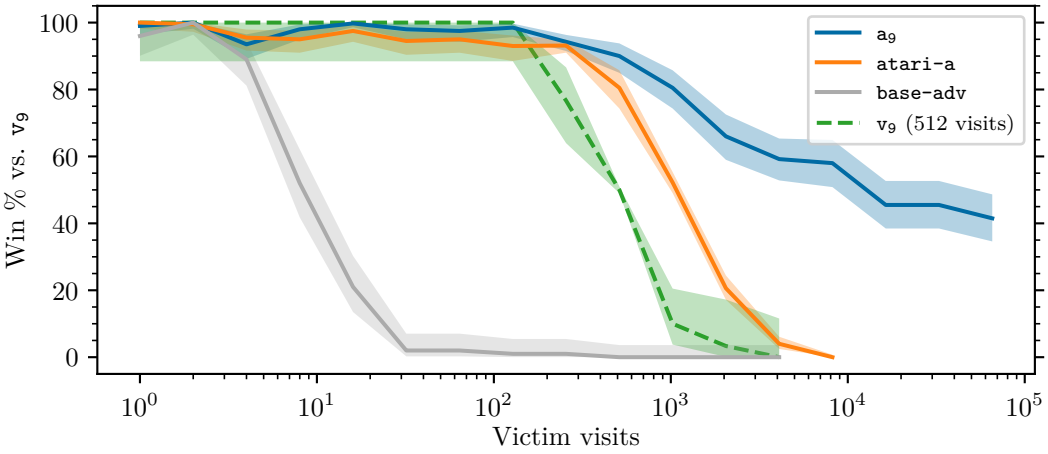
Our third and final definition of robustness captures the criterion that a robust system should be able to effectively correct its own mistakes given enough time to check its work at inference-time.

We formalize the **inference-compute-robustness** of a Go policy π against an opponent π_{adv} as the rate at which π's win-rate against π_{adv} increases as a function of π's inference-compute. The faster the rate of increase, the more inference-compute-robustness π has. Taking the slowest win-rate scaling trend over all π_{adv} in some threat-model set yields an aggregate measure of inference-compute-robustness.

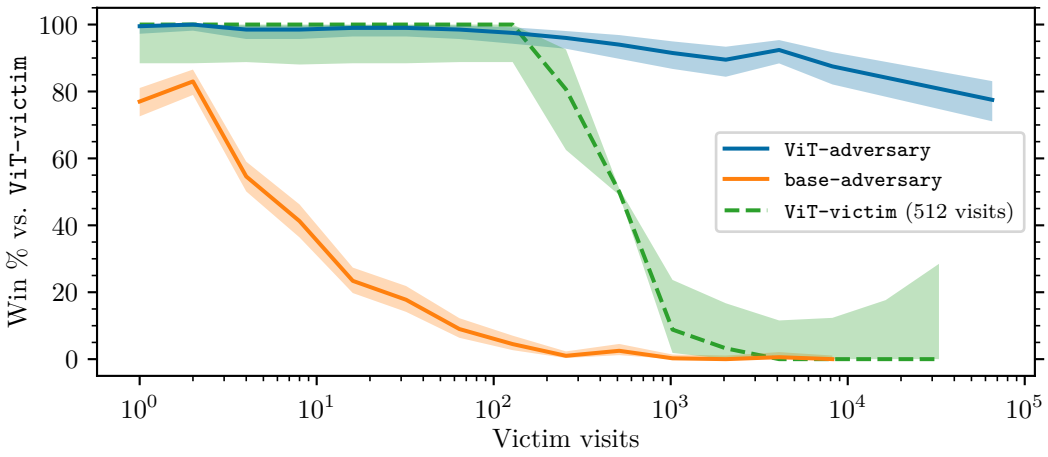
An upper-bound on inference-compute-robustness can be obtained by pitting a policy against a version of *itself* with a fixed inference budget. Any scaling trend which is significantly slower than this baseline scaling trend is an indication that a policy lacks inference-compute robustness. We use this baseline scaling trend in Figure B.1 to show that none of our defended victims have strong inference-compute-robustness against our strongest adversaries. The same figure does, however, show that our defenses are fairly inference-compute-robust with respect to some of our adversaries, like `atari-a` and `gift-adv`, even though these same adversaries demonstrate the defenses' lack of human-robustness and train-compute-robustness.



(a) Positional Adversarial Training



(b) Iterated Adversarial Training



(c) Vision Transformer (ViT)

Figure B.1: A version of Figure 3.1 with additional baseline scaling trends for the victim vs. itself. We note that the strongest adversaries stay above the baseline curve (i.e. the victim must spend more inference compute to beat them than it needs to beat itself). This is an indication that none of our defended victims have strong inference-compute-robustness.

C Training parameters

C.1 Training window

KataGo generates training data from self-play games. The model then trains on a sample from a sliding window of the most recent training data. The default starting window size is $m_0 = 250,000$ samples or "rows," and when there are N total training rows, the window size m scales as a power law in N:[9]

$$m = \frac{.4m_0^{.35}}{.65} \cdot \left(N^{.65} - m_0^{.65}\right) + m_0. \tag{1}$$

Each training "epoch" consumes approximately 250,000 data rows and performs 1 million training steps.

All of our models, besides our self-play ViT models, involve *warm-starting* from KataGo models or models trained by Wang et al. [50]. Warm-starting from a model, or fine-tuning a model, means we initialize our training from that model and pre-seed the training data with that model's training history. The pre-seeding increases the window size by increasing N in Eq. (1), and it populates the training window with the pre-existing training data. Without pre-seeding the data, the default starting window size would be small and cause over-fitting. We could also increase the training window size by increasing m without pre-seeding, but then there is a high initial cost to generate enough new data to populate the starting window.

C.2 Configuration parameters

The board size varies randomly between training games, allowing KataGo to learn to play Go on various board sizes. Because we focus on 19x19 games in our evaluations, we train our adversaries primarily on 19x19 games: 53.6% to be precise, matching the distribution used in recent KataGo training. This contrasts with the attack of Wang et al. [50] who set only 35% of games to be 19x19, following a distribution of board sizes matching those used for early KataGo training of small 6-block and 10-block models.

When training our adversaries, we disable the variance time loss (`vtimeloss` in the KataGo code), an auxiliary loss on a model output predicting uncertainty in the game's outcome. We disabled it following Wang et al.'s finding that this stabilized their early training runs, although we did not confirm its impact on our training.

Like Wang et al., our adversary training uses curricula in which the adversary plays against increasingly strong victims, switching to a stronger victim once the adversary win rate exceeds a certain threshold. We usually set the threshold to 75%. However, we increased the threshold to 90% for higher visit count victims (typically 512 or more) since at that point higher victim visit counts substantially increase the cost of generating games, making it more computationally efficient to train at a slightly lower sample efficiency but with cheaper samples.

When training adversaries against a victim using fewer than 100 victim visits, we enable Wang et al.'s pass-alive defense to prevent the adversary from learning the degenerate "pass attack" that they encountered in low-visit victims.

We change several training configuration parameters listed below compared to Wang et al., usually tweaking these parameters partway into training runs since we only identified or began experimenting with them after launching the runs.

Enabling selecting moves by the lower-confidence bound (LCB) on their utility. Selecting moves by LCB is the default in evaluation but is disabled in training because the creator of KataGo found that enabling LCB reduced self-play training progress despite making evaluation stronger.[10]. We found that having LCB disabled led to a large strength

[9]The sliding window is implemented by the script at `https://github.com/lightvector/KataGo/blob/eaaddd82339750d9defc70f566e6c59d7068b7b3/python/shuffle.py`, and the `--help` documentation string for the script gives this equation.

[10]The creator of KataGo details their LCB experiments at `https://github.com/leela-zero/leela-zero/issues/2411`.

gap between training and evaluation. We preferred to keep train and evaluation similar so that we could be more confident that training progress correlated with evaluation strength.

Adjusting other victim configuration parameters to more closely match the settings used during evaluation. For example, parameters that govern exploration vs. exploration trade-offs (like temperature), or the KataGo "optimism" feature[11].

In some training runs, we only changed a subset of these parameters because we had not yet discovered all of these parameters disparities. The full list of parameters we change in the final runs is:

```
antiMirror = true
chosenMoveTemperature = 0.10
chosenMoveTemperatureEarly = 0.50
conservativePass = true
cpuctExploration = 1.0
cpuctExplorationLog = 0.45
cpuctUtilityStdevScale = 0.85
dynamicScoreCenterScale = 0.75
dynamicScoreCenterZeroWeight = 0.2
dynamicScoreUtilityFactor = 0.3
enablePassingHacks = true
fillDameBeforePass = true
policyOptimism = 1.0
rootDesiredPerChildVisitsCoeff = 0
rootFpuReductionMax = 0.1
rootNoiseEnabled = false
rootNumSymmetriesToSample = 1
rootPolicyOptimism = 0.2
rootPolicyTemperature = 1.0
rootPolicyTemperatureEarly = 1.0
staticScoreUtilityFactor = 0.1
subtreeValueBiasFactor = 0.45
subtreeValueBiasWeightExponent = 0.85
useNoisePruning = true
useNonBuggyLcb = true
useUncertainty = true
valueWeightExponent = 0.25
```

Adjust adversary configuration parameters to more closely match the settings used in the latest KataGo training runs. This involves slight adjustments in exploration and utility computation, as well as a small bugfix related to LCB. We made these changes under the assumption that later KataGo configurations are superior to early ones our initial parameter settings were based on, although we did not check if this made a major difference in our training. The full list of parameters we change is:

```
cpuctExploration = 1.05
cpuctExplorationLog = 0.28
dynamicScoreCenterScale = 0.75
dynamicScoreUtilityFactor = 0.30
rootPolicyTemperatureEarly = 1.5
staticScoreUtilityFactor = 0.05
subtreeValueBiasFactor = 0.30
useNonBuggyLcb = true
```

[11] Policy optimism is described at `https://github.com/lightvector/KataGo/blob/828f1bc27617f9a7dc881d11a7296856ef7c4fc0/docs/KataGoMethods.md#optimistic-policy`. Wang et al. used a version of KataGo that had not yet introduced this feature.

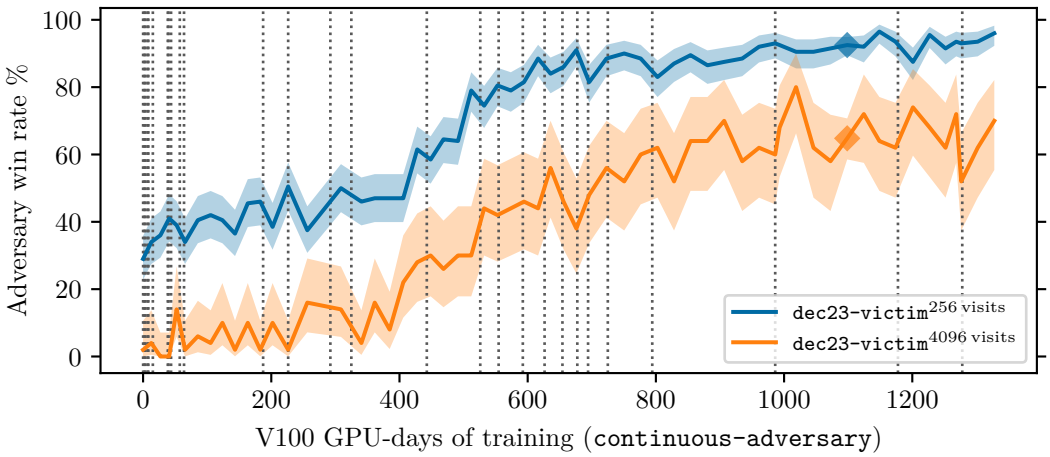


Figure D.1: Win rate (%) of `continuous-adversary` (marked ♦) against `dec23-victim` throughout fine-tuning against `dec23-victim`. The zero of the x-axis represents the win rate of `attack-may23` against `dec23-victim` before the fine-tuning against `dec23-victim` began.

D Positional adversarial training

D.1 Continuous attack

This section gives more details on the `continuous-adversary` attacking `dec23-victim` described in Section 3.2.

We warm-started from `attack-may23` since Wang et al. found it to be effective against KataGo's `b18` networks. `attack-may23` was trained to attack an adversarially trained KataGo network `may23-victim` (`b60c320-s7702m`) released on May 17, 2023 (see Table A.1). We trained the adversary for a further 1098 V100 GPU-days and 630 million training steps, for a total of 4476 GPU days and 1343 million training steps (Table A.2).

We started the curriculum at 1 victim visit, doubling the visits when a win rate threshold was reached. We set the threshold to 75% up to 256 visits, and 90% after that due to the increased cost of generating training games against high visit-count victims. We periodically updated the KataGo `b18` checkpoint used.

Figure D.1 shows `continuous-adversary`'s win rate against `dec23-victim` throughout adversary training. Figure D.2 shows `continuous-adversary`'s win rate against several `b18` KataGo networks. We see that `continuous-adversary` successfully attacks all `b18` networks up until `b18-s9432m` when `continuous-adversary` positions were introduced into KataGo's training data, at which point `continuous-adversary`'s win rate drops quickly. This decline was faster than when KataGo initially introduced positions from `base-adversary` into KataGo's training data—at that time, it took several hundred million training steps to make `base-adversary`'s win rate to dramatically drop, see Wang et al. [50, Figure L.2].

The full curriculum was:

- `b18-s7283m` (released August 17, 2023), 1–16 visits.
- `b18-s7313m`, 16–32 visits.
- `b18-s7343m`, 32–256 visits.
- `b18-s7373m`, 256 visits.
- `b18-s7500m`, 256–512 visits.
- `b18-s7590m`, 1024 visits.
- `b18-s7620m`, 256 visits. (Here we reverted visits to 256 because earlier visit increases were due to non-representative samples of games skewing our curriculum advancement script into giving inaccurate win rate estimates.)

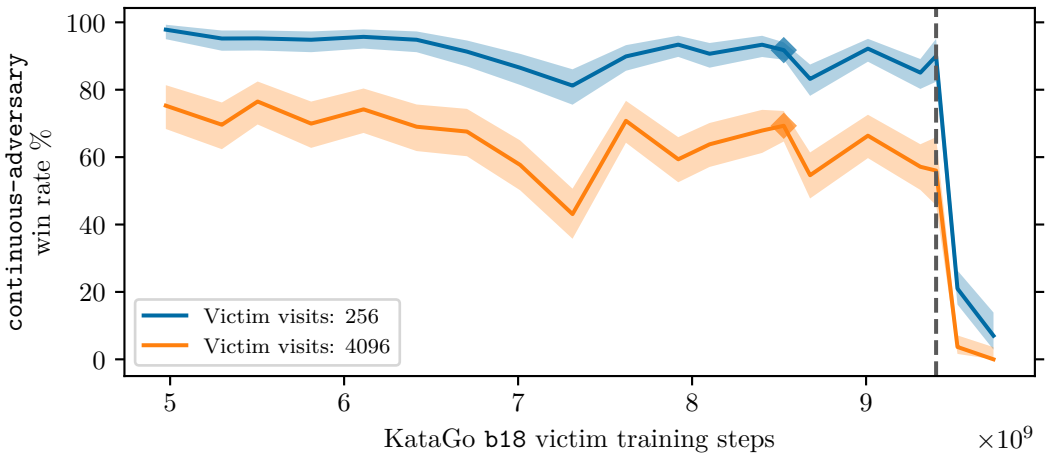


Figure D.2: The win rate (%) of `continuous-adversary` against the main KataGo training run between networks `b18-s4975m` and `b18-s9732m`. The marked point ♦ is `dec23-victim`. At the dashed line, the KataGo developers added positions from `continuous-adversary` and `gift-adversary` into KataGo's adversarial training data, which caused the win rate to drop.

- `b18-s7680m`, 256 visits.
- `b18-s7740m`, 256 visits.
- `b18-s7830m`, 256 visits.
- `b18-s7890m`, 256 visits.
- `b18-s7950m`, 256 visits.
- `b18-s8010m`, 256 visits.
- `b18-s8071m`, 256–512 visits.
- `b18-s8191m`, 512 visits.
- `b18-s8282m`, 512 visits.
- `b18-s8463m` (released Dec 11 2023), 512 visits. This is the last curriculum checkpoint that our chosen adversary checkpoint `continuous-adversary` at 1098 V100 GPU-days saw. The remaining curriculum checkpoints were seen by adversary checkpoints beyond the one we chose for main evaluations in this paper.
- `b18-s8588m-v512`
- `b18-s8678m-v512` (released Jan 9 2024)

We initially observed a large win rate gap between training and evaluation. To close this gap, we made only two changes to the training configuration, rather than all the changes listed in Appendix C.2: we enabled LCB move selection and activated optimism for the victim.

D.2 Gift attack

This section gives more details on the `gift-adversary` attacking `dec23-victim` described in Section 3.3.

The adversary is warm-started from `base-adv-early`. The curriculum began with `dec23-victim` at 4 visits, increasing up to 8 visits in 1 visit increments, then doubling visits each time until 512 visits. We added the extra victim visits between 4 and 8 because after finding that a direct increase from 4 to 8 visits led to a large win rate drop and minimal training progress. The adversary was trained for a further 1697 V100 GPU-days and 651 million training steps, totalling 1861 GPU-days and 878 million steps (Table A.2).

Figure D.3 shows the win rate of `gift-adversary` throughout training. Figure D.4 shows `gift-adversary`'s win rate against several `b18` KataGo networks. Either `gift-adversary` is

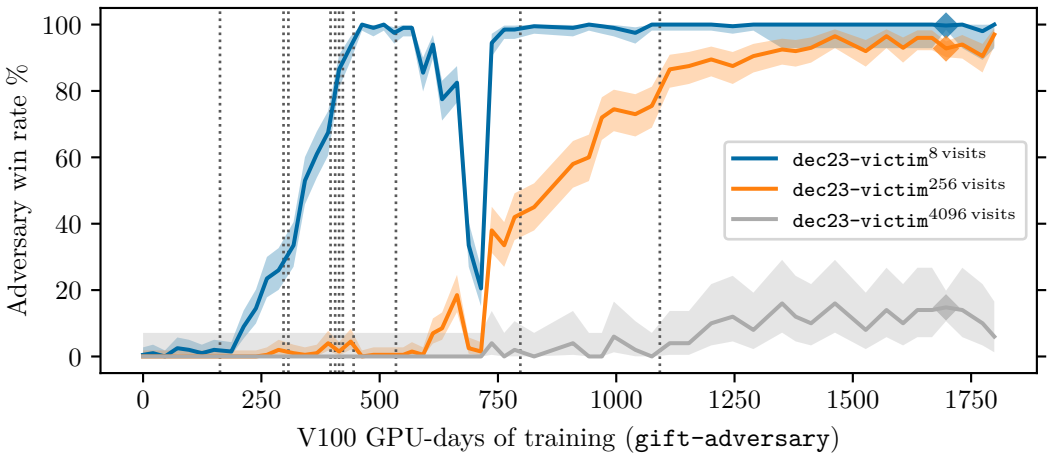


Figure D.3: Win rate (%) of `gift-adversary` (marked ♦) against `dec23-victim` throughout fine-tuning against `dec23-victim`. The zero of the x-axis represents the win rate of `base-adv-early` against `dec23-victim` before the fine-tuning against `dec23-victim` began. The large drop in win rate at 700 GPU-days occurred when the curriculum prematurely increased from 128 visits to 256 visits. The adversary's win rate against `dec23-victim` at 256 visits was poor, and it was not learning well. After we reverted the curriculum back to 128 visits, the win rate surprisingly recovered, seemingly without hindering training progress.

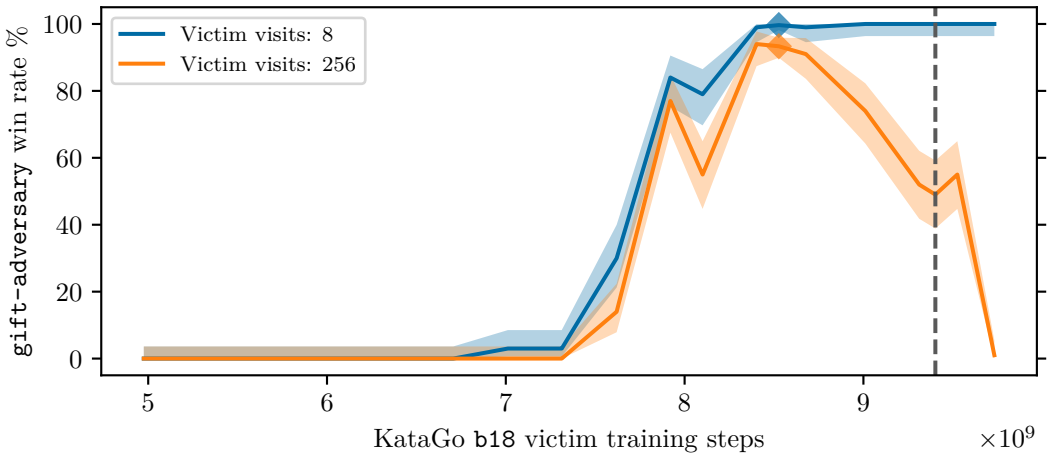


Figure D.4: The win rate (%) of `gift-adversary` against the main KataGo training run between networks `b18-s4975m` and `b18-s9732m`. The marked point ♦ is `dec23-victim`. At the dashed line, the KataGo developers added positions from `continuous-adversary` and `gift-adversary` into KataGo's adversarial training data.

highly specialized to setting up the gift attack against `dec23-victim`, or the gift vulnerability only appears in recent KataGo `b18` nets. David Wu, the main developer of KataGo, suggests the former is more likely. After we disclosed this vulnerability, he examined older KataGo nets and found that they also misjudge board positions produced by `gift-adversary`.

At 163 V100 GPU-days (79 million training steps), we adjusted victim configuration parameters to more closely match evaluation as described in Appendix C.2.

At 170 V100 GPU-days (81 million training steps), we reduced the training move limit per game from KataGo's default of 1600 moves to $900 * (\text{board area})/(19^2)$ moves since we noticed several games dragging out to hit the move limit due us enabling the pass-alive defense (Appendix C.2), which lengthens games, on low-visit victims during training. This is before the adversary had discovered the gift attack, and games were not noticeably longer than `atari-adversary`'s games at a similar point in `atari-adversary`'s training. Still, we hypothesized this would increase training efficiency by cutting the duration of lengthy games, which cost compute and generate an excessive amount of end-game policy training data.

At 475 V100 GPU-days (220 million steps), we noticed that the adversary learned to prolong a significant portion of games using extended ko fights to hit the 900-move limit. Normally during training, such games are scored and assigned a winner based on the final board state. Not only does having lots of games hit the move limit significantly slow down training, but we were also worried that the final board state score does not necessarily reflect what the score would have been had the game been played out to completion. The adversary could reward hack by stalling in a state that is winning if scored prematurely but is losing if played to the end.

We therefore at 632 V100 GPU-days (307 million steps) began scoring games that hit the move limit with a score of 0 and marked them as losing for the adversary (-1 utility). This drove the hit-move-limit rate from 59% to 22%. At 836 V100 GPU-days (393 million steps), we reduced the utility of such games even further to -1.6, below the worst typically possible utility -1.35 resulting from losing a game and having the opponent control all territory on the board. This drove the hit-move-limit down to 0%.

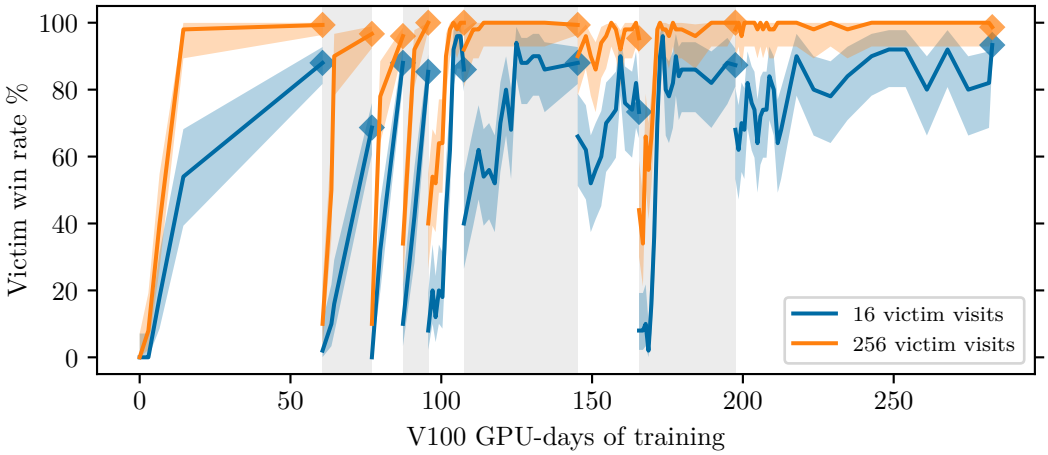


Figure E.1: The victim $\mathtt{v}_n$ win rate (%) against $\mathtt{a}_{n-1}$ throughout iterated adversarial training. Iterations are signified by alternating between a white and grey background. The curves for $\mathtt{v}_1$ to $\mathtt{v}_4$ only have a few data points along the x-axis as intermediate checkpoints were lost.

E Iterated adversarial training

E.1 Defense

At each iteration, we train a victim $\mathtt{v}_n$ to defend against a fixed adversary $\mathtt{a}_{n-1}$. Figure E.1 shows the training progress of each $\mathtt{v}_n$ against $\mathtt{a}_{n-1}$. Figure L.1 shows the same information but with a separate plot for each iteration. We see that the victim always made rapid progress in defending against the adversary, but continued to lose a significant fraction of the time at 16 victim visits, and still suffered occasional losses at 256 victim visits.

The first $\mathtt{v}_1$ is warm-started from $\mathtt{v}_0$ (`base-victim`) and is trained against `base-adversary`. We do not use a curriculum in victim training. We reduced the learning rate by a factor of 10 from KataGo's default since the base model `base-victim` had been trained with a lower learning rate scale as well. We found that fine-tuning with the default learning rate led to a large initial drop in model strength.

The victim plays with 300 MCTS visits, while the adversary plays with 600 A-MCTS visits. We chose 600 A-MCTS visits for the adversary to follow the default number of visits for adversary training and evaluation used by Wang et al. [50]. We chose 300 MCTS visits for the victim because it keeps the inference cost of the victim similar to the adversary's—roughly $600/2 = 300$ visits of the adversary's A-MCTS invoke the victim model, with the remaining visits invoking the smaller, cheaper adversary model.

The training window size begins at 68 million rows to match the window size of `base-victim`.[12] This is large enough that throughout defense training, all games generated in prior defense iterations remain in the training window. Although keeping all the games in the window was not an intentional design choice, it likely contributes to each $\mathtt{v}_n$ defending well against every $\mathtt{a}_m$ with $m < n$.

The victim was trained with a mixture of self-play games and games against the adversary. Self-play games help preserve general Go strength, whereas games against the adversary focus on overcoming specific attacks. We set the game mix to 82% self-play and 18% against the adversary. This proportion was based on preliminary experiments suggesting that training on 90% self-play data and 10% adversary data makes rapid progress in overcoming the adversary without compromising general Go strength (estimated via win rate against `base-victim`). Self-play games generate twice as much policy training data as games against the adversary

[12]The window size was calculated from Eq. (1) using the fact that `base-victim` was trained on 2.9 billion rows of data.

Iteration	Victim		Adversary	
n	GPU-days	Steps (M)	GPU-days	Steps (M)
1	61	61	238	150
2	16	22	439	253
3	10	16	273	213
4	8	11	1195	983
5	12	10	862	535
6	38	20	304	228
7	20	13	491	372
8	32	32	308	230
9	85	71	1005	622
Total	**282**	**256**	**5114**	**3587**

Table E.1: The cost of training the victim $\mathtt{v}_n$ and adversary $\mathtt{a}_n$ at each iteration n of iterated adversarial training.

because the model only trains on its own moves in adversary games. Setting the proportion of selfplay games to 82% makes the generated game data roughly match 90% from self-play.

For simplicity, we only used Tromp-Taylor rules, since the adversaries were also only trained on these rules. We also disabled many KataGo self-play flags (auto-komi, komi randomization, handicap games, game forking, cheap search, reduced search when winning, playing initial moves directly from policy) to simplify implementation.

In each iteration, we hand-select the final model for the subsequent iteration based on expected strength. All else equal, we choose the checkpoint with the highest win rate against the adversary. However, as the win rate against the adversary tends to plateau, we additionally favor checkpoints from *stable* periods of training where immediately preceding and succeeding checkpoints also have high win rates. We break ties in favor of earlier checkpoints.

E.1.1 Defense per-iteration

In this section we discuss each individual iteration in more detail. We provide the training cost (in training steps and V100 GPU-days) of each iteration in Table E.1. Additionally, we discuss any configuration changes or notable results that occurred in iterations below.

Defense iteration 1: The win rate at 300 victim visits against the cyclic adversary already plateaued after 14 of the 61 GPU-days (16 of 61 million steps), but we continued training in hopes of achieving a consistent 100% win rate against the adversary.

Defense iteration 4: An error occurred in populating the training history, where extra data from running the previous defense iteration was added for an additional 58 million steps beyond our selected checkpoint $\mathtt{v}_3$. We identified this error and removed the extraneous data for subsequent defense rounds.

Defense iteration 6: In iteration 6 and 7 we unintentionally generated games faster than we were training on them, which is why the GPU-days relative to the number of training steps is higher.

Defense iteration 9: We ran this iteration longer than usual because it was our final defense iteration. We also noticed that its win rate at 8 visits increased modestly (from 49% to 74% at the end of training), even though the training win rate at 300 visits against $\mathtt{a}_8$ remained around 97% for the entire run.

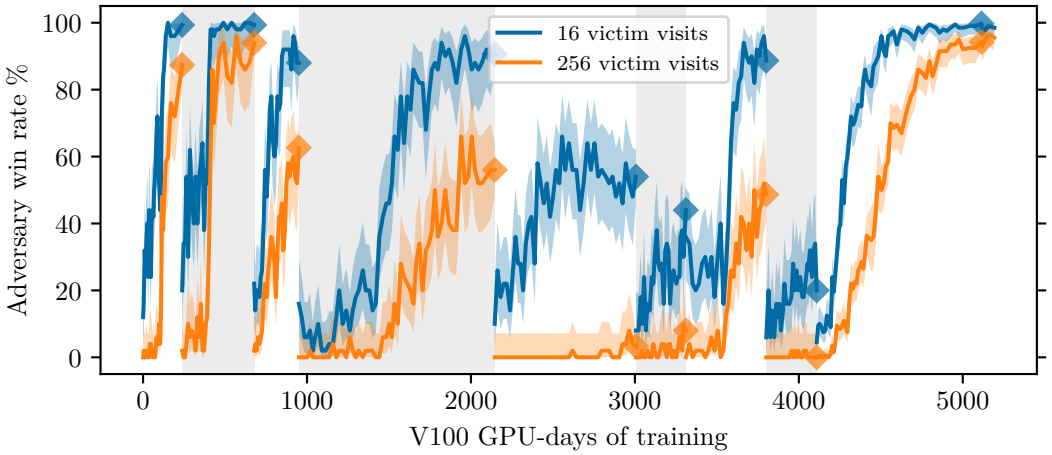


Figure E.2: The adversary $\texttt{a}_n$ win rate (%) against $\texttt{v}_n$ throughout iterated adversarial training. Iterations are signified by alternating between a white and grey background.

E.2 Attack

At each iteration, we train an adversary $\texttt{a}_n$ to attack $\texttt{v}_n$ warm-starting from the previous adversary $\texttt{a}_{n-1}$. The very first iteration $\texttt{a}_1$ is warm-started from $\texttt{a}_0 = \texttt{base-adversary}$. Figure E.2 shows the training progress of each $\texttt{a}_n$ against $\texttt{v}_n$. Figure L.2 shows the same information but with a separate plot for each iteration.

As can be seen in Fig. 4.1, the adversaries at iterations 5, 6 and 8 perform especially poorly for victim visits of 16 or above. The main reason for this is that training progress significantly slowed down; the number of victim visits reached by each iteration from $\texttt{a}_5$ onwards was at most 64. Additionally, in iterations 6 and 8, those adversaries were trained for relatively brief periods. The computational expense of training potent attacks is a bottleneck to performing a large number of iterations of adversarial training.

The adversary plays with 600 A-MCTS visits. Initially the curriculum for each adversary $\texttt{a}_n$ consisted of intermediate checkpoints from $\texttt{v}_n$'s training run before advancing to $\texttt{v}_n$ with doubling visit counts. We manually selected intermediate checkpoints by looking at the win rates of $\texttt{v}_n$'s intermediate checkpoints against $\texttt{a}_{n-1}$ and sampling checkpoints with varied win rates. Later we found that the victim $\texttt{v}_n$ at 1 visit was always vulnerable to attack, so we simplified the curriculum by no longer using intermediate checkpoints and instead started the curriculum at the final $\texttt{v}_n$ checkpoint with very low visit counts.

The final $\texttt{a}_n$ model we select from a training run is always the latest model checkpoint since win rate increases fairly consistently with more training. We did not have a consistent stopping criterion for each iteration, but we generally targeted either a high win rate at a particular number of visits or restricted the run to a rough training step budget. Unlike in defense training, because the adversary training is longer and the training window is smaller, the data from the previous iteration fully exits the training window (within 132 million training steps) in every iteration.

E.2.1 Attack strategies

All of the adversaries $\texttt{a}_n$ exploit a cyclic group, but there are still some qualitative differences. In particular, $\texttt{a}_1$ emphasizes a small alive group inside the victim's cyclic group and coaxes the victim to form a cyclic group with an eye, in contrast to the original cyclic attack of Wang et al. [50]. $\texttt{a}_2$ creates a very large group inside the victim's cyclic group. In the middle iterations $\texttt{a}_4$ to $\texttt{a}_6$, the inside group is small. In most attacks, the adversary sets up the inside group early and allows the victim stake out territory around it. However, in $\texttt{a}_4$ to $\texttt{a}_6$, the adversary instead stakes out its own territory with the destined inside group on the edge. It then allows the victim come into its territory, resulting in it separating off the inside group and forming the cycle.

Meanwhile, in iterations 7 through 8, the adversary forms an inside group with kos. For $\texttt{a}_7$ and $\texttt{a}_8$ there are between 1 and 3 kos – in small sample analysis, there were most often 2 or 3 kos for $\texttt{a}_7$ and 1 for $\texttt{a}_8$, with more variable inside group shape. With $\texttt{a}_9$, it initially converged to 2 kos and a highly consistent inside group shape, but then abandoned the kos and started making a diamond, "ponnuki"-like inside shape, which the victim surrounds with a square shape. This results in a small, nearly minimal inside group at the time of the final capture. An example of this is shown in Fig. 3.2c.

In Appendix K, we plot heatmaps of the inside and cyclic group locations. Paralleling the qualitative analysis above, we observe differences in where they are concentrated and their sizes. We also notice some variations in victim stone concentration. Overall, we find clear but constrained evolution in the attacks. To humans, the differences do not change the difficulty of gameplay—the attacks all fit very well in the same overall type (cyclic attacks) so knowing how to beat one would almost certainly mean knowing how to beat them all. But to the KataGo victims, the representations learned do not appear to generalize smoothly between these variations.

E.2.2 Attack per-iteration

In this section we discuss each individual iteration in more detail. We provide the training cost (in training steps and V100 GPU-days) of each iteration in Table E.1. Additionally, we discuss any configuration changes or notable results that occurred in the iterations below. We denote an intermediate checkpoint S million training steps into the n-th iteration of defense training as $\texttt{v}_n\texttt{-sSm}$.

Attack iteration 1: The curriculum consisted of $\texttt{v}_1\texttt{-s4m}$ with 128 visits, $\texttt{v}_1\texttt{-s16m}$ with 32–128 visits, and $\texttt{v}_1$ with 32–1024 visits, with a win rate threshold of 75%. We stopped the run due to hitting a large number of victim visits, which slowed the generation of training data.

In this iteration, we made an error when warm-starting from the original cyclic adversary. When we copied the original cyclic adversary's training history, timestamps were erased. Therefore, at the start of the run, the training window contained the original cyclic adversary's After 122 of 238 V100 GPU-days (41 of 150 million training steps), all these data left the window. The most likely effect of this error was hindering early training progress, though it is also possible that it inadvertently helped encourage exploration in early training.

Attack iteration 2: The curriculum consisted of $\texttt{v}_2\texttt{-s4m}$ with 128 visits, $\texttt{v}_2\texttt{-s5m}$ with 64–128 visits, and $\texttt{v}_2$ with 64–1024 visits, with a win rate threshold of 75%. Once again we stopped the run due to hitting a large number of victim visits.

Attack iteration 3: The curriculum consisted of $\texttt{v}_3\texttt{-s5m}$ with 128 visits and $\texttt{v}_3$ with 64–256 visits. Here, we stopped at a 75% training win rate against 256 victim visits because we had used about as much compute as in previous attack iterations and considered 256 visits to be a sufficiently large number of visits that the attack likely transfers, at least somewhat, to high visits.

Attack iteration 4: The curriculum consisted of $\texttt{v}_4\texttt{-s5m}$ with 128 visits and $\texttt{v}_4$ with 4–256 visits. We aimed for a training win rate of 75% against $\texttt{v}_4$ at 256 visits but halted early as we found training progress to be much slower than in previous iterations.

Initially, the curriculum jumped from $\texttt{v}_4\texttt{-s5m}^{128\text{ visits}}$ to $\texttt{v}_4{}^{64\text{ visits}}$ but the win rate against $\texttt{v}_4{}^{64\text{ visits}}$ was very low at 3%, and it did not appear to be trending upwards. After this, we reverted the curriculum back to $\texttt{v}_4\texttt{-s5m}^{128\text{ visits}}$ and enabled LCB move selection, which remained enabled in all subsequent attack iterations as well. The result of enabling LCB move selection was a lower training win rate, presumably due to the victim playing more strongly.

The hope was that training against the earlier checkpoint $\texttt{v}_4\texttt{-s5m}$ for longer would yield stronger performance once the curriculum advanced to $\texttt{v}_4$, but we still had a low win rate of 5% when we reached $\texttt{v}_4{}^{64\text{ visits}}$.

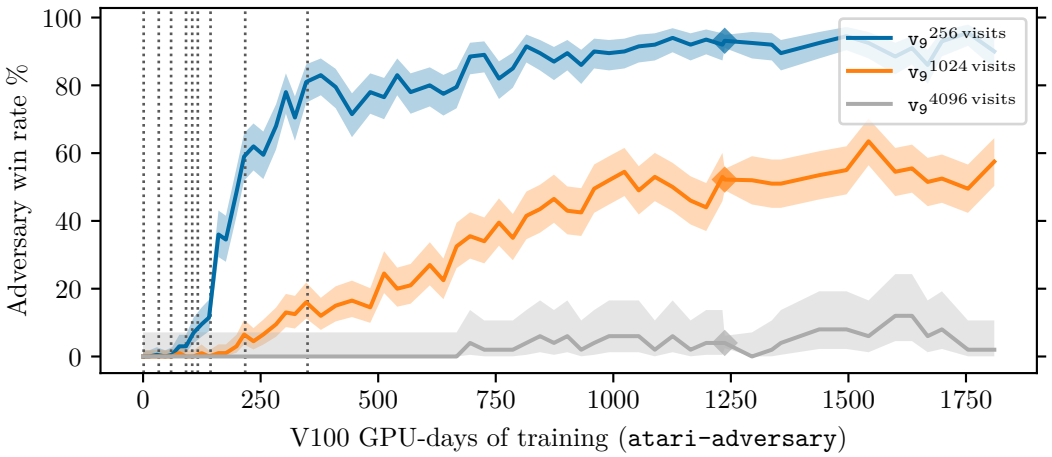


Figure E.3: Win rate of `atari-adversary` (♦) against $\texttt{v}_9$ throughout `atari-adversary` training, warm-starting (at $x = 0$) from `base-adv-early`. Curriculum changes are denoted by a vertical dotted line.

We then changed the curriculum to reduce the starting visit count for $\texttt{v}_4$ from 64 to 4, which worked better. We suspect that we could have skipped the $\texttt{v}_4$`-s5m` intermediate checkpoint entirely and immediately initialized the curriculum at $\texttt{v}_4$ with 4 visits, saving 420 V100 GPU-days and 360 million steps of training. We therefore stopped using intermediate curricula checkpoints in subsequent attack iterations and switched to starting the curriculum against the target victim $\texttt{v}_n$ at very low visits.

Attack iteration 5: The curriculum consisted of $\texttt{v}_5$ with 4–32 visits. We halted this run because progress was slow, and it did not look like we would reach higher victim visits within our compute budget.

Attack iteration 6: From this point onward, we did not anticipate reaching high victim visits during training, so we decided to run each attack iteration for around 250 million training steps, although it is not entirely clear that running defense iterations against weak adversaries provides useful training signal on the defense side. We trained $\texttt{a}_6$ against a curriculum of $\texttt{v}_6$ with 4–16 visits.

Attack iteration 7: We trained $\texttt{a}_7$ against a curriculum of $\texttt{v}_7$ with 4–64 visits. We initially intended to train $\texttt{a}_7$ for only 200 million steps, but the win rate in both training and evaluation started increasing noticeably faster at 190 million steps (272 V100 GPU-days), so we extended the training duration.

Attack iteration 8: We trained $\texttt{a}_8$ with a curriculum of $\texttt{v}_8$ with 4–16 visits.

Attack iteration 9: We trained $\texttt{a}_9$ against a curriculum of $\texttt{v}_9$ with 4–512 visits, raising the win rate threshold from 75% to 90% at 256 visits. At 82 GPU-days (74 million steps), we adjusted the victim configuration parameters to match evaluation settings as described in Appendix C.2 due to a large gap between training and evaluation win rates, and because a higher training win rate was not leading to stronger evaluation strength. We also updated adversary configuration parameters as described in Appendix C.2.

No defense iteration trains against $\texttt{a}_9$. We trained $\texttt{a}_9$ to see whether it could successfully attack $\texttt{v}_9$.

E.3 Validation attack

In Section 4.2.2, we found that the final iterated adversarially trained victim $\texttt{v}_9$ can be readily exploited at low visits by the validation attack `atari-adversary`. This attacker

was warm-started from `base-adv-early`. We used a curriculum of $\texttt{v}_9$ starting at 1 victim visits and doubling until it reached 512 visits (curriculum changes denoted by dotted lines in Fig. E.3). The curriculum win rate threshold was 75% until reaching 256 visits, at which point the threshold increased to 90%. We modified the bot configurations as described in Appendix C.2 to make the victim configuration closer to evaluation settings and the adversary configuration closer to the latest KataGo training runs.

Although `atari-adversary` beats $\texttt{v}_9$ at low visit counts, we did not find an attack that achieved a high win rate against $\texttt{v}_9$ at high visit counts. However, `atari-adversary` was only trained for 6% as much compute as $\texttt{v}_9$, raising the question: how well would the attack perform were we to continue this training run? More generally, can we predict how much more compute it would take to scale `atari-adversary` to achieve, say, a 10% win rate against $\texttt{v}_9$ at 65,536 visits?

Unfortunately, training dynamics are hard to forecast in advance. For instance, `base-adversary` made little progress for a few hundred GPU-days before abruptly finding a strategy that generalized to attack `base-victim` at high visits [14, 50]. Looking at the training progress for `atari-adversary` (Fig. E.3), `atari-adversary` makes fairly consistent progress until it stalls out late in the training run. This training curve is consistent both with it plateauing and never achieving high win rates against high victim visits, or it suddenly hitting a phase change in training like `base-adversary` did and shooting up.

Therefore even with the slowing progress of `atari-adversary` late in its training and its inability to win against $\texttt{v}_9$ at 8192 visits, we cannot conclude that $\texttt{v}_9$ is invulnerable at high visits. Indeed, $\texttt{a}_9$ achieves a 42% win rate against $\texttt{v}_9$ at 65536 visits, showing that there is available attack surface at high visits. This is despite $\texttt{a}_9$ only training against $\texttt{v}_9$ up to 512 visits.

This brings up another question: how can we encourage adversary training to find strategies that are likely to generalize against high visits? `continuous-adversary`, `gift-adversary`, $\texttt{a}_9$, and `atari-adversary` only trained against their victims up to 512 visits. Yet `continuous-adversary` and $\texttt{a}_9$ generalize to high visits, whereas `gift-adversary` and `atari-adversary` do not. Likewise, Wang et al. [50] found one strategy that generalizes to high visits (`base-adversary`) and one that does not (their "pass-adversary").

One hypothesis is that `continuous-adversary` and $\texttt{a}_9$ simply used more training compute. Another is that training is highly path dependent, so initialization, training curriculum, or randomness matter—`continuous-adversary` and $\texttt{a}_9$ were initialized from later adversary checkpoints and had a curriculum involving intermediate victim model checkpoints, whereas `gift-adversary` and `atari-adversary` were initialized from `base-adv-early` and had coarser curricula not involving intermediate victim models. `continuous-adversary` and $\texttt{a}_9$ were initialized from checkpoints `attack-may23` and `base-adversary` that already work against some strong high-visit victim, which may be important in biasing them away from discovering another vastly different fragile strategy that only works at low visits.

An additional point of evidence for training being path dependent is that warm-starting the adversary from `base-victim` failed to beat `base-victim`. In particular, the training win rate remained flat for 50 million training steps. Although we cannot rule out that the adversary would have eventually improved, we observed clear training progress before this point in other adversary training runs.

F Vision transformers

In this appendix, we provide an overview of our vision transformer (ViT) architecture and describe our ViT training procedure. For full architectural details, see our PyTorch implementation: `https://github.com/AlignmentResearch/KataGo-custom/blob/stable/python/model_pytorch.py`.

F.1 ViT inputs

Our ViTs take in the same inputs as standard KataGo CNNs, namely two tensors of spatial and global features.

The spatial features are represented by a three-dimensional binary tensor $\mathbf{S}$ taking values in $\{0,1\}^{\texttt{height}\times\texttt{width}\times 22}$, where `height` and `width` are the maximum Go board dimensions the model supports (usually 19). In other words, each point of the Go board has 22 binary features associated with it. These features encode various properties such as whether a point is occupied, the color of a stone on a point, move history, and more complicated features like whether a point is involved in a potential ladder. For an exact specification of these features, see this source file: `KataGo/cpp/neuralnet/nninputs.cpp`.

The global features are represented by a real-valued vector $\mathbf{G}$ taking values in $\mathbb{R}^{19}$. These 19 features encode properties like which of the past 5 moves were passes, and the particular ruleset the current game is using.

F.2 ViT architecture

Our ViT network replaces the KataGo CNN backbone with a transformer-based backbone, but reuses the same output layers as KataGo's networks (Figure F.1). Both our transformer backbone and the KataGo's CNN backbone output a real-valued tensor with dimensions $\texttt{height} \times \texttt{width} \times c$, where c is the embedding / residual-stream dimension of the network. This embedding tensor is processed by the standard KataGo output layers to produce the outputs KataGo expects networks to have: a scalar that estimates the value function, a vector that represents the next-move policy, etc. Our transformer backbone is built using a standard HuggingFace `transformers.ViTModel`.

Input preprocessing We zero-pad the spatial dimensions of $\mathbf{S}$ so that they are divisible by our ViT patch size (`patch_size=2`). We then expand $\mathbf{G}$ so it has shape $\texttt{padded\_width} \times \texttt{padded\_height} \times 19$ and concatenate it with $\mathbf{S}$ to form the actual input to our `ViTModel` of shape $\texttt{padded\_width} \times \texttt{padded\_height} \times 41$.

Unembedding The Huggingface `ViTModel` outputs a tensor of shape $\texttt{n\_patches} \times c$. We linearly project this tensor to one of size $\texttt{height} \times \texttt{width} \times c$ in the canonical way that preserves spatial locality.

Architecture hyperparameters We tried a few different ViT architecture hyperparameters and measured how quickly they trained with supervised learning on training data from `katagotraining.org`. We found the following hyperparameters to work fairly well:

Patch size	# Attn. heads	Embedding dim.	MLP dim.
2	6	384	1536

We trained networks of varying depths, ranging from a 4-layer ViT to a 16-layer ViT. See Appendix section F.4 for more details.

F.3 ViT implementation

KataGo implements its architectures in Python for training and C++ for self-play. Because implementing models in C++ is fairly complex, we only implement the ViT in Python using PyTorch. To use ViTs during inference, we export the PyTorch model as a TorchScript model and modify KataGo's C++ code to be able to invoke TorchScript models. Since

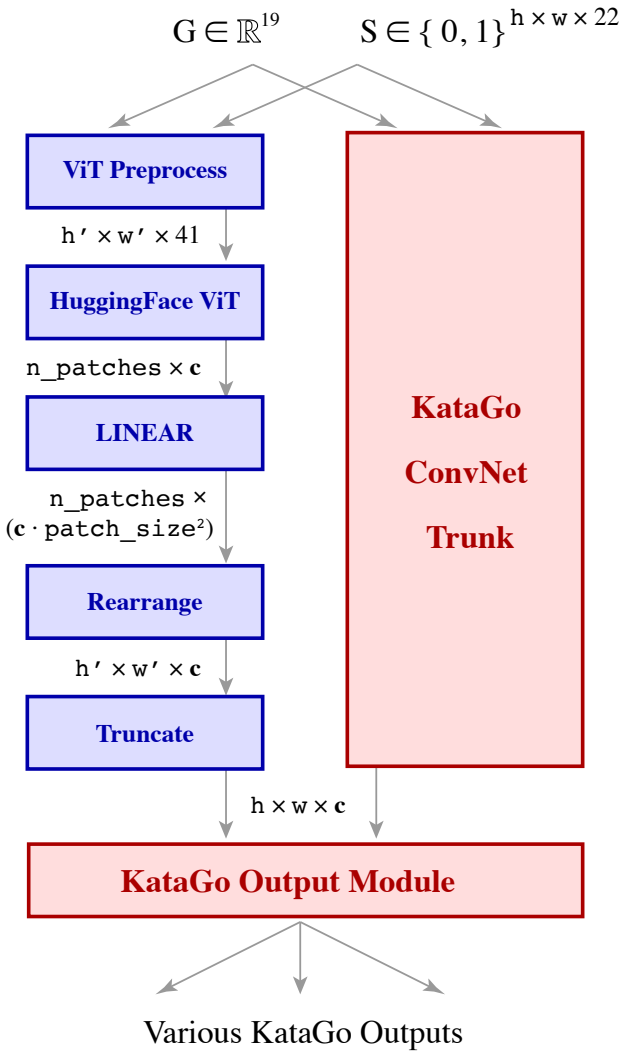


Figure F.1: A diagram comparing our ViT architecture to the standard KataGo CNN architecture. Our ViT architecture replaces the KataGo **CNN backbone** with a **transformer backbone**, and reuses the KataGo CNN output layers. Boxes denote neural network components and unboxed quantities denote tensor shapes.

TorchScript models are made to be serialized and executed independently of Python, this removes the need to implement the ViT itself in C++.

However, this comes at the cost of slower inference. On our machines, running inference for a `b18c384nbt` or `b20c256` KataGo CNN model using TorchScript incurred a 43% and 28% slowdown, respectively, compared to running them with KataGo's C++ CUDA implementation.

F.4 Self-play training

In this section we describe our self-play training process for our ViT agent `ViT-victim`.

F.4.1 Network scaling

In our ViT training run, we start with a small 4-block ViT network that is quick to generate data. When the smaller model hits capacity, we switched to an 8-block network, and then

Name	Model type	# Layers	Embedding dim.	# Parameters
`ViT-b4`	ViT	4	384	7,952,501
`ViT-b8`	ViT	8	384	15,050,357
`ViT-b16`	ViT	16	384	29,246,069
`b6c96`	CNN	6	96	1,001,613
`b10c128`	CNN	10	128	2,959,329
`b15c192`	CNN	15	192	9,875,893
`b20c256`	CNN	20	256	23,413,525
`b40c256`	CNN	40	256	46,632,501
`b18c384nbt`	CNN	18	384	26,389,941

Table F.1: Comparison of our ViT nets to KataGo CNNs in terms of depth, width, and parameter count.

finally switch to a 16-block network. We perform each switch by first pre-training the larger model on the smaller model's data, using the typical sliding training window method described in Appendix C.1 to sample training data, but using the smaller model's existing self-play games as data. We copy the smaller model's data into the pre-training dataset gradually in chronological order, copying roughly one epoch's worth of data after each pre-training epoch, so pre-training ends by training on the latest and presumably strongest data. We discard some of the smaller model's early data under the assumption that training on it would take extra time without much benefit due to the data being lower quality. We still increment N in Eq. (1) to keep the window size as large as it would have been had we not discarded the data.

F.4.2 Training configuration

Our configuration parameters matched those suggested by KataGo's example self-play configurations available in its codebase, except that we only used Tromp-Taylor rules instead of having rule variation, did not play any games on rectangular boards, and increased the percentage of 19x19 games to 53.6% to match the latest KataGo training runs. We trained exclusively on Tromp-Taylor rules because we always evaluate models under these rules, and our adversaries like `base-adversary` were all trained only under Tromp-Taylor rules.

KataGo also seeds 14% of its self-play games from custom positions that are rarely encountered in typical self-play [57]. This improves play on tricky positions like Mi Yuting's Flying Dagger joseki and improves analysis on human games [58]. Since we do not have access to this set of games, we do not include it in our ViT training run.

F.4.3 ViT training run

Figure F.2 shows the strength of our networks throughout self-play training. We successively trained three ViT networks (4-block, 8-block and 16-block) along with a control 10-block CNN. Larger ViT networks reached a higher Elo but quickly saturated. However, the ViT networks did not necessarily reach model capacity, as we were able to reach still higher Elo ratings by distilling KataGo CNN self-play training games into the ViT network.

We started with training a 4-block ViT with 600 visits using a configuration matching an example KataGo configuration, similar to the actual configuration used for training KataGo's 6-block and early 10-block networks.[13] We trained for 64 GPU-days and 213 million steps.

We then switched to a 8-block ViT, pre-training it on the 4-block ViT's latest 24.9 million data rows (100 million training steps). After pre-training (2 V100 GPU-days, 92 million steps), the 8-block ViT was about 175 Elo stronger than the 4-block ViT at 256 visits. We then began self-play with the 8-block ViT. After 20 V100 GPU-days and 48 million steps of self-play, we increased the number of self-play visits to 1000 visits by swapping our

[13]KataGo example configuration: `https://github.com/lightvector/KataGo/blob/7488c47b6f6952f9703d9209f9afbd8d38a8afb5/cpp/configs/training/selfplay1.cfg`

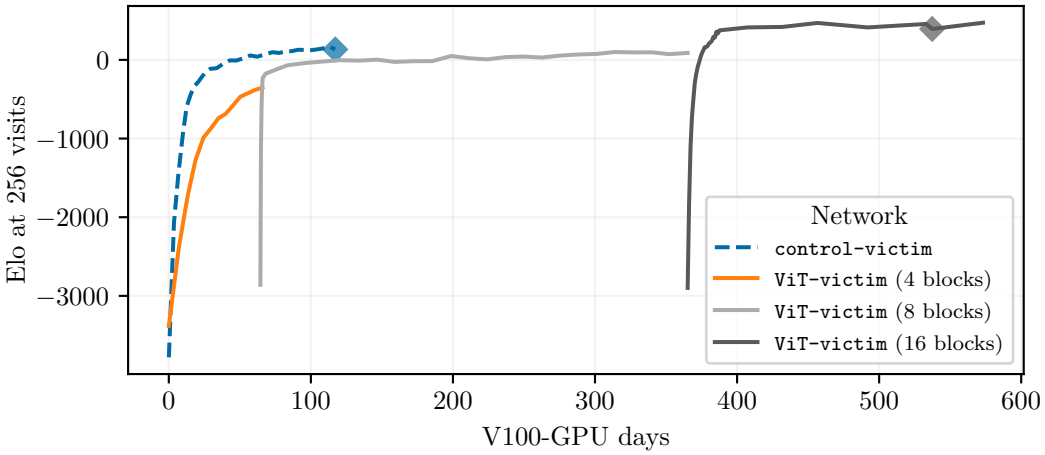


Figure F.2: The strength of our ViTs (`ViT-victim` = black ♦) throughout their training as well as a control 10-block CNN `control-victim` (blue ♦) trained with the same settings. Playing strength was estimated by playing the models against each other as well as against a few KataGo networks.

configuration to make it similar to that used for training KataGo's 10-block and 15-block nets.[14] At 461 million steps with self-play, we gained about 264 Elo at 256 visits. At this point we spent 301 V100 GPU-days on training the 8-block ViT. It was still making slow training progress, but we decided to switch to a larger architecture in hopes of achieving a faster increase in playing strength.

When we switched to a 16-block ViT, we pre-trained on the latest 24.9 million data rows generated by the 4-block ViT as well as all the data rows generated by the 8-block ViT, totaling 139 million data rows. For this pre-training, we used data-parallel training on 8 GPUs to decrease wall-clock training time. After training had reached 78% of the pre-training data, we noticed signs of overfitting: playing strength decreased, and the ViT's value loss (loss on the model's prediction of whether a position will lead to a win or a loss) was decreasing on training data yet increasing on validation data. We mitigated this by increasing the minimum window size m from 250000 to 10 million in Eq. (1), which roughly quadrupled the current window size. After training on 93% of the data, we reduced the learning rate by a factor of 2 since the loss plateaued.

After pre-training (22 V100 GPU-days, 532 million steps), the 16-block ViT was 286 Elo stronger than the 8-block ViT at 256 visits. We then started self-play. After 75 V100 GPU-days and 74 million steps of self-play, we increased the visits to 2000 and matched a configuration similar to that used for training KataGo's b18 models.[15] At 126 V100 GPU-days and 104 million steps of self-play, we reduced the learning rate by another factor of 2. We stopped self-play at 118 million steps of self-play, at which point we had spent 172 V100 GPU-days on 16-block training and gained another 18 Elo at 256 visits. The resulting model is `ViT-victim`. We likely could have trained the ViT for longer—strength was still increasing, albeit slowly. Moreover, when we trained a separate 16-block ViT on training data from `katagotraining.org` generated by KataGo's stronger CNN networks, the resulting model was an estimated 277 Elo stronger than `ViT-victim` at 300 visits, suggesting there is still capacity in `ViT-victim`'s architecture.

F.4.4 Control CNN training run

As a control run, we train a model `control-victim` with a 10-block CNN architecture (`b10c128` in Table F.1). In total, we train it for 121 V100 GPU-days and 419 million steps.

[14] KataGo example configuration: `https://github.com/lightvector/KataGo/blob/7488c47b6f6952f9703d9209f9afbd8d38a8afb5/cpp/configs/training/selfplay8b.cfg`

[15] KataGo example configuration: `https://github.com/lightvector/KataGo/blob/7488c47b6f6952f9703d9209f9afbd8d38a8afb5/cpp/configs/training/selfplay8mainb18.cfg`

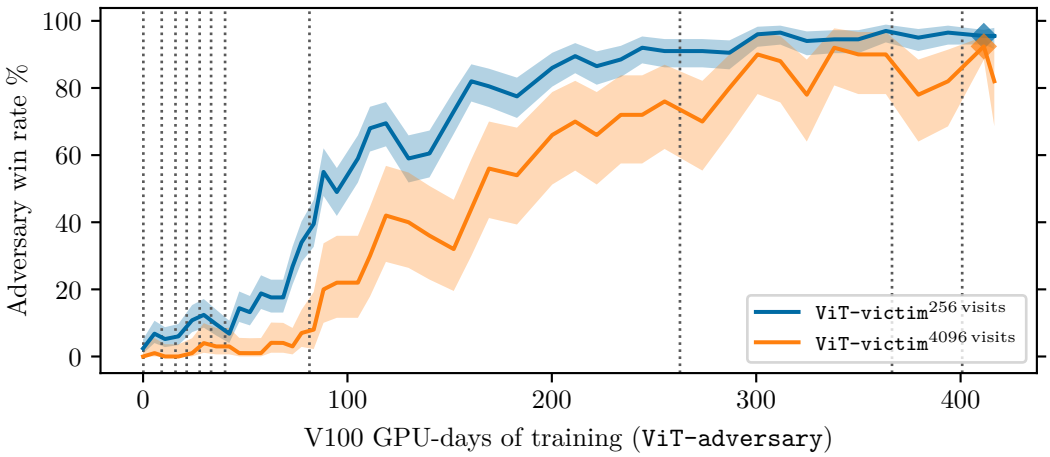


Figure F.3: Win rate (%) of `ViT-adversary` (⧫) against our superhuman ViT agent `ViT-victim` throughout `ViT-adversary` training. The zero of the x-axis represents the win rate of the warm-start `base-adversary` against `ViT-victim` before the fine-tuning against `ViT-victim` began. Dotted lines represent victim visit increases.

We started with the same 600-visit configuration used by the 4-block ViT, and at 29 V100 GPU-days (147 million steps), we switched to the 1000-visit configuration used by the 8-block ViT. At 35 GPU-days and 64 GPU-days (166 million and 251 million steps), we cut the learning rate in half, and at 110 GPU-days (396 million steps), we reduced the learning rate by 40%. By the end of the training run, the model was about as strong as the 8-block ViT.

In Fig. F.2, we see that `control-victim` learned quicker than the 4-block ViT and plateaued at about the same strength as the 8-block ViT, despite the 8-block ViT having five times as many parameters.

F.5 Training `ViT-adversary`

Figure F.3 shows the win rate of `ViT-adversary` against `ViT-victim` throughout `ViT-adversary`'s training. We trained the adversary for 409 V100 GPU-days and 328 million steps, stopping the run once we had high win rates against `ViT-victim` at 32768 visits, which we estimate to be just shy of superhuman (Appendix G). We fine-tuned `ViT-adversary` from `base-adversary` after observing that `base-adversary` is able to win against the final ViT at low victim visits. We used a curriculum of `ViT-victim` starting with 1 visit and doubling until 2048 visits. The curriculum win rate threshold was 75% until the curriculum reached 256 visits, after which the threshold was increased to 90%.

At 262 V100 GPU-days (206 million time steps) the curriculum reached 1024 victim visits. However, we noticed that the training win rate was higher than the evaluation win rate by about 14%, and also that the drop in win rate when the curriculum moved on to a higher visit victim was small. We considered it desirable to train for longer at lower victim visits since it would be cheaper to generate training data and high win rates at low visits were likely to translate to high win rates at high visits. We therefore changed the configuration parameters to bring the victim closer to evaluation settings as described in Appendix C.2. This reduced training win rate, so we rewound the curriculum from 1024 visits to 256 visits. With more training, the curriculum eventually reached 1024 visits again.

F.6 ViT vulnerability throughout training

Figures F.4 and F.5 show the vulnerability of `ViT-victim` to `base-adversary` and `ViT-adversary`. We observe that vulnerability to both adversaries develops early in training and shows no sign of decreasing. Figures F.6 and F.7 show the same results for the control CNN model `control-victim`.

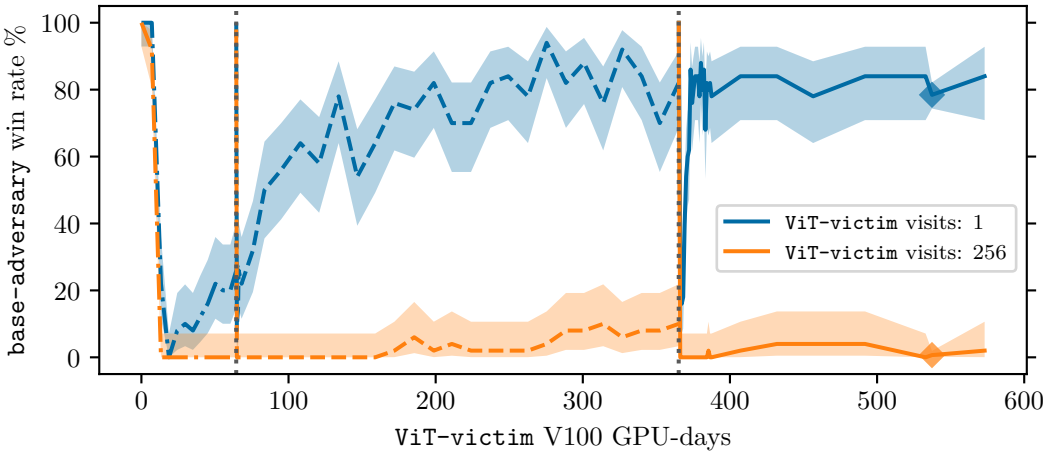


Figure F.4: Vulnerability of `ViT-victim` to `base-adversary` throughout `ViT-victim` training. A dotted gray line represents switching to a larger ViT architecture, at which point the vulnerability drops as the larger architecture is initialized randomly but then quickly rises during pre-training.

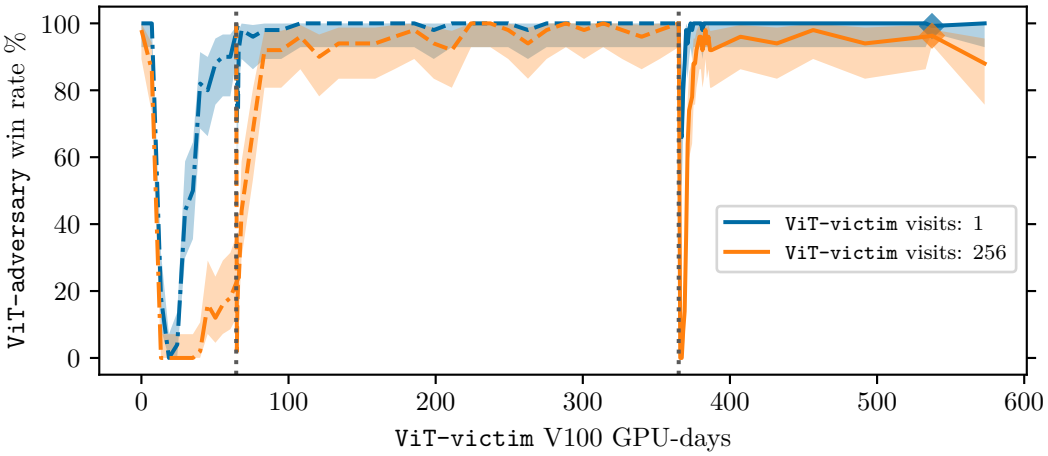


Figure F.5: Vulnerability of `ViT-victim` to `ViT-adversary` throughout `ViT-victim` training. A dotted gray line represents switching to a larger ViT architecture, at which point the vulnerability drops as the larger architecture is initialized randomly but then quickly rises during pre-training.

Figure F.8 shows the win rate of `control-victim` against `ViT-adversary` and `base-adversary` at varying amounts of `control-victim` visits (the corresponding plot for `ViT-victim` is Fig. 3.1). We see that `control-victim` is also highly vulnerable to `ViT-adversary`, indicating that `ViT-adversary` is not conducting an architecture-specific attack.

In Figs. F.9 and F.10, we plot how the playing strength of `ViT-victim` and `control-victim` throughout training compares to their vulnerability to `base-adversary`. More training yields greater strength but also increased vulnerability. `control-victim` develops vulnerability to `base-adversary` at a weaker strength than `ViT-victim`, suggesting that ViTs may be marginally more robust than CNNs against cyclic attacks at a given strength.

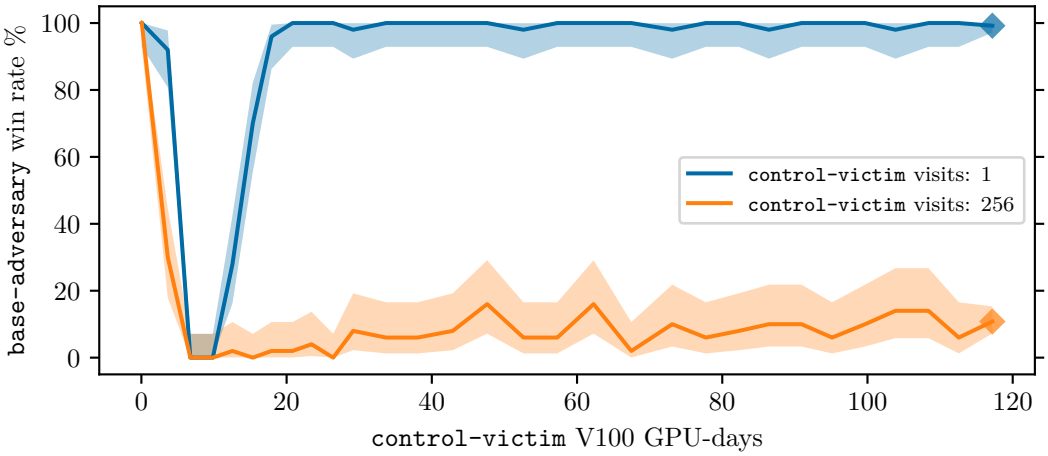


Figure F.6: Vulnerability of `control-victim` to `base-adversary` throughout `control-victim` training.

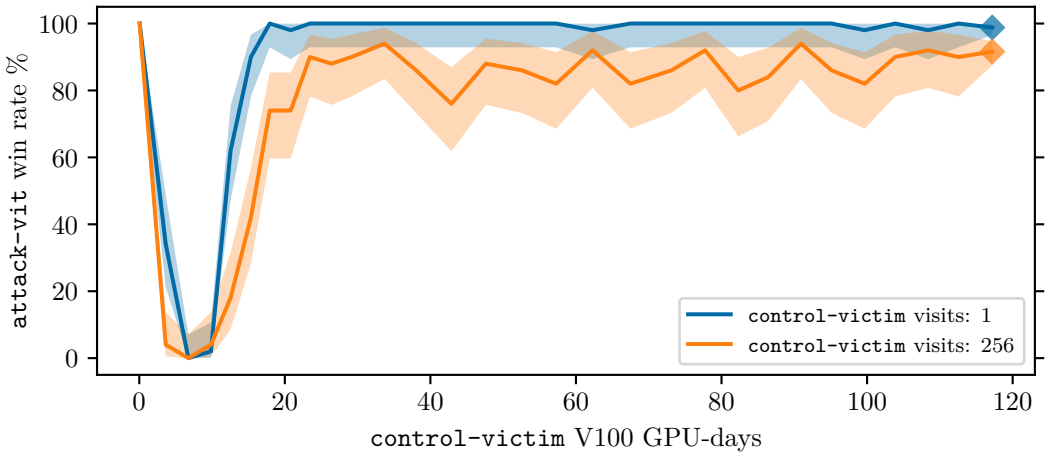


Figure F.7: Vulnerability of `control-victim` to `ViT-adversary` throughout `control-victim` training.

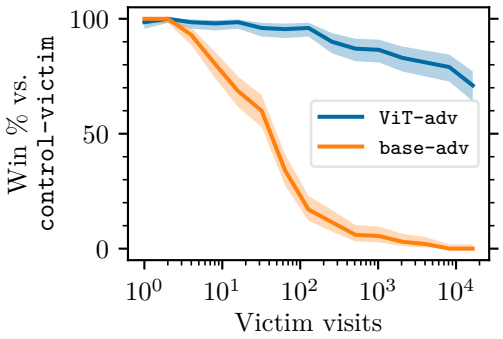


Figure F.8: Win rate (%) for `control-victim` against `ViT-adversary` and `base-adversary`, with varying victim visits.

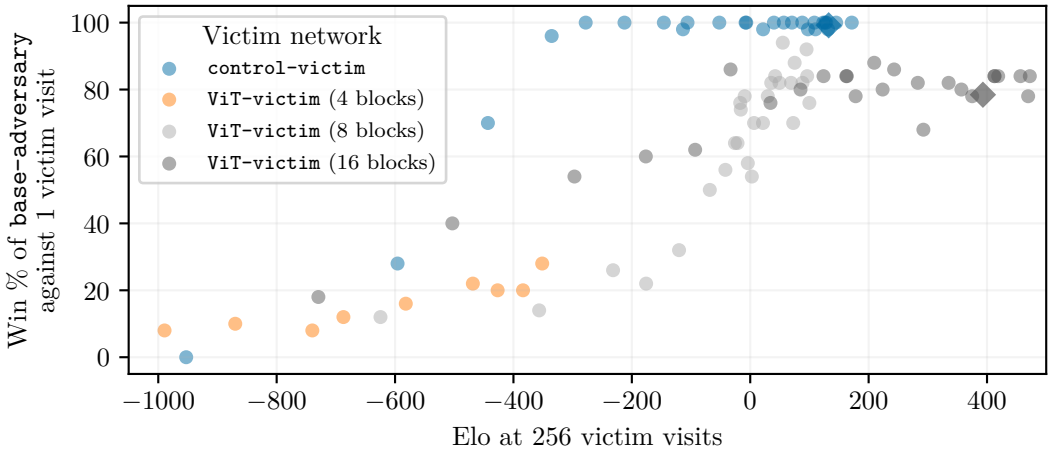


Figure F.9: Plot of several `ViT-victim` (black ♦) and `control-victim` (blue ♦) training checkpoints with their playing strength on the x-axis and their vulnerability to `base-adversary` at 1 victim visit on the y-axis.

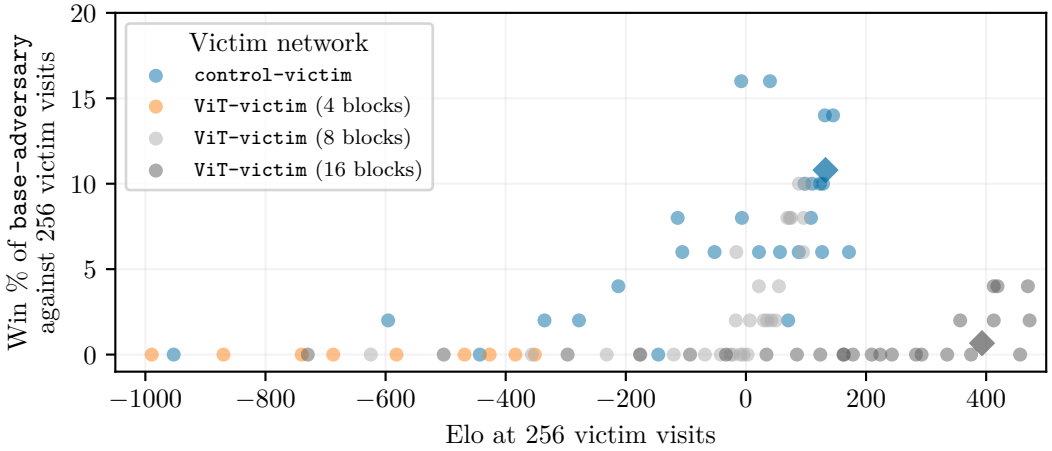


Figure F.10: Same as Fig. F.9 but with vulnerability at 256 victim visits on the y-axis.

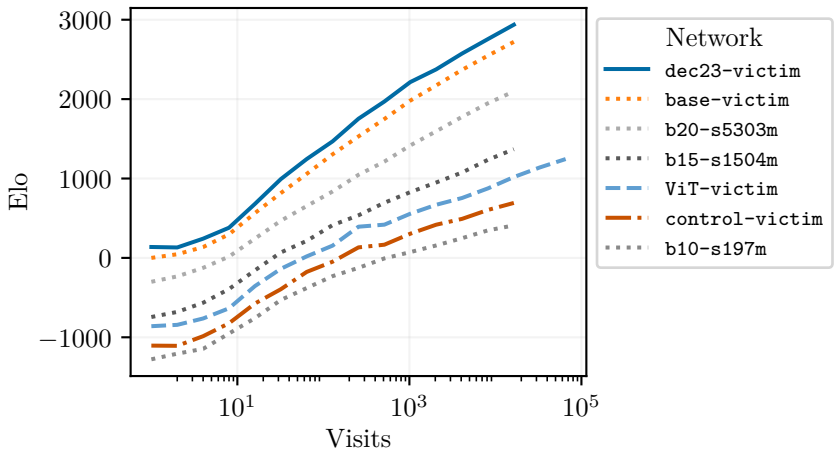


Figure G.1: Elo strength of networks (different colored lines) by visit count (x-axis). The four dotted lines are KataGo networks.

G Network strength

G.1 Performance of defenses vs base KataGo networks

We estimate the strength of the defended victims at playing regular Go games by pitting them against regular KataGo networks. We find that the defended victims `ViT-victim`, `dec23-victim` and `v9` all possess superhuman Go capabilities.

We evaluate `dec23-victim` (positional adversarial training; Section 3) and `ViT-victim` (vision transformer; Section 5) by playing games against several KataGo networks at varying visit counts and then running a Bayesian Elo estimation algorithm. We plot the results in Fig. G.1. The KataGo networks we use are `b10-s197m`, `b15-s1504m`, `b20-s5303m`, and `base-victim`, which Wang et al. [50] refer to `cp79`, `Original`, `cp127`, and `Latest` respectively.

We estimate that `ViT-victim` at 32768 visits is 1139 Elo stronger than `base-victim` at 1 visit. Using Wang et al. [50]'s estimate that `base-victim` at 1 visit would have an Elo of 2738 on `goratings.org`, `ViT-victim` at 32768 visits has an estimated Elo of 3877. This is just shy of superhuman, as the strongest historical Elo rating on `goratings.org` is 3877 at the time of writing (as of 2024-05-02). At 65536 visits, `ViT-victim` has an estimated Elo of 3983, which is superhuman.

Likewise, `dec23-victim` at 64 visits is 1245 Elo stronger than `base-victim` at 1 visit, giving it a superhuman estimated Elo of 3983.

We estimate the strength of `v9` by playing against `base-victim` at varying visit counts. We plot the results in Fig. G.2. We estimate that `v9` has an Elo of 4997 at 4096 visits, which is 110 points weaker than `base-victim` but still clearly superhuman.

G.2 Performance of `ViT-victim` against human players

We also deployed a 64-thread, 65536 visit / move version of `ViT-victim` on the KGS Online Go server [19]. From the previous section, we estimate this bot has a `goratings.org` Elo of 3855, around the level of a top human professional.[16] Our results support this: our

[16] In the previous section we estimated a `goratings.org` Elo of 3983 for `ViT-victim` at 65536 visits. However, KataGo's internal benchmarks suggest that above 5000 visits, each search thread decreases performance by around 2 Elo (see `https://github.com/lightvector/KataGo/blob/v1.13.0/cpp/program/playutils.cpp#L868`). Adjusting for this gives an Elo of $3983 - 2 * 64 = 3855$. Using multiple search threads parallelizes inference, decreasing inference latency at the cost of overall strength. That is to say, for a fixed number of visits, using fewer threads generally leads to a stronger agent. All of our training and evaluation runs in the paper are done with a single search thread unless noted otherwise.

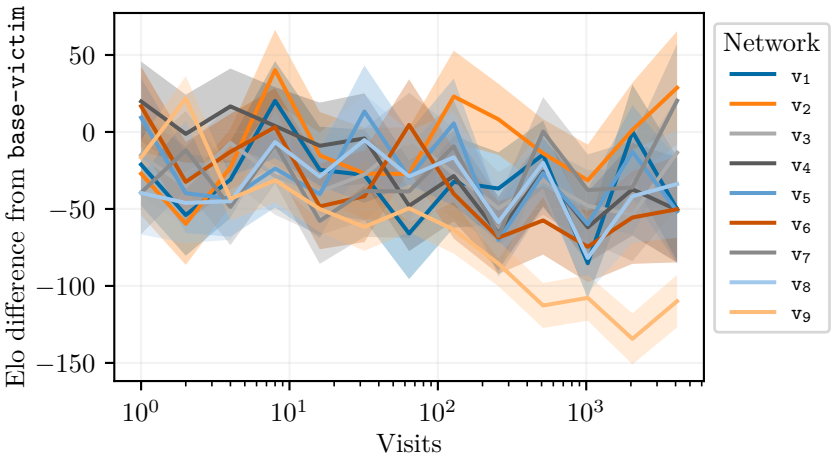


Figure G.2: Elo difference between each v_n to `base-victim` at visit counts up to 4096. Shaded regions are the standard deviation of the Elo estimate. Each v_n is slightly weaker than `base-victim`.

`ViT-victim` bot achieved a peak ranking of 9th on KGS [19], ahead of many KataGo bots but behind others playing with stronger settings. Since professional players rarely play on KGS, we also commissioned three games against strong professionals: our bot won two out of three, losing one largely due to a weakness that also affected early versions of KataGo (see discussion below).

G.2.1 Public KGS games

Our bot played 1000 ranked games on the KGS website with members of the public, achieving a peak rating of 10.98 dan on the KGS website [19].[17] We note that bots are common on the server; we follow the standard best practice of notifying players that they are playing a bot, and our bot was approved for ranked games by the KGS administrators. Indeed, the top ranked players on KGS are dominated by bots: our ranking of 9th puts us ahead of several KataGo bots and behind several others, though the exact configuration settings of these bots are unknown. However, the majority of ranked games played by our `ViT-victim` bot were against human players, usually with our bot giving 1 to 6 stones of handicap to the human player.

Despite our `ViT-victim` bot having a strong showing on the KGS Online Go server, it is understood within the Go community that strong professional players rarely play on KGS. Thus our results on KGS only show that a 64-thread, 65536 visit / move version of `ViT-victim` is much stronger than many strong amateur Go players.

G.2.2 Games against professional Go players

We therefore also commissioned a game against the 7 dan professional Yilun Yang and two games against the 4 dan professional Ryan Li. The players were informed that they were playing a bot and agreed to acknowledgement in the paper. They were also compensated at a rate greater than 4x the minimum wage in the relevant jurisdiction.

Yilun Yang played with 90 minutes base time for each player, and 5 periods of 30 seconds byo-yomi overtime. `ViT-victim` won, with Yang feeling he may have gotten behind early and missed some better ways to play in the middle game.

[17] The KGS website has a special rating system (`www.gokgs.com/help/rmath.html`). Official ranks are discrete and only go up to 9 dan, but KGS computes an internal Elo for all players with a minimum number of ranked games. These internal Elos can go past 9 dan. See the top rated accounts at `www.gokgs.com/top100.jsp` to see some examples of this.

Ryan Li played with 5 minutes base time per player, and the same 5x30 byo-yomi overtime. `ViT-victim` lost the first game and won the second. In the first game, Li played the "Flying Dagger" joseki, a notoriously difficult opening corner sequence, and obtained a substantial early advantage after `ViT-victim` misplayed. Li played accurately for the rest of the game and `ViT-victim` never caught up. This joseki was a known weakness in early versions of KataGo as well; it was eventually corrected through manually adding positions from the sequence to the training run. In our training, we did not include those positions (Appendix F.4.2), and it seems `ViT-victim` developed a similar weakness.

In the second game Li played, we requested and Li agreed to avoid that joseki, and with that constraint `ViT-victim` won.

Overall, these results indicate `ViT-victim` has some weaknesses that might lead to a lower Elo. But in general it plays at a strong professional level, in line with our original estimate. Explore the games on the accompanying project website.

H Human replication of attacks

A Go expert author (Kellin Pelrine) was also able to replicate several of our attacks after studying the game records but without AI assistance at attack time. Full game records, along with additional commentary on the play, are available on our website and linked in the following sections.

H.1 Human replication of the continuous adversary

This attack was the most challenging to replicate, requiring multiple components chained together. In addition to carefully engineering the shapes of the attack, a key discovery was that the final step approaching the capture seems to require obfuscation. That is, the attack failed many times after seemingly achieving the salient features of the cyclic group like the distinctive double cut formation highlighted in Fig. 3.2a. To succeed, it appears that the final threat against the cyclic group needs to be a natural move for a purpose other than attacking the cyclic group. This, we hypothesize, leads `dec23-victim` to be less likely to search follow-up sequences attacking its cyclic group, and consequently miss the danger that it is in.

The successful attack was performed against `dec23-victim` playing with 512 visits. Although the final obfuscation is likely to become more challenging against higher visits, we believe it should still be possible for humans to achieve, as it is possible to engineer situations where the final threat has a very large threat against something besides the cyclic group and appears very natural. For example, the critical move in the successful game is also (mis-)played by KataGo with 4096 visits. Meanwhile, the other components of the attack do not seem related to search depth and should not be harder to achieve. We plan to test human attacks against higher visits in future work.

H.2 Human replication of the gift adversary

Unlike the preceding attack, setting up the apparent shapes for this attack is relatively straightforward. It was not too challenging to produce a successful attack against 1 visit. In particular, it was quite simple to induce `dec23-victim` to make errors—the challenge was ensuring `dec23-victim`'s lead was sufficiently narrow for the errors to change the game outcome.

Scaling to higher visits, however, proved difficult. Multiple attempts at 256 and 512 visits failed. We hypothesize that this is because the victim must assign enough value to the sending-two "gift" move to play it, but at the same time not keep searching locally and see disaster coming after the adversary's next move. These requirements are conflicting: if there are valuable areas to play elsewhere then the victim is likely to play those instead of sending-two, but if there are none, then there are none for the adversary either, so the victim is more likely to expect the adversary to continue locally and accept the gift – and then to see the danger.

This need for some but not too much local search so that the victim plays the local "gift" move is in stark contrast with all versions of the cyclic attack, where the attack is more likely to succeed the less search is allocated by the victim to the locality of the vulnerability. Furthermore, at least in the versions of the attack observed so far, the number of moves that the victim needs to look ahead locally, between its deciding move and realized loss (adversary group living), is fixed and small. This again contrasts with the cyclic attack, where the deciding move can take place a virtually arbitrary amount of moves ahead of realized loss (cyclic group captured or something else lost while saving the cyclic group).

This requirement means the attack needs to balance search probabilities over the entire board to a greater and greater degree at higher visits. By contrast, in the cyclic attack it suffices to control the local situation to make the attack more hidden, requiring a greater victim search depth needed to notice the attack. This also fits with our empirical observations: humans can perform the attack at one visit but seemingly not at 256+, while `gift-adversary` can reach 512 visits and somewhat beyond but falls off very sharply after 1024 visits (Fig. 3.1).

`gift-adversary` is likely able to balance search probabilities of the victim with much higher precision than humans can, but it becomes prohibitively difficult at high enough visits.

H.3 Human cyclic attack on `ViT-victim`

Pelrine was also able to use a cyclic attack to beat `ViT-victim`. This attack was the easiest to execute of those discussed in this section. It was performed against a 64-thread, 65536 visit / move version of `ViT-victim`, the same used in the strength evaluation in Appendix G.2. The shape used for the inside group paralleled some of the wins by `base-adversary` against this victim. The attack emphasized ensuring lots of liberties for the groups surrounding the cyclic one so that `ViT-victim` would have to see the danger early to have a way out.

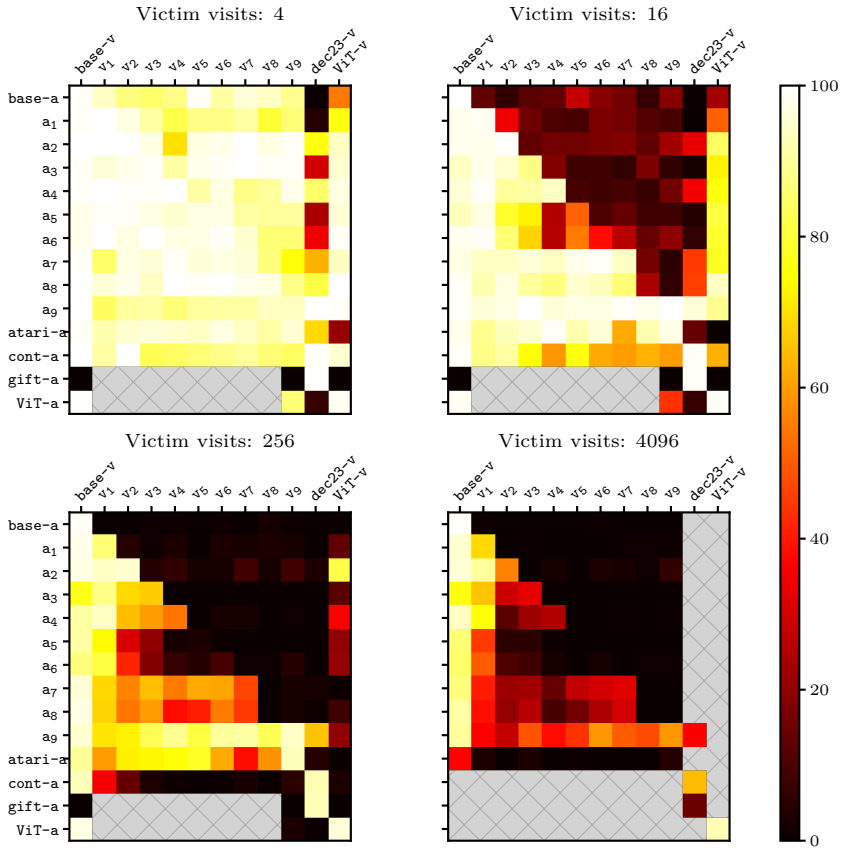


Figure I.1: We extend Fig. 4.1's plot of adversaries' win rates against various victims to include more adversaries on the y-axis and more victims on the x-axis.

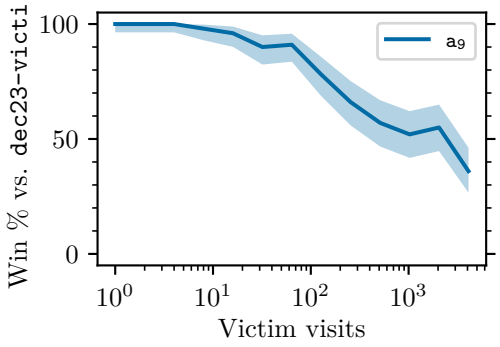


Figure I.2: Win rate (y-axis) of `a9` versus `dec23-victim` at varying victim visits (x-axis), demonstrating considerable transfer performance.

I Transfer

Figure I.1 shows the result of playing adversaries against a variety of victims. The ability of victims to defeat adversaries they were not trained against provides evidence of their robustness.

Victims: We find all victims remain vulnerable at extremely low amounts of search (4 victim visits), although `dec23-victim` does better than others. `base-victim` through

v_4 progressively improve at defending against `continuous-adversary`, after which their performance plateaus.

Adversaries: a_9, trained against v_9, transfers surprisingly well to defeat `dec23-victim`, winning 66% of games at 256 visits and 36% at 4096 visits (Fig. I.2). `atari-adversary`, trained against v_9, wins 4% of games against `dec23-victim` at 256 victim visits. By contrast, `continuous-adversary`, trained against `dec23-victim`, wins 5% of games against v_9. `gift-adversary` does not transfer at all to other victims, achieving no wins even at 4 visits against `base-victim` and `ViT-victim`.

J Compute resources

J.1 Compute infrastructure

We ran experiments using cloud computing infrastructure orchestrated with Kubernetes configured with the Kueue batch scheduler. We used A6000 GPUs for nearly all our training runs. The main exception is that $\texttt{v}_1$, $\texttt{v}_2$, $\texttt{v}_3$, and $\texttt{v}_4$ used A100 80GB GPUs as we were trying a different compute platform. We also used some H100 GPUs during the `gift-adversary` and $\texttt{a}_9$ runs, but they were mainly run on A6000 GPUs.

J.2 Compute for our training runs

We convert our compute numbers to V100 GPU-days so that our numbers can be straightforwardly compared to the V100-based compute estimates of Wang et al. [50]. According to Wang et al.'s conversion estimates, one A100 80GB GPU-day is 1.873 A6000 GPU-days and one A6000 GPU-day is 1.704 V100 GPU-days. We estimate that one H100 GPU-day generated as much training data as 0.369 A6000 GPU-days. Note we did not tune our H100 setup as we made minimal use of these GPUs.

Most of our compute estimates are measured by parsing our training logs. However, when training $\texttt{a}_1$, ViT, and `ViT-adversary`, we made sub-optimal configuration choices that slowed down our training runs. For these runs, we provide idealized compute estimates by benchmarking the slow-down caused by the poor configuration and scaling our compute estimates downwards accordingly.

Our error in $\texttt{a}_1$ training was using too few game threads, a parameter controlling how many victim-play games are played at once. We were using 16–32 game threads rather than the 128–256 game threads that we used in later training runs, which gave higher training throughput. $\texttt{a}_1$ used 703 V100 GPU-days, and we estimate that with higher game threads it would have cost 238 V100 GPU-days instead.

Our error in training our ViT networks and `ViT-adversary` was using single-precision floating point rather than half-precision floating point for ViT inference. Inference with half-precision floating point is significantly faster. Our actual compute cost for training ViT with single-precision floating point was 128 V100 GPU-days for the 4-block ViT, 661 V100 GPU-days for the 8-block ViT, and 457 V100 GPU-days for the 16-block ViT, totalling 1247 V100 GPU-days. We estimate that with half-precision floating point, the cost would have been 537 V100 GPU-days instead. For `ViT-adversary`, we switched to half-precision floating point near the end of the run and spent 711.0 V100 GPU-days. We estimate that had we used half-precision floating point for the entire training run, it would have been 409 V100 GPU-days instead.

J.3 Compute for KataGo models

`base-victim`, `may23-victim`, and `dec23-victim` all come from KataGo's ongoing distributed training run, which was initialized from KataGo's "third major run." Wu [63] reports training compute estimates for the third major run, from which we can extrapolate the training cost of models from the distributed training run. (Our compute estimate calculations are similar to those of Wang et al. [50], except in our estimates we do not anchor on the initial 38.5 days of the third major run that generated data with smaller models, and we account for the greater search used in the distributed training run.)

In the last 118.5 out of 157 days in the third major run, the run switched from using `b20c256` nets to using `b40c256` and `b30c320` nets for self-play data generation. The final `b20c256` net used for self-play was trained on 468,617,949 data rows whereas the third major run generated 1,229,425,124 rows in total, so over the course of those 118.5 days, the run generated $1{,}229{,}425{,}124 - 468{,}617{,}949 = 760{,}807{,}175$ rows. This segment of the run used 46 V100 GPUs, costing $118.5 \times 46 = 5451$ V100 GPU-days. The total cost of the the third run across all 157 days is 6,730 V100 GPU-days [50].

The distributed run generates data with `b40c256`, `b60c320`, and `b18c384nbt` nets, all of which have similar or higher inference cost to the `b40c256` and `b30c320` used in the third

major run.[18] Therefore, we estimate the average inference from the distributed run is at least as expensive as the average inference from the last 118.5 days of the third major run.

Moreover, the distributed training run uses more inferences to generate each data row. The third major run used 1000 full-search visits or 200 cheap-search visits per move, where full searches are used to generate high-quality policy data and cheap searches are used to play games quickly [61]. The distributed training run started with 1500 full-search visits and 250 cheap-search visits [55]. Assuming inference count scales proportionally with search and that data row compute cost scales proportionally with inference count, we crudely estimate each training row generated with these search parameters costs 1.25× as much as each training row from the third major run. The distributed run switched to 2000 full-search visits and 350 cheap-search visits in March 2023 [53], after about 3.211 billion data rows (including the 1.2 billion from the third major run) were generated. We estimate each row generated with these parameters costs 1.75× as much as each third-major-run row.

Putting this all together, our training compute estimate in V100 GPU-days for a model from KataGo's distributed training run that has trained on $D \geq 1{,}229{,}425{,}124$ rows is

$$6730 + \frac{(\min\{D, 3211000000\} - 1229425124) \cdot 1.25 + \max\{D - 3211000000, 0\} \cdot 1.75}{760807175} \cdot 5451.$$

Since `base-victim` trained on 2,898,845,681 data rows, its estimated cost is 21681 V100 GPU-days. `may23-victim` trained on 3,323,518,127 rows, giving a cost of 25888 V100 GPU-days, and `dec23-victim` trained on 3,929,217,702 rows, giving a cost of 33482 V100 GPU-days.

For adversarial training, the last KataGo network before adversarial training began was trained with 3,057,177,418 data rows [52]. Based solely on the number of data rows, `dec23-victim` has had $(3929217702 - 3057177418)/(3323518127 - 3057177418) = 3.3$ times as much adversarial training as `may23-victim`.

[18] `b60c320` is a strictly larger in width and depth than `b40c256` and therefore has a higher inference cost. `https://github.com/lightvector/KataGo/blob/v1.14.1/python/modelconfigs.py#L1384` states that `b40c256`, `b30c320`, and `b18c384nbt` have similar inference costs.

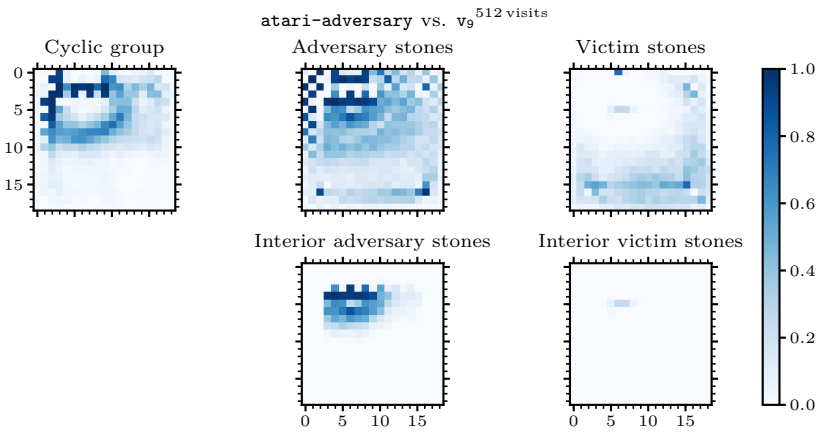


Figure K.1: Heat map showing the cyclic attack made by `atari-adversary` against v_9.

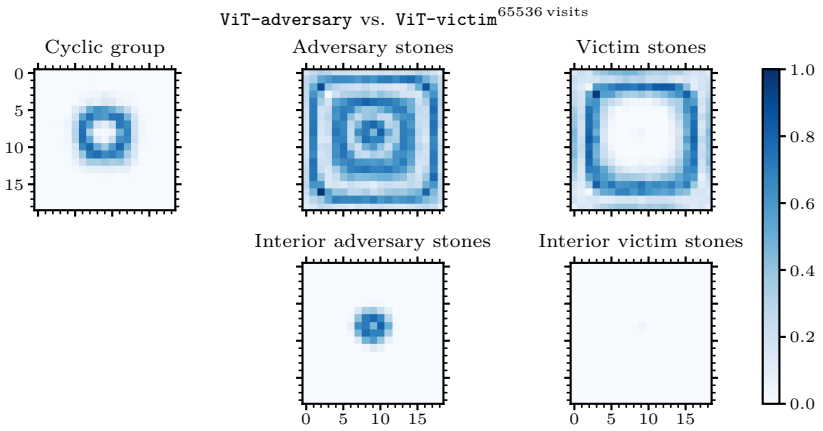


Figure K.2: Heat map showing the cyclic attack made by `ViT-adversary` against `ViT-victim`.

K Heat maps of cyclic attacks

In this section, we present heat maps illustrating the cyclic shapes constructed by each of our cyclic adversaries. We also plot differences between heat maps to show changes in the cyclic group constructed by different adversaries.

To construct the heat maps for an adversary, we took games where the adversary beats the victim it was trained against. We then inspect the board state during the move at which a large cyclic group of victim stones is captured. To remove board symmetries, we rotate the game board so that the center of the cyclic group is in the top-left quadrant of the board, and flip across the major diagonal of the board to keep the center of the group above the major diagonal. We then plot the frequency of each board square being in the captured cyclic group. We also plot the adversary's stones, the victim's other stones, and the adversary and victims' stones falling in the interior of the cyclic group at the time of capture.

Figure K.1 shows heat maps for `atari-adversary` against v_9, with the dark squares in the cyclic group being the bamboo joints discussed in Section 4.2.2. We also see a checkerboard pattern of adversary stones near the cyclic group. These are likely isolated pieces, mentioned in the same discussion, that could be captured if the victim saw the danger its cyclic group was in.

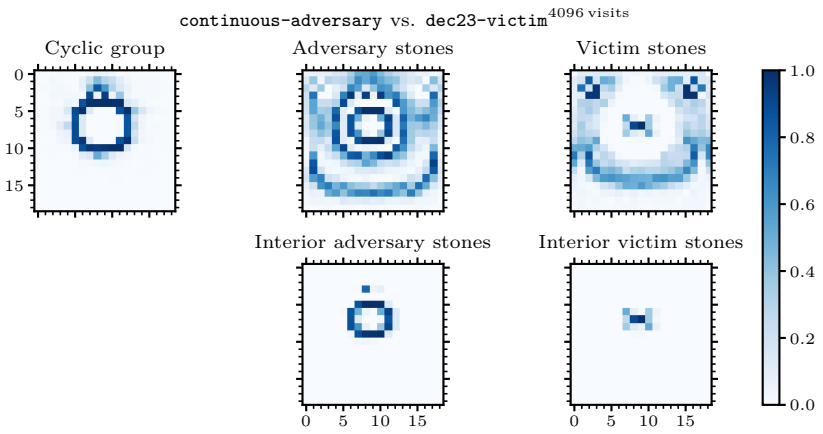


Figure K.3: Heat map showing the cyclic attack made by `continuous-adversary` against `dec23-victim` with 4096 victim visits of search.

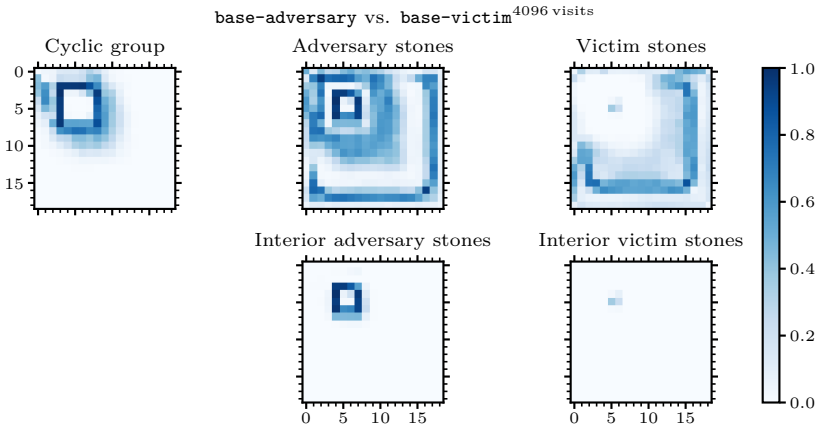


Figure K.4: Heat map showing the cyclic attack made by `base-adversary` against `base-victim` with 4096 victim visits of search.

Figure K.2 shows heat maps for `ViT-adversary` against `ViT-victim`, which moves the cycle into the center and forms another boundary of stones around it.

`continuous-adversary`'s attack against `dec23-victim` (Fig. K.3) shows less variation in the cyclic group than the attack made by `base-adversary` (Fig. K.4): the cyclic stone heat map (Fig. K.3) is deeply colored throughout with few lightly colored squares. We also see a larger and consistent shape of interior adversary stones for `continuous-adversary`, along with a pattern in interior victim stones that isn't present for the `base-adversary`.

For `base-adversary`, using more victim visits does not substantially affect the shape of the cyclic attack. Fig. K.5 plots the difference between the heat map for 4096 (Fig. K.4) and 16 (Fig. K.6) victim visits, finding minimal differences.

Figures K.8 to K.24 show heat maps for each adversary $\texttt{a}_1$ through $\texttt{a}_9$ trained in iterated adversarial training against their corresponding victims at 16 victim visits:

- $\texttt{a}_1$ (Figs. K.7 and K.8) has a less consistent structure to the stones outside the cyclic group than `base-adversary`, which tends to form a boundary of stones near the edge of the board outside the cycle.
- $\texttt{a}_2$ (Figs. K.9 and K.10) forms a larger cycle than $\texttt{a}_1$.

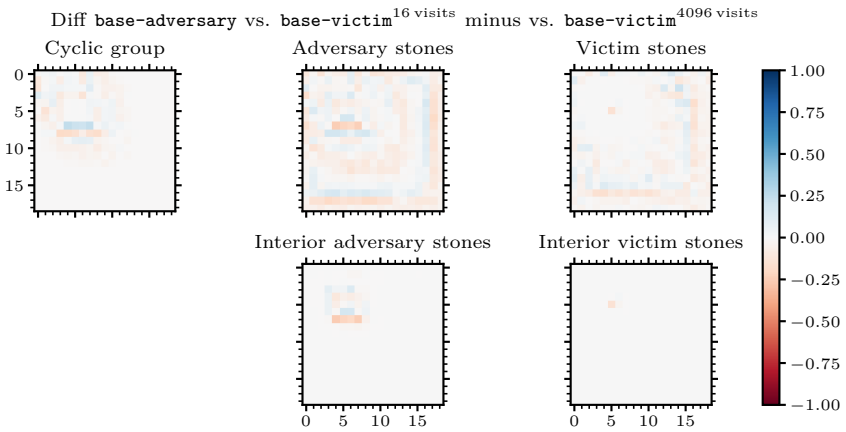


Figure K.5: Difference between the heat maps of `base-adversary` against `base-victim` with 16 (Fig. K.6) and 4096 (Fig. K.4) victim visits of search. `base-adversary`'s attack does not change much when victim visits are increased.

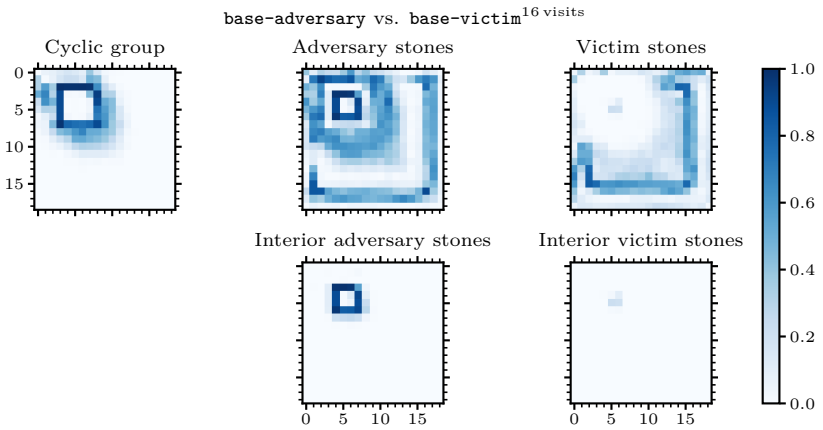


Figure K.6: Heat map showing the cyclic attack made by `base-adversary` against `base-victim` with 16 victim visits of search.

- a_3 (Figs. K.11 and K.12) moves the cycle towards the center on one axis, and the cycle shrinks again but with less consistent shapes.
- a_4 (Figs. K.13 and K.14) moves the cycle towards the center along the other axis.
- a_5 (Figs. K.15 and K.16) makes the cycle larger.
- a_6 (Figs. K.17 and K.18) does not show much qualitative difference in the heat maps.
- a_7 (Figs. K.19 and K.20) tends to place stones on board locations of a particular parity near the boundaries of the board, leading to a checkerboard pattern in the heat map.
- a_8 (Figs. K.21 and K.22) does not show much change.
- a_9 (Figs. K.23 and K.24) shrinks the cycle slightly.

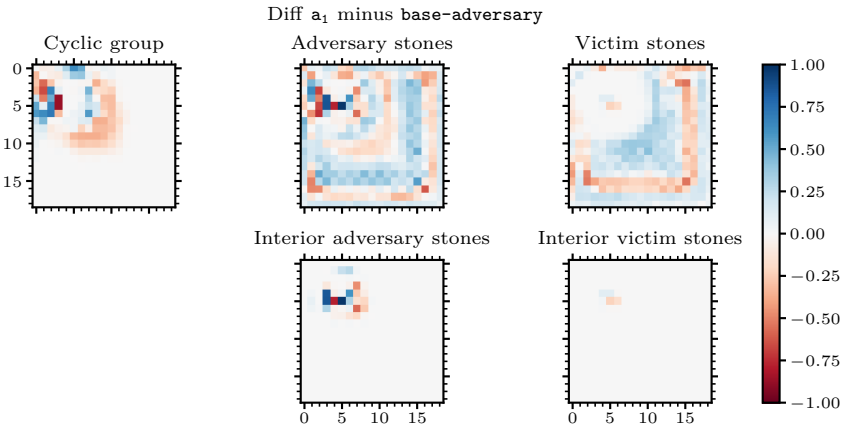


Figure K.7: Difference between the heat maps of $\texttt{a}_1$ (Fig. K.8) and `base-adversary` (Fig. K.6).

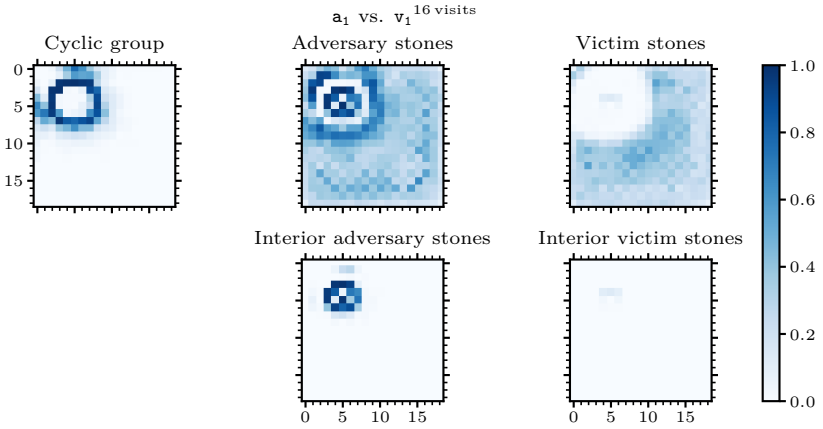


Figure K.8: Heat map showing the cyclic attack made by $\texttt{a}_1$ against $\texttt{v}_1$.

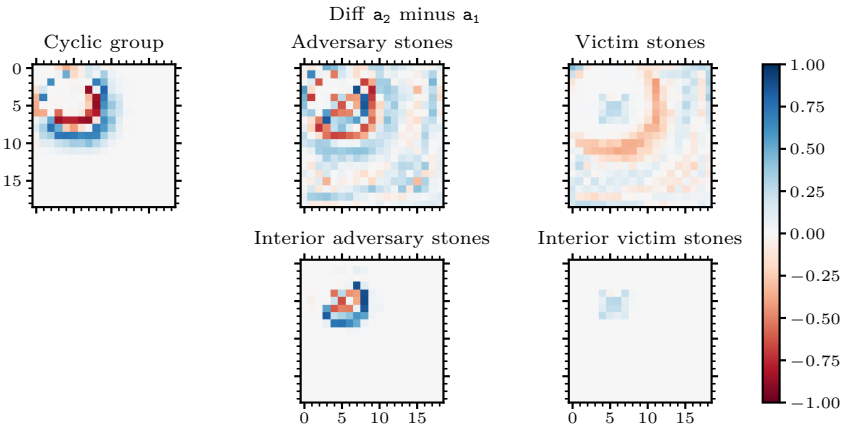


Figure K.9: Difference between the heat maps of $\texttt{a}_2$ (Fig. K.10) and $\texttt{a}_1$ (Fig. K.8).

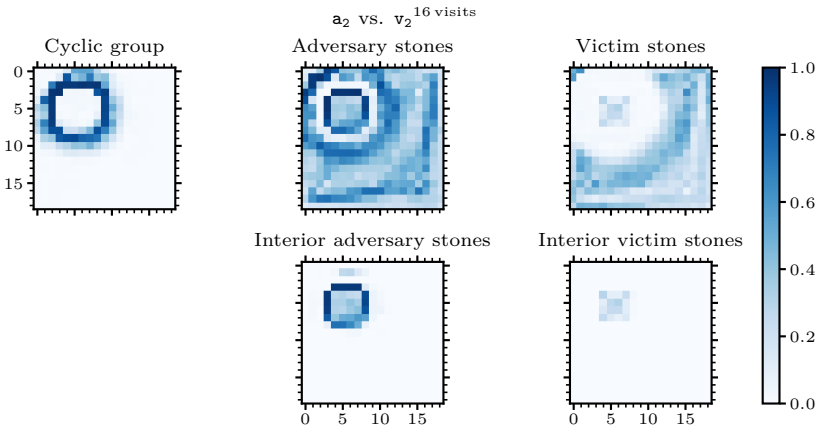


Figure K.10: Heat map showing the cyclic attack made by $\mathtt{a}_2$ against $\mathtt{v}_2$.

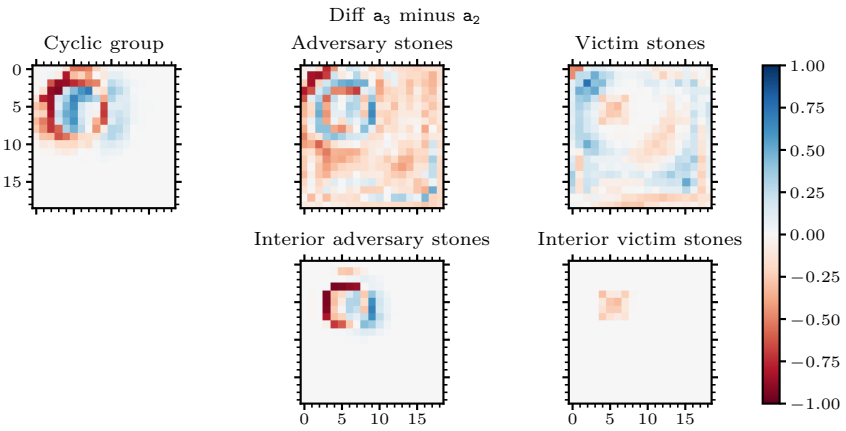


Figure K.11: Difference between the heat maps of $\mathtt{a}_3$ (Fig. K.12) and $\mathtt{a}_2$ (Fig. K.10).

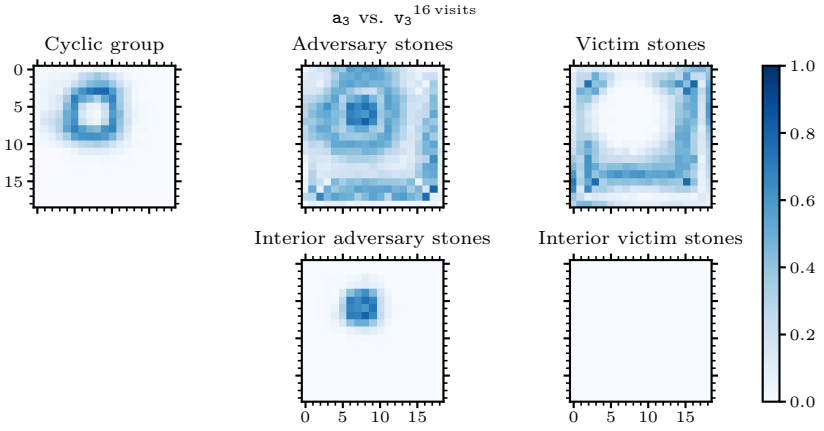


Figure K.12: Heat map showing the cyclic attack made by $\mathtt{a}_3$ against $\mathtt{v}_3$.

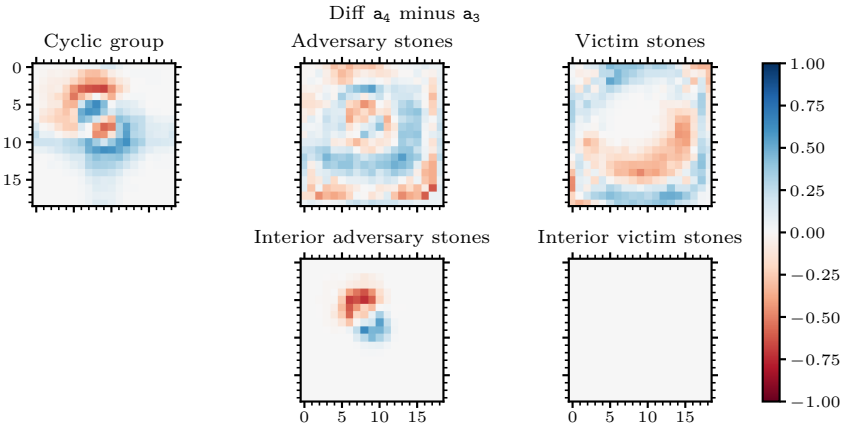


Figure K.13: Difference between the heat maps of $\mathtt{a}_4$ (Fig. K.14) and $\mathtt{a}_3$ (Fig. K.12).

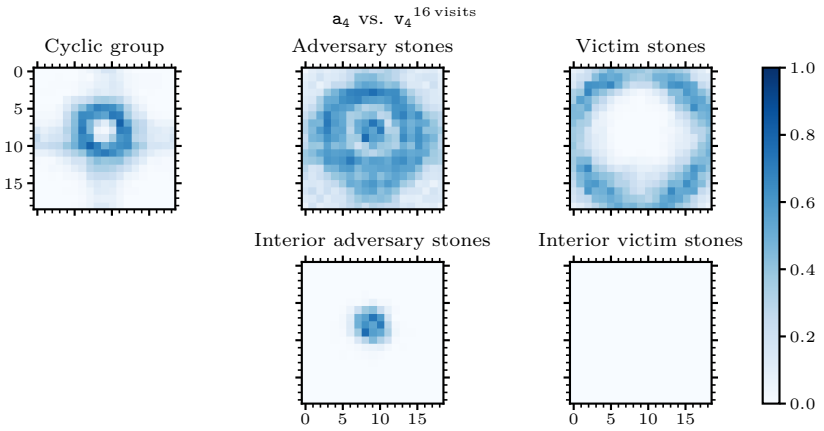


Figure K.14: Heat map showing the cyclic attack made by $\mathtt{a}_4$ against $\mathtt{v}_4$.

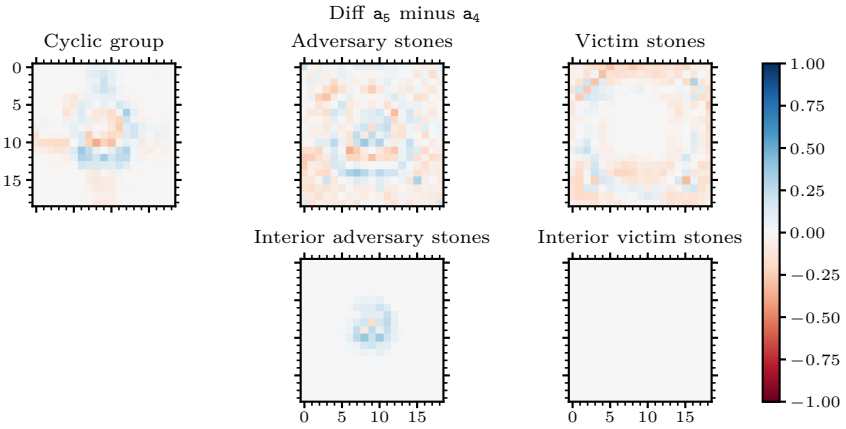


Figure K.15: Difference between the heat maps of $\mathtt{a}_5$ (Fig. K.16) and $\mathtt{a}_4$ (Fig. K.14).

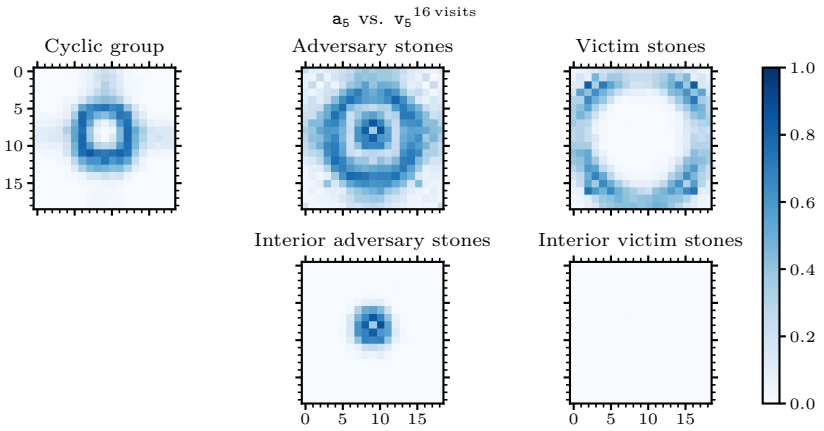


Figure K.16: Heat map showing the cyclic attack made by $\mathtt{a}_5$ against $\mathtt{v}_5$.

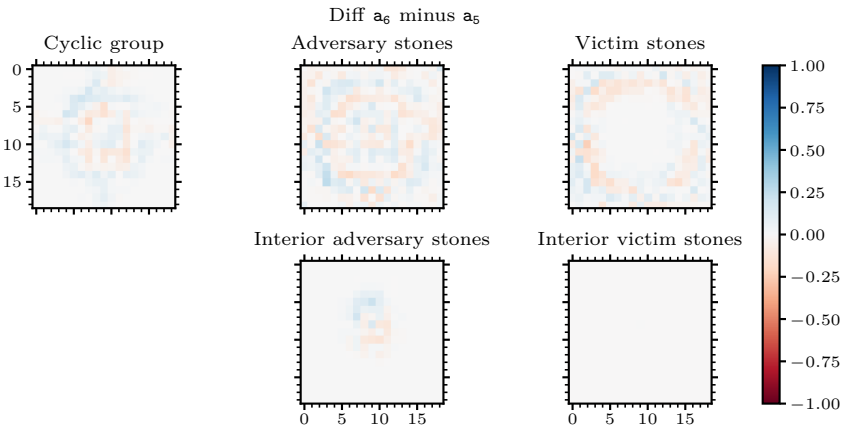


Figure K.17: Difference between the heat maps of $\mathtt{a}_6$ (Fig. K.18) and $\mathtt{a}_5$ (Fig. K.16).

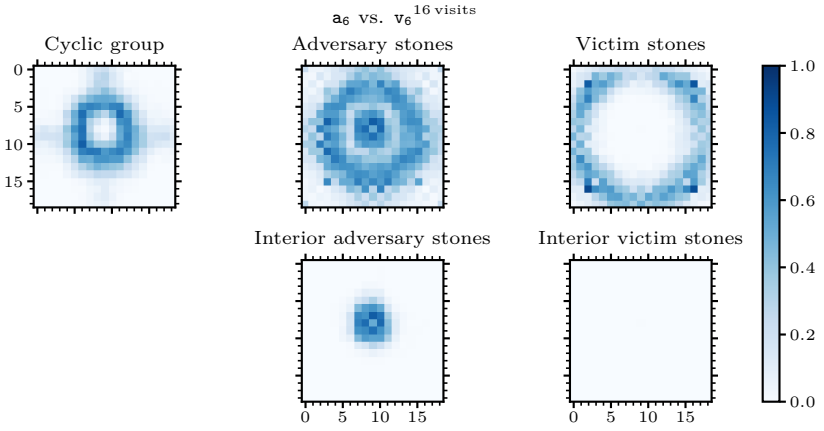


Figure K.18: Heat map showing the cyclic attack made by $\mathtt{a}_6$ against $\mathtt{v}_6$.

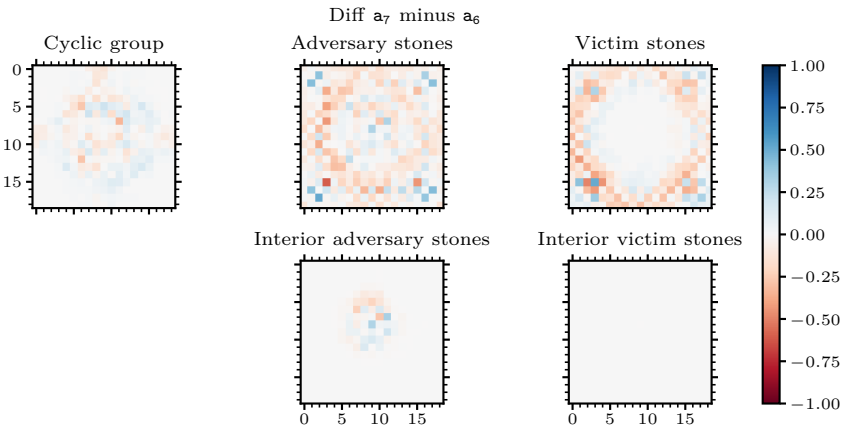


Figure K.19: Difference between the heat maps of $\mathtt{a}_7$ (Fig. K.20) and $\mathtt{a}_6$ (Fig. K.18).

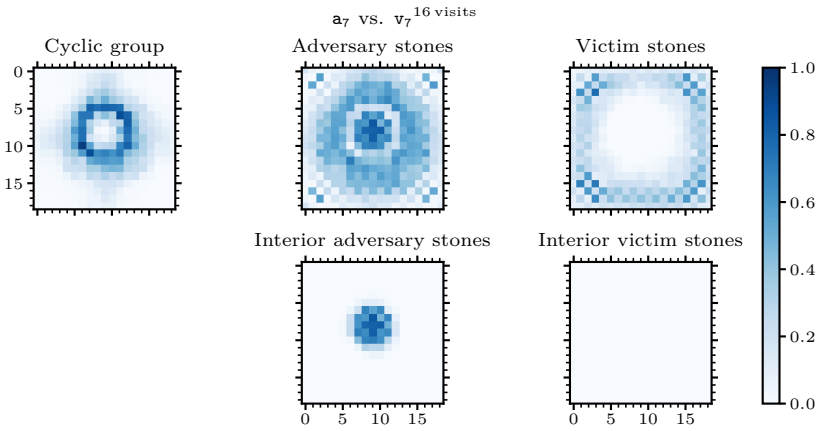


Figure K.20: Heat map showing the cyclic attack made by $\mathtt{a}_7$ against $\mathtt{v}_7$.

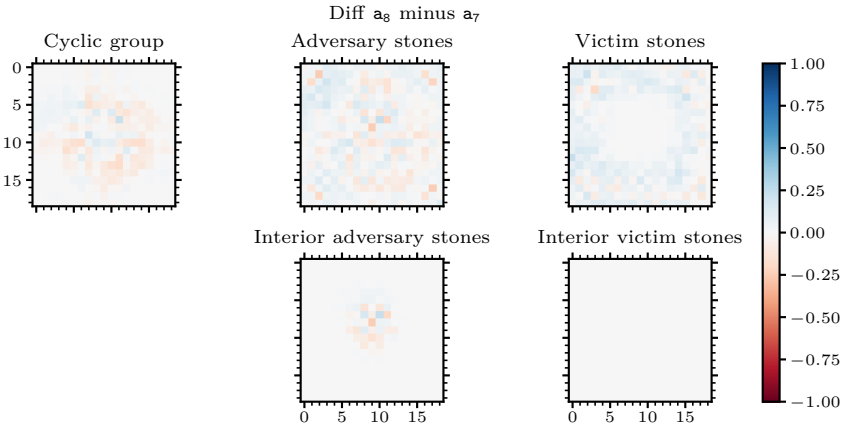


Figure K.21: Difference between the heat maps of $\mathtt{a}_8$ (Fig. K.22) and $\mathtt{a}_7$ (Fig. K.20).

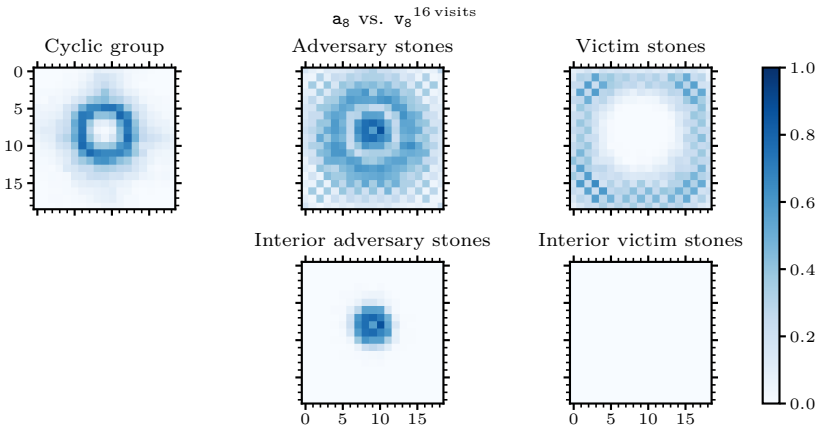


Figure K.22: Heat map showing the cyclic attack made by $\mathtt{a}_8$ against $\mathtt{v}_8$.

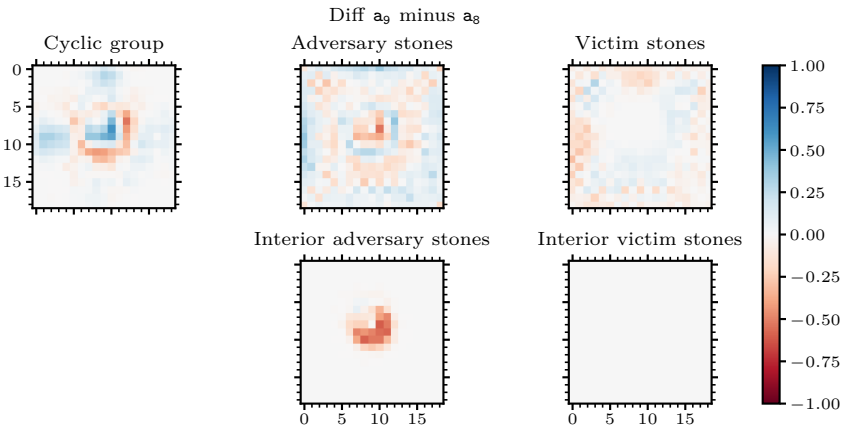


Figure K.23: Difference between the heat maps of $\mathtt{a}_9$ (Fig. K.24) and $\mathtt{a}_8$ (Fig. K.22).

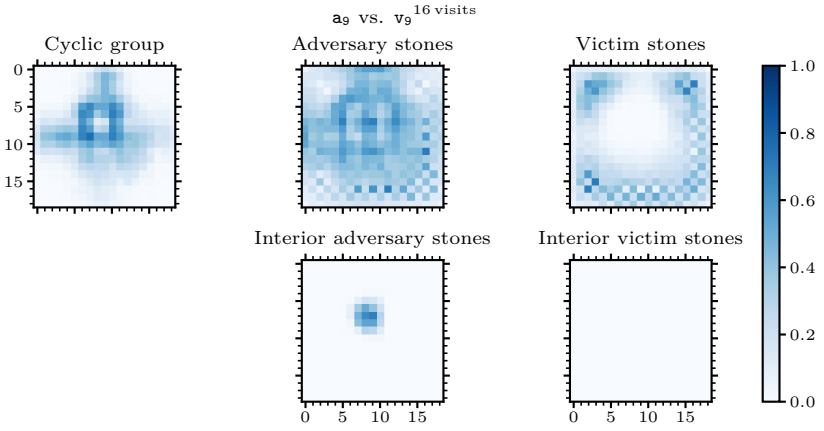


Figure K.24: Heat map showing the cyclic attack made by $\mathtt{a}_9$ against $\mathtt{v}_9$.

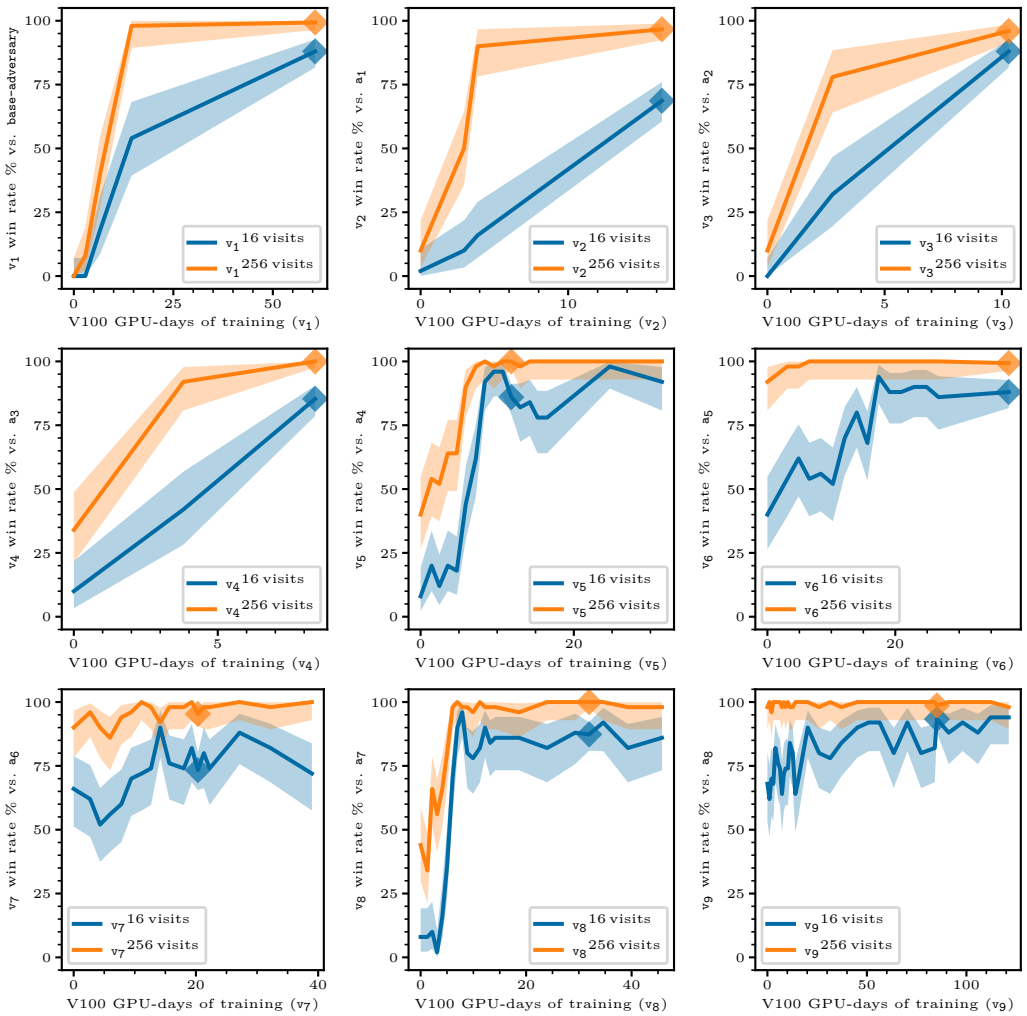


Figure L.1: Win rate (y-axis) of each $\texttt{v}_N$ (♦) against $\texttt{a}_{N-1}$ throughout $\texttt{v}_N$'s training (x-axis). The curves for $\texttt{v}_1$ to $\texttt{v}_4$ only have a few data points along the x-axis as intermediate checkpoints were lost.

L Extra experimental plots

This section collects additional, visually large plots referenced in previous sections.

L.1 Individual iterated adversarial training plots

Whereas Figs. E.1 and E.2 concatenate all defense iterations into one plot and all attack iterations into another plot, Figs. L.1 and L.2 give training progress plots for each iteration separately.

L.2 Training steps plots

In Figs. L.3 to L.15 we display versions of previous plots but use victim-play or self-play training steps on the x-axis to measure training time instead of GPU-days. We tended to use GPU-days throughout this paper since it is a unit that is more understandable for readers, but as GPU-days are machine-dependent, training steps may be more useful for other researchers who want to compare our runs to other KataGo-like training runs.

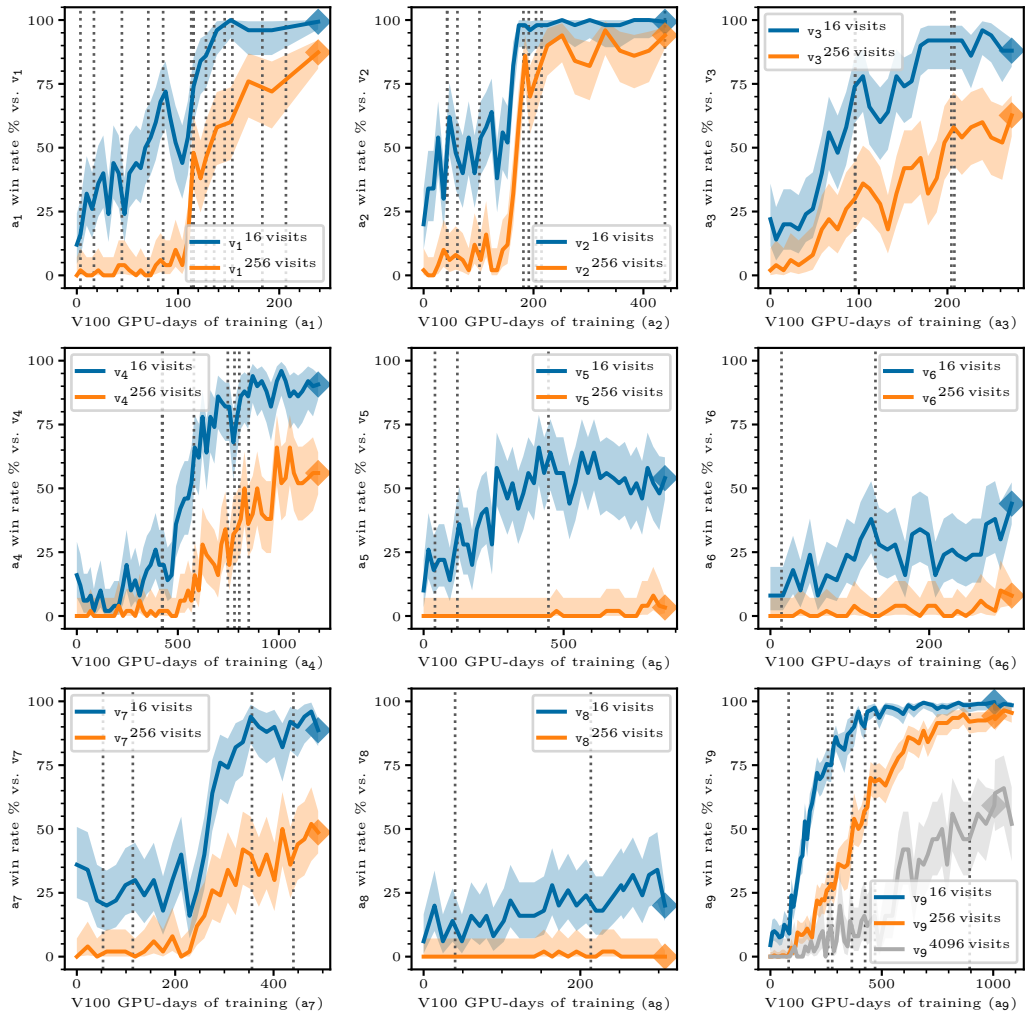


Figure L.2: Win rate (y-axis) of each $\texttt{a}_N$ against $\texttt{v}_N$ throughout $\texttt{a}_N$'s training (x-axis). Dotted lines represent advancing to the next victim in the curriculum.

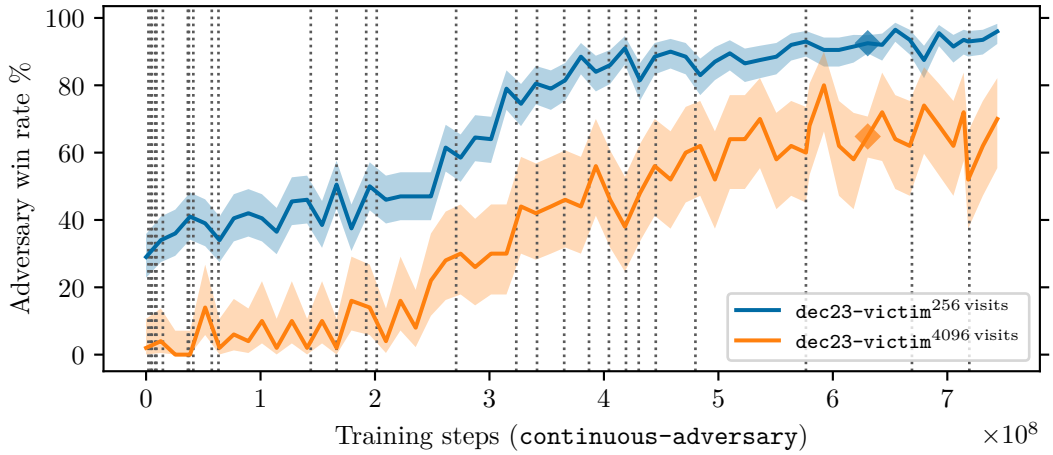


Figure L.3: This plot is the same as Fig. D.1 but with training steps on the x-axis instead of GPU-days.

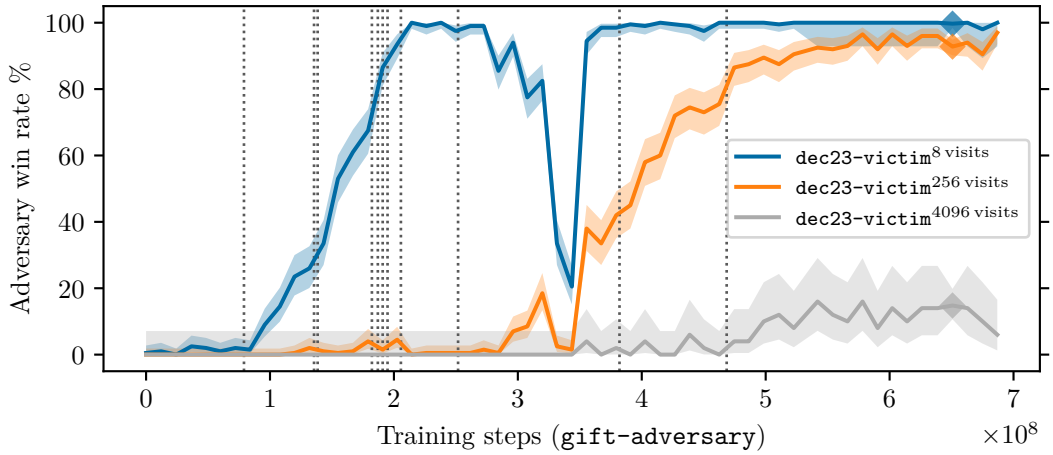


Figure L.4: This plot is the same as Fig. D.3 but with training steps on the x-axis instead of GPU-days.

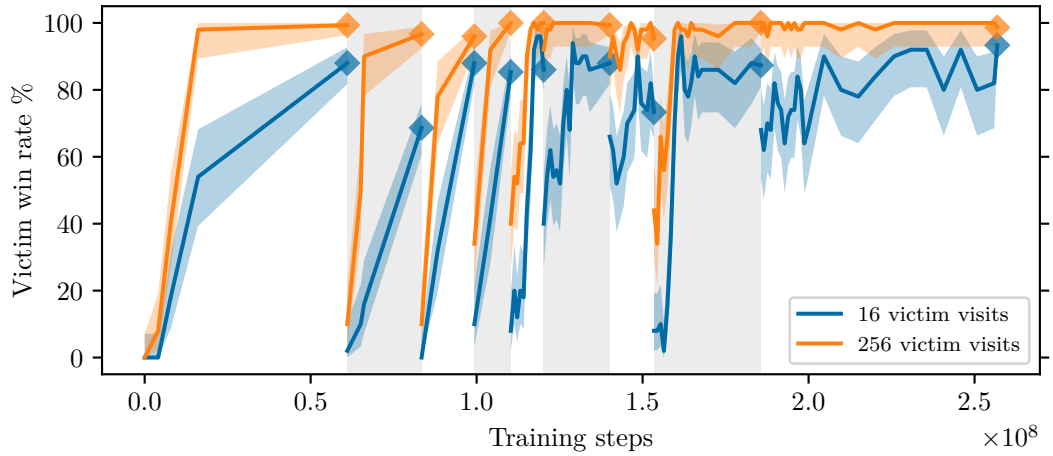


Figure L.5: This plot is the same as Fig. E.1 but with training steps on the x-axis instead of GPU-days.

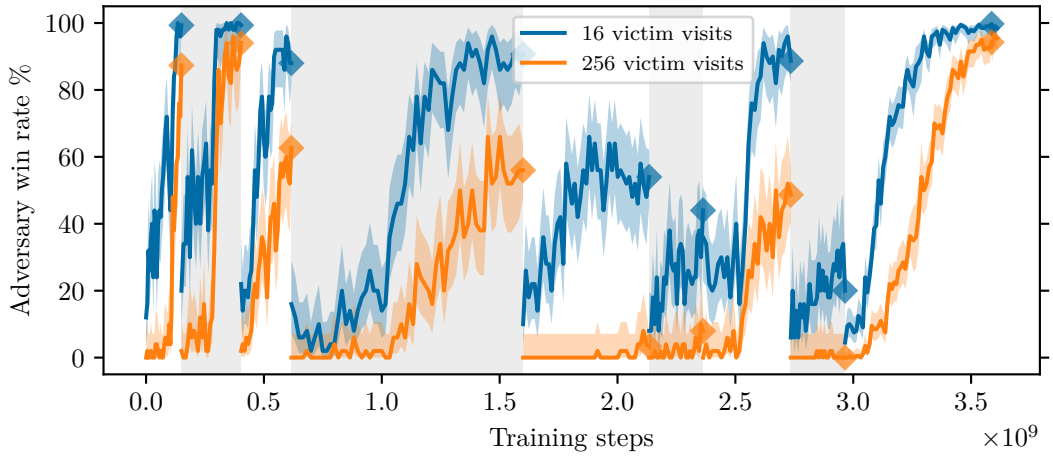


Figure L.6: This plot is the same as Fig. E.2 but with training steps on the x-axis instead of GPU-days.

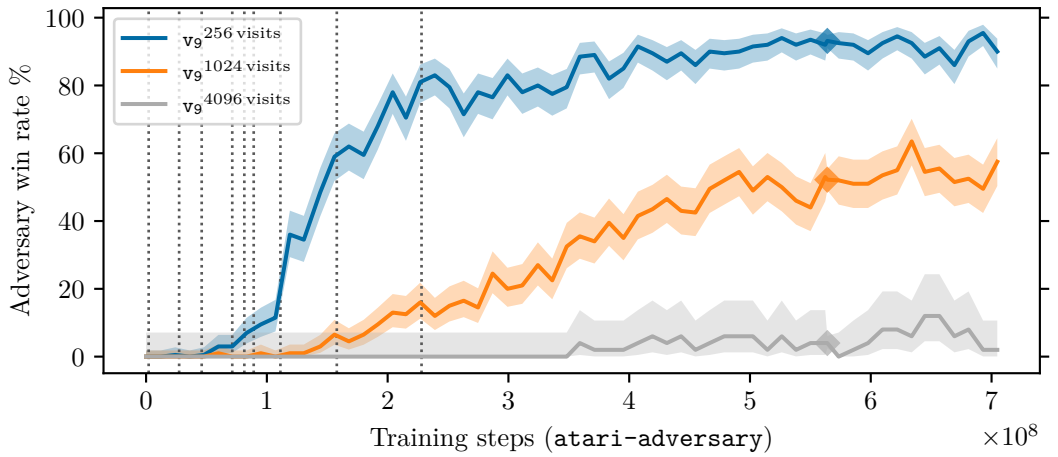


Figure L.7: This plot is the same as Fig. E.3 but with training steps on the x-axis instead of GPU-days.

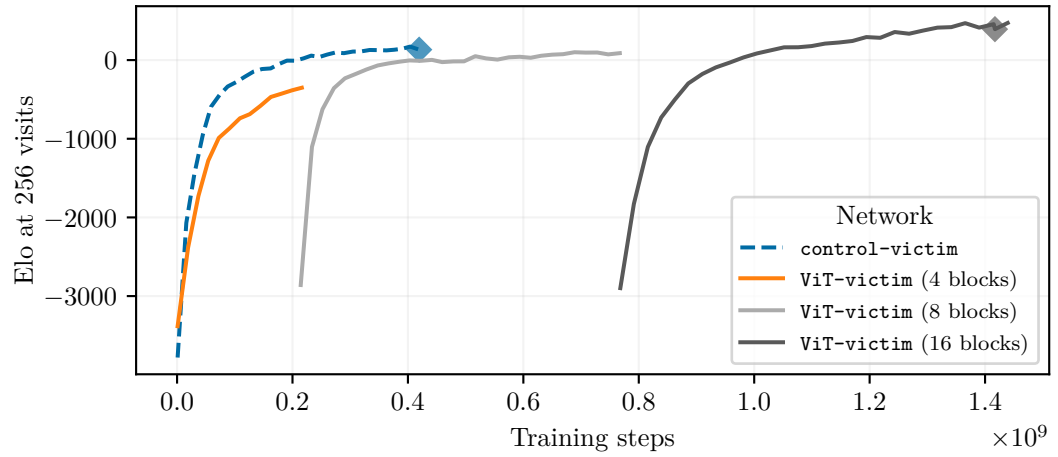


Figure L.8: This plot is the same as Fig. F.2 but with training steps on the x-axis instead of GPU-days.

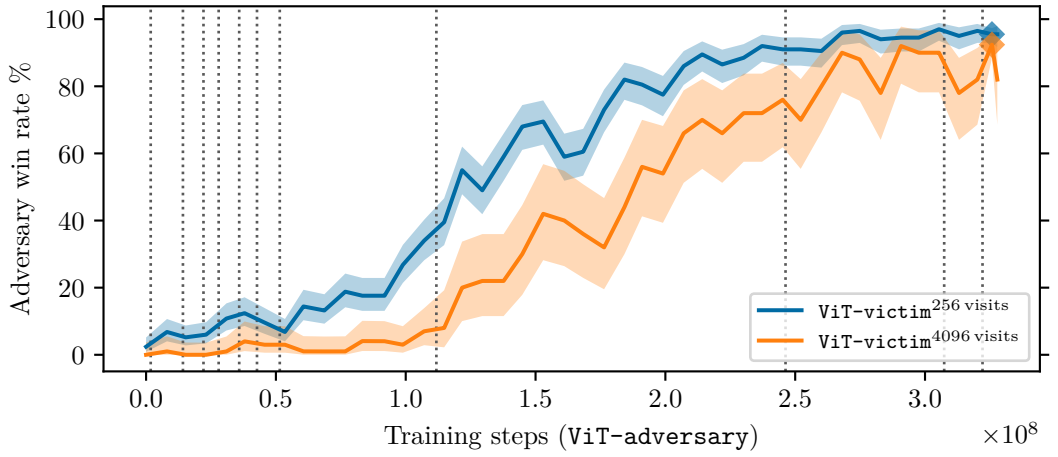


Figure L.9: This plot is the same as Fig. F.3 but with training steps on the x-axis instead of GPU-days.

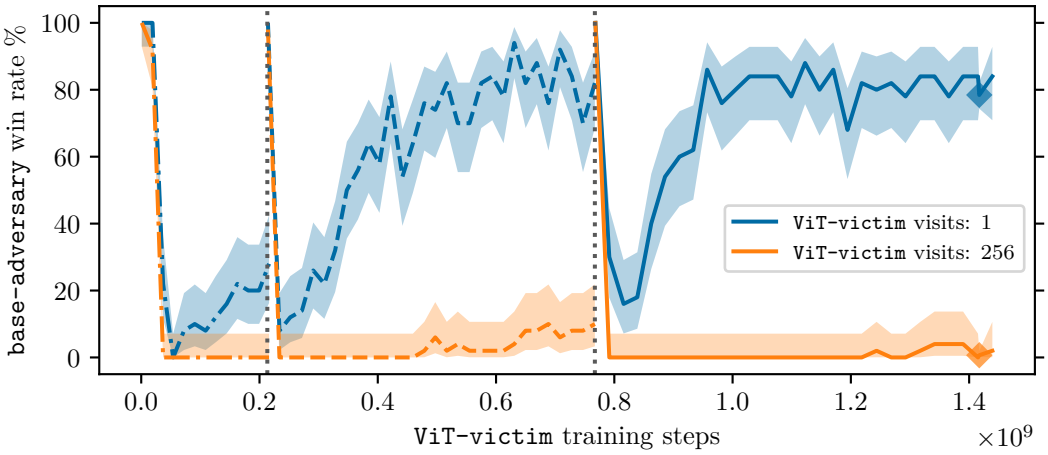


Figure L.10: This plot is the same as Fig. F.4 but with training steps on the x-axis instead of GPU-days.

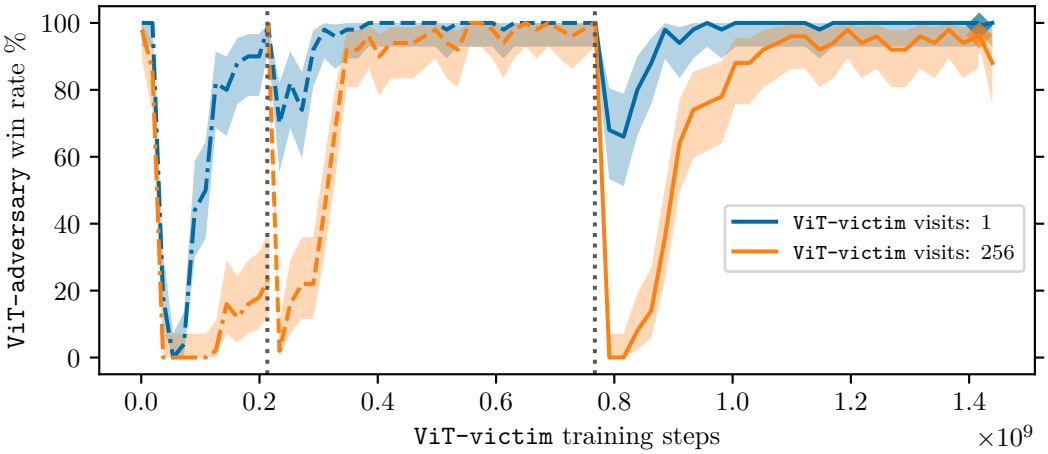


Figure L.11: This plot is the same as Fig. F.5 but with training steps on the x-axis instead of GPU-days.

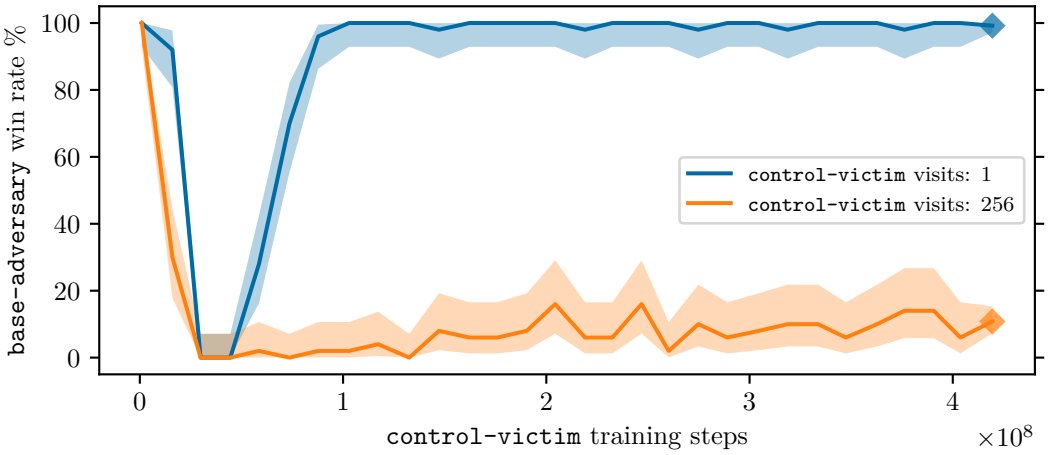


Figure L.12: This plot is the same as Fig. F.6 but with training steps on the x-axis instead of GPU-days.

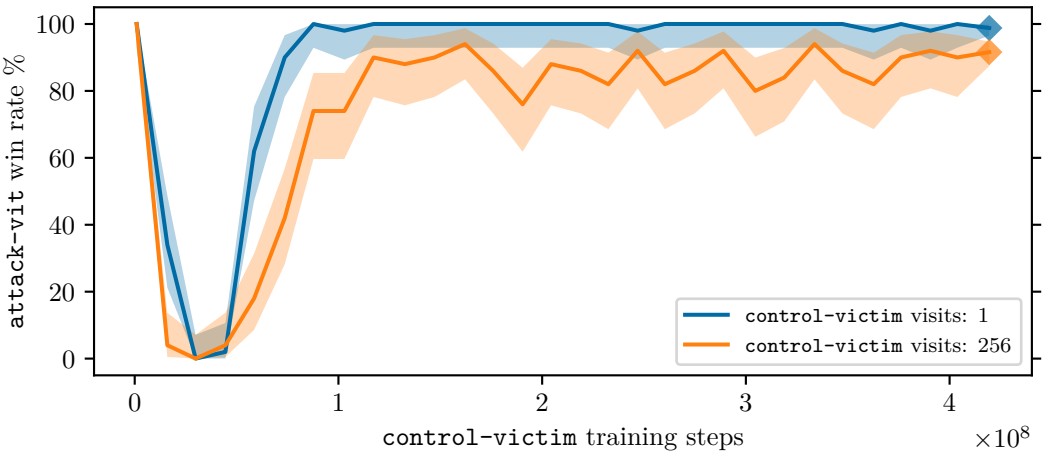


Figure L.13: This plot is the same as Fig. F.7 but with training steps on the x-axis instead of GPU-days.

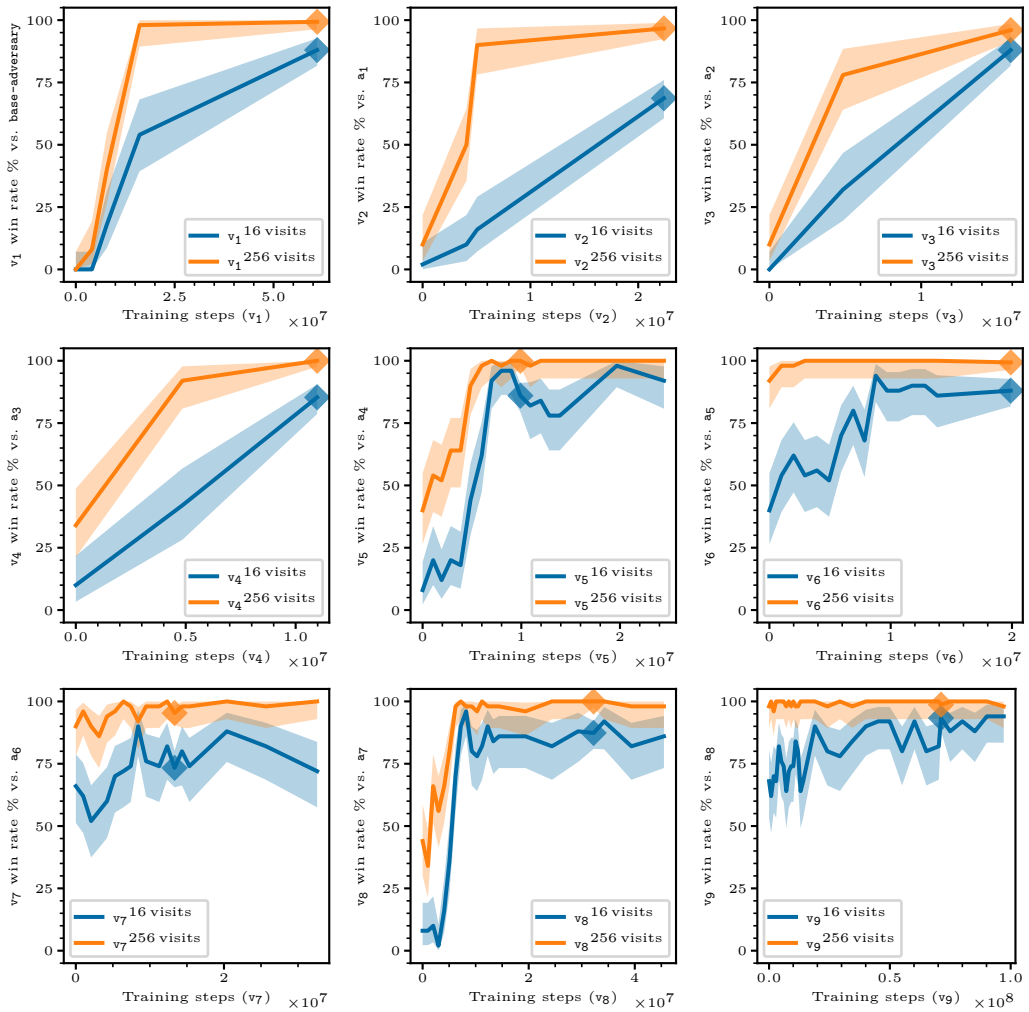


Figure L.14: This plot is the same as Fig. L.1 but with training steps on the x-axis instead of GPU-days.

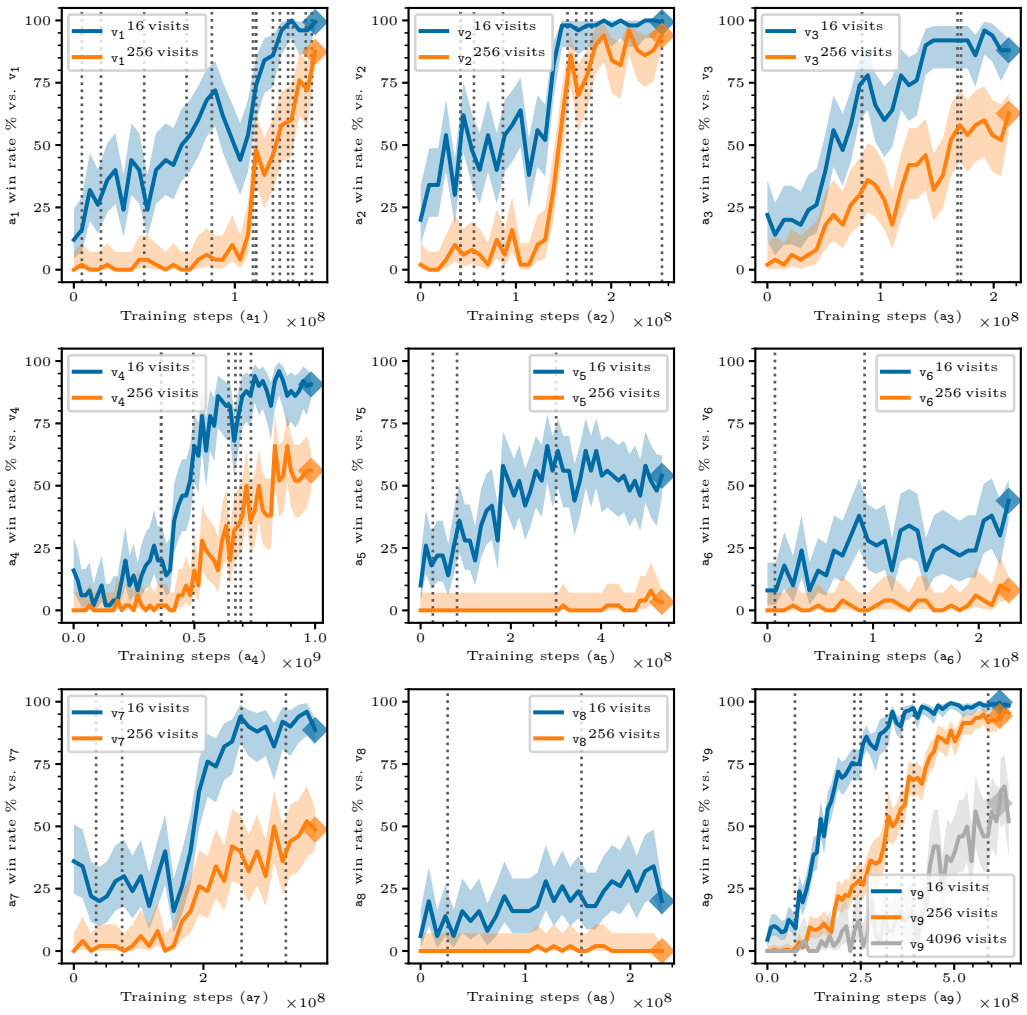


Figure L.15: This plot is the same as Fig. L.2 but with training steps on the x-axis instead of GPU-days.