Playing Coinche with Deep Counterfactual Regret Minimization

Sophia Günlük **Théo Saulus**
Université de Montréal, Mila

Abstract

Coinche (or *belote coinchée*) is a French card game that opposes two teams of two players with imperfect information. In this project, we design a game environment for reinforcement learning (RL) agents to play and learn, and study different approaches to train a model to play Coinche. In particular, we extend previous open-source implementations of the game to account for the bidding phase which is otherwise left out, and implement the Deep Counterfactual Regret minimization (Deep CFR) method from scratch. Through analysis of the results, we are able to recover a Nash equilibrium, and we identify some stereotypical behaviors.

1 Introduction

Coinche stands among the most popular card games in France; while we were looking for related works to the game, our google searches found many "irrelevant" dissertations that thanked Coinche in their acknowledgments for providing an outlet during stressful times. For our project, we were interested in exploring various reinforcement learning (RL) algorithms applied to this setting, and understanding which strategies the algorithms find optimal. Due to its 2 vs. 2 structure, it provides a framework that is well suited to study both adversarial and cooperative learning dynamics, and due to its large state space and incomplete information aspects, it provides a non-trivial framework for RL algorithms to learn in.

First, we had to properly model the problem with reward, transitions, probabilities, etc, in order to set up the agent's objective and state space. There has been some prior work into formalizing the Coinche game process, but we found that they usually simplify or misrepresent some phase of the game, such as the bidding phase. We, however, wanted to implement a version that accurately captures all (or at least almost all) of the complexities of the game, specifically the necessity for cooperation without the ability to freely communicate. We designed a new Gym RL environment that is a complete implementation of the game Coinche, extending from a pre-existing open-source environment.

We then explored two different RL algorithms and analyze the optimal strategies they learn. We trained agents using PPO, a standard RL training algorithm, via the Stable-Baselines3 [Raffin et al., 2021] package. Using the learned policies, we analyzed what strategies it was able to learn when playing against different players with pre-determined strategies, designed by our own Coinche expert Théo! We also trained agents using the Deep Counterfactual Regret minimization (Deep CFR) method [Brown et al., 2019], which we coded from scratch. Deep-CFR is a variant of Counterfactual Regret minimization (CFR) [Zinkevich et al., 2007] and Monte Carlo CFR (MCCFR) [Lanctot et al., 2009] that is better suited for large state spaces and incomplete information, and converges to a ε-Nash equilibrium. Using self-play, where agents have a shared policy and learn while playing against themselves, we were able to identify the trivial Nash equilibrium that the Deep CFR algorithm converges to. We were also able to get some results from agents trained with Deep CFR against other players.

Our contributions are the following:

- Extend the open-source implementation proposed by Picot [2020] to model the full game mechanics, including the bidding phase, the end-of-game reward, and three agents relying

on heuristics. Our implementation can be found on https://github.com/theosaulus/coinche. See Section 2.

- Implement deep counterfactual regret minimization (Deep CFR; Brown et al. [2019]) from scratch. See Section 3.
- Train policies in a format where all players are learning to play from scratch. We identify and recover some simple Nash equilibrium, and discuss the learned policies. See Section 4.
- Train policies in a format where two agents learn to collaborate against two heuristics agents. We show how dynamics of learning specific to Coinche may prevent an efficient learning of the game when playing against knowledgeable agents, and identify a few stereotypical behaviors. See Section 5.
- Discuss future work enabled by our implementation in Section 6.

2 Game rules and implementation

The game is divided in two parts: first, the players can bid the number of points they intend to achieve, along with the trumps suit. Then, they play a sequence of 8 tricks to earn points. Rules of Coinche vary slightly between regions, so we choose to follow the rules of *belote contrée* from the French Belote Federation (FFB), and we count points following the *points annoncés* method[1].

Our implementation is a heavily modified fork of Picot [2020] repository, using essentially numpy [Harris et al., 2020], PyTorch [Ansel et al., 2024], and Gymnasium [Towers et al., 2024].

Let $(North, East, South, West)$ denote the four players, such that $(North, South)$ and $(East, West)$ form the two teams. We abbreviate the players identity as (N, E, S, W) in the following.

2.1 Dealing

Cards are dealt anti-clockwise by one of the player, and the dealer role rotates anti-clockwise after every game. Traditionally, the deck is never shuffled, only cut, so that players have better hands on average. To make implementation easier, and to follow the recommendation of the FFB, we instead consider the deck to be reshuffled at every game.

Before the next phase starts, each player p has 8 cards in their hand, with a number attribute $i \in \{7, 8, 9, J, Q, K, 10, A\}$, and a suit attribute $j \in \{\spadesuit, \heartsuit, \clubsuit, \diamondsuit\}$. We encode the hand as a one-hot vector of size 32. Since there are 4 players, there exist $\frac{32!}{8!^2} \approx 10^{17}$ initial combinations.

2.2 Bidding

All players sequentially bid a contract they aim to fulfill, knowing that there are 162 points in total in the game. The order of bidding is anti-clockwise, and the first player to bid is the one on the right of the dealer. Each bid can be one of the 42 following possibilities:

- Pass (not announcing a bid at this turn)
- A number of points $n \in \mathbb{N} = \{80, 90, \dots, 160\}$ and a trump suit $s \in \mathbb{S} = \{\spadesuit, \heartsuit, \clubsuit, \diamondsuit\}$ [2] (aiming to do at least n points with s as the suit)
- Capot and a trump suit $\mathbb{S}$ (aiming to win all the tricks with s as the suit)
- Coinche (doubling down on the current bid, keeping it the same but doubling the amount of points at stake)
- Surcoinche (doubling down again on a coinched bid)

The bidding phase ends if no bid is placed and all 4 players pass from the start, or a bid is placed and 3 players pass in a row, or surcoinche is announced. The team placing the last bid is considered attacking, and the other one is defending. This results in a sequence of bids $B_1, \dots, B_b$ of length at

[1] https://www.ffbelote.org/belote-contree/

[2] We ignore here the "all trumps" and "no trump" suit bids for simplicity

most 37. The bids are encoded as an array of size 37 of integers between 0 and 43 representing every possible bid, plus a padding token for bids that did not happened yet.

From a human perspective, this phase is where most of the information is exchanged between partners, and it has a significant impact on how to play the following phase. E.g., a partner raising the bid from 100♠ to 110♠ usually indicates possession of an Ace of suit $\mathbb{S}\backslash\spadesuit$.

2.3 Tricks

Eight tricks are performed sequentially, with each player playing a card, rotating anti-clockwise. For the first turn, the first player is the one who spoke first during bidding phase, and for subsequent turns, it is the one who won the previous trick. A trick is won by placing the strongest card. A number of rules dictate what cards can or cannot be played at a given moment, and which card is the strongest. These rules can be found on the FFB website, and were already implemented by Picot [2020].

2.4 Environment implementation remarks

Implementing the bidding mechanism in the the environment of Picot [2020] required to augment the observation, modify the action space, modify the step function, and add a masking function for bids. Indeed, in all other implementations we found, only the last bid was provided to the agent, which does not allow to transmit or receive information as human players would do. The action space has now a double usage: during the bidding phase the bids are sampled among the 43 categories, and during the tricks phase the action is sampled among the 32 first categories only. This dual-use action space allows to have a single environment for the two phases of the game. Masking during bidding phase is performed according to the FFB rules of the game.

2.5 Points

Each card has a value, detailed in Table 1.

Card	7	8	9	J	Q	K	10	A
Trump	0	0	14	20	3	4	10	11
Non-trump	0	0	0	2	3	4	10	11

Table 1: Card values

The sum of all cards' value won by each team is their score. If the score of the attacking team is above (or equal to) their bid, then they win, otherwise they loose. Classically, a sequence of games is played, and points are tracked until one team reaches 1000 points. The amount of points earned at each game is detailed in Table 2. One can observe that loosing as an attacker (in French, *chuter*) is strongly penalized.

	Attack	Defense
Win	bid's value	160
Lose	0	0

Table 2: Points obtained. The bid's value of *capot* is 250 in both attack and defense.

Due to time constraints, we did not have time to work with sequences of games, although our environment could allow for it. Note however that each game is independent from the others (if dealing is random), so this sequential aspect should have little influence.

2.6 Formalization of the game model

Extending the work of Reyes [2023], we frame Coinche as a Partially Observable Markov Decision Process (POMDP) [Sondik, 1971, Drake, 1962]:

- The state space S is composed of the dealer index in (N, E, S, W), the index of the player currently bidding or playing a trick, the bids history $B_1, \ldots, B_b$, the player leading the

current trick, and the assignment of each 32 card to either a player, already played cards, or the current trick. Some of these attributes may be empty before the bidding phase has ended.

- The action space A is either the 42 bids, or the cards in the player's hand.
- The transition probabilities $P_a(s, s')$ depend on other players' strategy
- The reward $R_a(s, s') = \begin{cases} r & \text{if } s' \text{ is terminal} \\ 0 & \text{otherwise} \end{cases}$, where r is determined by Table 2. Changing this reward and attributing intermediate rewards could benefit training, and will be discussed later.
- The observations O consist in the dealer index in (N, E, S, W), the bids history $B_1, \ldots, B_b$, the player leading the current trick, and the assignment of each 32 card to either the player itself, already played cards, the current trick, or other players' hands.

2.7 Related work

A few other projects can be found on this game. Graux [2016] exhaustively describes the rules and common strategies in Coinche. Picot [2020] developed an environment to play Coinche with RL models. However, the bid is chosen at random or using simple predictors, without bidding history, which prevents the players to exchange information. A few other implementations in Python exist, but are not as complete and thorough as this one. Reyes [2023] proposes a formalization of the game and compares different algorithms for the betting phase and tricks phases. However, bidding is yet again done as a one-shot prediction without bidding history. Conreux and Soulaire [2023] attempt other strategies, but again do not model the bidding phase properly.

3 Deep Counterfactual Regret Minimization

In such settings such as Coinche, where there is incomplete information and points are only awarded at the end of rounds, it can be difficult to train a model due to bad/weak gradients. Because there is also an exponentially large number of states and action pairs, we found Deep Counterfactual Regret (Deep CFR) to be an appropriate algorithm for this setting.

With this method, instead of solving a value/regret function for each information set I and action a, they train multiple networks that learn to model players' advantages, which are then used to train a network to model the optimal policy that maximize those advantages, i.e. minimizes regrets. In order to handle the large state space, Deep CFR conducts a constant number K partial traversals of the game tree, with the path of the traversal determined according to external sampling. They show that this converges to an ε-Nash equilibrium.

A more detailed analysis of the method and how it was developed, as well as the pseudocode, can be found in Appendix A.

4 Training models without expert knowledge

4.1 Deep CFR

While Deep CFR is significantly more computationally feasible than CFR or MCCFR, we still had the problem of exploding branches, particularly in the bidding phase and the beginning of the tricks phase. It was also not trivial to parallelize the process since the algorithm solves from the bottom to the top of the tree, so it has to wait for every later traversal to complete before it can continue. Given all this, in order to reduce the explosion of branching, we implemented a CFR traversal that uses both External Sampling and Outcome Sampling, which is showed by the MCCFR paper to both have bounded overall regret with high probability [Lanctot et al., 2009].

Following this, we implemented 2 versions of Deep CFR: one that strictly follows the paper and one that follows the paper but samples actions for the learning player instead of fully traversing them (i.e. uses outcome sampling); the code can be found in the github repository [3]. Since the original paper's implementation was too expensive to run, we show only results from the outcome sampling

[3] See branch 7.

version. The hyperparameters (Table 3) and psuedocode describing the implementation of Deep CFR (Algorithm 1), their CFR traversal (Algorithm 2) and our version of traverse (Algorithm 3) can be found in Appendix A.

4.2 Recovering Nash equilibrium

As mentioned before, Deep CFR converges to an ε-Nash equilibrium, even with outcome sampling. Before we ran experiments, we hypothesized that passing all the time would be a trivial Nash equilibrium, and would likely be the first policy the Deep CFR algorithm would converge to. By betting, players, especially weak ones, open themselves up to losing points by failing to satisfy the bid, so they should be incentivized to pass continuously and safely get a reward of 0.

Our experiments confirmed this result, which we can conclude since the episode length goes to 4 (indicating that all players pass and the game terminates) and the policy loss stabilizes after that point:

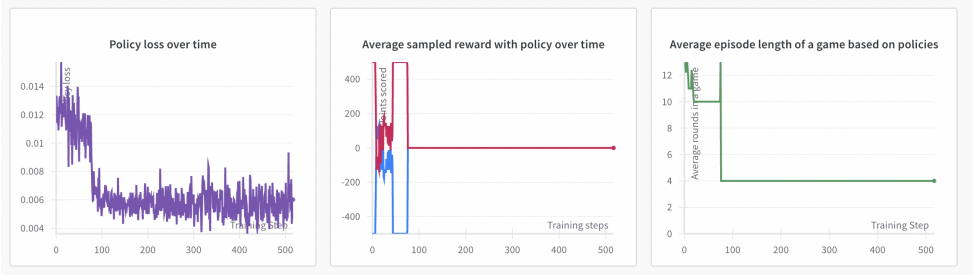

Figure 1: Trivial Nash equilibrium at all players passing during the bidding phase, resulting in 0 reward for all players.

This is a Nash equilibrium of Coinche in our zero-sum Coinche setting. More concretely, given that one team is always passing during the bidding phase, if either member of the other team bids, they will have to score at least that many points to win any points (since the other team will continue to pass). However, since the agents are initially random, they do not know how to play tricks properly and score, meaning they will almost definitely lose the round and get a negative reward (unless they randomly get amazing cards, bid the right suit, and play the right cards at the right time – but this is extremely improbable). Therefore, if any player changes their policy to anything but passing, it will result in a strictly worse (negative) reward, and so each player is individually incentivized to keep passing and score 0 points, providing a Nash equilibrium.

5 Training models against heuristics

5.1 Heuristics players

To train and evaluate our models, we designed simple agents with different levels of difficulty: one random player choosing actions uniformly over legal actions, one simple deterministic player using simple logic to bid and play, and one stronger deterministic player using refined logic to bid and much better logic to play tricks, inspired by the recommendations of Graux [2016]. The models are detailed in Appendix B.

5.2 Setup

To start with, we train one model to play as (N, S) against two identical heuristic players (E, W). For simplicity, and since the reward is shared among the players of the same team, we train one model only for both players (N, S). This also reduces the memory overhead (and allows for better parallelization when training with the PPO algorithm).

During training, we track many game related quantities, which allows us to identify different phases of training. To provide signal to the model when it looses an attack or a defense, we use the negative reward denoted as "good" in the next subsection, and add a per-trick reward weighted by 0.1. We focus on the following metrics, and provide details about our expectations on them, and educated values for a human player in Appendix C.

5.3 PPO

Before testing Deep CFR, we wanted to understand what strategies an algorithm could even learn in this setting. Therefore, we trained agents with a pre-implemented Proximal Policy Optimization (PPO) [Schulman et al., 2017] model from the package Stable-Baselines3 [Raffin et al., 2021]. This solution has the benefit to allow us to verify more easily that our environment is well behaved, and helped us find some initial bugs.

PPO is a classic reinforcement learning algorithm that learns both a policy $\pi(a \mid s)$ and a value function $V(s)$. Its specificity is to clip policy updates to avoid large updates and effectively stabilize training, making it a natural choice. Moreover, the pre-implemented version of Stable-Baselines3 natively allows to mask actions, contrary to other implementations. Hyperparameters are the default ones, except for the following values chosen with very little tuning: 64 environments in parallel, 512 steps before policy updates, and mini-batch size of 512.

In Figure 2, we plot the aforementioned metrics over time against the three heuristics. In those plots, note that capot is registered as a bid of value 250.

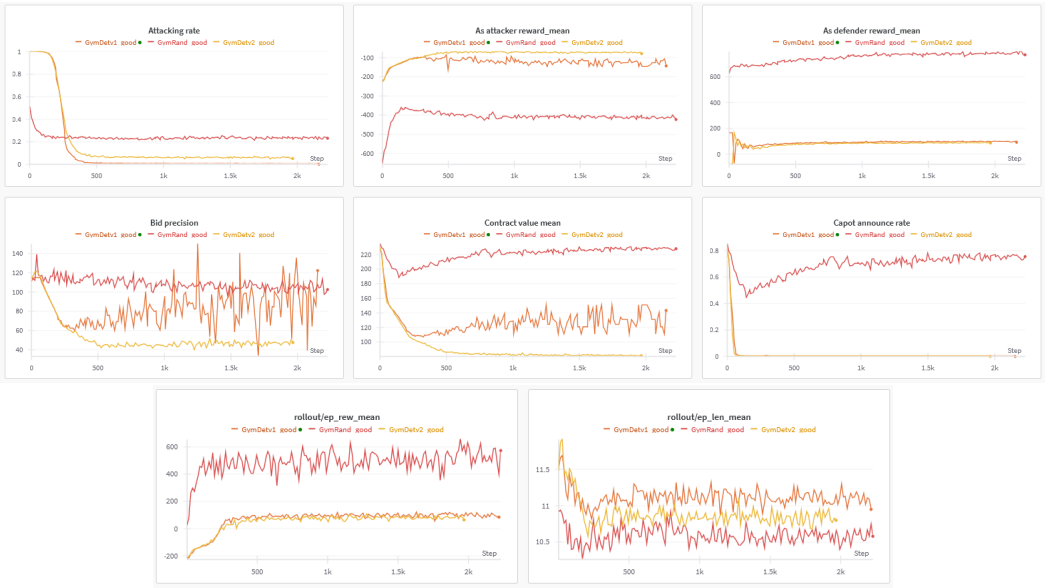

Figure 2: Game performance of the PPO model against 3 heuristics. GymRandom corresponds to the game PPO vs. random player, GymDetv1 is PPO vs. simple deterministic player, and GymDetv2 is PPO vs. stronger deterministic player. Note that bid precision stands for the absolute difference between the bet and points scored, which implies that lower is better.

The observations confirm some of our hypotheses, and showcase some limitations:

- **Against random players** does not provide a strong signal or incentive to learn how to bid (low attacker reward, low bidding precision). The model initially learns to *not* bid capot too often, which results in a reward increase as attacker. This reward as attacker then stabilizes, and the bidding value increases again. This suggests that the ability to play tricks improves over time.
- **Against either deterministic players** results in similar dynamics: early increase in bidding soundness (better precision, lower average value), and stabilization of the defensive abilities. In both case, it is clear that it is easier to defend than attack, and therefore few attacks are attempted. This is probably due to the basic trick heuristics of these players, especially the simple one.
- **Against simple deterministic player**, the ability to bid is not sustained until the end of training, and defending becomes the almost unique strategy (with a success rate of 75%), probably because the trick play heuristic is too simple.

- **Against stronger deterministic player**, the model attempts to attack more, and shows signs of learning better how to bid (better bid precision, lower bids on average). On the other hand, the defense success rate is slightly lower at 70%. This may hint that having an even stronger heuristic could help learning a good model.

Since all our experiments were with non-normalized rewards (i.e. rewards are between $0 - 500$), which is not ideal for RL algorithms, we have additional results where we analyzed learned behavior when trained against different opponents with different reward heuristics in Appendix D.

5.4 Deep CFR

Since Deep CFR converged to a trivial strategy when training in self-play, we wanted to see what it could learn when similarly playing against "experts," as defined by the heuristics above. Similar to PPO, the learning "Gym" agents are on one team, while the other team is made of up of either random players, simple deterministic players, or stronger deterministic players. The hyperparameters are the same as the self-play experiments, in the Appendix A.3.

In figure 3, we plot the same metrics from the results of different Deep CFR training sessions against the various types of players:

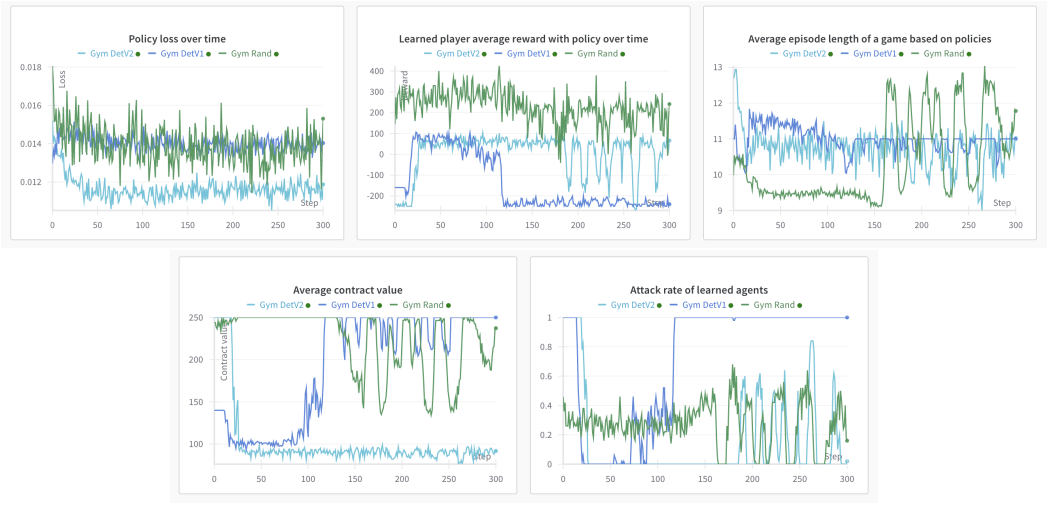

Figure 3: Game performance of the Deep CFR model against 3 heuristics. GymRandom corresponds to the game Deep CFR vs. random player, GymDetv1 is Deep CFR vs. simple deterministic player, and GymDetv2 is Deep CFR vs. stronger deterministic player.

- **Against random agents**, similar to PPO, the agents are able to learn a relatively "good" strategy by just waiting for the random players to badly attack, or attack with a reasonable contract value (i.e. not capot). It does not have to learn how to properly play tricks; since the random team is highly likely to make a bad move, it can just depend on winning points through bidding.
- **Against simple deterministic agents**, the agents learn to defend relatively well, but at some point prefers to always attack instead, which it doesn't do very successfully. We believe this occurs due to the various sampling our Deep CFR algorithm does, which results in a strong "advantage" signal when the learning player randomly happens to beat the deterministic player when it attacks. As such, it eventually chooses to always bid really high (since it may as well go all in) in hopes that it will sometimes attack successfully, whereas an algorithm that was purely trying to maximize rewards, like PPO, or had access to full traversals rather than just sampling, would prefer a more stable strategy.
- **Against stronger deterministic agents**, we see the agents similarly learn a more reasonable strategy before oscillating between strategies that sometimes attack vs never attack, resulting in a more stable expected reward. This is because the agents are not able to consistently rely on randomly winning attacks against the stronger deterministic player as much as it

could with the simple deterministic player. However, similarly to before, the agents only learn from partial traversals, which probably leads to this instability of choosing to attack sometimes.

As with PPO, it is clearly easier for the Deep CFR model to learn to defend rather than attack, leading the model to focus on strategizing the bidding phase rather than learning to play tricks properly. However, due to the sparse, non-normalized rewards and sampling, it takes advantage of randomly winning large rewards rather than refining its bidding strategy.

We also ran experiments where there were 3 learning agents in an environment with 1 pre-defined agent, as well as 2 learning agents that are on opposite teams (i.e. each paired with some pre-defined agent) but did not have the time or space to include it in the report.

6 Conclusion and future work

In our project, we created a novel Gym environment suitable for RL training that is a complete implementation of the card game Coinche. We then trained two algorithms, PPO and Deep CFR, and analyzed the optimal strategies learned in self-play and when training against other players. PPO was able to learn successful strategies against various players, which we were able to analyze using our own knowledge of the game, and Deep CFR was successfully able to find the Nash equilibrium that we expected, as well as learn to play against various players, somewhat unsuccessfully, which we also analyzed using our knowledge of the game and method.

Future directions include:

- Play Coinche as a sequential game: original Coinche is played as a sequence of games, where the first team to reach a given threshold wins. This implies some sense of game memory, like in sequential social dilemmas: even though the aim of each game is to win, knowing previous games results may influence the best strategy to adopt.
- Use imitation learning, like some have done for Poker, by using a vision-based model to transcribe games, and then learn via imitation from those expert games.
- Once strong AI players are developed, a deeper study of the game dynamics could be performed, to look for specific game patterns: correlation between bids and specific cards in hand (e.g., bidding 90 and possessing the Jack), or classic tricks move (e.g., giving hints about which suit is one player's Ace, a.k.a. *appel*). In particular, one could try to train two separate models, and look at their ability to collaborate.
- Another line of work could be related to representation learning, by adding more tasks than only winning the game: since human players use the bidding phase to transmit information to their partner, one could add a reconstruction loss over the hand of the partner, to incentivize this communication. It would however be more satisfying if such a behavior emerged naturally.
- Late thoughts about our information encoding made us realize that encoding the bidding history as an array of integers in $[1, 42]$ implicitly implies a spurious order between the bets, and is a format known to be inefficient for MLPs. Instead, one-hot encoding of all bids into a 37×42 list, would be more suitable for training, although high-dimensional. Another option would be to encode the bidding phase with a sequence model such as an LSTMs, and feed this embedding to a model specialized for the tricks phase.
- Optimize the environment and the DeepCFR method would allow for faster training, and searches with more breadth/less outcome sampling.

Acknowledgments

We acknowledge the use of LLM-based assistants for coding (GitHub Copilot). This project was enabled in part by compute resources provided by Mila (mila.quebec).

References

Jason Ansel, Edward Yang, Horace He, Natalia Gimelshein, Animesh Jain, Michael Voznesensky, Bin Bao, Peter Bell, David Berard, Evgeni Burovski, et al. Pytorch 2: Faster machine learning through dynamic python bytecode transformation and graph compilation. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, pages 929–947, 2024.

Noam Brown, Adam Lerer, Sam Gross, and Tuomas Sandholm. Deep counterfactual regret minimization, 2019. URL https://arxiv.org/abs/1811.00164.

Louis Conreux and Tom Soulaire. The ultimate coinche bot: Strategy exploration. Technical Report AA228, Stanford University, Fall 2023. URL https://aa228.stanford.edu/old-projects/. Decision Making Under Uncertainty.

Alvin W Drake. *Observation of a Markov process through a noisy channel*. PhD thesis, Massachusetts Institute of Technology, 1962.

Damien Graux. La coinche: vers un système efficace. Technical report, Inria, 2016.

Charles R Harris, K Jarrod Millman, Stéfan J Van Der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J Smith, et al. Array programming with numpy. *Nature*, 585(7825):357–362, 2020.

Marc Lanctot, Kevin Waugh, Martin Zinkevich, and Michael Bowling. Monte carlo sampling for regret minimization in extensive games. In *Advances in Neural Information Processing Systems*, volume 22, pages 1078–1086, 2009.

Eric Picot. gym-coinche. GitHub repository, 2020. URL https://github.com/EricPicot/gym-coinche. An environment for the Coinche card game.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

Emilio Reyes. Decision-making algorithms for winning coinche. Technical Report AA228, Stanford University, Winter 2023. URL https://aa228.stanford.edu/old-projects/. Decision Making Under Uncertainty.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Edward Jay Sondik. *The optimal control of partially observable Markov processes*. Stanford University, 1971.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulao, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007. URL https://proceedings.neurips.cc/paper_files/paper/2007/file/08d98638c6fcd194a4b1e6992063e944-Paper.pdf.

A Deep Counterfactual Regret

A.1 In-depth Analysis of Deep CFR

In such settings such as Coinche, where there is incomplete information and points are only awarded at the end of rounds, it can be difficult to train a model due to bad gradients. We decided to look at Counterfactual Regret Minimization (CFR) [Zinkevich et al., 2007], a method for maximizing reward by minimizing the expected regret of playing an action. To do this, they first provide context by formally defining regret minimization, which had been more extensively studied at that time. They define the overall regret of a player p as:

$$R_p^T = \frac{1}{T} \max_{\sigma_p^* \in \Sigma_p} \sum_{t=1}^{T} \Big(u_p(\sigma_p^*, \sigma_{-p}^t) - u_p(\sigma^t) \Big)$$

where u_p is player p's expected utility if they and other players follow the given strategy, and a player's average strategy at information set I for each action is defined as a we:

$$\bar{\sigma}_p^T(I)(a) = \frac{\sum_{t=1}^{T} \pi_{\sigma_p^t}(I)\, \sigma^t(I)(a)}{\sum_{t=1}^{T} \pi_{\sigma_p^t}(I)}$$

Regret minimization is closely linked to to Nash equilibrium, so they conclude that *in a zero-sum game at time T, if both players average overall regret is less than ε, then $\bar{\sigma}^T$ is a 2ε equilibrium.*

A.1.1 Counterfactual regret minimization

This led them to define the notion of counterfactual regret minimization, propose a method that would converge to Nash equilibrium, even in incomplete information settings where the state space was extremely large, such as limit Texas Hold'em (as many as 10^{12} states).

They do this by first considering one particular information set $I \in \mathcal{I}_p$ and player is choices made in that information set and define the expected utility of that player $u_p(\sigma, h)$ given that the history h is reached (full round completed) and all players played using strategy σ. They then define the *counterfactual utility* to be the expected utility given that information set I is reached and all players play using strategy σ except player p; formally, if $\pi^\sigma(h, h')$ is the probability of going from history h to history h' (due to either chance nodes or other player's strategies), then

$$u_p(\sigma, I) = \frac{\sum_{h \in I} \sum_{h' \in Z} \pi_{-p}^\sigma(h)\, \pi^\sigma(h, h')\, u_p(h')}{\pi_{-p}^\sigma(I)}.$$

This means *immediate counterfactual regret* is the maximum utility that the player is expected to miss out on by following the policy rather than taking another action. Formally, if for all actions $a \in A(I)$, $\sigma_{I \to a}$ is the a "counterfactual strategy" i.e. strategy profile identical to σ except that player i always chooses action a when in information set I, then:

$$R_{i,\text{imm}}^T(I) = \frac{1}{T} \max_{a \in A(I)} \sum_{t=1}^{T} \pi_{-p}^{\sigma^t}(I) \big(u_p(\sigma_{I \to a}^t, I) - u_p(\sigma^t, I) \big).$$

By clipping the immediate counterfactual regret $R_{i,\text{imm}}^{T,+}(I)$, they show that by minimizing immediate counterfactual regret also minimizes the overall regret, which provides a corresponding method to find an approximate Nash equilibrium since $R_p^T \leq \sum_{I \in \mathcal{I}_p} R_{i,\text{imm}}^{T,+}(I)$.

Since minimizing immediate counterfactual regret minimizes the overall regret, it enables us to find an approximate Nash equilibrium if we can only minimize the immediate counterfactual regret, which can be done by only by controlling $\sigma_p(I)$.

This leads them to their method: for all $I \in \mathcal{I}_p$ and all $a \in A_p$:

$$R_p^T(I,a) = \frac{1}{T}\sum_{t=1}^{T} \pi_{-p}^{\sigma^t}(I)\big(u_p(\sigma^t_{I\to a}, I) - u_p(\sigma^t, I)\big).$$

Taking $R_p^{T,+}(I,a)$ as the clipped positive version, then the strategy for the next timestep $T+1$ should be proportion to the counterfactual regret (which needs to be clipped to be positive) for not playing that action (and if no actions have any positive immediate counterfactual regret, then the action is selected uniformly at random to encourage exploration):

$$\sigma_p^{T+1}(I)(a) = \begin{cases} \dfrac{R_p^{T,+}(I,a)}{\sum_{a'\in A(I)} R_p^{T,+}(I,a')}, & \text{if } \sum_{a'\in A(I)} R_p^{T,+}(I,a') > 0, \\ \dfrac{1}{|A(I)|}, & \text{otherwise (no regrets).} \end{cases}$$

They conclude that since the overall regret is bounded by the sum of counterfactual regret and the counterfactual regret can be minimized at each information set independently, the CFR method computes a Nash equilibrium when in self-play (i.e. policy network is shared).

A.1.2 Deep Counterfactual Regret Minimization

While this method is very elegant and works well for relatively large state spaces, it does require the game to learn regrets $R_p^{T,+}(I,a)$ and $\sigma_p^{T+1}(I)(a)$ for each information set $I \in \mathcal{I}_p$ and action $a \in A_p$ for each player p of a game. In Coinche, this explodes due to the combination of bidding phase, which can be up to 37 turns (with different bidding combinations), and the tricks phase, which requires learning policies for playing 32 specific cards in response to the bidding and other players. As such, it was not realistic to implement CFR for this setting, but luckily other researchers also found this algorithm intriguing and extended this work to Deep Counterfactual Regret Minimization (Deep CFR) [Brown et al., 2019].

Instead of solving for each information set I and action a, they develop a method that essentially iteratively samples traversals and fits a neural network to model players' advantages (inverse of regrets) and fit another neural network an method for learning the optimal policy to maximize those advantages, which minimizes regrets.

More specifically, at each training step, Deep CFR conducts a constant number K of partial traversals of the game tree, with the path of the traversal determined according to external sampling Monte Carlo CFR (MCCFR), which is an extension of CFR that only does partial traversals by externally sampling chance nodes (such as dealt cards) and their opponents' actions. This also converges to an equilibrium: for any $\rho \in (0,1]$, with probability $1-\rho$, the total regret is bounded by

$$R_p^T \;\le\; \left(1+\sqrt{\tfrac{2}{\rho}}\right)\frac{1}{\delta}\,\Delta_{u,i}\,M_p\,\frac{\sqrt{|A_p|}}{\sqrt{T}} \;\le\; \left(1+\sqrt{\tfrac{2}{\rho}}\right)|\mathcal{I}_p|\,\Delta\frac{\sqrt{|A_p|}}{\sqrt{T}}$$

With this method, the sampled instantaneous regrets are an unbiased estimator of the advantage, which is the difference in expected rewards for playing a instead of following the policy at information set I, assuming both players follow the policy everywhere else:

$$\mathbb{E}_{Q\in\mathcal{Q}_t}\big[\tilde{r}_p^{\sigma^t}(I,a) \;\big|\; Q \text{ is complete game}\big] = \frac{v^{\sigma^t}(I,a) - v^{\sigma^t}(I)}{\pi_{-p}^{\sigma^t}(I)}.$$

This gives convergence bounds for Deep CFR: assuming that at each iteration $t = 1,\dots,T$: if the difference between the average MSE loss for samples of the the advantage network at time t and the minimum loss for any advantage network is bounded by ε_L, which means the values memories are large enough to make sampling error negligible, then with probability $1-\rho$, the average regret at time T is bounded by: $R_p^T \;\le\; \left(1+\frac{\sqrt{2}}{\sqrt{\rho\,K}}\right)\Delta\,|\mathcal{I}_p|\,\sqrt{\frac{|A|}{T}} \;+\; 4\,\Delta\,|\mathcal{I}_p|\,\sqrt{|A|\,\varepsilon_L}$. Therefore as $T\to\infty$, average regret R_p^T is bounded by $4\,\Delta\,|\mathcal{I}_p|\,\sqrt{|A|\,\varepsilon_L}$ with high probability, implying convergence to a ε-Nash equilibrium.

A.2 Psuedocode

Algorithm 1 Deep Counterfactual Regret Minimization

Initialize each advantage network $V(I, a \mid \theta_p)$ with $\theta_p \leftarrow 0$
Initialize reservoir memories $\mathcal{M}_{V,1}, \mathcal{M}_{V,2}$ and strategy memory $\mathcal{M}_\Pi$
for $t = 1$ **to** T **do**
  for each player p **do**
    for $k = 1$ **to** K **do**
      TRAVERSE($\emptyset, p, \theta_1, \theta_2, \mathcal{M}_{V,p}, \mathcal{M}_\Pi, t$) ▷ externalsampling traversal
    end for
    Train θ_p from scratch on

$$\mathcal{L}(\theta_p) \;=\; \mathbb{E}_{(I,t',\tilde{r}^{t'})\sim\mathcal{M}_{V,p}}\big[t' \sum_a \big(\tilde{r}^{t'}(a) - V(I, a \mid \theta_p)\big)\big]^2$$

  end for
  Train θ_Π on

$$\mathcal{L}(\theta_\Pi) \;=\; \mathbb{E}_{(I,t',\sigma^{t'})\sim\mathcal{M}_\Pi}\big[t' \sum_a \big(\sigma^{t'}(a) - \Pi(I, a \mid \theta_\Pi)\big)\big]^2$$

end for
return θ_Π

Algorithm 2 CFR Traversal with External Sampling

function TRAVERSE($h, p, \theta_1, \theta_2, \mathcal{M}_V, \mathcal{M}_\Pi, t$)
  if h is terminal **then**
    return payoff$_p(h)$
  else if h is a chance node **then**
    $a \sim \sigma(h)$
    return TRAVERSE($h{\cdot}a, p, \theta_1, \theta_2, \mathcal{M}_V, \mathcal{M}_\Pi, t$)
  else if $P(h) = p$ **then** ▷ traversers turn
    Compute $\sigma^t(I)$ via regret matching on $V(I(h), \cdot \mid \theta_p)$
    for all $a \in A(h)$ **do**
      $v[a] \leftarrow$ TRAVERSE($h{\cdot}a, p, \dots$)
    end for
    for all $a \in A(h)$ **do**
      $\tilde{r}(a) \leftarrow v[a] \;-\; \sum_{a'} \sigma^t(I, a')\, v[a']$
    end for
    $\mathcal{M}_V$.insert($I, t, \tilde{r}$)
    return $\sum_a \sigma^t(I, a)\, v[a]$
  else ▷ opponents turn
    Compute $\sigma^t(I)$ on $V(I(h), \cdot \mid \theta_{-p})$
    $\mathcal{M}_\Pi$.insert($I, t, \sigma^t(I)$)
    $a \sim \sigma^t(I)$
    return TRAVERSE($h{\cdot}a, p, \dots$)
  end if
end function

Algorithm 3 CFR Traversal with Outcome and External Sampling

function TRAVERSE($h, p, \theta_1, \theta_2, \mathcal{M}_V, \mathcal{M}_\Pi, t$)
  if h is terminal **then**
    return payoff$_p(h)$
  else if h is a chance node **then**
    $a \sim \sigma(h)$
    return TRAVERSE($h{\cdot}a, p, \theta_1, \theta_2, \mathcal{M}_V, \mathcal{M}_\Pi, t$)
  else if $P(h) = p$ **then** ▷ traversers turn
    Compute $\sigma^t(I)$ via regret matching on $V(I(h), \cdot \mid \theta_p)$
    for $s = 1$ **to** $A_{samples}$ **do** ▷ instead of searching over all actions,
      $a_s \sim \sigma^t(I)$ ▷ sample based on strategy
      $v[a_s] \leftarrow$ TRAVERSE($h{\cdot}a_s, p, \dots$)
    end for
    for $s = 1$ **to** $A_{samples}$ **do**
      $\tilde{r}(a_s) \leftarrow v[a_s] \; - \; \sum_{a'} \sigma^t(I, a')\, v[a']$
    end for
    $\mathcal{M}_V$.insert($I, t, \tilde{r}$)
    return $\sum_a \sigma^t(I, a)\, v[a]$
  else ▷ opponents turn
    Compute $\sigma^t(I)$ on $V(I(h), \cdot \mid \theta_{-p})$
    $\mathcal{M}_\Pi$.insert($I, t, \sigma^t(I)$)
    $a \sim \sigma^t(I)$
    return TRAVERSE($h{\cdot}a, p, \dots$)
  end if
end function

A.3 Hyperparameters

Attribute	Value
num_iterations	5,000
log_interval	10
batch_size	256
adv_hidden	[64, 64]
adv_lr	3e-4
adv_mem_size	100,000
pol_hidden	[64, 64]
pol_lr	3e-4
pol_mem_size	500,000
num_traversals	256
adv_epochs	4
pol_epochs	4
A_samples	4

Table 3: Hyperparameters for our Deep CFR implementation

B Heuristics players

B.1 Random player

During bidding phase, it chooses a bid at random among all valid bids.

During tricks phase, it picks a card uniformly at random among valid cards.

B.2 Simple deterministic player

Helped by our own experience as players, we propose the following intuitive and simple rules to play Coinche.

During bidding phase, the model follows simple rules, mimicking a beginner player. The player starts by counting its number of Jack, 9, Aces, and the number of cards in each suit. Then it performs the following:

- If the partner has not bid yet,
  - If opponents have not bid yet, then,
    - If the player finds a suit where it has Jack and 9, and either another card in this suit or an Ace elsewhere, then it bids 90 in this suit.
    - Otherwise, if the player finds a suit where it has either Jack or 9, and either two other cards in this suit or a card in this suit and an Ace elsewhere, then it bids 80 in this suit.
  - If opponent's largest bid is $n = 80$, then bid 10 more points compared to the above rule.
- If the partner has bid either 80 or 90, then increase the bid and keep the same suit,
  - If the player has the Jack, raise by 20,
  - And if the player has the 9, raise by 10,
  - And raise by 10 for each Ace in other suits.

This bidding is often suboptimal, as it is too aggressive in some aspects (e.g., the player adds 10 points to its partner for an alone 9), although this is all matter of strategies.

During tricks phase, play the highest legal card possible, ranked by value. Trump suit is preferred (when legal), and other suits are ordered randomly. Note that such a strategy is far from optimal as it results in potentially loosing strong cards when the opponent is winning the trick with a stronger card.

B.3 Stronger deterministic player

Inspired by the guide of Graux [2016], we propose a stronger set of logics to play Coinche. We do not implement the exact recommendations since they become quickly quite complex.

During bidding phase, the player counts its number of Jack, 9, Aces, 10, and the number of cards in each suit. Then it performs the following:

- If the partner has not bid yet,
  - If opponents have not bid yet, then,
    - If the player finds a suit where it has Jack, 9, Ace, and 10, then it bids 110 in this suit.
    - If the player finds a suit where it has Jack and 9, and either another card in this suit, then it bids 90 in this suit.
    - If the player finds a suit where it has either Jack or 9, and either the Ace or two other cards in this suit, then it bids 80 in this suit.
  - If opponents have bid 80, then bid 10 more points compared to the above rule.
- If the partner has bid either 80 or 90, then increase the bid and keep the same suit,
  - If the player has the Jack, raise by 20,
  - And if the player has the 9, raise by 10, except if it is alone,
  - And raise by 10 for each Ace in other suits.

These strategies are more conservative than the previous player, and more aligned with the behaviour of more experienced players.

During tricks phase, the player follows more complex patterns:

- If it is the first to play in this trick,

  - If it is in the attacking team,
    * If it has trumps, play the largest
    * If it has non-trump Aces, play the Ace in the longest side suit
    * Otherwise, play the card on the smallest side suit
  - If it is in the defending team,
    * If it has non-trump Aces, play the Ace on the shortes side suit
    * If other non-trump cards are available, play the card of smallest value
    * Otherwise, play the lowest trump card
- If it is not the first to play,
  - If the player can play cards stronger than the current strongest card, it plays the smallest one among them
  - Else, play the card of lowest value

These rules are closer to what a knowledgeable player would do, although it is still suboptimal in many situations.

C Metrics

We track the following metrics:

- Attacking rate: how often do the model attack. Since the bids are random at the beginning of the training, we expect the model to initially bid almost all the time. A human player would attack around 40% of the time.
- Average reward as attacker. Initially, the number of points lost will be high since high bids will be failed, but as the model learns to bid, the average reward should grow. A human player should win strictly more than 0 points on average, especially against those relatively week heuristics players.
- Average reward as defender. Against deterministic players, defending well should indicate a decent competency of the tricks phase, and very high reward is expected against random players which fail to behave reasonably. A human player should be able to defend well against the heuristics players and get a positive defender reward average.
- Average reward: summarizes the three previous metrics.
- Average success rate as attacker/defender.
- Bid precision: the absolute value between the bid placed as an attacker and the number of points obtained in the end (lower is better). Learning how to bid is expected against the stronger deterministic player, which has less chance of loosing tricks dumbly, while other models do not incentivize much to learn how to bid properly. Experienced human players should reach around 10 points difference.
- Average contract value. We expect to see a clear decrease at first, when learning to not go for sky high bids. Like the bidding precision, this metric is expected to get better with deterministic players which incentivize learning how to bid. A human player bids on average in the $100 - 120$ range.
- Capot announce rate. This metric confirms that basic bidding logic is mastered: capot is quite rare for human players.
- Average episode length: number of table turns, for both bidding and tricks play. The model will bid too much at first, so a decrease is expected at first. Human players usually play for 10 table turns (8 is fixed for the tricks phase, and approximately 2 for bids).

D PPO training models against heuristics (continued)

Behavior with different rewards Noticeably in the previous experiment, we identify that no player learns to attack as often as a human player would do. Our interpretation is that the reward matrix of the original game (Table 2) is very punitive in case of bidding failure: the opponent scores 160 points, one of the most difficult bid to do as an attacker. In other words, the reward matrix incentivizes weak

players to stay defensive, and this could be seen as a local optimal strategy. To test this hypothesis, we tweaked the reward matrix, and checked how the model reacts to those changes. We attempt the reward matrices described in Tables 4.

	Attack	Defense
Win	bid	160
Lose	-160	-bid

	Attack	Defense
Win	bid	0
Lose	-160	-bid

	Attack	Defense
Win	bid	160
Lose	-bid	-bid

Table 4: Alternative rewards matrices, denoted as mirror, no_defense, and good in plots. The bid's value of *capot* is 250 in both attack and defense.

In Figure 4, we plot these metrics over time against the stronger deterministic model. In those plots, note that capot is registered as a bid of value 250.

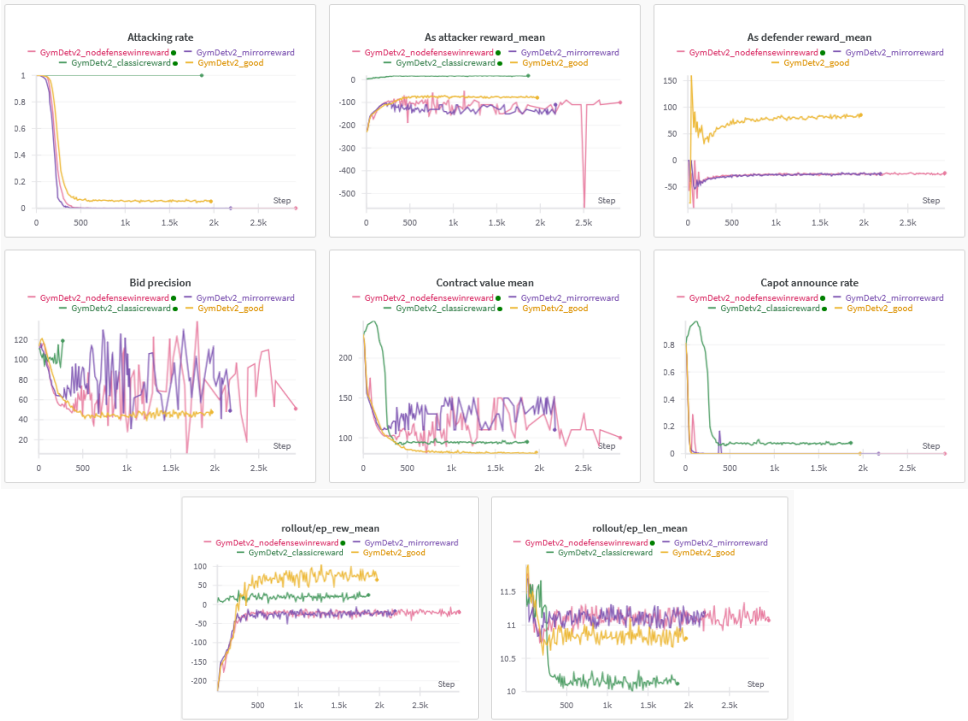

Figure 4: Game performance of the PPO model against the stronger determinist model with different rewards. Names of the models are detailed in the legend of Table 4. Note that bid precision stands for the absolute difference between the bet and points scored, except when the coinche bid is placed.

We can identify two main patterns, keeping in mind that the rewards are not equivalent and one can only compare the dynamics and not absolute values:

- Win by defending. The models trained with the mirror, no_defense, and good rewards exhibit similar behaviors as described in the previous experiment.
- Win by coinching. The model trained with the classic reward finds an alternative strategy, which is in fact close to the previous one: the model systematically coinche its opponent (the coinche rate is indeed close to 1), allowing it to earn at least 160 points in case of victory. Since the deterministic player is agnostic to the coinche bid, it will behave the same, and this behavior is indeed more optimal than the previous one. Thus, coinching makes the model become attacker without having to learn how to bid properly.

As a side note, we attempted to mask extreme bids in a variety of ways, but none resulted in a player learning to bid with good precision.