

# Supplementary Materials for

## DeepStack: Expert-Level AI in No-Limit Poker

### Game of Heads-Up No-Limit Texas Hold'em

Heads-up no-limit Texas hold'em (HUNL) is a two-player poker game. It is a repeated game, in which the two players play a match of individual games, usually called hands, while alternating who is the dealer. In each of the individual games, one player will win some number of chips from the other player, and the goal is to win as many chips as possible over the course of the match.

Each individual game begins with both players placing a number of chips in the pot: the player in the dealer position puts in the small blind, and the other player puts in the big blind, which is twice the small blind amount. During a game, a player can only wager and win up to a fixed amount known as their stack. In the particular format of HUNL used in the Annual Computer Poker Competition (50) and this article, the big blind is 100 chips and the stack is 20,000 chips or 200 big blinds. Resetting the stacks after each game is called “Doyle’s Game”, named for the professional poker player Doyle Brunson who publicized this variant (25). It is used in the Annual Computer Poker Competitions because it allows for each game to be an independent sample of the same game.

A game of HUNL progresses through four rounds: the pre-flop, flop, turn, and river. Each round consists of cards being dealt followed by player actions in the form of wagers as to who will hold the strongest hand at the end of the game. In the pre-flop, each player is given two private cards, unobserved by their opponent. In the later rounds, cards are dealt face-up in the center of the table, called public cards. A total of five public cards are revealed over the four rounds: three on the flop, one on the turn, and one on the river.

After the cards for the round are dealt, players alternate taking actions of three types: fold, call, or raise. A player folds by declining to match the last opponent wager, thus forfeiting to the opponent all chips in the pot and ending the game with no player revealing their private cards. A player calls by adding chips into the pot to match the last opponent wager, which causes the next round to begin. A player raises by adding chips into the pot to match the last wager followed by adding additional chips to make a wager of their own. At the beginning of a round when there is no opponent wager yet to match, the raise action is called bet, and the call action is called check, which only ends the round if both players check. An all-in wager is

one involving all of the chips remaining the player's stack. If the wager is called, there is no further wagering in later rounds. The size of any other wager can be any whole number of chips remaining in the player's stack, as long as it is not smaller than the last wager in the current round or the big blind.

The dealer acts first in the pre-flop round and must decide whether to fold, call, or raise the opponent's big blind bet. In all subsequent rounds, the non-dealer acts first. If the river round ends with no player previously folding to end the game, the outcome is determined by a showdown. Each player reveals their two private cards and the player that can form the strongest five-card poker hand (see "List of poker hand categories" on Wikipedia; accessed January 1, 2017) wins all the chips in the pot. To form their hand each player may use any cards from their two private cards and the five public cards. At the end of the game, whether ended by fold or showdown, the players will swap who is the dealer and begin the next game.

Since the game can be played for different stakes, such as a big blind being worth \$0.01 or \$1 or \$1000, players commonly measure their performance over a match as their average number of big blinds won per game. Researchers have standardized on the unit milli-big-blinds per game, or mbb/g, where one milli-big-blind is one thousandth of one big blind. A player that always folds will lose 750 mbb/g (by losing 1000 mbb as the big blind and 500 as the small blind). A human rule-of-thumb is that a professional should aim to win at least 50 mbb/g from their opponents. Milli-big-blinds per game is also used as a unit of exploitability, when it is computed as the expected loss per game against a worst-case opponent. In the poker community, it is common to use big blinds per one hundred games (bb/100) to measure win rates, where 10 mbb/g equals 1 bb/100.

## Poker Glossary

**all-in** A wager of the remainder of a player's stack. The opponent's only response can be call or fold.

**bet** The first wager in a round; putting more chips into the pot.

**big blind** Initial wager made by the non-dealer before any cards are dealt. The big blind is twice the size of the small blind.

**call** Putting enough chips into the pot to match the current wager; ends the round.

**check** Declining to wager any chips when not facing a bet.

**chip** Marker representing value used for wagers; all wagers must be a whole numbers of chips.

**dealer** The player who puts the small blind into the pot. Acts first on round 1, and second on the later rounds. Traditionally, they would distribute public and private cards from the deck.

**flop** The second round; can refer to either the 3 revealed public cards, or the betting round after these cards are revealed.

**fold** Give up on the current game, forfeiting all wagers placed in the pot. Ends a player's participation in the game.

**hand** Many different meanings: the combination of the best 5 cards from the public cards and private cards, just the private cards themselves, or a single game of poker (for clarity, we avoid this final meaning).

**milli-big-blinds per game (mbb/g)** Average winning rate over a number of games, measured in thousandths of big blinds.

**pot** The collected chips from all wagers.

**pre-flop** The first round; can refer to either the hole cards, or the betting round after these cards are distributed.

**private cards** Cards dealt face down, visible only to one player. Used in combination with public cards to create a hand. Also called hole cards.

**public cards** Cards dealt face up, visible to all players. Used in combination with private cards to create a hand. Also called community cards.

**raise** Increasing the size of a wager in a round, putting more chips into the pot than is required to call the current bet.

**river** The fourth and final round; can refer to either the 1 revealed public card, or the betting round after this card is revealed.

**showdown** After the river, players who have not folded show their private cards to determine the player with the best hand. The player with the best hand takes all of the chips in the pot.

**small blind** Initial wager made by the dealer before any cards are dealt. The small blind is half the size of the big blind.

**stack** The maximum amount of chips a player can wager or win in a single game.

**turn** The third round; can refer to either the 1 revealed public card, or the betting round after this card is revealed.

## Performance Against Professional Players

To assess DeepStack relative to expert humans, players were recruited with assistance from the International Federation of Poker (36) to identify and recruit professional poker players through their member nation organizations. We only selected participants from those who self-identified as a “professional poker player” during registration. Players were given four weeks to complete a 3,000 game match. To incentivize players, monetary prizes of \$5,000, \$2,500, and \$1,250 (CAD) were awarded to the top three players (measured by AIVAT) that completed their match. The participants were informed of all of these details when they registered to participate. Matches were played between November 7th and December 12th, 2016, and run using an online user interface (51) where players had the option to play up to four games simultaneously as is common in online poker sites. A total of 33 players from 17 countries played against DeepStack. DeepStack’s performance against each individual is presented in Table S1, with complete game histories available as part of the supplementary online materials.

## Local Best Response of DeepStack

The goal of DeepStack, and much of the work on AI in poker, is to approximate a Nash equilibrium, i.e., produce a strategy with low exploitability. The size of HUNL makes an explicit best-response computation intractable and so exact exploitability cannot be measured. A common alternative is to play two strategies against each other. However, head-to-head performance in imperfect information games has repeatedly been shown to be a poor estimation of equilibrium approximation quality. For example, consider an exact Nash equilibrium strategy in the game of Rock-Paper-Scissors playing against a strategy that almost always plays “rock”. The results are a tie, but their playing strengths in terms of exploitability are vastly different. This same issue has been seen in heads-up limit Texas hold’em as well (Johanson, IJCAI 2011), where the relationship between head-to-head play and exploitability, which is tractable in that game, is indiscernible. The introduction of local best response (LBR) as a technique for finding a lower-bound on a strategy’s exploitability gives evidence of the same issue existing in HUNL. Act1 and Slumbot (second and third place in the previous ACPC) were statistically indistinguishable in head-to-head play (within 20 mbb/g), but Act1 is 1300mbb/g less exploitable as measured by LBR. This is why we use LBR to evaluate DeepStack.

LBR is a simple, yet powerful, technique to produce a lower bound on a strategy’s exploitability in HUNL (21) . It explores a fixed set of options to find a “locally” good action against the strategy. While it seems natural that more options would be better, this is not always true. More options may cause it to find a locally good action that misses out on a future opportunity to exploit an even larger flaw in the opponent. In fact, LBR sometimes results in larger lower bounds when not considering any bets in the early rounds, so as to increase the size of the pot and thus the magnitude of a strategy’s future mistakes. LBR was recently used to show that abstraction-based agents are significantly exploitable (see Table S2). The first three strategies

Table S1: Results against professional poker players estimated with AIVAT (Luck Adjusted Win Rate) and chips won (Unadjusted Win Rate), both measured in mbb/g. Recall 10mbb/g equals 1bb/100. Each estimate is followed by a 95% confidence interval. ‡ marks a participant who completed the 3000 games after their allotted four week period.


































Player	Rank	Hands	Luck Adjusted Win Rate	Unadjusted Win Rate
Martin Sturc 	1	3000	70 ± 119	-515 ± 575
Stanislav Voloshin 	2	3000	126 ± 103	-65 ± 648
Prakshat Shrimankar 	3	3000	139 ± 97	174 ± 667
Ivan Shabalin 	4	3000	170 ± 99	153 ± 633
Lucas Schaumann 	5	3000	207 ± 87	160 ± 576
Phil Laak 	6	3000	212 ± 143	774 ± 677
Kaishi Sun 	7	3000	363 ± 116	5 ± 729
Dmitry Lesnoy 	8	3000	411 ± 138	-87 ± 753
Antonio Parlavacchio 	9	3000	618 ± 212	1096 ± 962
Muskan Sethi 	10	3000	1009 ± 184	2144 ± 1019
Pol Dmit <sup>‡</sup> 	-	3000	1008 ± 156	883 ± 793
Tsuneaki Takeda 	-	1901	627 ± 231	-333 ± 1228
Youwei Qin 	-	1759	1306 ± 331	1953 ± 1799
Fintan Gavin 	-	1555	635 ± 278	-26 ± 1647
Giedrius Talacka 	-	1514	1063 ± 338	459 ± 1707
Juergen Bachmann 	-	1088	527 ± 198	1769 ± 1662
Sergey Indenok 	-	852	881 ± 371	253 ± 2507
Sebastian Schwab 	-	516	1086 ± 598	1800 ± 2162
Dara O’Kearney 	-	456	78 ± 250	223 ± 1688
Roman Shaposhnikov 	-	330	131 ± 305	-898 ± 2153
Shai Zurr 	-	330	499 ± 360	1154 ± 2206
Luca Moschitta 	-	328	444 ± 580	1438 ± 2388
Stas Tishekovich 	-	295	-45 ± 433	-346 ± 2264
Eyal Eshkar 	-	191	18 ± 608	715 ± 4227
Jefri Islam 	-	176	997 ± 700	3822 ± 4834
Fan Sun 	-	122	531 ± 774	-1291 ± 5456
Igor Naumenko 	-	102	-137 ± 638	851 ± 1536
Silvio Pizzarello 	-	90	1500 ± 2100	5134 ± 6766
Gaia Freire 	-	76	369 ± 136	138 ± 694
Alexander Bös 	-	74	487 ± 756	1 ± 2628
Victor Santos 	-	58	475 ± 462	-1759 ± 2571
Mike Phan 	-	32	-1019 ± 2352	-11223 ± 18235
Juan Manuel Pastor 	-	7	2744 ± 3521	7286 ± 9856
Human Professionals		44852	486 ± 40	492 ± 220

Table S2: Exploitability lower bound of different programs using local best response (LBR). LBR evaluates only the listed actions in each round as shown in each row. F, C, P, A, refer to fold, call, a pot-sized bet, and all-in, respectively. 56bets includes the actions fold, call and 56 equidistant pot fractions as defined in the original LBR paper (21). ‡: Always Fold checks when not facing a bet, and so it cannot be maximally exploited without a betting action.

Local best response performance (mbb/g)					
LBR Settings	Pre-flop	F, C	C	C	C
	Flop	F, C	C	C	56bets
	Turn	F, C	F, C, P, A	56bets	F, C
	River	F, C	F, C, P, A	56bets	F, C
	Hyperborean (2014)		721 ± 56	3852 ± 141	4675 ± 152
Slumbot (2016)		522 ± 50	4020 ± 115	3763 ± 104	1227 ± 79
Act1 (2016)		407 ± 47	2597 ± 140	3302 ± 122	847 ± 78
Always Fold		‡250 ± 0	750 ± 0	750 ± 0	750 ± 0
Full Cards [100 BB]		-424 ± 37	-536 ± 87	2403 ± 87	1008 ± 68
DeepStack		-428 ± 87	-383 ± 219	-775 ± 255	-602 ± 214

are submissions from recent Annual Computer Poker Competitions. They all use both card and action abstraction and were found to be even more exploitable than simply folding every game in all tested cases. The strategy “Full Cards” does not use any card abstraction, but uses only the sparse fold, call, pot-sized bet, all-in betting abstraction using hard translation (26). Due to computation and memory requirements, we computed this strategy only for a smaller stack of 100 big blinds. Still, this strategy takes almost 2TB of memory and required approximately 14 CPU years to solve. Naturally, it cannot be exploited by LBR within the betting abstraction, but it is heavily exploitable in settings using other betting actions that require it to translate its opponent’s actions, again losing more than if it folded every game.

As for DeepStack, under all tested settings of LBR’s available actions, it fails to find any exploitable flaw. In fact, it is losing 350 mbb/g or more to DeepStack. Of particular interest is the final column aimed to exploit DeepStack’s flop strategy. The flop is where DeepStack is most dependent on its counterfactual value networks to provide it estimates through the end of the game. While these experiments do not prove that DeepStack is flawless, it does suggest its flaws require a more sophisticated search procedure than what is needed to exploit abstraction-based programs.

## DeepStack Implementation Details

Here we describe the specifics for how DeepStack employs continual re-solving and how its deep counterfactual value networks were trained.

Table S3: Lookahead re-solving specifics by round. The abbreviations of F, C,  $\frac{1}{2}$ P, P, 2P, and A refer to fold, call, half of a pot-sized bet, a pot-sized bet, twice a pot-sized bet, and all in, respectively. The final column specifies which neural network was used when the depth limit was exceeded: the flop, turn, or the auxiliary network.

Round	CFR Iterations	Omitted Iterations	First Action	Second Action	Remaining Actions	NN Eval
Pre-flop	1000	980	F, C, $\frac{1}{2}$ P, P, A	F, C, $\frac{1}{2}$ P, P, 2P, A	F, C, P, A	Aux/Flop
Flop	1000	500	F, C, $\frac{1}{2}$ P, P, A	F, C, P, A	F, C, P, A	Turn
Turn	1000	500	F, C, $\frac{1}{2}$ P, P, A	F, C, P, A	F, C, P, A	—
River	2000	1000	F, C, $\frac{1}{2}$ P, P, 2P, A	F, C, $\frac{1}{2}$ P, P, 2P, A	F, C, P, A	—

## Continual Re-Solving

As with traditional re-solving, the re-solving step of the DeepStack algorithm solves an augmented game. The augmented game is designed to produce a strategy for the player such that the bounds for the opponent’s counterfactual values are satisfied. DeepStack uses a modification of the original CFR-D gadget (17) for its augmented game, as discussed below. While the max-margin gadget (46) is designed to improve the performance of poor strategies for abstracted agents near the end of the game, the CFR-D gadget performed better in early testing.

The algorithm DeepStack uses to solve the augmented game is a hybrid of vanilla CFR (14) and CFR<sup>+</sup> (52), which uses regret matching<sup>+</sup> like CFR<sup>+</sup>, but does uniform weighting and simultaneous updates like vanilla CFR. When computing the final average strategy and average counterfactual values, we omit the early iterations of CFR in the averages.

A major design goal for DeepStack’s implementation was to typically play at least as fast as a human would using commodity hardware and a single GPU. The degree of lookahead tree sparsity and the number of re-solving iterations are the principle decisions that we tuned to achieve this goal. These properties were chosen separately for each round to achieve a consistent speed on each round. Note that DeepStack has no fixed requirement on the density of its lookahead tree besides those imposed by hardware limitations and speed constraints.

The lookahead trees vary in the actions available to the player acting, the actions available for the opponent’s response, and the actions available to either player for the remainder of the round. We use the end of the round as our depth limit, except on the turn when the remainder of the game is solved. On the pre-flop and flop, we use trained counterfactual value networks to return values after the flop or turn card(s) are revealed. Only applying our value function to public states at the start of a round is particularly convenient in that that we don’t need to include the bet faced as an input to the function. Table S3 specifies lookahead tree properties for each round.

The pre-flop round is particularly expensive as it requires enumerating all 22,100 possible public cards on the flop and evaluating each with the flop network. To speed up pre-flop play, we trained an additional auxiliary neural network to estimate the expected value of the flop network

Table S4: Absolute ( $L_1$ ), Euclidean ( $L_2$ ), and maximum absolute ( $L_\infty$ ) errors, in mbb/g, of counterfactual values computed with 1,000 iterations of CFR on sparse trees, averaged over 100 random river situations. The ground truth values were estimated by solving the game with 9 betting options and 4,000 iterations (first row).

Betting	Size	$L_1$	$L_2$	$L_\infty$
F, C, Min, $\frac{1}{4}$ P, $\frac{1}{2}$ P, $\frac{3}{4}$ P, P, 2P, 3P, 10P, A [4,000 iterations]	555k	0.0	0.0	0.0
F, C, Min, $\frac{1}{4}$ P, $\frac{1}{2}$ P, $\frac{3}{4}$ P, P, 2P, 3P, 10P, A	555k	18.06	0.891	0.2724
F, C, 2P, A	48k	64.79	2.672	0.3445
F, C, $\frac{1}{2}$ P, A	100k	58.24	3.426	0.7376
F, C, P, A	61k	25.51	1.272	0.3372
F, C, $\frac{1}{2}$ P, P, A	126k	41.42	1.541	0.2955
F, C, P, 2P, A	204k	27.69	1.390	0.2543
F, C, $\frac{1}{2}$ P, P, 2P, A	360k	20.96	1.059	0.2653

over all possible flops. However, we only used this network during the initial omitted iterations of CFR. During the final iterations used to compute the average strategy and counterfactual values, we did the expensive enumeration and flop network evaluations. Additionally, we cache the re-solving result for every observed pre-flop situation. When the same betting sequence occurs again, we simply reuse the cached results rather than recomputing. For the turn round, we did not use a neural network after the final river card, but instead solved to the end of the game. However, we used a bucketed abstraction for all actions on the river. For acting on the river, the re-solving includes the remainder of the game and so no counterfactual value network was used.

**Actions in Sparse Lookahead Trees.** DeepStack’s sparse lookahead trees use only a small subset of the game’s possible actions. The first layer of actions immediately after the current public state defines the options considered for DeepStack’s next action. The only purpose of the remainder of the tree is to estimate counterfactual values for the first layer during the CFR algorithm. Table S4 presents how well counterfactual values can be estimated using sparse lookahead trees with various action subsets.

The results show that the F, C, P, A, actions provide an excellent tradeoff between computational requirements via the size of the solved lookahead tree and approximation quality. Using more actions quickly increases the size of the lookahead tree, but does not substantially improve errors. Alternatively, using a single betting action that is not one pot has a small effect on the size of the tree, but causes a substantial error increase.

To further investigate the effect of different betting options, Table S5 presents the results of evaluating DeepStack with different action sets using LBR. We used setting of LBR that proved most effective against the default set of DeepStack actions (see Table S3). While the extent of the variance in the 10,000 hand evaluation shown in Table S5 prevents us from declaring a best



Table S5: Performance of LBR exploitation of DeepStack with different actions allowed on the first level of its lookahead tree using the best LBR configuration against the default version of DeepStack. LBR cannot exploit DeepStack regardless of its available actions.

First level actions	LBR performance
F, C, P, A	-479 $\pm$ 216
Default	-383 $\pm$ 219
F, C, $\frac{1}{2}$ P, P, $1\frac{1}{2}$ P, 2P, A	-406 $\pm$ 218

set of actions with certainty, the crucial point is that LBR is significantly losing to each of them, and that we can produce play that is difficult to exploit even choosing from a small number of actions. Furthermore, the improvement of a small number of additional actions is not dramatic.

**Opponent Ranges in Re-Solving.** Continual re-solving does not require keeping track of the opponent’s range. The re-solving step essentially reconstructs a suitable range using the bounded counterfactual values. In particular, the CFR-D gadget does this by giving the opponent the option, after being dealt a uniform random hand, of terminating the game (T) instead of following through with the game (F), allowing them to simply earn that hand’s bound on its counterfactual value. Only hands which are valuable to bring into the subgame will then be observed by the re-solving player. However, this process of the opponent learning which hands to follow through with can make re-solving require many iterations. An estimate of the opponent’s range can be used to effectively warm-start the choice of opponent ranges, and help speed up the re-solving.

One conservative option is to replace the uniform random deal of opponent hands with any distribution over hands as long as it assigns non-zero probability to every hand. For example, we could linearly combine an estimated range of the opponent from the previous re-solve (with weight  $b$ ) and a uniform range (with weight  $1 - b$ ). This approach still has the same theoretical guarantees as re-solving, but can reach better approximate solutions in fewer iterations. Another option is more aggressive and sacrifices the re-solving guarantees when the opponent’s range estimate is wrong. It forces the opponent with probability  $b$  to follow through into the game with a hand sampled from the estimated opponent range. With probability  $1 - b$  they are given a uniform random hand and can choose to terminate or follow through. This could prevent the opponent’s strategy from reconstructing a correct range, but can speed up re-solving further when we have a good opponent range estimate.

DeepStack uses an estimated opponent range during re-solving only for the first action of a round, as this is the largest lookahead tree to re-solve. The range estimate comes from the last re-solve in the previous round. When DeepStack is second to act in the round, the opponent has already acted, biasing their range, so we use the conservative approach. When DeepStack is first to act, though, the opponent could only have checked or called since our last re-solve. Thus, the lookahead has an estimated range following their action. So in this case, we use the

Table S6: Thinking times for both humans and DeepStack. DeepStack’s extremely fast pre-flop speed shows that pre-flop situations often resulted in cache hits.

Round	Thinking Time (s)			
	Humans		DeepStack	
	Median	Mean	Median	Mean
Pre-flop	10.3	16.2	0.04	0.2
Flop	9.1	14.6	5.9	5.9
Turn	8.0	14.0	5.4	5.5
River	9.5	16.2	2.2	2.1
Per Action	9.6	15.4	2.3	3.0
Per Hand	22.0	37.4	5.7	7.2

aggressive approach. In both cases, we set  $b = 0.9$ .

**Speed of Play.** The re-solving computation and neural network evaluations are both implemented in Torch7 (53) and run on a single NVIDIA GeForce GTX 1080 graphics card. This makes it possible to do fast batched calls to the counterfactual value networks for multiple public subtrees at once, which is key to making DeepStack fast.

Table S6 reports the average times between the end of the previous (opponent or chance) action and submitting the next action by both humans and DeepStack in our study. DeepStack, on average, acted considerably faster than our human players. It should be noted that some human players were playing up to four games simultaneously (although few players did more than two), and so the human players may have been focused on another game when it became their turn to act.

## Deep Counterfactual Value Networks

DeepStack uses two counterfactual value networks, one for the flop and one for the turn, as well as an auxiliary network that gives counterfactual values at the end of the pre-flop. In order to train the networks, we generated random poker situations at the start of the flop and turn. Each poker situation is defined by the pot size, ranges for both players, and dealt public cards. The complete betting history is not necessary as the pot and ranges are a sufficient representation. The output of the network are vectors of counterfactual values, one for each player. The output values are interpreted as fractions of the pot size to improve generalization across poker situations.

The training situations were generated by first sampling a pot size from a fixed distribution which was designed to approximate observed pot sizes from older HUNL programs.<sup>1</sup> The

<sup>1</sup>The fixed distribution selects an interval from the set of intervals  $\{[100, 100), [200, 400), [400, 2000)\}$ ,

player ranges for the training situations need to cover the space of possible ranges that CFR might encounter during re-solving, not just ranges that are likely part of a solution. So we generated pseudo-random ranges that attempt to cover the space of possible ranges. We used a recursive procedure  $R(S, p)$ , that assigns probabilities to the hands in the set  $S$  that sum to probability  $p$ , according to the following procedure.

1. If  $|S| = 1$ , then  $\Pr(s) = p$ .
2. Otherwise,
  - (a) Choose  $p_1$  uniformly at random from the interval  $(0, p)$ , and let  $p_2 = p - p_1$ .
  - (b) Let  $S_1 \subset S$  and  $S_2 = S \setminus S_1$  such that  $|S_1| = \lfloor |S|/2 \rfloor$  and all of the hands in  $S_1$  have a hand strength no greater than hands in  $S_2$ . Hand strength is the probability of a hand beating a uniformly selected random hand from the current public state.
  - (c) Use  $R(S_1, p_1)$  and  $R(S_2, p_2)$  to assign probabilities to hands in  $S = S_1 \cup S_2$ .

Generating a range involves invoking  $R(\text{all hands}, 1)$ . To obtain the target counterfactual values for the generated poker situations for the main networks, the situations were approximately solved using 1,000 iterations of CFR<sup>+</sup> with only betting actions fold, call, a pot-sized bet, and all-in. For the turn network, ten million poker turn situations (from after the turn card is dealt) were generated and solved with 6,144 CPU cores of the Calcul Québec MP2 research cluster, using over 175 core years of computation time. For the flop network, one million poker flop situations (from after the flop cards are dealt) were generated and solved. These situations were solved using DeepStack’s depth limited solver with the turn network used for the counterfactual values at public states immediately after the turn card. We used a cluster of 20 GPUS and one-half of a GPU year of computation time. For the auxiliary network, ten million situations were generated and the target values were obtained by enumerating all 22,100 possible flops and averaging the counterfactual values from the flop network’s output.

**Neural Network Training.** All networks were trained using built-in Torch7 libraries, with the Adam stochastic gradient descent procedure (34) minimizing the average of the Huber losses (35) over the counterfactual value errors. Training used a mini-batch size of 1,000, and a learning rate 0.001, which was decreased to 0.0001 after the first 200 epochs. Networks were trained for approximately 350 epochs over two days on a single GPU, and the epoch with the lowest validation loss was chosen.

**Neural Network Range Representation.** In order to improve generalization over input player ranges, we map the distribution of individual hands (combinations of public and private cards) into distributions of buckets. The buckets were generated using a clustering-based abstraction

---

$[2000, 6000)$ ,  $[6000, 19950]$  with uniform probability, followed by uniformly selecting an integer from within the chosen interval.

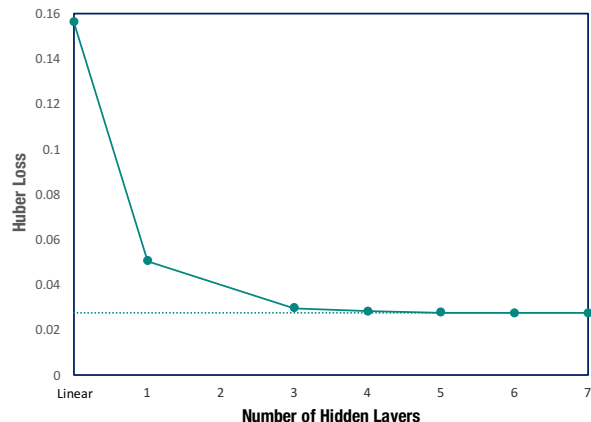


Figure S1: Huber loss with different numbers of hidden layers in the neural network.

technique, which cluster strategically similar hands using  $k$ -means clustering with earth mover’s distance over hand-strength-like features (28, 54). For both the turn and flop networks we used 1,000 clusters and map the original ranges into distributions over these clusters as the first layer of the neural network (see Figure 3 of the main article). This bucketing step was not used on the auxiliary network as there are only 169 strategically distinct hands pre-flop, making it feasible to input the distribution over distinct hands directly.

**Neural Network Accuracies.** The turn network achieved an average Huber loss of 0.016 of the pot size on the training set and 0.026 of the pot size on the validation set. The flop network, with a much smaller training set, achieved an average Huber loss of 0.008 of the pot size on the training set, but 0.034 of the pot size on the validation set. Finally, the auxiliary network had average Huber losses of 0.000053 and 0.000055 on the training and validation set, respectively. Note that there are, in fact, multiple Nash equilibrium solutions to these poker situations, with each giving rise to different counterfactual value vectors. So, these losses may overestimate the true loss as the network may accurately model a different equilibrium strategy.

**Number of Hidden Layers.** We observed in early experiments that the neural network had a lower validation loss with an increasing number of hidden layers. From these experiments, we chose to use seven hidden layers in an attempt to tradeoff accuracy, speed of execution, and the available memory on the GPU. The result of a more thorough experiment examining the turn network accuracy as a function of the number of hidden layers is in Figure S1. It appears that seven hidden layers is more than strictly necessary as the validation error does not improve much beyond five. However, all of these architectures were trained using the same ten million turn situations. With more training data it would not be surprising to see the larger networks see a further reduction in loss due to their richer representation power.

## Proof of Theorem 1

The formal proof of Theorem 1, which establishes the soundness of DeepStack’s depth-limited continual re-solving, is conceptually easy to follow. It requires three parts. First, we establish that the exploitability introduced in a re-solving step has two linear components; one due to approximately solving the subgame, and one due to the error in DeepStack’s counterfactual value network (see Lemmas 1 through 5). Second, we enable estimates of subgame counterfactual values that do not arise from actual subgame strategies (see Lemma 6). Together, parts one and two enable us to use DeepStack’s counterfactual value network for a single re-solve.<sup>2</sup> Finally, we show that using the opponent’s values from the best action, rather than the observed action, does not increase overall exploitability (see Lemma 7). This allows us to carry forward estimates of the opponent’s counterfactual value, enabling continual re-solving. Put together, these three parts bound the error after any finite number of continual re-solving steps, concluding the proof. We now formalize each step.

There are a number of concepts we use throughout this section. We use the notation from Burch et al. (17) without any introduction here. We assume player 1 is performing the continual re-solving. We call player 2 the opponent. We only consider the re-solve player’s strategy  $\sigma$ , as the opponent is always using a best response to  $\sigma$ . All values are considered with respect to the opponent, unless specifically stated. We say  $\sigma$  is  $\epsilon$ -exploitable if the opponent’s best response value against  $\sigma$  is no more than  $\epsilon$  away from the game value for the opponent.

A public state  $S$  corresponds to the root of an imperfect information subgame. We write  $\mathcal{I}_2^S$  for the collection of player 2 information sets in  $S$ . Let  $G\langle S, \sigma, w \rangle$  be the subtree gadget game (the re-solving game of Burch et al. (17)), where  $S$  is some public state,  $\sigma$  is used to get player 1 reach probabilities  $\pi_{-2}^\sigma(h)$  for each  $h \in S$ , and  $w$  is a vector where  $w_I$  gives the value of player 2 taking the terminate action (T) from information set  $I \in \mathcal{I}_2^S$ . Let

$$\mathbf{BV}_I(\sigma) = \max_{\sigma_2^*} \sum_{h \in I} \pi_{-2}^\sigma(h) u^{\sigma, \sigma_2^*}(h) / \pi_{-2}^\sigma(I),$$

be the counterfactual value for  $I$  given we play  $\sigma$  and our opponent is playing a best response. For a subtree strategy  $\sigma^S$ , we write  $\sigma \rightarrow \sigma^S$  for the strategy that plays according to  $\sigma^S$  for any state in the subtree and according to  $\sigma$  otherwise. For the gadget game  $G\langle S, \sigma, w \rangle$ , the gadget value of a subtree strategy  $\sigma^S$  is defined to be:

$$\mathbf{GV}_{w, \sigma}^S(\sigma^S) = \sum_{I \in \mathcal{I}_2^S} \max(w_I, \mathbf{BV}_I(\sigma \rightarrow \sigma^S)),$$

and the underestimation error is defined to be:

$$\mathbf{U}_{w, \sigma}^S = \min_{\sigma^S} \mathbf{GV}_{w, \sigma}^S(\sigma^S) - \sum_{I \in \mathcal{I}_2^S} w_I.$$

---

<sup>2</sup>The first part is a generalization and improvement on the re-solving exploitability bound given by Theorem 3 in Burch et al. (17), and the second part generalizes the bound on decomposition regret given by Theorem 2 of the same work.

**Lemma S1** *The game value of a gadget game  $G\langle S, \sigma, w \rangle$  is*

$$\sum_{I \in \mathcal{I}_2^S} w_I + U_{w, \sigma}^S.$$

**Proof.** Let  $\tilde{\sigma}_2^S$  be a gadget game strategy for player 2 which must choose from the F and T actions at starting information set  $I$ . Let  $\tilde{u}$  be the utility function for the gadget game.

$$\begin{aligned} \min_{\sigma_1^S} \max_{\tilde{\sigma}_2^S} \tilde{u}(\sigma_1^S, \tilde{\sigma}_2^S) &= \min_{\sigma_1^S} \max_{\sigma_2^S} \sum_{I \in \mathcal{I}_2^S} \frac{\pi_{-2}^\sigma(I)}{\sum_{I' \in \mathcal{I}_2^S} \pi_{-2}^\sigma(I')} \max_{a \in \{F, T\}} \tilde{u}^{\sigma^S}(I, a) \\ &= \min_{\sigma_1^S} \max_{\sigma_2^S} \sum_{I \in \mathcal{I}_2^S} \max(w_I, \sum_{h \in I} \pi_{-2}^\sigma(h) u^{\sigma^S}(h)) \end{aligned}$$

A best response can maximize utility at each information set independently:

$$\begin{aligned} &= \min_{\sigma_1^S} \sum_{I \in \mathcal{I}_2^S} \max(w_I, \max_{\sigma_2^S} \sum_{h \in I} \pi_{-2}^\sigma(h) u^{\sigma^S}(h)) \\ &= \min_{\sigma_1^S} \sum_{I \in \mathcal{I}_2^S} \max(w_I, \mathbf{BV}_I(\sigma \rightarrow \sigma_1^S)) \\ &= U_{w, \sigma}^S + \sum_{I \in \mathcal{I}_2^S} w_I \end{aligned}$$

■

**Lemma S2** *If our strategy  $\sigma^S$  is  $\epsilon$ -exploitable in the gadget game  $G\langle S, \sigma, w \rangle$ , then  $\mathbf{GV}_{w, \sigma}^S(\sigma^S) \leq \sum_{I \in \mathcal{I}_2^S} w_I + U_{w, \sigma}^S + \epsilon$*

**Proof.** This follows from Lemma S1 and the definitions of  $\epsilon$ -Nash,  $U_{w, \sigma}^S$ , and  $\mathbf{GV}_{w, \sigma}^S(\sigma^S)$ . ■

**Lemma S3** *Given an  $\epsilon_0$ -exploitable  $\sigma$  in the original game, if we replace a subgame with a strategy  $\sigma^S$  such that  $\mathbf{BV}_I(\sigma \rightarrow \sigma^S) \leq w_I$  for all  $I \in \mathcal{I}_2^S$ , then the new combined strategy has an exploitability no more than  $\epsilon_0 + \mathbf{EXP}_{w, \sigma}^S$  where*

$$\mathbf{EXP}_{w, \sigma}^S = \sum_{I \in \mathcal{I}_2^S} \max(\mathbf{BV}_I(\sigma), w_I) - \sum_{I \in \mathcal{I}_2^S} \mathbf{BV}_I(\sigma)$$

**Proof.** We only care about the information sets where the opponent's counterfactual value increases, and a worst case upper bound occurs when the opponent best response would reach every such information set with probability 1, and never reach information sets where the value decreased.

Let  $Z[S] \subseteq Z$  be the set of terminal states reachable from some  $h \in S$  and let  $v_2$  be the game value of the full game for player 2. Let  $\sigma_2$  be a best response to  $\sigma$  and let  $\sigma_2^S$  be the part of  $\sigma_2$  that plays in the subtree rooted at  $S$ . Then necessarily  $\sigma_2^S$  achieves counterfactual value  $\text{BV}_I(\sigma)$  at each  $I \in \mathcal{I}_2^S$ .

$$\begin{aligned}
& \max_{\sigma_2^*} (u(\sigma \rightarrow \sigma^S, \sigma_2^*)) \\
&= \max_{\sigma_2^*} \left[ \sum_{z \in Z[S]} \pi_{-2}^{\sigma \rightarrow \sigma^S}(z) \pi_2^{\sigma_2^*}(z) u(z) + \sum_{z \in Z \setminus Z[S]} \pi_{-2}^{\sigma \rightarrow \sigma^S}(z) \pi_2^{\sigma_2^*}(z) u(z) \right] \\
&= \max_{\sigma_2^*} \left[ \sum_{z \in Z[S]} \pi_{-2}^{\sigma \rightarrow \sigma^S}(z) \pi_2^{\sigma_2^*}(z) u(z) - \sum_{z \in Z[S]} \pi_{-2}^{\sigma}(z) \pi_2^{\sigma_2^* \rightarrow \sigma_2^S}(z) u(z) \right. \\
&\quad \left. + \sum_{z \in Z[S]} \pi_{-2}^{\sigma}(z) \pi_2^{\sigma_2^* \rightarrow \sigma_2^S}(z) u(z) + \sum_{z \in Z \setminus Z[S]} \pi_{-2}^{\sigma}(z) \pi_2^{\sigma_2^*}(z) u(z) \right] \\
&\leq \max_{\sigma_2^*} \left[ \sum_{z \in Z[S]} \pi_{-2}^{\sigma \rightarrow \sigma^S}(z) \pi_2^{\sigma_2^*}(z) u(z) - \sum_{z \in Z[S]} \pi_{-2}^{\sigma}(z) \pi_2^{\sigma_2^* \rightarrow \sigma_2^S}(z) u(z) \right] \\
&\quad + \max_{\sigma_2^*} \left[ \sum_{z \in Z[S]} \pi_{-2}^{\sigma}(z) \pi_2^{\sigma_2^* \rightarrow \sigma_2^S}(z) u(z) + \sum_{z \in Z \setminus Z[S]} \pi_{-2}^{\sigma}(z) \pi_2^{\sigma_2^*}(z) u(z) \right] \\
&\leq \max_{\sigma_2^*} \left[ \sum_{I \in \mathcal{I}_2^S} \sum_{h \in I} \pi_{-2}^{\sigma}(h) \pi_2^{\sigma_2^*}(h) u^{\sigma^S, \sigma_2^*}(h) \right. \\
&\quad \left. - \sum_{I \in \mathcal{I}_2^S} \sum_{h \in I} \pi_{-2}^{\sigma}(h) \pi_2^{\sigma_2^*}(h) u^{\sigma, \sigma_2^S}(h) \right] + \max_{\sigma_2^*} (u(\sigma, \sigma_2^*))
\end{aligned}$$

By perfect recall  $\pi_2(h) = \pi_2(I)$  for each  $h \in I$ :

$$\begin{aligned}
&\leq \max_{\sigma_2^*} \left[ \sum_{I \in \mathcal{I}_2^S} \pi_2^{\sigma_2^*}(I) \left( \sum_{h \in I} \pi_{-2}^{\sigma}(h) u^{\sigma^S, \sigma_2^*}(h) - \sum_{h \in I} \pi_{-2}^{\sigma}(h) u^{\sigma, \sigma_2^S}(h) \right) \right] \\
&\quad + v_2 + \epsilon_0 \\
&= \max_{\sigma_2^*} \left[ \sum_{I \in \mathcal{I}_2^S} \pi_2^{\sigma_2^*}(I) \pi_{-2}^{\sigma}(I) \left( \text{BV}_I(\sigma \rightarrow \sigma^S) - \text{BV}_I(\sigma) \right) \right] + v_2 + \epsilon_0 \\
&\leq \left[ \sum_{I \in \mathcal{I}_2^S} \max(\text{BV}_I(\sigma \rightarrow \sigma^S) - \text{BV}_I(\sigma), 0) \right] + v_2 + \epsilon_0
\end{aligned}$$

$$\begin{aligned}
&\leq \left[ \sum_{I \in \mathcal{I}_2^S} \max(w_I - \mathbf{BV}_I(\sigma), \mathbf{BV}_I(\sigma) - \mathbf{BV}_I(\sigma)) \right] + v_2 + \epsilon_O \\
&= \left[ \sum_{I \in \mathcal{I}_2^S} \max(\mathbf{BV}_I(\sigma), w_I) - \sum_{I \in \mathcal{I}_2^S} \mathbf{BV}_I(\sigma) \right] + v_2 + \epsilon_O
\end{aligned}$$

■

**Lemma S4** *Given an  $\epsilon_O$ -exploitable  $\sigma$  in the original game, if we replace the strategy in a subgame with a strategy  $\sigma^S$  that is  $\epsilon_S$ -exploitable in the gadget game  $G^S(S, \sigma, w)$ , then the new combined strategy has an exploitability no more than  $\epsilon_O + \text{EXP}_{w, \sigma}^S + U_{w, \sigma}^S + \epsilon_S$ .*

**Proof.** We use that  $\max(a, b) = a + b - \min(a, b)$ . From applying Lemma S3 with  $w_I = \mathbf{BV}_I(\sigma \rightarrow \sigma^S)$  and expanding  $\text{EXP}_{\mathbf{BV}(\sigma \rightarrow \sigma^S), \sigma}^S$  we get exploitability no more than  $\epsilon_O - \sum_{I \in \mathcal{I}_2^S} \mathbf{BV}_I(\sigma)$  plus

$$\begin{aligned}
&\sum_{I \in \mathcal{I}_2^S} \max(\mathbf{BV}_I(\sigma \rightarrow \sigma^S), \mathbf{BV}_I(\sigma)) \\
&\leq \sum_{I \in \mathcal{I}_2^S} \max(\mathbf{BV}_I(\sigma \rightarrow \sigma^S), \max(w_I, \mathbf{BV}_I(\sigma))) \\
&= \sum_{I \in \mathcal{I}_2^S} (\mathbf{BV}_I(\sigma \rightarrow \sigma^S) + \max(w_I, \mathbf{BV}_I(\sigma)) \\
&\quad - \min(\mathbf{BV}_I(\sigma \rightarrow \sigma^S), \max(w_I, \mathbf{BV}_I(\sigma)))) \\
&\leq \sum_{I \in \mathcal{I}_2^S} (\mathbf{BV}_I(\sigma \rightarrow \sigma^S) + \max(w_I, \mathbf{BV}_I(\sigma)) \\
&\quad - \min(\mathbf{BV}_I(\sigma \rightarrow \sigma^S), w_I)) \\
&= \sum_{I \in \mathcal{I}_2^S} (\max(w_I, \mathbf{BV}_I(\sigma)) + \max(w_I, \mathbf{BV}_I(\sigma \rightarrow \sigma^S)) - w_I) \\
&= \sum_{I \in \mathcal{I}_2^S} \max(w_I, \mathbf{BV}_I(\sigma)) + \sum_{I \in \mathcal{I}_2^S} \max(w_I, \mathbf{BV}_I(\sigma \rightarrow \sigma^S)) - \sum_{I \in \mathcal{I}_2^S} w_I
\end{aligned}$$

From Lemma S2 we get

$$\leq \sum_{I \in \mathcal{I}_2^S} \max(w_I, \mathbf{BV}_I(\sigma)) + U_{w, \sigma}^S + \epsilon_S$$

Adding  $\epsilon_O - \sum_I \mathbf{BV}_I(\sigma)$  we get the upper bound  $\epsilon_O + \text{EXP}_{w, \sigma}^S + U_{w, \sigma}^S + \epsilon_S$ . ■



**Lemma S5** Assume we are performing one step of re-solving on subtree  $S$ , with constraint values  $w$  approximating opponent best-response values to the previous strategy  $\sigma$ , with an approximation error bound  $\sum_I |w_I - \mathbf{BV}_I(\sigma)| \leq \epsilon_E$ . Then we have  $\text{EXP}_{w,\sigma}^S + \text{U}_{w,\sigma}^S \leq \epsilon_E$ .

**Proof.**  $\text{EXP}_{w,\sigma}^S$  measures the amount that the  $w_I$  exceed  $\mathbf{BV}_I(\sigma)$ , while  $\text{U}_{w,\sigma}^S$  bounds the amount that the  $w_I$  underestimate  $\mathbf{BV}_I(\sigma \rightarrow \sigma^S)$  for any  $\sigma^S$ , including the original  $\sigma$ . Thus, together they are bounded by  $|w_I - \mathbf{BV}_I(\sigma)|$ :

$$\begin{aligned}
\text{EXP}_{w,\sigma}^S + \text{U}_{w,\sigma}^S &= \sum_{I \in \mathcal{I}_2^S} \max(\mathbf{BV}_I(\sigma), w_I) - \sum_{I \in \mathcal{I}_2^S} \mathbf{BV}_I(\sigma) \\
&\quad + \min_{\sigma^S} \sum_{I \in \mathcal{I}_2^S} \max(w_I, \mathbf{BV}_I(\sigma \rightarrow \sigma^S)) - \sum_{I \in \mathcal{I}_2^S} w_I \\
&\leq \sum_{I \in \mathcal{I}_2^S} \max(\mathbf{BV}_I(\sigma), w_I) - \sum_{I \in \mathcal{I}_2^S} \mathbf{BV}_I(\sigma) \\
&\quad + \sum_{I \in \mathcal{I}_2^S} \max(w_I, \mathbf{BV}_I(\sigma)) - \sum_{I \in \mathcal{I}_2^S} w_I \\
&= \sum_{I \in \mathcal{I}_2^S} [\max(w_I - \mathbf{BV}_I(\sigma), 0) + \max(\mathbf{BV}_I(\sigma) - w_I, 0)] \\
&= \sum_{I \in \mathcal{I}_2^S} |w_I - \mathbf{BV}_I(\sigma)| \leq \epsilon_E
\end{aligned}$$

■

**Lemma S6** Assume we are solving a game  $G$  with  $T$  iterations of CFR-D where for both players  $p$ , subtrees  $S$ , and times  $t$ , we use subtree values  $v_I$  for all information sets  $I$  at the root of  $S$  from some suboptimal black box estimator. If the estimation error is bounded, so that  $\min_{\sigma_s^* \in \text{NE}_S} \sum_{I \in \mathcal{I}_2^S} |v^{\sigma_s^*}(I) - v_I| \leq \epsilon_E$ , then the trunk exploitability is bounded by  $k_G/\sqrt{T} + j_G \epsilon_E$  for some game specific constant  $k_G, j_G \geq 1$  which depend on how the game is split into a trunk and subgames.

**Proof.** This follows from a modified version the proof of Theorem 2 of Burch et al. (17), which uses a fixed error  $\epsilon$  and argues by induction on information sets. Instead, we argue by induction on entire public states.

For every public state  $s$ , let  $N_s$  be the number of subgames reachable from  $s$ , including any subgame rooted at  $s$ . Let  $\text{Succ}(s)$  be the set of our public states which are reachable from  $s$  without going through another of our public states on the way. Note that if  $s$  is in the trunk, then every  $s' \in \text{Succ}(s)$  is in the trunk or is the root of a subgame. Let  $D_{TR}(s)$  be the set of our trunk public states reachable from  $s$ , including  $s$  if  $s$  is in the trunk. We argue that for any public state  $s$  where we act in the trunk or at the root of a subgame

$$\sum_{I \in s} R_{full}^{T,+}(I) \leq \sum_{s' \in D_{TR}(s)} \sum_{I \in s'} R^{T,+}(I) + TN_s \epsilon_E \tag{S1}$$

First note that if no subgame is reachable from  $s$ , then  $N_s = 0$  and the statement follows from Lemma 7 of (I4). For public states from which a subgame is reachable, we argue by induction on  $|D_{TR}(s)|$ .

For the base case, if  $|D_{TR}(s)| = 0$  then  $s$  is the root of a subgame  $S$ , and by assumption there is a Nash Equilibrium subgame strategy  $\sigma_S^*$  that has regret no more than  $\epsilon_E$ . If we implicitly play  $\sigma_S^*$  on each iteration of CFR-D, we thus accrue  $\sum_{I \in s} R_{full}^{T,+}(I) \leq T\epsilon_E$ .

For the inductive hypothesis, we assume that (S1) holds for all  $s$  such that  $|D_{TR}(s)| < k$ . Consider a public state  $s$  where  $|D_{TR}(s)| = k$ . By Lemma 5 of (I4) we have

$$\begin{aligned} \sum_{I \in s} R_{full}^{T,+}(I) &\leq \sum_{I \in s} \left[ R^T(I) + \sum_{I' \in Succ(I)} R_{full}^{T,+}(I') \right] \\ &= \sum_{I \in s} R^T(I) + \sum_{s' \in Succ(s)} \sum_{I' \in s'} R_{full}^{T,+}(I') \end{aligned}$$

For each  $s' \in Succ(s)$ ,  $D(s') \subset D(s)$  and  $s \notin D(s')$ , so  $|D(s')| < |D(s)|$  and we can apply the inductive hypothesis to show

$$\begin{aligned} \sum_{I \in s} R_{full}^{T,+}(I) &\leq \sum_{I \in s} R^T(I) + \sum_{s' \in Succ(s)} \left[ \sum_{s'' \in D(s')} \sum_{I \in s''} R^{T,+}(I) + TN_{s' \in E} \right] \\ &\leq \sum_{s' \in D(s)} \sum_{I \in s'} R^{T,+}(I) + T\epsilon_E \sum_{s' \in Succ(s)} N_{s'} \\ &= \sum_{s' \in D(s)} \sum_{I \in s'} R^{T,+}(I) + T\epsilon_E N_s \end{aligned}$$

This completes the inductive argument. By using regret matching in the trunk, we ensure  $R^T(I) \leq \Delta\sqrt{AT}$ , proving the lemma for  $k_G = \Delta|\mathcal{I}_{TR}|\sqrt{A}$  and  $j_G = N_{root}$ . ■

**Lemma S7** *Given our strategy  $\sigma$ , if the opponent is acting at the root of a public subtree  $S$  from a set of actions  $A$ , with opponent best-response values  $BV_{I,a}(\sigma)$  after each action  $a \in A$ , then replacing our subtree strategy with any strategy that satisfies the opponent constraints  $w_I = \max_{a \in A} BV_{I,a}(\sigma)$  does not increase our exploitability.*

**Proof.** If the opponent is playing a best response, every counterfactual value  $w_I$  before the action must either satisfy  $w_I = BV_I(\sigma) = \max_{a \in A} BV_{I,a}(\sigma)$ , or not reach state  $s$  with private information  $I$ . If we replace our strategy in  $S$  with a strategy  $\sigma'_S$  such that  $BV_{I,a}(\sigma'_S) \leq BV_I(\sigma)$  we preserve the property that  $BV_I(\sigma') = BV_I(\sigma)$ . ■

**Theorem S1** *Assume we have some initial opponent constraint values  $w$  from a solution generated using at least  $T$  iterations of CFR-D, we use at least  $T$  iterations of CFR-D to solve each resolving game, and we use a subtree value estimator such that  $\min_{\sigma_S^* \in NE_S} \sum_{I \in \mathcal{I}_2^S} |v^{\sigma_S^*}(I) - v_I| \leq$*

$\epsilon_E$ , then after  $d$  re-solving steps the exploitability of the resulting strategy is no more than  $(d+1)k/\sqrt{T} + (2d+1)j\epsilon_E$  for some constants  $k, j$  specific to both the game and how it is split into subgames.

**Proof.** Continual re-solving begins by solving from the root of the entire game, which we label as subtree  $S_0$ . We use CFR-D with the value estimator in place of subgame solving in order to generate an initial strategy  $\sigma_0$  for playing in  $S_0$ . By Lemma S6, the exploitability of  $\sigma_0$  is no more than  $k_0/\sqrt{T} + j_0\epsilon_E$ .

For each step of continual re-solving  $i = 1, \dots, d$ , we are re-solving some subtree  $S_i$ . From the previous step of re-solving, we have approximate opponent best-response counterfactual values  $\widetilde{BV}_I(\sigma_{i-1})$  for each  $I \in \mathcal{I}_2^{S_{i-1}}$ , which by the estimator bound satisfy  $|\sum_{I \in \mathcal{I}_2^{S_{i-1}}} BV_I(\sigma_{i-1}) - \widetilde{BV}_I(\sigma_{i-1})| \leq \epsilon_E$ . Updating these values at each public state between  $S_{i-1}$  and  $S_i$  as described in the paper yields approximate values  $\widetilde{BV}_I(\sigma_{i-1})$  for each  $I \in \mathcal{I}_2^{S_i}$  which by Lemma S7 can be used as constraints  $w_{I,i}$  in re-solving. Lemma S5 with these constraints gives us the bound  $\text{EXP}_{w_i, \sigma_{i-1}}^{S_i} + U_{w_i, \sigma_{i-1}}^{S_i} \leq \epsilon_E$ . Thus by Lemma S4 and Lemma S6 we can say that the increase in exploitability from  $\sigma_{i-1}$  to  $\sigma_i$  is no more than  $\epsilon_E + \epsilon_{S_i} \leq \epsilon_E + k_i/\sqrt{T} + j_i\epsilon_E \leq k_i/\sqrt{T} + 2j_i\epsilon_E$ .

Let  $k = \max_i k_i$  and  $j = \max_i j_i$ . Then after  $d$  re-solving steps, the exploitability is bounded by  $(d+1)k/\sqrt{T} + (2d+1)j\epsilon_E$ . ■

## Best-response Values Versus Self-play Values

DeepStack uses self-play values within the continual re-solving computation, rather than the best-response values described in Theorem S1. Preliminary tests using CFR-D to solve smaller games suggested that strategies generated using self-play values were generally less exploitable and had better one-on-one performance against test agents, compared to strategies generated using best-response values. Figure S2 shows an example of DeepStack’s exploitability in a particular river subgame with different numbers of re-solving iterations. Despite lacking a theoretical justification for its soundness, using self-play values appears to converge to low exploitability strategies just as with using best-response values.

One possible explanation for why self-play values work well with continual re-solving is that at every re-solving step, we give away a little more value to our best-response opponent because we are not solving the subtrees exactly. If we use the self-play values for the opponent, the opponent’s strategy is slightly worse than a best response, making the opponent values smaller and counteracting the inflationary effect of an inexact solution. While this optimism could hurt us by setting unachievable goals for the next re-solving step (an increased  $U_{w, \sigma}^S$  term), in poker-like games we find that the more positive expectation is generally correct (a decreased  $\text{EXP}_{w, \sigma}^S$  term.)

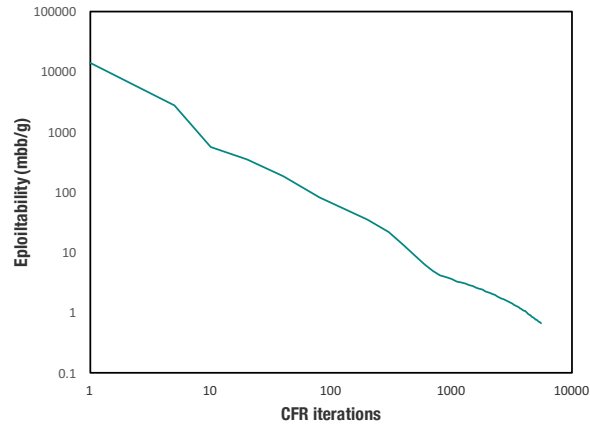


Figure S2: DeepStack’s exploitability within a particular public state at the start of the river as a function of the number of re-solving iterations.

## Pseudocode

Complete pseudocode for DeepStack’s depth-limited continual re-resolving algorithm is in Algorithm S1. Conceptually, DeepStack can be decomposed into four functions: RE-SOLVE, VALUES, UPDATESUBTREESTRATEGIES, and RANGEADGET. The main function is RE-SOLVE, which is called every time DeepStack needs to take an action. It iteratively calls each of the other functions to refine the lookahead tree solution. After  $T$  iterations, an action is sampled from the approximate equilibrium strategy at the root of the subtree to be played. According to this action, DeepStack’s range,  $\vec{r}_1$ , and its opponent’s counterfactual values,  $\vec{v}_2$ , are updated in preparation for its next decision point.

---

**Algorithm S1** Depth-limited continual re-solving

---

**INPUT:** Public state  $S$ , player range  $\mathbf{r}_1$  over our information sets in  $S$ , opponent counterfactual values  $\mathbf{v}_2$  over their information sets in  $S$ , and player information set  $I \in S$

**OUTPUT:** Chosen action  $a$ , and updated representation after the action  $(S(a), \mathbf{r}_1(a), \mathbf{v}_2(a))$

```
1: function RE-SOLVE( $S, \mathbf{r}_1, \mathbf{v}_2, I$ )
2:    $\sigma^0 \leftarrow$  arbitrary initial strategy profile
3:    $\mathbf{r}_2^0 \leftarrow$  arbitrary initial opponent range
4:    $R_G^0, R^0 \leftarrow \mathbf{0}$  ▷ Initial regrets for gadget game and subtree
5:   for  $t = 1$  to  $T$  do
6:      $\mathbf{v}_1^t, \mathbf{v}_2^t \leftarrow$  VALUES( $S, \sigma^{t-1}, \mathbf{r}_1, \mathbf{r}_2^{t-1}, 0$ )
7:      $\sigma^t, R^t \leftarrow$  UPDATESUBTREESTRATEGIES( $S, \mathbf{v}_1^t, \mathbf{v}_2^t, R^{t-1}$ )
8:      $\mathbf{r}_2^t, R_G^t \leftarrow$  RANGEGADGET( $\mathbf{v}_2, \mathbf{v}_2^t(S), R_G^{t-1}$ )
9:   end for
10:   $\bar{\sigma}^T \leftarrow \frac{1}{T} \sum_{t=1}^T \sigma^t$  ▷ Average the strategies
11:   $a \sim \bar{\sigma}^T(\cdot|I)$  ▷ Sample an action
12:   $\mathbf{r}_1(a) \leftarrow \langle \mathbf{r}_1, \sigma(a|\cdot) \rangle$  ▷ Update the range based on the chosen action
13:   $\mathbf{r}_1(a) \leftarrow \mathbf{r}_1(a) / \|\mathbf{r}_1(a)\|_1$  ▷ Normalize the range
14:   $\mathbf{v}_2(a) \leftarrow \frac{1}{T} \sum_{t=1}^T \mathbf{v}_2^t(a)$  ▷ Average of counterfactual values after action  $a$ 
15:  return  $a, S(a), \mathbf{r}_1(a), \mathbf{v}_2(a)$ 
16: end function

17: function VALUES( $S, \sigma, \mathbf{r}_1, \mathbf{r}_2, d$ ) ▷ Gives the counterfactual values of the subtree  $S$  under  $\sigma$ ,  
computed with a depth-limited lookahead.
18:   if  $S$  is terminal then
19:      $\mathbf{v}_1(S) \leftarrow U_S \mathbf{r}_2$  ▷ Where  $U_S$  is the matrix of the bilinear utility function at  $S$ ,
20:      $\mathbf{v}_2(S) \leftarrow \mathbf{r}_1^\top U_S$   $U(S) = \mathbf{r}_1^\top U_S \mathbf{r}_2$ , thus giving vectors of counterfactual values
21:     return  $\mathbf{v}_1(S), \mathbf{v}_2(S)$ 
22:   else if  $d = \text{MAX-DEPTH}$  then
23:     return NEURALNETEVALUATE( $S, \mathbf{r}_1, \mathbf{r}_2$ )
24:   end if
25:    $\mathbf{v}_1(S), \mathbf{v}_2(S) \leftarrow \mathbf{0}$ 
26:   for action  $a \in S$  do
27:      $\mathbf{r}_{\text{Player}(S)}(a) \leftarrow \langle \mathbf{r}_{\text{Player}(S)}, \sigma(a|\cdot) \rangle$  ▷ Update range of acting player based on strategy
28:      $\mathbf{r}_{\text{Opponent}(S)}(a) \leftarrow \mathbf{r}_{\text{Opponent}(S)}$ 
29:      $\mathbf{v}_1(S(a)), \mathbf{v}_2(S(a)) \leftarrow$  VALUE( $S(a), \sigma, \mathbf{r}_1(a), \mathbf{r}_2(a), d + 1$ )
30:      $\mathbf{v}_{\text{Player}(S)}(S) \leftarrow \mathbf{v}_{\text{Player}(S)}(S) + \sigma(a|\cdot) \mathbf{v}_{\text{Player}(S)}(S(a))$  ▷ Weighted average
31:      $\mathbf{v}_{\text{Opponent}(S)}(S) \leftarrow \mathbf{v}_{\text{Player}(S)}(S) + \mathbf{v}_{\text{Opponent}(S)}(S(a))$  ▷ Unweighted sum, as our strategy is already in-  
cluded in opponent counterfactual values
32:   end for
33:   return  $\mathbf{v}_1, \mathbf{v}_2$ 
34: end function
```

---

---

```

35: function UPDATESUBTREESTRATEGIES( $S, \mathbf{v}_1, \mathbf{v}_2, R^{t-1}$ )
36:   for  $S' \in \{S\} \cup \text{SubtreeDescendants}(S)$  with  $\text{Depth}(S') < \text{MAX-DEPTH}$  do
37:     for action  $a \in S'$  do
38:        $R^t(a|\cdot) \leftarrow R^{t-1}(a|\cdot) + \mathbf{v}_{\text{Player}(S')}(S'(a)) - \mathbf{v}_{\text{Player}(S')}(S')$ 
                                      $\triangleright$  Update acting player's regrets
39:     end for
40:     for information set  $I \in S'$  do
41:        $\sigma^t(\cdot|I) \leftarrow \frac{R^t(\cdot|I)^+}{\sum_a R^t(a|I)^+}$ 
                                      $\triangleright$  Update strategy with regret matching
42:     end for
43:   end for
44:   return  $\sigma^t, R^t$ 
45: end function

46: function RANGEGADGET( $\mathbf{v}_2, \mathbf{v}_2^t, R_G^{t-1}$ )
                                      $\triangleright$  Let opponent choose to play in the subtree or
                                     receive the input value with each hand (see
                                     Burch et al. (17))
47:    $\sigma_G(\mathbf{F}|\cdot) \leftarrow \frac{R_G^{t-1}(\mathbf{F}|\cdot)^+}{R_G^{t-1}(\mathbf{F}|\cdot)^+ + R_G^{t-1}(\mathbf{T}|\cdot)^+}$ 
                                      $\triangleright$  F is Follow action, T is Terminate
48:    $\mathbf{r}_2^t \leftarrow \sigma_G(\mathbf{F}|\cdot)$ 
49:    $\mathbf{v}_G^t \leftarrow \sigma_G(\mathbf{F}|\cdot)\mathbf{v}_2^{t-1} + (1 - \sigma_G(\mathbf{F}|\cdot))\mathbf{v}_2$ 
                                      $\triangleright$  Expected value of gadget strategy
50:    $R_G^t(\mathbf{T}|\cdot) \leftarrow R_G^{t-1}(\mathbf{T}|\cdot) + \mathbf{v}_2 - v_G^{t-1}$ 
                                      $\triangleright$  Update regrets
51:    $R_G^t(\mathbf{F}|\cdot) \leftarrow R_G^{t-1}(\mathbf{F}|\cdot) + \mathbf{v}_2^t - v_G^t$ 
52:   return  $\mathbf{r}_2^t, R_G^t$ 
53: end function

```

---