A List of contributions

This paper makes several contributions, which we summarize here.

- **RL+Search in two-player zero-sum imperfect-information games.** Prior work has developed RL+Search for two-player zero-sum perfect-information games. There has also been prior work on learning value functions in fully cooperative imperfect-information games [19] and limited subsets of zero-sum imperfect-information games [29]. However, we are not aware of any prior RL+Search algorithms for two-player zero-sum games in general. We view this as the central contribution of this paper.
- **Alternative to safe search techniques.** Theorem 3 proves that, when doing search at test time with an accurate PBS value function, one can empirically play according to a Nash equilibrium by sampling a random iteration and passing down the beliefs produced by that iteration's policy. This result applies regardless of how the value function was trained and therefore applies to earlier techniques that use a PBS value function, such as DeepStack [40].
- **Subgame decomposition via CFR-AVG.** We describe the CFR-AVG algorithm in Appendix I. CFR-D [16] is a way to conduct depth-limited solving of a subgame with CFR when given a value function for PBSs. CFR-D is theoretically sound but has certain properties that may reduce performance in a self-play setting. CFR-AVG is a theoretically sound alternative to CFR-D that does not have these weaknesses. However, in order to implement CFR-AVG efficiently, in our experiments we modify the algorithm in a way that is not theoretically sound but empirically performs well in poker. Whether or not this modified form of CFR-AVG is theoretically sound remains an open question.
- **Connection between PBS gradients and infostate values.** Theorem 1 proves that all of the algorithms described in this paper can in theory be conducted using only V_1, not $\hat{v}$ (that is, a value function that outputs a single value for a PBS, rather than a vector of values for the infostates in the PBS). While this connection does not have immediate practical consequences, it does point toward a way of deploying the ideas in this paper to settings with billions or more infostates per PBS.
- **Fictitious Linear Optimistic Play (FLOP).** Section H introduces FLOP, a novel variant of Fictitious Play that is inspired by recent work on regret minimization algorithms [13]. We show that FLOP empirically achieves near-$O(\frac{1}{T})$ convergence in the limit in both poker and Liar's Dice, does far better than any previous variant of FP, and in some domains is a reasonable alternative to CFR.

B Pseudocode for ReBeL

Algorithm 2 presents ReBeL in more detail.

We define the average of two policies to be the policy that is, in expectation, identical to picking one of the two policies and playing that policy for the entire game. Formally, if $\pi = \alpha\pi_1 + (1-\alpha)\pi_2$, then $\pi(s_i) = \frac{\big(x_i^{\pi_1}(s_i)\alpha\big)\pi_1(s_i) + \big(x_i^{\pi_2}(s_i)(1-\alpha)\big)\pi_2(s_i)}{x_i^{\pi_1}(s_i)\alpha + x_i^{\pi_2}(s_i)(1-\alpha)}$ where $x_i^{\pi_1}(s_i)$ is the product of the probabilities for all agent i actions leading to s_i. Formally, $x_i^{\pi}(s_i)$ of infostate $s_i = (O_i^0, a_i^0, O_i^1, a_i^1, ..., O_i^t)$ is $x_i^{\pi}(s_i) = \Pi_t(a_i^t)$.

C Description of Games used for Evaluation

C.1 Heads-up no-limit Texas hold'em poker (HUNL)

HUNL is the two-player version of no-limit Texas hold'em poker, which is the most popular variant of poker in the world. For each "hand" (game) of poker, each player has some number of chips (the *stack*) in front of them. In our experiments, stack size varies during training between \$5,000 and \$25,000 but during testing is always \$20,000, as is standard in the AI research community. Before play begins, Player 1 commits a *small blind* of \$50 to the pot and Player 2 commits a *big blind* of \$100 to the pot.

Algorithm 2 ReBeL

```
function REBEL-LINEAR-CFR-D($\beta_r, \theta^v, \theta^\pi, D^v, D^\pi$)    ▷ $\beta_r$ is the current PBS
    while !ISTERMINAL($\beta_r$) do
        $G \leftarrow$ CONSTRUCTSUBGAME($\beta_r$)
        $\bar{\pi}, \pi^{t_{\text{warm}}} \leftarrow$ INITIALIZEPOLICY($G, \theta^\pi$)    ▷ $t_{\text{warm}} = 0$ and $\pi^0$ is uniform if no warm start
        $G \leftarrow$ SETLEAFVALUES($\beta_r, \pi^{t_{\text{warm}}}, \theta^v$)
        $v(\beta_r) \leftarrow$ COMPUTEEV($G, \pi^{t_{\text{warm}}}$)
        $t_{sample} \sim \text{linear}\{t_{\text{warm}} + 1, T\}$    ▷ Probability of sampling iteration $t$ is proportional to $t$
        for $t = (t_{\text{warm}} + 1)..T$ do
            if $t = t_{sample}$ then
                $\beta'_r \leftarrow$ SAMPLELEAF($G, \pi^{t-1}$)    ▷ Sample one or multiple leaf PBSs
            $\pi^t \leftarrow$ UPDATEPOLICY($G, \pi^{t-1}$)
            $\bar{\pi} \leftarrow \frac{t}{t+2}\bar{\pi} + \frac{2}{t+2}\pi^t$
            $G \leftarrow$ SETLEAFVALUES($\beta_r, \pi^t, \theta^v$)
            $v(\beta_r) \leftarrow \frac{t}{t+2}v(\beta_r) + \frac{2}{t+2}$ COMPUTEEV($G, \pi^t$)
        Add $\{\beta_r, v(\beta_r)\}$ to $D^v$    ▷ Add to value net training data
        for $\beta \in G$ do    ▷ Loop over the PBS at every public state in $G$
            Add $\{\beta, \bar{\pi}(\beta)\}$ to $D^\pi$    ▷ Add to policy net training data (optional)
        $\beta_r \leftarrow \beta'_r$

function SETLEAFVALUES($\beta, \pi, \theta^v$)
    if ISLEAF($\beta$) then
        for $s_i \in \beta$ do    ▷ For each infostate $s_i$ corresponding to $\beta$
            $v(s_i) = \hat{v}(s_i|\beta, \theta^v)$
    else
        for $a \in \mathcal{A}(\beta)$ do
            SETLEAFVALUES($\mathcal{T}(\beta, \pi, a), \pi, \theta^v$)

function SAMPLELEAF($G, \pi$)
    $i^* \sim \text{unif}\{1, N\}, h \sim \beta_r$ ▷ Sample a history randomly from the root PBS and a random player
    while !ISLEAF($h$) do
        $c \sim \text{unif}[0, 1]$
        for $i = 1..N$ do
            if $i == i^*$ and $c < \epsilon$ then    ▷ we set $\epsilon = 0.25$ during training, $\epsilon = 0$ at test time
                sample an action $a_i$ uniform random
            else
                sample an action $a_i$ according to $\pi_i(s_i(h))$
        $h \sim \tau(h, a)$
    return $\beta_h$    ▷ Return the PBS corresponding to leaf node $h$
```

Once players commit their blinds, they receive two private cards from a standard 52-card deck. The first of four rounds of betting then occurs. On each round of betting, players take turns deciding whether to fold, call, or raise. If a player folds, the other player receives the money in the pot and the hand immediately ends. If a player calls, that player matches the opponent's number of chips in the pot. If a player raises, that player adds more chips to the pot than the opponent. The initial raise of the round must be at least \$100, and every subsequent raise on the round must be at least as large as the previous raise. A player cannot raise more than either player's stack size. A round ends when both players have acted in the round and the most recent player to act has called. Player 1 acts first on the first round. On every subsequent round, player 2 acts first.

Upon completion of the first round of betting, three *community* cards are publicly revealed. Upon completion of the second betting round, another community card is revealed, and upon completion of the third betting round a final fifth community card is revealed. After the fourth betting round, if no player has folded, then the player with the best five-card poker hand, formed from the player's two private cards and the five community cards, is the winner and takes the money in the pot. In case of a tie, the money is split.

C.2 Turn endgame hold'em (TEH)

TEH is identical to HUNL except both players automatically call for the first two betting rounds, and there is an initial \$1,000 per player in the pot at the start of the third betting round. We randomize the stack sizes during training to be between \$5,000 and \$50,000 per player. The action space of TEH is reduced to at most three raise sizes ($0.5\times$ pot, $1\times$ pot, or all-in for the first raise in a round, and $0.75\times$ pot or all-in for subsequent raises), but the raise sizes for non-all-in raises are randomly perturbed by up to $\pm 0.1\times$ pot each game during training. Although we train on randomized stack sizes, bet sizes, and board cards, we measure exploitability on the case of both players having \$20,000, unperturbed bet sizes, and the first four board cards being $3\spadesuit 7\heartsuit T\diamondsuit K\spadesuit$. In this way we can train on a massive game while still measuring NashConv tractably. Even without the randomized stack and bet sizes, TEH has roughly $2 \cdot 10^{11}$ infostates.

C.3 Liar's Dice

Liar's Dice is a two-player zero-sum game in our experiments, though in general it can be played with more than two players. At the beginning of a game each player privately rolls d dice with f faces each. After that a betting stage starts where players take turns trying to predict how many dice of a specific kind there are among all the players, e.g., 4 dice with face 5. A player's bid must either be for more dice than the previous player's bid, or the same number of dice but a higher face. The round ends when a player challenges the previous bid (a call of *liar*). If all players together have at least as many dice of the specified face as was predicted by the last bid, then the player who made the bid wins. Otherwise the player who challenged the bid wins. We use the highest face as a *wild* face, i.e., dice with this face count towards a bid for any face.

D Domain Knowledge Leveraged in our Poker AI Agent

The most prominent form of domain knowledge in our ReBeL poker agent is the simplification of the action space during self play so that there are at most 8 actions at each decision point. The bet sizes are hand-chosen based on conventional poker wisdom and are fixed fractions of the pot, though each bet size is perturbed by $\pm 0.1\times$ pot during training to ensure diversity in the training data.

We specifically chose not to leverage domain knowledge that has been widely used in previous poker AI agents:

- All prior top poker agents, including DeepStack [40], Libratus [12], and Pluribus [14], have used *information abstraction* to bucket similar infostates together based on domain-specific features [30, 21, 6]. Even when computing an exact policy, such as during search or when solving a poker game in its entirety [24, 4], past agents have used *lossless abstraction* in which strategically identical infostates are bucketed together. For example, a flush of spades may be strategically identical to a flush of hearts.

  Our agent does not use any information abstraction, whether lossy or lossless. The agent computes a unique policy for each infostate. The agent's input to its value and policy network is a probability distribution over pairs of cards for each player, as well as all public board cards, the amount of money in the pot relative to the stacks of the players, and a flag for whether a bet has occurred on this betting round yet.

- DeepStack trained its value network on random PBSs. In addition to reducing the dimensionality of its value network input by using information abstraction, DeepStack also sampled PBSs according to a handcrafted algorithm that would sample more realistic PBSs compared to sampling uniform random. We show in Section 8 that training on PBSs sampled uniformly randomly without information abstraction results in extremely poor performance in a value network.

  Our agent collects training data purely from self play without any additional heuristics guiding which PBSs are sampled, other than an exploration hyperparameter that was set to $\epsilon = 0.25$ in all experiments.

- In cases where both players bet all their chips before all board cards are revealed, past poker AIs compute the exact expected value of all possible remaining board card outcomes. This is expensive to do in real time on earlier rounds, so past agents pre-compute this expected value and look it up during training and testing. Using the exact expected value reduces variance and makes learning easier.

  Our agent does not use this shortcut. Instead, the agent learns these "all-in" expected values on its own. When both agents have bet all their chips, the game proceeds as normal except neither player is allowed to bet.

- The search space in DeepStack [40] extends to the start of the next betting round, except for the third betting round (out of four) where it instead extends to the end of the game. Searching to the end of the game on the third betting round was made tractable by using information abstraction on the fourth betting round (see above). Similarly, Libratus [11], Modicum [15], and Pluribus [14] all search to the end of the game when on the third betting round. Searching to the end of the game has the major benefit of not requiring the value network to learn values for the end of the third betting round. Thus, instead of the game being three "levels" deep, it is only two levels deep. This reduces the potential for propogation of errors.

  Our agent always solves to the end of the current betting round, regardless of which round it is on.

- The depth-limited subgames in DeepStack extended to the start of the next betting round on the second betting round. On the first betting round, it extended to the end of the first betting round for most of training and to the start of the next betting round for the last several CFR iterations. Searching to the start of the next betting round was only tractable due to the abstractions mentioned previously and due to careful optimizations, such as implementing CFR on a GPU.

  Our agent always solves to the end of the current betting round regardless of which round it is on. We implement CFR only on a single-thread CPU and avoid any abstractions. Since a subgame starts at the beginning of a betting round and ends at the start of the next betting round, our agent must learn six "layers" of values (end of first round, start of second round, end of second round, start of third round, end of third round, start of fourth round) compared to three for DeepStack (end of first round, start of second round, start of third round).

- DeepStack used a separate value network for each of the three "layers" of values (end of first round, start of second round, start of third round). Our agent uses a single value network for all situations.

E Hyper parameters

In this section we provide details of the value and policy networks and the training procedures.

We approximate the value and policy functions using artificial neural networks. The input to the value network consists of three components for both games: agent index, representation of the public state, and a probability distribution over infostates for both agents. For poker, the public state representation consists of the board cards and the common pot size divided by stack size; for Liar's Dice it is the last bid and the acting agent. The output of the network is a vector of values for each possible infostate of the indexed agent, e.g., each possible poker hand she can hold.

We trained a policy network only for poker. The policy network state representation additionally contains pot size fractions for both agents separately as well as a flag for whether there have been any bets so far in the round. The output is a probability distribution over the legal actions for each infostate.

As explained in section 7 we use Multilayer perceptron with GeLU [27] activation functions and LayerNorm [3] for both value and policy networks.

For poker we represent the public state as a concatenation of a vector of indices of the board cards, current pot size relative to the stack sizes, and binary flag for the acting player. The size of the full input is

$$1(\text{agent index}) + 1(\text{acting agent}) + 1(\text{pot}) + 5(\text{board}) + 2 \times 1326(\text{infostate beliefs})$$

We use card embedding for the board cards similar to [7] and then apply MLP. Both the value and the policy networks contain 6 hidden layers with 1536 layers each. For all experiments we set the probability to explore a random action to $\epsilon = 25\%$ (see Section 5.2). To store the training data we use a simple circular buffer of size 12M and sample uniformly. Since our action abstraction contains at most 9 legal actions, the size of the target vector for the policy network is 9 times bigger than one used for the value network. In order to make it manageable, we apply linear quantization to the policy values. As initial data is produced with a random value network, we remove half of the data from the replay buffer after 20 epochs.

For the full game we train the network with Adam optimizer with learning rate 3×10^{-4} and halved the learning rate every 800 epochs. One epoch is 2,560,000 examples and the batch size 1024. We used 90 DGX-1 machines, each with $8 \times$ 32GB Nvidia V100 GPUs for data generation. We report results after 1,750 epochs. For TEH experiments we use higher initial learning rate 4×10^{-4}, but halve it every 100 epochs. We report results after 300 epochs.

For Liar's Dice we represent the state as a concatenation of a one hot vector for the last bid and binary flag for the acting player. The size of the full input is

$$1(\text{agent index}) + 1(\text{acting agent}) + n_{\text{dice}} n_{\text{faces}}(\text{last bid}) + 2 {n_{\text{faces}}}^{n_{\text{dice}}}(\text{infostate beliefs}).$$

The value network contains 2 hidden layers with 256 layers each. We train the network with Adam optimizer with learning rate 3×10^{-4} and halved the learning rate every 400 epochs. One epoch is 25,600 examples and the batch size 512. During both training and evaluation we run the search algorithm for 1024 iterations. We use single GPU for training and 60 CPU threads for data generation. We trained the network for 1000 epochs. To reduce the variance in RL+Search results, we evaluated the three last checkpoints and reported averages in table 2.

E.1 Human Experiments for HUNL

We evaluated our HUNL agent against Dong Kim, a top human professional specializing in HUNL. Kim was one of four humans that played against Libratus [12] in the man-machine competition which Libratus won. Kim lost the least to Libratus. However, due to high variance, it is impossible to statistically compare the performance of the individual humans that participated in the competition.

A total of 7,500 hands were played between Kim and the bot. Kim was able to play from home at his own pace on any schedule he wanted. He was also able to play up to four games simultaneously against the bot. To incentivize strong play, Kim was offered a base compensation of $\$1 \pm \$0.05x$ for each hand played, where x signifies his average win/loss rate in terms of big blinds per hundred hands played. Kim was guaranteed a minimum of \$0.75 per hand and could earn no more than \$2 per hand. Since final compensation was based on the variance-reduced score rather than the raw score, Kim was not aware of his precise performance during the experiment.

The bot played at an extremely fast pace. No decision required more than 5 seconds, and the bot on average plays faster than 2 seconds per hand in self play. To speed up play even further, the bot cached subgames it encountered on the preflop. When the same subgame was encountered again, it would simply reuse the solution it had already computed previously.

Kim's variance-reduced score, which we report in Section 8, was a loss of 165 ± 69 where the $\pm$ indicates one standard error. His raw score was a loss of 358 ± 188.

F Proof Related to Value Functions (Theorem 1)

We start by proving some preliminary Lemmas. For simplicity, we will sometimes prove results for only one player, but the results hold WLOG for both players.

For some policy profile $\pi = (\pi_1, \pi_2)$, let $v_i^\pi(\beta) : \mathcal{B} \to \mathbb{R}^{|S_i|}$ be a function that takes as input a PBS and outputs infostate values for player i.

Lemma 1. *Let $V_1^{\pi_2}(\beta)$ be player 1's value at β assuming that player 2 plays π_2 in a 2p0s game. $V_1^{\pi_2}(\beta)$ is linear in β_1.*

Proof. This follows directly from the definition of $v_i^\pi(s_i|\beta)$ along with the definition of V_1,

$$V_1^{\pi_2}(\beta) = \sum_{s_1 \in S_1(s_{\text{pub}})} \beta_1(s_1) v_1(s_1|\beta, (BR(\pi_2), \pi_2))$$

□

Lemma 2. $V_1(\beta) = \min_{\pi_2} V_1^{\pi_2}(\beta)$*, and the set of π_2 that attain $V_1(\beta)$ at β_0 are precisely the Nash equilibrium policies at β_0. This also implies that $V_1(\beta)$ is concave.*

Proof. By definition, the Nash equilibrium at β is the minimum among all choices of π_2 of the value to player 1 of her best response to π_2. Any π_2 that achieves this Nash equilibrium value when playing a best response is a Nash equilibrium policy.

From Lemma 1, we know that each $V_1^{\pi_2}(\beta)$ is linear, which implies that $V_1(\beta)$ is concave since any function that is the minimum of linear functions is concave. □

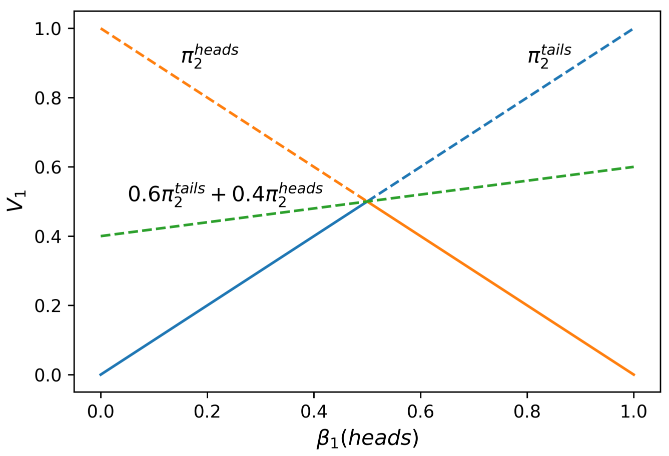

Figure 3: Illustration of Lemma 2. In this simple example, the subgame begins with some probability $\beta(heads)$ of a coin being heads-up, which player 1 observes. Player 2 then guesses if the coin is heads or tails, and wins if he guesses correctly. The payoffs for Player 2's pure strategies are shown as the lines marked π_2^{heads} and π_2^{tails}. The payoffs for a mixed strategy is a linear combination of the pure strategies. The value for player 1 is the minimum among all the lines corresponding to player 2 strategies, denoted by the solid lines.

Now we can turn to proving the Theorem.

Consider a function $\tilde{V}_1$ that is an extension of V_1 to unnormalized probability distributions over S_1 and S_2; i.e. $\tilde{V}_i((s_{pub}, b_1, b_2)) = V_i((s_{\text{pub}}, b_1/|b_1|_1, b_2/|b_2|_1))$. $\tilde{V}_i = V_i$ on the simplex of valid beliefs, but we extend it in this way to $\mathbb{R}_{\geq 0}^{|s_1|} \setminus \vec{0}$ so that we can consider gradients w.r.t. $p(s_1)$.

We will use the term 'supergradient' to be the equivalent of the subgradient for concave functions. Formally, g is a supergradient of concave function F at x_0 iff for any x in the domain of F,

$$F(x) - F(x_0) \leq g \cdot (x - x_0).$$

Also, $superg(F) = -subg(-F)$.

Theorem (Restatement of Theorem 1)**.** *For any belief $\beta = (\beta_1, \beta_2)$ (for the beliefs over player 1 and 2 infostates respectively) and any policy π^* that is a Nash equilibrium of the subgame rooted at β,*

$$v_1^{\pi^*}(s_1|\beta) = V_1(\beta) + \bar{g} \cdot \hat{s}_1 \tag{3}$$

for some supergradient $\bar{g}$ of $\tilde{V}_1(\beta)$ with respect to β_1, where $\hat{s}_1$ is the unit vector in direction s_1.

Proof. Lemma 2 shows that $V_1(\beta)$ is a concave function in β_1, and its extension $\tilde{V}$ off the simplex is constant perpendicular to the simplex, so $\tilde{V}$ is concave as well. Therefore the notion of a supergradient is well-defined.

Now, consider some policy profile $\pi^* = (\pi_1^*, \pi_2^*)$ that is a Nash equilibrium of $G(\beta_0)$. $V_1^{\pi_2^*}$ is a linear function and tangent to V_1 at β_0; therefore, its gradient is a supergradient of V_1 at β_0. Its gradient is

$$\nabla_{\beta_1} V_1^{\pi_2^*}(\beta) = \nabla_{\beta_1} \sum_{s_1 \in S_1(s_{\text{pub}})} \beta_1(s_1) v_1^{\pi_2^*}(s_1|\beta) \tag{4}$$

$$= \sum_{s_1 \in S_1(s_{\text{pub}})} \hat{s}_1 v_1^{\pi_2^*}(s_1|\beta) + \beta(s_1) \nabla_{\beta_1} v^{\pi_2^*}(s_1|\beta) \tag{5}$$

$$= \sum_{s_1 \in S_1(s_{\text{pub}})} \hat{s}_1 v_1^{\pi_2^*}(s_1|\beta) \tag{6}$$

Equation 6 follows from the fact that for a fixed opponent policy π_2^*, each player 1 infostate s_1 is an independent MDP whose value $v_1^{\pi_2^*}(s_1|\beta)$ doesn't depend on the probabilities β_1 of being in the different infostates (although it might depend on β_2 since player 1 doesn't observe these).

Note also that the gradient of $V_1^{\pi_2^*}$ is correct even when some infostates s_1 have probability 0, due to the fact that we defined $v_1^{\pi_2}(s_1|\beta)$ as the value of player 1 playing a *best response* to π_2 at each infostate s_1 (rather than just playing the equilibrium policy π_1^*, which may play arbitrarily at unvisited infostates).

Finally, let's compute $g \cdot \hat{s}_1$ at some β_1 on the simplex (i.e. $|\beta_1|_1 = 1$).

$$g = \nabla_{\beta_1/|\beta_1|_1} V_1^{\pi^*}(s_{\text{pub}}, \beta_1/|\beta_1|, \beta_2) \cdot \frac{d}{d\beta_1}\left(\frac{\beta_1}{|\beta_1|_1}\right) \qquad \text{(chain rule)} \tag{7}$$

$$= \left(\sum_{s_1' \in S_1(s_{\text{pub}})} \hat{s}_1' v_1^{\pi^*}(s_1'|\beta) \right) \cdot (|\beta_1|_1 - \beta_1)/(|\beta_1|_1)^2 \qquad \text{(Eq. 4)} \tag{8}$$

$$= \left(\sum_{s_1' \in S_1(s_{\text{pub}})} \hat{s}_1' v_1^{\pi^*}(s_1'|\beta) \right) \cdot (1 - \beta_1) \qquad (\text{since } |\beta_1|_1 = 1) \tag{9}$$

$$= \sum_{s_1' \in S_1(s_{\text{pub}})} \hat{s}_1' v_1^{\pi^*}(s_1'|\beta) - \sum_{s_1' \in S_1(s_{\text{pub}})} \beta_1(s_1') v_1^{\pi^*}(s_1'|\beta) \tag{10}$$

$$= \sum_{s_1' \in S_1(s_{\text{pub}})} \hat{s}_1' v_1^{\pi^*}(s_1'|\beta) - V_1(\beta) \tag{11}$$

$$g \cdot \hat{s}_1 = v_1^{\pi^*}(s_1|\beta) - V_1(\beta) \tag{12}$$

And we're done.

□

G Proofs Related to Subgame Solving (Theorems 2 and 3)

Lemma 3. *Running Algorithm 1 for $N \to \infty$ times in a depth-limited subgame rooted at PBS β_r will compute an infostate value vector $v_i^{\pi^*}(\beta_r)$ corresponding to the values of the infostates when π^**

is played in the (not depth-limited) subgame rooted at β_r, where π^* is a $\frac{C}{\sqrt{T}}$-Nash equilibrium for some constant C.

Proof. A key part of our proof is the insight that Algorithm 1 resembles the CFR-D algorithm from [16] if Algorithm 1 were modified such that there was no random sampling and every call to the value network was replaced with a recursive call to the CFR-D algorithm.

Suppose the subgame rooted at β_r extends to the end of the game and therefore there are no calls to the neural network. Then CFR is proven to compute a $\frac{C}{\sqrt{T}}$-Nash equilibrium [61], which we will call π^*, and Algorithm 1 will indeed learn a value vector $v_i^{\pi^*}(\beta_r)$ for β_r corresponding to the infostate values of β_r when π^* is played in the subgame. Thus, the base case for the inductive proof holds.

Now suppose we have a depth-limited subgame rooted at β_r. Assume that for every leaf PBS $\beta_z^{\pi^t}$ for iterations $t \leq T$, where π^t is the policy in the subgame on iteration t, that we have already computed an infostate value vector $v_i^{\pi^*}(\beta_z^{\pi^t})$ corresponding to the values of the infostates in $\beta_z^{\pi^t}$ when π^* is played in the subgame rooted at $\beta_z^{\pi^t}$, where π^* is a $\frac{C}{\sqrt{T}}$-Nash equilibrium for some constant C. Assume also that Lemma 3 holds for all leaf PBSs $\beta_z^{\pi^{T+1}}$.

First, for leaf PBSs that neither player reaches with positive probability, the values for the leaf PBS have no affect on CFR or the values computed for the root infostates because CFR weights the values by the probability the player reaches the PBS [61].

Now consider a leaf PBS $\beta_z^{\pi^{T+1}}$ that some player reaches with positive probability. Since $v_i^{\pi^*}(\beta_z^{\pi^t})$ has already been computed for all $\beta_z^{\pi^t}$ and since we are running a deterministic algorithm, so $v_i^{\pi^*}(\beta_z^{\pi^t})$ will not change with subsequent calls of Algorithm 1 on $\beta_z^{\pi^t}$ for all $t \leq T$. Thus, π^t will be the same for all $t \leq T$. Since Algorithm 1 samples a random CFR iteration, and since the leaf PBS $\beta_z^{\pi^{T+1}}$ is sampled with positive probability for some player when iteration $T+1$ is sampled, so the algorithm will sample $\beta_z^{\pi^{T+1}}$ N' times, where $N' \to \infty$ as $N \to \infty$. Since Lemma 3 holds for $\beta_z^{\pi^{T+1}}$, so eventually $v_i^{\pi^*}(\beta_z^{\pi^{T+1}})$ will be computed for $\beta_z^{\pi^{T+1}}$. Therefore, due to CFR-D [16], Lemma 3 will hold for β_r and the inductive step is proven. □

Theorem (Restatement of Theorem 2)**.** *Consider an idealized value approximator that returns the most recent sample of the value for sampled PBSs, and 0 otherwise. Running Algorithm 1 with T iterations of CFR in each subgame will produce a value approximator that produces values that correspond to a $\frac{C}{\sqrt{T}}$-equilibrium policy for any PBS that could be encountered during play, where C is a game-dependent constant.*

Proof. Since we run Algorithm 1 for $N \to \infty$ times at the root of the game, so by Lemma 3, Theorem 2 is true. □

Theorem (Restatement of Theorem 3)**.** *If Algorithm 1 is run at test time with no off-policy exploration, a value network that has error at most δ for any leaf PBS, and with T iterations of CFR being used to solve subgames, then the algorithm plays a $(\delta C_1 + \frac{\delta C_2}{\sqrt{T}})$-Nash equilibrium, where C_1, C_2 are game-specific constants.*

Proof. We prove the theorem inductively. Consider first a subgame near the end of the game that is not depth-limited. I.e., it has no leaf nodes. Clearly, the policy π^* that Algorithm 1 using CFR plays in expectation is a $\frac{k_1}{\sqrt{T}}$-Nash equilibrium for game-specific constant k_1 in this subgame.

Rather than play the average policy over all T iterations $\bar{\pi}^T$, one can equivalently pick a random iteration $t \sim \mathrm{uniform}\{1, T\}$ and play according to π^t, the policy on iteration t. This algorithm is also a $\frac{k_1}{\sqrt{T}}$-Nash equilibrium in expectation.

Next, consider a depth-limited subgame G such that for any leaf PBS β^t on any CFR iteration t, the policy that Algorithm 1 plays in the subgame rooted at β^t is in expectation a δ-Nash equilibrium in the subgame. If one computes a policy for G using tabular CFR-D [16] (or, as discussed in Section I, using CFR-AVG), then by Theorem 2 in [16], the average policy over all iterations is $k_2\delta + \frac{k_3}{\sqrt{T}}$-Nash equilibrium.

Just as before, rather than play according to this average policy $\bar{\pi}^T$, one can equivalently pick a random iteration $t \sim \text{uniform}\{1, T\}$ and play according to π^t. Doing so would also result in a $k_2\delta + \frac{k_3}{\sqrt{T}}$-Nash equilibrium in expectation. This is exactly what Algorithm 1 does.

Since there are a finite number of "levels" in a game, which is a game-specific constant, Algorithm 1 plays according to a $\delta C_1 + \frac{\delta C_2}{\sqrt{T}}$-Nash equilibrium.

□

H Fictitious Linear Optimistic Play

Fictitious Play (FP) [5] is an extremely simple iterative algorithm that is proven to converge to a Nash equilibrium in two-player zero-sum games. However, in practice it does so at an extremely slow rate. On the first iteration, all agents choose a uniform policy π_i^0 and the average policy $\bar{\pi}_i^0$ is set identically. On each subsequent iteration t, agents compute a best response to the other agents' average policy $\pi_i^t = \text{argmax}_{\pi_i} v_i(\pi_i, \bar{\pi}_{-i}^{t-1})$ and update their average policies to be $\bar{\pi}_i^t = \frac{t-1}{t}\bar{\pi}_i^{t-1} + \frac{1}{t}\pi_i^t$. As $t \to \infty$, $\bar{\pi}^t$ converges to a Nash equilibrium in two-player zero-sum games.

It has also been proven that a family of algorithms similar to FP known as **generalized weakened fictitious play (GWFP)** also converge to a Nash equilibrium so long as they satisfy certain properties [60, 38], mostly notably that in the limit the policies on each iteration converge to best responses.

In this section we introduce a novel variant of FP we call **Fictitious Linear Optimistic Play (FLOP)** which is a form of GWFP. FLOP is inspired by related variants in CFR, in particular Linear CFR [13]. FLOP converges to a Nash equilibrium much faster than FP while still being an extremely simple algorithm. However, variants of CFR such as Linear CFR and Discounted CFR [13] still converge much faster in most large-scale games.

In FLOP, the initial policy π_i^0 is uniform. On each subsequent iteration t, agents compute a best response to an *optimistic* [18, 47, 58] form of the opponent's average policy in which π_{-i}^{t-1} is given extra weight: $\pi_i^t = \text{argmax}_{\pi_i} v_i(\pi_i, \frac{t}{t+2}\bar{\pi}_{-i}^{t-1} + \frac{2}{t+2}\pi_{-i}^{t-1})$. The average policy is updated to be $\bar{\pi}_i^t = \frac{t-1}{t+1}\bar{\pi}_i^{t-1} + \frac{2}{t+1}\pi_i^t$. Theorem 4 proves that FLOP is a form of GWFP and therefore converges to a Nash equilibrium as $t \to \infty$.

Theorem 4. *FLOP is a form of Generalized Weakened Fictitious Play.*

Proof. Assume that the range of payoffs in the game is M. Since $\pi_i^t = \text{argmax}_{\pi_i} v_i(\pi_i, \frac{t}{t+2}\bar{\pi}_{-i}^{t-1} + \frac{2}{t+2}\pi_{-i}^{t-1})$, so π_i^t is an ϵ_t-best response to $\bar{\pi}_{-i}^{t-1}$ where $\epsilon_t < M\frac{2}{t+2}$ and $\epsilon_t \to 0$ as $t \to \infty$. Thus, FLOP is a form of GWFP with $\alpha_t = \frac{2}{t}$. □

I CFR-AVG: CFR Decomposition using Average Strategy

On each iteration t of CFR-D, the value of every leaf node z is set to $\hat{v}(s_i(z)|\beta_z^{\pi^t})$. Other than changing the values of leaf nodes every iteration, CFR-D is otherwise identical to CFR. If T iterations of CFR-D are conducted with a value network that has error at most δ for each infostate value, then $\bar{\pi}^T$ has exploitability of at most $k_1\delta + k_2/\sqrt{T}$ where k_1 and k_2 are game-specific constants [40].

Since it is the *average* policy profile $\bar{\pi}^t$, not π^t, that converges to a Nash equilibrium as $t \to \infty$, and since the leaf PBSs are set based on π^t, the input to the value network $\hat{v}$ may span the entire domain of inputs even as $t \to \infty$. For example, suppose in a Nash equilibrium π^* the probability distribution at $\beta_z^{\pi^*}$ was uniform. Then the probability distribution at $\beta_z^{\pi^t}$ for any individual iteration t could be *anything*, because regardless of what the probability distribution is, the average over all iterations could still be uniform in the end. Thus, $\hat{v}$ may need to be accurate over the entire domain of inputs rather than just the subspace near $\beta_z^{\pi^*}$.

In **CFR-AVG**, leaf values are instead set according to the *average policy* $\bar{\pi}^t$ on iteration t. When a leaf PBS is sampled, the leaf node is sampled with probability determined by π^t, but the PBS itself is defined using $\bar{\pi}^t$.

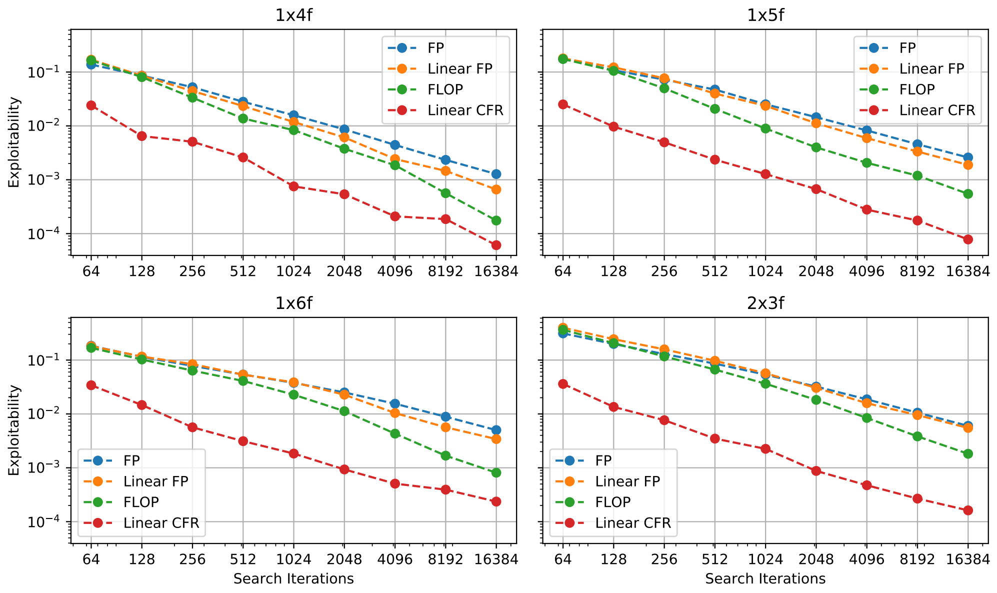

Figure 4: Exploitability of different algorithms of 4 variants of Liar's Dice: 1 die with 4, 5, or 6 faces and 2 dice with 3 faces. For all games FLOP outperforms Linear FP, but does not match the convergence of Linear CFR.

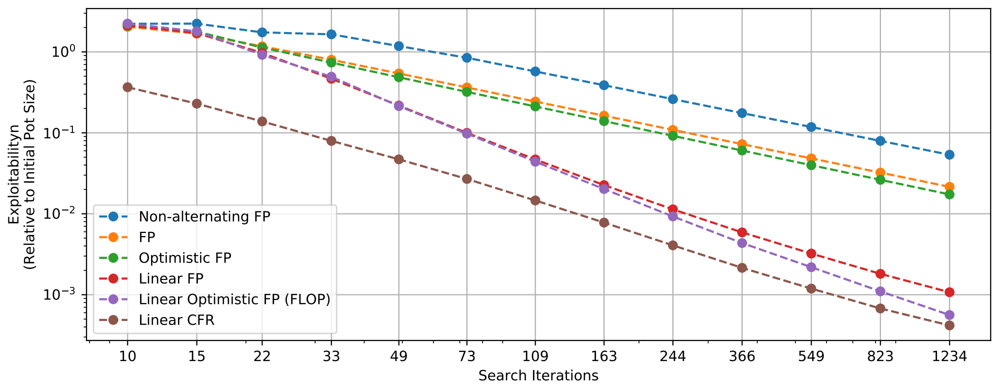

Figure 5: Exploitability of different algorithms for Turn Endgame Hold'em.

We first describe the tabular form of CFR-D [16]. Consider a game G' and a depth-limited subgame G, where both G' and G share a root but G extends only a limited number of actions into the future. Suppose that T iterations of a modified form of CFR are conducted in G'. On each iteration $t \leq T$, the policy $\pi(s_i)$ is set according to CFR for each $s_i \in G$. However, for every infostate $s'_i \in G' \setminus G$, the policy is set differently than what CFR would call for. At each leaf public state s'_{pub} of G, we solve a subgame rooted at $\beta^{\pi^t}_{s'_{\text{pub}}}$ by running T' iterations of CFR. For each s'_i in the subgame rooted at $\beta^{\pi^t}_{s'_{\text{pub}}}$, we set $\pi^t(s'_i) = \bar{\pi}^T(s'_i)$ (where $\pi^t(s'_i)$ is the policy for the infostate in G and $\bar{\pi}^T(s'_i)$ is the policy for the infostate in the subgame rooted at $\beta^{\pi^t}_{s'_{\text{pub}}}$). It is proven that as $T' \to \infty$, CFR-D converges to a $O(\frac{1}{\sqrt{T}})$-Nash equilibrium [16].

CFR-AVG is identical to CFR-D, except the subgames that are solved on each iteration t are rooted at $\beta^{\bar{\pi}^t}_{s'_{\text{pub}}}$ rather than $\beta^{\pi^t}_{s'_{\text{pub}}}$. Theorem 5 proves that CFR-AVG achieves the same bound on convergence to a Nash equilibrium as CFR-D.

Theorem 5. *Suppose that T iterations of CFR-AVG are run in a depth-limited subgame, where on each iteration $t \leq T$ the subgame rooted at each leaf PBS $\beta^{\bar{\pi}^t}_{s'_{pub}}$ is solved completely. Then $\bar{\pi}^T$ is a $\frac{C}{\sqrt{T}}$-Nash equilibrium for a game-specific constant C.*

CFR-AVG has a number of potential benefits over CFR-D:

- Since $\bar{\pi}^t$ converges to a Nash equilibrium as $t \to \infty$, CFR-AVG allows $\hat{v}$ to focus on being accurate over a more narrow subspace of inputs.
- When combined with a policy network (as introduced in Section 5.3), CFR-AVG may allow $\hat{v}$ to focus on an even more narrow subspace of inputs.
- Since $\bar{\pi}^{t+1}$ is much closer to $\bar{\pi}^t$ than π^{t+1} is to π^t, in practice as t becomes large one can avoid querying the value network on every iteration and instead recycle the values from a previous iteration. This may be particularly valuable for Monte Carlo versions of CFR.

While CFR-AVG is theoretically sound, we modify its implementation in our experiments to make it more efficient in a way that has not been proven to be theoretically sound. The reason for this is that while the *input* to the value network is $\beta^{\bar{\pi}^t}_{s'_{\text{pub}}}$ (i.e., the leaf PBS corresponding to $\bar{\pi}^t$ being played in G, the *output* needs to be the value of each infostate s_i given that π^t is played in G. Thus, unlike CFR-D and FP, in CFR-AVG there is a mismatch between the input policy and the output policy.

One way to cope with this is to have the input consist of both $\beta^{\bar{\pi}^t}_{s'_{\text{pub}}}$ and $\beta^{\pi^t}_{s'_{\text{pub}}}$. However, we found this performed relatively poorly in preliminary experiments when trained through self play.

Instead, on iteration $t-1$ we store the output from $\hat{v}(s_i|\beta^{\bar{\pi}^{t-1}}_{s'_{\text{pub}}})$ for each s_i and on iteration t we set $v^t(s_i)$ to be $t\hat{v}(s_i|\beta^{\bar{\pi}^t}_{s'_{\text{pub}}}) - (t-1)\hat{v}(s_i|\beta^{\bar{\pi}^{t-1}}_{s'_{\text{pub}}})$ (in vanilla CFR). The motivation for this is that $\pi^t = t\bar{\pi}^t - (t-1)\bar{\pi}^{t-1}$. If $v^t(h) = v^{t-1}(h)$ for each history h in the leaf PBS, then this modification of CFR-AVG is sound. Since $v^t(h) = v^{t-1}(h)$ when h is a full-game terminal node (i.e., it has no actions), this modified form of CFR-AVG is identical to CFR in a non-depth-limited game. However, that is not the case in a depth-limited subgame, and it remains an open question whether this modified form of CFR-AVG is theoretically sound in depth-limited subgames. Empirically, however, we found that it converges to a Nash equilibrium in turn endgame hold'em for every set of parameters (e.g., bet sizes, stack sizes, and initial beliefs) that we tested.

Figure 6 shows the performance of CFR-D, CFR-AVG, our modified form of CFR-AVG, and FP in TEH when using an oracle function for the value network. It also shows the performance of CFR-D, our modified form of CFR-AVG, and FP in TEH when using a value network trained through self-play. Surprisingly, the theoretically sound form of CFR-AVG does worse than CFR-D when using an oracle function. However, the modified form of CFR-AVG does better than CFR-D when using an oracle function and also when trained through self play.

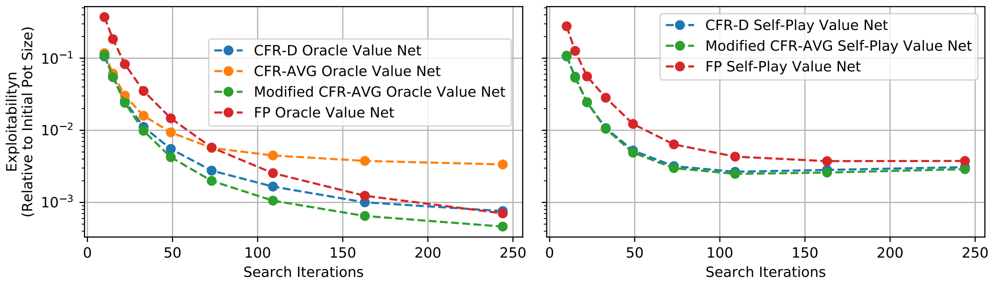

Figure 6: **Left:** comparison of CFR-D, CFR-AVG, modified CFR-AVG, and FP using an oracle value network which returns exact values for leaf PBSs. **Right:** comparison of CFR-D, modified CFR-AVG, and FP using a value network learned through 300 epochs of self play.

We also trained a model on HUNL with training parameters that were identical to the one reported in Section 8, but using CFR-D rather than CFR-AVG. That model lost to BabyTartanian8 by 10 ± 3 whereas the CFR-AVG model won by 9 ± 4. The CFR-D model also beat Slumbot by 39 ± 6 whereas the CFR-AVG model won by 45 ± 5.

I.1 Proof of Theorem 5

Our proof closely follows from [16] and [40].

Proof. Let $R^t(s_i)$ be the (cumulative) regret of infostates s_i on iteration t. We show that the regrets of all infostates in G' are bounded by $O(\sqrt{T})$ and therefore the regret of the entire game is bounded by $O(\sqrt{T})$.

First, consider the infostates in G. Since their policies are chosen according to CFR each iteration, their regrets are bounded by $O(\sqrt{T})$ regardless of the policies played in descendant infostates.

Next consider an infostate $s_i \in G' \setminus G$. We prove inductively that $R^t(s_i) \leq 0$. Let β^{π^t} be the PBS at the root of the subgame containing s_i in CFR-D, and $\beta^{\bar{\pi}^t}$ be the PBS at the root of the subgame containing s_i in CFR-AVG. On the first iteration, $\beta^{\pi^t} = \beta^{\bar{\pi}^t}$. Since we assume CFR-AVG computes an exact equilibrium in the subgame rooted at $\beta^{\bar{\pi}^t} = \beta^{\pi^t}$, so $R^t(s_i) = 0$ on the first iteration.

Next, we prove $R^{t+1}(s_i) \leq R^t(s_i)$. We define $a^{*,t}$ as

$$a_i^{*,t} = \operatorname*{argmax}_{a_i} \sum_{t'=0}^{t} v^{t'}(s_i, a_i) \tag{13}$$

By definition of regret,

$$R^{t+1}(s_i) = \sum_{t'=0}^{t+1} \big(v^{t'}(s_i, a_i^{*,t+1}) - v^{t'}(s_i)\big) \tag{14}$$

Separating iteration $t+1$ from the summation we get

$$R^{t+1}(s_i) = \sum_{t'=0}^{t} \big(v^{t'}(s_i, a_i^{*,t+1}) - v^{t'}(s_i)\big) + \big(v^{t+1}(s_i, a_i^{*,t+1}) - v^{t+1}(s_i)\big) \tag{15}$$

By definition of $a_i^{*,t}$ we know $\sum_{t'=0}^{t} v^{t'}(s_i, a_i^{*,t+1}) \leq \sum_{t'=0}^{t} v^{t'}(s_i, a_i^{*,t})$, so

$$R^{t+1}(s_i) \leq \sum_{t'=0}^{t} \big(v^{t'}(s_i, a_i^{*,t}) - v^{t'}(s_i)\big) + \big(v^{t+1}(s_i, a_i^{*,t+1}) - v^{t+1}(s_i)\big) \tag{16}$$

Since $\sum_{t'=0}^{t} \big(v^{t'}(s_i, a_i^{*,t}) - v^{t'}(s_i)\big)$ is the definition of $R^t(s_i)$ we get

$$R^{t+1}(s_i) \leq R^t(s_i) + \big(v^{t+1}(s_i, a_i^{*,t+1}) - v^{t+1}(s_i)\big) \tag{17}$$

Since $\pi^{t+1} = \pi^{*,t+1}$ in the subgame where $\pi^{*,t+1}$ is an exact equilibrium of the subgame rooted at $\beta^{\bar{\pi}^{t+1}}$, so π^{t+1} is a best response to $\bar{\pi}^{t+1}$ in the subgame and therefore $v^{t+1}(s_i, a_i^{*,t+1}) = v^{t+1}(s_i)$. Thus,

$$R^{t+1}(s_i) \leq R^t(s_i) \tag{18}$$

$\square$

J CFR Warm Start Algorithm Used

Our warm start technique for CFR is based on [10], which requires only a policy profile to warm start CFR soundly. That techniques computes a "soft" best response to the policy profile, which results in instantaneous regrets for each infostate. Those instantaneous regrets are scaled up to be equivalent to some number of CFR iterations. However, that technique requires careful parameter tuning to achieve good performance in practice.

We instead use a simplified warm start technique in which an exact best response to the policy profile is computed. That best response results in instantaneous regrets at each infostate. Those regrets are scaled up by a factor of 15 to imitate 15 CFR iterations. Similarly, the average policy effectively assumes that the warm start policy was played for the first 15 iterations of CFR. CFR then proceeds as if 15 iterations have already occurred.