Cite as: N. Brown, T. Sandholm, *Science*
10.1126/science.aao1733 (2017).

Superhuman AI for heads-up no-limit poker: Libratus beats top professionals

Noam Brown and Tuomas Sandholm*

Computer Science Department, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA 15213, USA.

***Corresponding author. Email: sandholm@cs.cmu.edu**

No-limit Texas hold'em is the most popular form of poker. Despite AI successes in perfect-information games, the private information and massive game tree have made no-limit poker difficult to tackle. We present Libratus, an AI that, in a 120,000-hand competition, defeated four top human specialist professionals in heads-up no-limit Texas hold'em, the leading benchmark and long-standing challenge problem in imperfect-information game solving. Our game-theoretic approach features application-independent techniques: an algorithm for computing a blueprint for the overall strategy, an algorithm that fleshes out the details of the strategy for subgames that are reached during play, and a self-improver algorithm that fixes potential weaknesses that opponents have identified in the blueprint strategy.

In recent years the field of artificial intelligence (AI) has advanced considerably. The measure of this progress has, in many cases, been marked by performance against humans in benchmark games. AI programs have defeated top humans in checkers (*1*), chess (*2*), and Go (*3*). In these perfect-information games both players know the exact state of the game at every point. In contrast, in imperfect-information games, some information about the state of the game is hidden from a player—for example, the opponent may hold hidden cards. Hidden information is ubiquitous in real-world strategic interactions, such as business strategy, negotiation, strategic pricing, finance, cybersecurity, and military applications, which makes research on general-purpose techniques for imperfect-information games particularly important.

Hidden information makes a game far more complex for a number of reasons. Rather than simply search for an optimal sequence of actions, an AI for imperfect-information games must determine how to balance actions appropriately, so that the opponent never finds out too much about the private information the AI has. For example, bluffing is a necessary feature in any competitive poker strategy, but bluffing all the time would be a bad strategy. In other words, the value of an action depends on the probability it is played.

Another key challenge is that different parts of the game cannot be considered in isolation; the optimal strategy for a given situation may depend on the strategy that would be played in situations that have not occurred (*4*). As a consequence, a competitive AI must always consider the strategy for the game as a whole.

Poker has a long history as a challenge problem for developing AIs that can address hidden information (*5*–*11*). No-limit Texas hold'em is the most popular form of poker in the world. The heads-up (that is, two-player) variant prevents opponent collusion and kingmaker scenarios where a bad player causes a mediocre player to shine, and therefore allows a clear winner to be determined. Due to its large size and strategic complexity, heads-up no-limit Texas hold'em (HUNL) has been the primary benchmark and challenge problem for imperfect-information game solving for several years. No prior AI has defeated top human players in this game.

In this paper we introduce Libratus, (*12*) an AI that takes a distinct approach to addressing imperfect-information games. In a 20-day, 120,000-hand competition featuring a $200,000 prize pool, it defeated top human professionals in HUNL. The techniques in Libratus do not use expert domain knowledge or human data and are not specific to poker; thus they apply to a host of imperfect-information games.

Game-solving approach in Libratus

Libratus features three main modules:

(i) The first module computes an abstraction of the game, which is smaller and easier to solve, and then computes game-theoretic strategies for the abstraction. The solution to this abstraction provides a detailed strategy for the early rounds of the game, but only an approximation for how to play in the more numerous later parts of the game. We refer to the solution of the abstraction as the blueprint strategy.

(ii) When a later part of the game is reached during play, the second module of Libratus constructs a finer-grained abstraction for that subgame and solves it in real time (*13*). Unlike subgame-solving techniques in perfect-information games, Libratus does not solve the subgame abstraction in isolation; instead, it ensures that the fine-grained solution to

the subgame fits within the larger blueprint strategy of the whole game. The subgame solver has several key advantages over prior subgame-solving techniques (*14*, *15*, *16*). Whenever the opponent makes a move that is not in the abstraction, a subgame is solved with that action included. We call this nested subgame solving. This technique comes with a provable safety guarantee.

(iii) The third module of Libratus—the self-improver—enhances the blueprint strategy. It fills in missing branches in the blueprint abstraction and computes a game-theoretic strategy for those branches. In principle, one could conduct all such computations in advance, but the game tree is way too large for that to be feasible. To tame this complexity, Libratus uses the opponents' actual moves to suggest where in the game tree such filling is worthwhile.

In the following three subsections, we present these three modules in more detail.

Abstraction and equilibrium finding: Building a blueprint strategy

One solution to the problem of imperfect information is to simply reason about the entire game as a whole, rather than just pieces of it. In this approach, a solution is pre-computed for the entire game, possibly using a linear program (*10*) or an iterative algorithm (*17*–*21*). For example, an iterative algorithm called counterfactual regret minimization plus (CFR+) was used to near-optimally solve heads-up limit Texas hold'em, a relatively simple version of poker, which has about 10^{13} unique decision points (*11*, *22*).

In contrast, HUNL (*23*) has 10^{161} decision points (*24*), so traversing the entire game tree even once is impossible. Precomputing a strategy for every decision point is infeasible for such a large game.

Fortunately, many of those decision points are very similar. For example, there is little difference between a bet of $100 and a bet of $101. Rather than consider every possible bet between $100 and $20,000, we could instead just consider increments of $100. This is referred to as action abstraction. An abstraction is a smaller, simplified game that retains as much as possible the strategic aspects of the original game. This drastically reduces the complexity of solving the game. If an opponent bets $101 during an actual match, then the AI may simply round this to a bet of $100 and respond accordingly (*25*–*27*). Most of the bet sizes included in Libratus's action abstraction were nice fractions or multiples of the pot [roughly determined by analyzing the most common bet sizes at various points in the game taken by prior top AIs in the Annual Computer Poker Competition (ACPC) (*28*)]. However, certain bet sizes early in the game tree were determined by an application-independent parameter optimization algorithm that converged to a locally optimal set of bet sizes (*29*).

An additional form of abstraction is abstraction of actions taken by chance, that is, card abstraction in the case of poker. Similar hands are grouped together and treated identically. Intuitively, there is little difference between a King-high flush and a Queen-high flush. Treating those hands as identical reduces the complexity of the game and thus makes it computationally easier. Nevertheless, there are still differences even between a King-high flush and a Queen-high flush. At the highest levels of play, those distinctions may be the difference between winning and losing. Libratus does not use any card abstraction on the first and second betting rounds. The last two betting rounds, which have a significantly larger number of states, are abstracted only in the blueprint strategy. The 55 million different hand possibilities on the third round were algorithmically grouped into 2.5 million abstract buckets, and the 2.4 billion different possibilities on the fourth round were algorithmically grouped into 1.25 million abstract buckets. However, the AI does not follow the blueprint strategy in these rounds and instead applies nested subgame solving, described in the next section, which does not use any card abstraction. Thus, each poker hand is considered individually during actual play. The card abstraction algorithm that we used was similar to that used in our prior AIs Baby Tartanian8 (*30*), which won the 2016 ACPC, and Tartanian7 (*31*–*33*), which won the 2014 ACPC (there was no ACPC in 2015).

Once the abstraction was constructed, we computed the blueprint strategy for Libratus by having the AI play simulated games of poker against itself (while still exploring the hypothetical outcomes of actions not chosen) using an improved version of an algorithm called Monte Carlo Counterfactual Regret Minimization (MCCFR). MCCFR (*17*, *34*, *35*) has a long history of use in successful poker AIs (*30*, *31*, *36*, *37*). MCCFR maintains a regret value for each action. Intuitively, regret represents how much the AI regrets having not chosen that action in the past. When a decision point is encountered during self play, the AI chooses actions with higher regret with higher probability (*38*). As more and more games are simulated, MCCFR guarantees that with high probability a player's average regret for any action (total regret divided by the number of iterations played) approaches zero. Thus, the AI's average strategy over all simulated games gradually improves. We will now describe the equilibrium-finding algorithm (*4*).

On each simulated game, MCCFR chooses one player (who we refer to as the traverser) that will explore every possible action and update his regrets, while the opponent simply plays according to the strategy determined by the current regrets. The algorithm switches the roles of the two players after each game, that is, a single hand of poker. Every time either player is faced with a decision point in a simulated game, the player will choose a probability distribution over actions based on regrets on those actions (which are determined by what he had learned in earlier games when he had

been in that situation). For the first game, the AI has not learned anything yet and therefore uses a uniform random distribution over actions. At traverser decision points, MCCFR explores every action in a depth-first manner. At opponent decision points, MCCFR samples an action based on the probability distribution. This process repeats at every decision point until the game is over and a reward is received, which is passed up. When a reward is returned by every action at a traverser decision point, MCCFR calculates the weighted average reward for that decision point based on the probability distribution over actions. The regret for each action is then updated by adding the value returned by that action, and subtracting the weighted average reward for the decision point. The weighted average reward is then passed up to the preceding decision point, and so on.

Our improved version of MCCFR traverses a smaller portion of the game tree on each iteration. Intuitively, there are many clearly suboptimal actions in the game, and repeatedly exploring them wastes computational resources that could be better used to improve the strategy elsewhere. Rather than explore every hypothetical alternative action to see what its reward would have been, our algorithm probabilistically skips over unpromising actions that have very negative regret as it proceeds deeper into the tree during a game (*30*, *39*). This led to a factor of three speedup of MCCFR in practice and allowed us to solve larger abstractions than were otherwise possible.

This skipping also mitigates the problems that stem from imperfect recall. The state-of-the-art practical abstractions in the field, including ours, are imperfect-recall abstractions where some aspects of the cards on the path of play so far are intentionally forgotten in order to be able to computationally afford to have a more detailed abstraction of the present state of cards (*30*–*32*, *40*). Since all decisions points in a single abstract card bucket share the same strategy, updating the strategy for one of them leads to updating the strategy for all of them. This is not an issue if all of them share the same optimal strategy at the solution reached, but in practice there are differences between their optimal strategies and they effectively "fight" to push the bucket's strategy toward their own optimal strategy. Skipping negative-regret actions means that decision points that will never be reached in actual play will no longer have their strategies updated, thereby allowing the decision points that will actually occur during play to move the bucket's strategy closer to their optimal strategies.

We ran our algorithm on an abstraction that is very detailed in the first two rounds of HUNL, but relatively coarse in the final two rounds. However, Libratus never plays according to the abstraction solution in the final two rounds. Rather, it uses the abstract blueprint strategy in those rounds only to estimate what reward a player should expect to receive with a particular hand in a subgame. This estimate is used to determine a more precise strategy during actual play, as described in the next section.

Nested safe subgame solving

Although purely abstraction-based approaches have produced strong AIs for poker (*25*, *30*, *32*, *41*), abstraction alone has not been enough to reach superhuman performance in HUNL. In addition to abstraction, Libratus builds upon prior research into subgame solving (*14*–*16*, *42*), in which a more detailed strategy is calculated for a particular part of the game that is reached during play. Libratus features many advances in subgame solving that proved critical to achieving superhuman performance (*43*).

Libratus plays according to the abstract blueprint strategy only in the early parts of HUNL, where the number of possible states is relatively small and we can afford the abstraction to be extremely detailed. Upon reaching the third betting round, or any earlier point in the game where the remaining game tree is sufficiently small (*44*), Libratus constructs a new, more detailed abstraction for the remaining subgame and solves it in real time.

However, there is a major challenge with subgame solving in imperfect-information games: a subgame cannot be solved in isolation because its optimal strategy may depend on other, unreached subgames (*4*). Prior AIs that used real-time subgame solving addressed this problem by assuming the opponent plays according to the blueprint strategy. However, the opponent can exploit this assumption by simply switching to a different strategy. For this reason, the technique may produce strategies that are far worse than the blueprint strategy and is referred to as unsafe subgame solving (*42*, *45*). Safe subgame solving techniques, on the other hand, guarantee that the subgame's new strategy makes the opponent no better off no matter what strategy the opponent might use (*14*). They accomplish this by ensuring that the new strategy for the subgame fits within the overarching blueprint strategy of the original abstraction. Ensuring the opponent is no better off relative to the blueprint strategy is trivially possible because we could just reuse the blueprint strategy. However, now that the abstraction is more detailed in the subgame and we can better distinguish the strategic nuances of the subgame, it may be possible to find an improvement over the prior strategy that makes the opponent worse off no matter what cards she is holding.

We now describe Libratus's core technique for determining an improved strategy in a subgame. For exposition, we assume Player 2 (P2) is determining an improved strategy against Player 1 (P1). Given that P2's strategy outside the subgame is σ_2, there exists some optimal strategy σ_2^* that P2

could play in the subgame. We would like to find or approximate σ_2^* in real time. We assume that, for each poker hand P1 might have, we have a good estimate of the value P1 receives in the subgame with that hand by playing optimally against σ_2^*, even though we do not know σ_2^* itself. Although we do not know these values exactly, we can approximate them with the values P1 receives in the subgame in the blueprint strategy. We later prove that if these estimates are approximately accurate, we can closely approximate σ_2^*.

To find a strategy close to σ_2^* in the subgame using only the values from the blueprint, we create an augmented subgame (Fig. 1) which contains the subgame and additional structures. At the start of the augmented subgame, P1 is privately dealt a random poker hand. Given that P2 plays according to σ_2 prior to the subgame, and given P1's dealt hand, there is a particular probability distribution over what hands P2 might have in this situation. P2 is dealt a poker hand according to this probability distribution. P1 then has the choice of entering the subgame (which is now far more detailed than in the blueprint strategy), or of taking an alternative payoff that ends the augmented subgame immediately. The value of the alternative payoff is our estimate, according to the blueprint strategy, of P1's value for that poker hand in that subgame. If P1 chooses to enter the subgame, then play proceeds normally until the end of the game is reached. We can solve this augmented subgame just as we did for the blueprint strategy (*46*).

For any hand P1 might have, P1 can do no worse in the augmented subgame than just choosing the alternative payoff (which awards our estimate of the expected value P1 could receive against σ_2^*). At the same time, P2 can ensure that for every poker hand P1 might have, he does no better than what he could receive against σ_2^*, because P2 can simply play σ_2^* itself. Thus, any solution to the augmented subgame must do approximately as well as σ_2^*—where the approximation error depends on how far off our estimates of P1's values are. P2 then uses the solution to the augmented subgame as P2's strategy going forward.

All of this relied on the assumption that we have accurate estimates of P1's values against σ_2^*. Although we do not know these values exactly, we can approximate them with values from the blueprint strategy. We now prove that if these estimates are approximately accurate, subgame solving will produce a strategy that is close to the quality of σ_2^*. Specifically, we define the exploitability of a strategy σ_2 as how much more σ_2 would lose, in expectation, against a worst-case opponent than what P2 would lose, in expectation, in an exact solution of the full game.

Theorem 1 uses a form of safe subgame solving we coin Estimated-Maxmargin. We define a margin for every P1 hand in a subgame as the expected value of that hand according to the blueprint minus what P1 could earn with that hand, in expectation, by entering the more-detailed subgame. Estimated-Maxmargin finds a strategy that maximizes the minimum margin among all P1 hands. It is similar to a prior technique called Maxmargin (*15*) except that the prior technique conservatively used as the margin what P1 could earn in the subgame, in expectation, by playing a best response to P2's blueprint strategy minus what P1 could earn, in expectation, by entering the more-detailed subgame.

Theorem 1. *Let σ_i be a strategy for a two-player zero-sum perfect-recall game, let S be a set of non-overlapping subgames in the game, and let σ_i^* be the least-exploitable strategy that differs from σ_i only in S. Assume that for any opponent decision point (hand in the case of poker) and any subgame in S, our estimate of the opponent's value in a best response to σ_i^* for that decision point in that subgame is off by at most Δ. Applying Estimated-Maxmargin subgame solving to any subgame in S reached during play results in overall exploitability at most 2Δ higher than that of σ_i^* (47).*

Although safe subgame solving techniques have been known for three years (*14*, *15*), they were not used in practice because empirically they performed significantly worse than unsafe subgame solving (*42*) head to head (*48*). Libratus features a number of advances to subgame solving that greatly improve effectiveness.

(i) Although we describe safe subgame solving as using estimates of P1 values, past techniques used upper bounds on those values (*14*, *15*). Using upper bounds guarantees that the subgame solution has exploitability no higher than the blueprint strategy. However, it tends to lead to overly conservative strategies in practice. Using estimates can, in theory, result in strategies with higher exploitability than the blueprint strategy, but Theorem 1 bounds how much higher this exploitability can be.

(ii) It arrives at better strategies in subgames than was previously thought possible. Past techniques ensured that the new strategy for the subgame made P1 no better off in that subgame for every situation. It turns out that this is an unnecessarily strong constraint. For example, 2♠7♥ is considered the worst hand in HUNL and should be folded immediately, which ends the game. Choosing any other action would result in an even bigger loss in expectation. Nevertheless, past subgame solving techniques would be concerned about P1 having 2♠7♥ in a subgame, which is unrealistic. Even if subgame solving resulted in a strategy that increased the value of 2♠7♥ a small amount in one subgame, that increase would not outweigh the cost of reaching the

subgame (that is, the cost of not folding with 2♠7♥). Thus, P2 can allow the value of some "unimportant" P1 hands to increase in subgames, so long as the increase is small enough that it is still a mistake for P1 to reach the subgame with that hand. We accomplish this by increasing the alternative reward of P1 hands in the augmented subgame by the extra cost to P1 of reaching the subgame, that is, the size of the mistake P1 would have to make to reach that subgame with that hand. By increasing the alternative reward in the augmented subgame of these "unimportant" hands, P2 develops a strategy in the subgame that better defends against hands P1 might actually have (*4*).

(iii) Libratus crafts a unique strategy in response to opponent bets, rather than rounding it to the nearest size in the abstraction. The optimal response to a bet of $101 is different from the optimal response to a bet of $100, but the difference is likely minor. For that reason, rounding an opponent bet of $101 to $100 is reasonable. But the optimal response to a bet of $150 is likely significantly different from the response to a bet of $100 or a bet of $200. In principle one could simply increase the number of actions in the abstraction, perhaps by considering bets in increments of $10 rather than $100, so that the error from rounding is smaller. However, the size of the abstraction, and the time needed to solve it, increases prohibitively as more actions are added.

Therefore, rather than round to the nearest action, Libratus calculates a unique response in real time to off-tree actions, that is, an action taken by an opponent that is not in the abstraction. Libratus attempts to make the opponent no better off, no matter what hand the opponent might have, for having chosen the off-tree action rather than an action in the abstraction. It does this by generating and solving an augmented subgame following the off-tree action where the alternative payoff is the best in-abstraction action the opponent could have taken (the best action may differ across hands).

Libratus repeats this for every subsequent off-tree action in a process we call nested subgame solving (see Fig. 2). Later we provide experiments that demonstrate that this technique improves the worst-case performance of poker AIs by more than an order of magnitude compared to the best technique for rounding opponent actions to a nearby in-abstraction action.

(iv) Because the subgame is solved in real time, the abstraction in the subgame can also be decided in real time and change between hands. Libratus leverages this feature by changing, at the first point of subgame solving, the bet sizes it will use in that subgame and every subsequent subgame of that poker hand, thereby forcing the opponent to continually adapt to new bet sizes and strategies (*49*).

The authors of the poker AI DeepStack independently and concurrently developed an algorithm similar to nested subgame solving, which they call continual re-solving (*50*). In an Internet experiment, DeepStack defeated human professionals who are not specialists in HUNL. However, DeepStack was never shown to outperform prior publicly-available top AIs in head-to-head performance, whereas Libratus beats the prior leading HUNL poker AI Baby Tartanian8 by a wide margin, as we discuss later.

Like Libratus, DeepStack computes in real time a response to the opponent's specific bet and uses estimates rather than upper bounds on the opponent's values. It does not share Libratus's improvement of de-emphasizing hands the opponent would only be holding if she had made an earlier mistake, and does not share the feature of changing the subgame action abstraction between hands.

DeepStack solves a depth-limited subgame on the first two betting rounds by estimating values at the depth limit via a neural network. This allows it to always calculate real-time responses to opponent off-tree actions, while Libratus typically plays according to its pre-computed blueprint strategy in the first two rounds.

Because Libratus typically plays according to a pre-computed blueprint strategy on the first two betting rounds, it rounds an off-tree opponent bet size to a nearby in-abstraction action. The blueprint action abstraction on those rounds is dense in order to mitigate this weakness. In addition, Libratus has a unique self-improvement module to augment the blueprint strategy over time, which we now introduce.

Self-improvement

The third module of Libratus is the self-improver. It enhances the blueprint strategy in the background. It fills in missing branches in the blueprint abstraction and computes a game-theoretic strategy for those branches. In principle, one could conduct all such computations in advance, but the game tree is way too large for that to be feasible. To tame this complexity, Libratus uses the opponents' actual moves to suggest where in the game tree such filling is worthwhile.

The way machine learning has typically been used in game playing is to try to build an opponent model, find mistakes in the opponent's strategy (e.g., folding too often, calling too often, etc.), and exploit those mistakes (*51–53*). The downside is that trying to exploit the opponent opens oneself to being exploited. (A certain conservative family of exploitation techniques constitutes the sole exception to this downside (*51–53*).) For that reason, to a first approximation, Libratus did not do opponent exploitation. Instead, it used the data of the bet sizes that the opponents used to suggest which branches should be added to the blueprint, and it then computed game-theoretic strategies for those branches in the background.

In most situations that can occur in the first two betting rounds, real-time subgame solving as used in Libratus would likely not produce a better strategy than the blueprint, because the blueprint already uses no card abstraction in those rounds and conducting subgame solving in real time so early in the game tree would require heavy abstraction in the subgame. For these reasons, Libratus plays according to the precomputed blueprint strategy in these situations. In those rounds there are many bet sizes in the abstraction, so the error from rounding to a nearby size is small. Still, there is some error, and this could be reduced by including more bet sizes in the abstraction. In the experiment against human players described in the next section, Libratus analyzed the bet sizes in the first betting round most heavily used by its opponents in aggregate during each day of the competition. Based on the frequency of the opponent bet sizes and their distance from the closest bet size in the abstraction, Libratus chose k bet sizes for which it would try to calculate a response overnight (*54*). Each of those bet sizes for which reasonable convergence had been reached by the morning was then added to the blueprint strategy together with the newly-computed strategy following that bet size. In this way Libratus was able to progressively narrow its gaps as the competition proceeded by leveraging the humans' ability to find potential weaknesses. Furthermore, these fixes to its strategy are universal: they work against all opponents, not just the opponents that Libratus has faced.

Libratus's self-improvement comes in two forms. For one of them, when adding one of the k bet sizes, a default sibling bet size is also used during the equilibrium finding so as to not assume that the opponent necessarily only uses the bet size that will be added. For the other, a default bet size is not used. This can be viewed as more risky and even exploitative, but Libratus mitigates the risk by using that part of the strategy during play only if the opponent indeed uses that bet size most of the time (*4*).

Experimental evaluation

To evaluate the strength of the techniques used in Libratus, we first tested the overall approach of the AI on scaled-down variants of poker before proceeding to tests on full HUNL. These moderate-sized variants consisted of only two or three rounds of betting rather than four, and at most three bet sizes at each decision point. The smaller size of the games allowed us to precisely calculate exploitability, the distance from an optimal strategy. Performance was measured in milli-big blinds per hand (mbb/hand), the average number of big blinds won per 1,000 hands.

In the first experiment, we compared using no subgame solving, unsafe subgame solving (*42*) (in which a subgame is solved in isolation with no theoretical guarantees on performance), and safe subgame solving just once upon reaching the final betting round of the game. Both players were constrained to choosing among only two different bet sizes, so off-tree actions were not an issue in this first experiment. The results are shown in Table 1. In all cases, safe subgame solving reduced exploitability by more than a factor of 4 relative to no subgame solving. In one case, unsafe subgame solving led to even lower exploitability, while in another it increased exploitability by nearly an order of magnitude more than if no subgame solving had been used. This demonstrates that although unsafe subgame solving may produce strong strategies in some games, it may also lead to far worse performance. Safe subgame solving, in contrast, reduced exploitability in all games.

In the second experiment, we constructed an abstraction of a game which only includes two of the three available bet sizes. If the opponent played the missing bet size, the AI either used action translation [in which the bet is rounded to a nearby size in the abstraction; we compared against the leading action translation technique (*27*)], or nested subgame solving. The results are shown in Table 2. Nested subgame solving reduced exploitability by more than an order of magnitude relative to action translation.

Next we present experiments in full HUNL. After constructing Libratus, we tested the AI against the prior leading HUNL poker AI, our 2016 bot Baby Tartanian8, which had defeated all other poker AIs with statistical significance in the most recent ACPC (*55*). We report average win rates followed by the 95% confidence interval. Using only the raw blueprint strategy, Libratus lost to Baby Tartanian8 by 8 ± 15 mbb/hand. Adding state-of-the-art post-processing on the 3rd and 4th betting rounds (*31*), such as eliminating low-probability actions that are likely only positive owing to insufficient time to reach convergence, led to the Libratus blueprint strategy defeating Baby Tartanian8 by 18 ± 21 mbb/hand. Eliminating low-probability actions empirically leads to better performance against non-adjusting AIs. However, it also increases the exploitability of the AI because its strategy becomes more predictable. The full Libratus agent did not use post-processing on the third and fourth betting rounds. On the first two rounds, Libratus primarily used a new, more robust, form of post-processing (*4*).

The next experiment evaluated nested subgame solving (with no post-processing) using only actions that are in Baby Tartanian8's action abstraction. Libratus won by 59 ± 28 mbb/hand (*56*). Finally, applying the nested subgame solving structure used in the competition resulted in Libratus defeating Baby Tartanian8 by 63 ± 28 mbb/hand. The results are shown in Table 3. In comparison, Baby Tartanian8 defeated the next two strongest AIs in the ACPC by 12 ± 10 mbb/hand and 24 ± 20 mbb/hand.

Finally, we tested Libratus against top humans. In January 2017, Libratus played against a team of four top HUNL

specialist professionals in a 120,000-hand Brains vs. AI challenge match over 20 days. The participants were Jason Les, Dong Kim, Daniel McCauley, and Jimmy Chou. A prize pool of $200,000 was allocated to the four humans in aggregate. Each human was guaranteed $20,000 of that pool. The remaining $120,000 was divided among them based on how much better the human did against Libratus than the worst-performing of the four humans. Libratus decisively defeated the humans by a margin of 147 mbb/hand, with 99.98% statistical significance and a p-value of 0.0002 (if the hands are treated as independent and identically distributed), see Fig. 3 (*57*). It also beat each of the humans individually.

Conclusions

Libratus presents an approach that effectively addresses the challenge of game-theoretic reasoning under hidden information in a large state space. The techniques that we developed are largely domain independent and can thus be applied to other strategic imperfect-information interactions, including non-recreational applications. Owing to the ubiquity of hidden information in real-world strategic interactions, we believe the paradigm introduced in Libratus will be important for the future growth and widespread application of AI.

REFERENCES AND NOTES

1. J. Schaeffer, *One Jump Ahead: Challenging Human Supremacy in Checkers* (Springer, 1997).
2. M. Campbell, A. J. Hoane Jr., F.-H. Hsu, Deep Blue. *Artif. Intell.* **134**, 57–83 (2002). doi:10.1016/S0004-3702(01)00129-1
3. D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis, Mastering the game of Go with deep neural networks and tree search. *Nature* **529**, 484–489 (2016). doi:10.1038/nature16961 Medline
4. See supplementary materials for more details.
5. J. Nash, "Non-cooperative games," thesis, Princeton University (1950).
6. J. F. Nash, L. S. Shapley, *Contributions to the Theory of Games*, H. W. Kuhn, A. W. Tucker, Eds. (Princeton Univ. Press, 1950), vol. 1, pp. 105–116.
7. D. A. Waterman, Generalization learning techniques for automating the learning of heuristics. *Artif. Intell.* **1**, 121–170 (1970). doi:10.1016/0004-3702(70)90004-4
8. J. Shi, M. Littman, in *CG '00: Revised Papers from the Second International Conference on Computers and Games* (Springer, 2002), pp. 333–345.
9. D. Billings *et al*., in *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)* (Morgan Kaufmann Publishers, San Francisco, 2003), pp. 661–668.
10. A. Gilpin, T. Sandholm, in *Proceedings of the National Conference on Artificial Intelligence (AAAI)* (AAAI Press, 2005), pp. 1684–1685.
11. M. Bowling, N. Burch, M. Johanson, O. Tammelin, Heads-up limit hold'em poker is solved. *Science* **347**, 145–149 (2015). doi:10.1126/science.1259433 Medline
12. Libratus is Latin and means balanced (for approximating Nash equilibrium) and forceful (for its powerful play style and strength).
13. An imperfect-information subgame (which we refer to simply as a subgame) is defined differently than how a subgame is usually defined in game theory. The usual definition requires that a subgame starts with the players knowing the exact state of the game, that is, no information is hidden from any player. Here, an imperfect-information subgame is determined by information that is common knowledge to the players. For example, in poker, a subgame is defined by the sequence of visible board cards and actions the players have taken so far. Every possible combination of private cards—that is, every node in the game tree which is consistent with the common knowledge—is a root of this subgame. Any node that descends from a root node is also included in the subgame. A formal definition is provided in the supplementary material.
14. N. Burch, M. Johanson, M. Bowling, in *AAAI Conference on Artificial Intelligence (AAAI)* (AAAI Press, 2014), pp. 602–608.
15. M. Moravcik, M. Schmid, K. Ha, M. Hladik, S. Gaukrodger, in *AAAI Conference on Artificial Intelligence (AAAI)* (AAAI Press, 2016).
16. E. Jackson, in *AAAI Workshop on Computer Poker and Imperfect Information* (AAAI Press, 2014).
17. M. Zinkevich, M. Johanson, M. H. Bowling, C. Piccione, in *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)* (Neural Information Processing Systems Foundation, Inc., 2007), pp. 1729–1736.
18. Y. Nesterov, Excessive gap technique in nonsmooth convex minimization. *SIAM J. Optim.* **16**, 235–249 (2005). doi:10.1137/S1052623403422285
19. S. Hoda, A. Gilpin, J. Peña, T. Sandholm, Smoothing techniques for computing Nash equilibria of sequential games. *Math. Oper. Res.* **35**, 494–512 (2010). doi:10.1287/moor.1100.0452
20. A. Gilpin, J. Peña, T. Sandholm, First-order algorithm with O(ln(1/ϵ)) convergence for ϵ-equilibrium in two-person zero-sum games. *Math. Program.* **133**, 279–298 (2012). doi:10.1007/s10107-010-0430-2.
21. C. Kroer, K. Waugh, F. Klnç-Karzan, T. Sandholm, in *Proceedings of the ACM Conference on Economics and Computation (EC)* (ACM, New York, 2017).
22. O. Tammelin, N. Burch, M. Johanson, M. Bowling, in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* (AAAI Press, 2015), pp. 645–652.
23. The version of HUNL that we refer to, which is used in the Annual Computer Poker Competition, allows bets in increments of $1, with each player having $20,000 at the beginning of a hand.
24. M. Johanson, "Measuring the size of large no-limit poker games," (Technical Report, Univ. of Alberta Libraries, 2013).
25. A. Gilpin, T. Sandholm, T. B. Sørensen, in *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems* (International Foundation for Autonomous Agents and Multiagent Systems, 2008), vol. 2, pp. 911–918.
26. D. Schnizlein, M. Bowling, D. Szafron, in *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence* (AAAI Press, 2009), pp. 278–284.
27. S. Ganzfried, T. Sandholm, in *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence* (AAAI Press, 2013), pp. 120–128.
28. Annual Computer Poker Competition; www.computerpokercompetition.org.
29. N. Brown, T. Sandholm, in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI)* (AAAI Press, 2014), pp. 594–601.
30. N. Brown, T. Sandholm, in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)* (AAAI Press, 2016), pp. 4238–4239.
31. N. Brown, S. Ganzfried, T. Sandholm, in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems* (International Foundation for Autonomous Agents and Multiagent Systems, 2015), pp. 7–15.
32. N. Brown, S. Ganzfried, T. Sandholm, in *AAAI Conference on Artificial Intelligence (AAAI)* (AAAI Press, 2015), pp. 4270–4271.
33. M. Johanson, N. Burch, R. Valenzano, M. Bowling, in *Proceedings of the 2013 International Conference on Autonomous Agents and Multiagent Systems* (International Foundation for Autonomous Agents and Multiagent Systems, 2013), pp. 271–278.
34. M. Lanctot, K. Waugh, M. Zinkevich, M. Bowling, in *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)* (Neural Information Processing Systems Foundation, Inc., 2009), pp. 1078–1086.
35. R. Gibson, M. Lanctot, N. Burch, D. Szafron, M. Bowling, in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence* (AAAI Press, 2012), pp. 1355–1361.
36. M. Johanson, N. Bard, M. Lanctot, R. Gibson, M. Bowling, in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems* (International Foundation for Autonomous Agents and Multiagent Systems, 2012), vol. 2, pp. 837–846.

37. R. Gibson, "Regret minimization in games and the development of champion multiplayer computer poker-playing agents," thesis, University of Alberta (2014).
38. There are a number of theoretically correct ways to choose actions on the basis of their regrets. The most common is regret matching, in which an action is chosen in proportion to its positive regret (*58*). Another common choice is hedge (*59*, *60*).
39. An action *a* with regret $R(a)$ that is below a threshold C (where C is negative) is sampled with probability $K/[K + C - R(a)]$, where K is a positive constant. There is additionally a floor on the sample probability. This sampling is only done for about the last half of iterations to be run; the first half is conducted using traditional external-sampling MCCFR. Other formulas can also be used.
40. K. Waugh *et al*., in *Symposium on Abstraction, Reformulation, and Approximation (SARA)* (AAAI Press, 2009).
41. M. Johanson, N. Bard, N. Burch, M. Bowling, in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence* (AAAI Press, 2012), pp. 1371–1379.
42. S. Ganzfried, T. Sandholm, in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (International Foundation for Autonomous Agents and Multiagent Systems, 2015), pp. 37–45.
43. N. Brown, T. Sandholm, *Adv. Neural Inf. Process. Syst.* **30**, 689–699 (2017).
44. In Libratus, we considered "sufficiently small" to be situations where no additional bets or raises could be made.
45. Despite lacking theoretical guarantees, unsafe subgame solving empirically performs well in certain situations and requires less information to be precomputed. For this reason, Libratus uses it once upon first reaching the third betting round, while using safe subgame solving in all subsequent situations.
46. We solved augmented subgames using a heavily optimized form of the CFR+ algorithm (*22*, *61*) because of the better performance of CFR+ in small games where a precise solution is desired. The optimizations we use keep track of all possible P1 hands rather than dealing out a single one at random.
47. Note that the theorem only assumes perfect recall in the actual game, not in the abstraction that is used for computing a blueprint strategy. Furthermore, applying Estimated-Maxmargin assumes that that subroutine maximizes the minimum margin; a sufficient condition for doing so is that there is no abstraction in the subgame.
48. Indeed, the original purpose of safe subgame solving was merely to reduce space usage by reconstructing subgame strategies rather than storing them.
49. Specifically, Libratus increased or decreased all its bet sizes by a percentage chosen uniformly at random between 0 and 8%.
50. M. Moravčík, M. Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, M. Bowling, DeepStack: Expert-level artificial intelligence in heads-up no-limit poker. *Science* **356**, 508–513 (2017). doi:10.1126/science.aam6960 Medline
51. D. Billings, D. Papp, J. Schaeffer, D. Szafron, in *Proceedings of the National Conference on Artificial Intelligence (AAAI)* (AAAI Press, 1998), pp. 493–499.
52. S. Ganzfried, T. Sandholm, in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (International Foundation for Autonomous Agents and Multiagent Systems, 2011).
53. S. Ganzfried, T. Sandholm, Safe opponent exploitation. *ACM Transaction on Economics and Computation (TEAC)* **3**, 1–28 (2015). doi:10.1145/2716322
54. Based on the available computing resources, we chose $k = 3$ so that the algorithm could typically fix three holes to reasonable accuracy in 24 hours.
55. Baby Tartanian8 and all other AIs in the ACPC are available to ACPC participants for benchmarking.
56. Baby Tartanian8 uses action translation in response to bet sizes that are not in its action abstraction. Our experiments above demonstrated that action translation performs poorly compared to subgame solving. Using only bet sizes in Baby Tartanian8's abstraction disentangles the effects of action translation from the improvement of nested subgame solving. Baby Tartanian8 still used actions that were not in Libratus's abstraction, and therefore the experiments can be considered conservative.
57. Because both the humans and the AI adapted over the course of the competition, treating the hands as independent is not entirely inappropriate. We include confidence figures to provide some intuition for the variance in HUNL. In any case, 147 mbb/hand over 120,000 hands is considered a massive and unambiguous victory in HUNL.
58. S. Hart, A. Mas-Colell, A simple adaptive procedure leading to correlated equilibrium. *Econometrica* **68**, 1127–1150 (2000). doi:10.1111/1468-0262.00153
59. N. Littlestone, M. K. Warmuth, The weighted majority algorithm. *Inf. Comput.* **108**, 212–261 (1994). doi:10.1006/inco.1994.1009
60. Y. Freund, R. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.* **55**, 119–139 (1997). doi:10.1006/jcss.1997.1504
61. M. Johanson, K. Waugh, M. Bowling, M. Zinkevich, in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* (AAAI Press, 2011), pp. 258–265.
62. L. Kocsis, C. Szepesvári, in *European Conference on Maching Learning (ECML)* (Springer, 2006), pp. 282–293.
63. R. Coulom, *Computers and Games* (Springer, 2007), pp. 72–83.
64. D. E. Knuth, R. W. Moore, An analysis of alpha-beta pruning. *Artif. Intell.* **6**, 293–326 (1975). doi:10.1016/0004-3702(75)90019-3
65. J. F. Nash, Equilibrium points in *n*-person games. *Proc. Natl. Acad. Sci. U.S.A.* **36**, 48–49 (1950). doi:10.1073/pnas.36.1.48 Medline
66. N. Brown, T. Sandholm, in *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)* (2015), pp. 1972–1980.
67. N. Brown, T. Sandholm, in *International Conference on Machine Learning* (Proceedings of Machine Learning Research, 2017).
68. S. Ganzfried, T. Sandholm, K. Waugh, in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (International Foundation for Autonomous Agents and Multiagent Systems, 2012), pp. 871–878.

ACKNOWLEDGMENTS

This material is based on research supported by the National Science Foundation under grants IIS-1718457, IIS-1617590, and CCF-1733556, and by the ARO under award W911NF-17-1-0082, as well as XSEDE computing resources provided by the Pittsburgh Supercomputing Center. The Brains vs. AI competition was sponsored by Carnegie Mellon University, Rivers Casino, GreatPoint Ventures, Avenue4Analytics, TNG Technology Consulting, Artificial Intelligence, Intel, and Optimized Markets, Inc. We thank Ben Clayman for computing statistics of the play of our AIs against humans. The data presented in this paper are shown in the main text and supplementary material. Additional data can be obtained from the corresponding author upon request. Because HUNL poker is played commercially, the risk associated with releasing the code outweighs the benefits. To aid reproducibility, we have included the pseudo-code for the major components of our program in (*4*). The technology has been exclusively licensed to Strategic Machine, Inc., and the authors have ownership interest in the company.

SUPPLEMENTARY MATERIALS

www.sciencemag.org/cgi/content/full/science.aao1733/DC1
Supplementary text
Figs. S1 and S2
Table S1
References (*62*–*68*)

22 June 2017; accepted 12 December 2017
Published online 17 December 2017
10.1126/science.aao1733

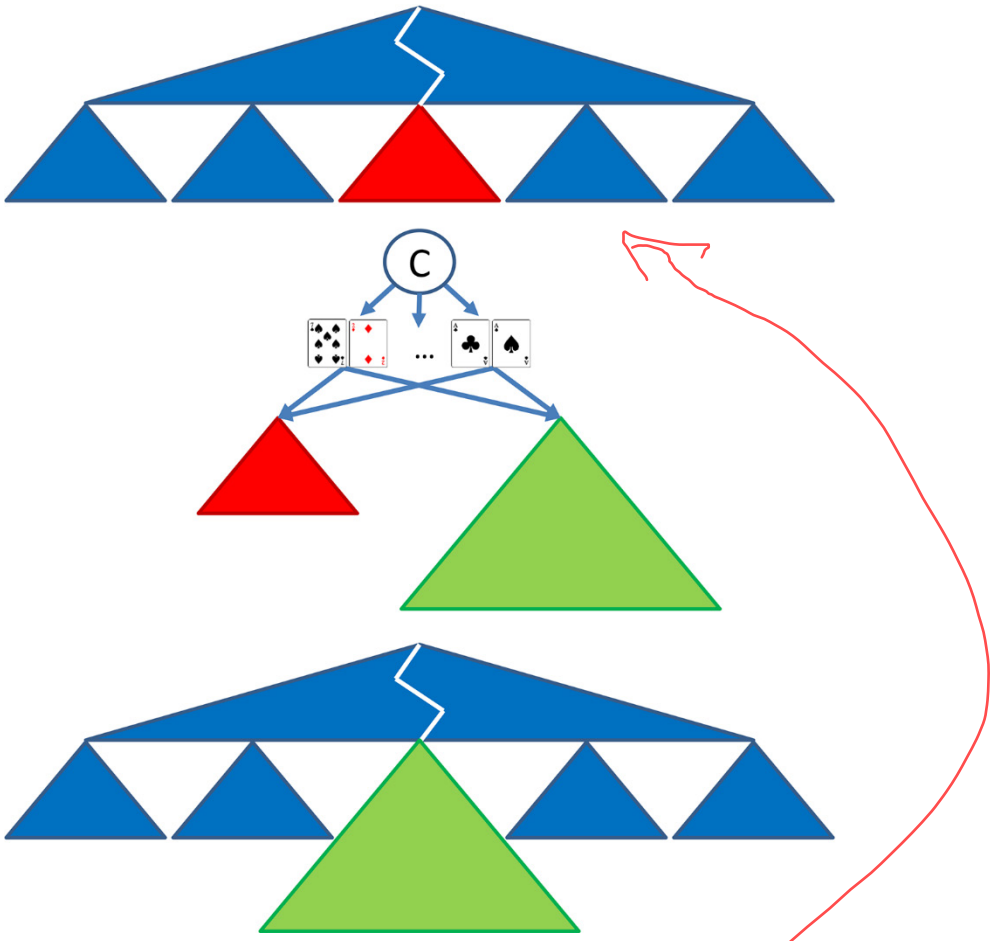


Fig. 1. Subgame solving. Top: A subgame is reached during play. Middle: A more detailed strategy for that subgame is determined by solving an augmented subgame, in which on each iteration the opponent is dealt a random poker hand and given the choice of taking the expected value of the old abstraction (red), or of playing in the new, finer-grained abstraction (green) where the strategy for both players can change. This forces Libratus to make the finer-grained strategy at least as good as in the original abstraction against every opponent poker hand. Bottom: The new strategy is substituted in place of the old one.

Fig. 2. A visualization of nested subgame solving. Every time a subgame is reached during play, a more detailed abstraction is constructed and solved just for that subgame, while fitting its solution within the overarching blueprint strategy.

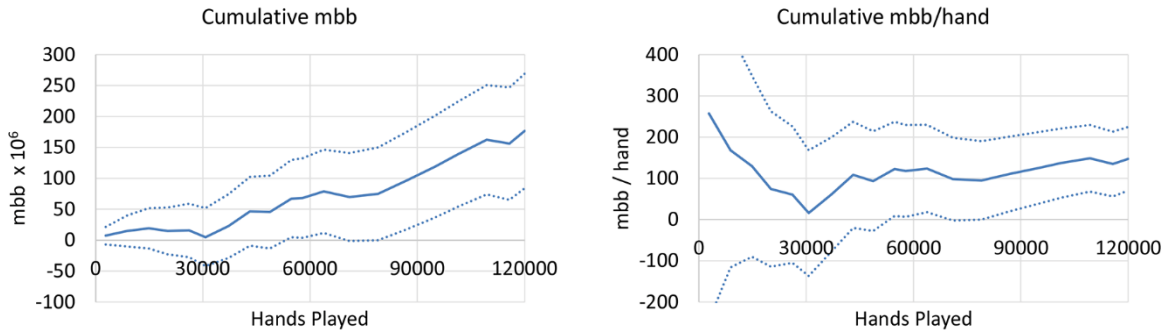


Fig. 3. Libratus performance against top humans. Shown are the results of the 2017 Brains vs. AI competition. The 95% confidence intervals (if the hands are treated as independent and identically distributed) are shown as dotted lines.

Table 1. Exploitability of subgame solving techniques on smaller poker variants.

Simplified Game:	Small 2-Round	Large 2-Round Hold'em	3-Round Hold'em
No subgame solving	91.3 mbb/hand	41.3 mbb/hand	346 mbb/hand
Unsafe subgame solving	5.51 mbb/hand	397 mbb/hand	79.3 mbb/hand
Safe subgame solving	22.6 mbb/hand	9.84 mbb/hand	72.6 mbb/hand

Table 2. Exploitability of nested subgame solving. Shown is the comparison to no nested subgame solving (which instead uses the leading action translation technique) in a small poker variant.

	Exploitability
No nested subgame solving	1,465 mbb/hand
Nested unsafe subgame solving	148 mbb/hand
Nested safe subgame solving	119 mbb/hand

Table 3. Head-to-head performance of Libratus. Shown are results for the Libratus blueprint strategy as well as forms of nested subgame solving against Baby Tartanian8 in HUNL.

	Performance against *Baby Tartanian8*
Blueprint	−8 ±15 mbb/hand
Blueprint with post-processing	18 ± 21 mbb/hand
On-tree nested subgame solving	59 ± 28 mbb/hand
Full nested subgame solving	63 ± 28 mbb/hand

Superhuman AI for heads-up no-limit poker: Libratus beats top professionals

Noam Brown and Tuomas Sandholm

published online December 17, 2017

www.sciencemag.org/cgi/content/full/science.aao1733/DC1

Supplementary Materials for

Superhuman AI for heads-up no-limit poker: Libratus beats top professionals

Noam Brown and Tuomas Sandholm*

*Corresponding author. Email: sandholm@cs.cmu.edu

Published 17 December 2017 on *Science* First Release
DOI: 10.1126/science.aao1733

This PDF file includes:

SUPPLEMENTARY MATERIAL

The challenge of hidden information

In this section we provide intuition for why imperfect-information games are difficult, and what it means to "solve" a game. Having perfect information is critical to all top-performing AIs in games such as checkers, chess, and Go because it allows the AI to only consider the subgame it is in—the "mini-game" formed by the current state of the game (for example, the current positions of all pieces on the board in chess) and all future states that can be reached from that point on (*62–64*). Other unreachable states, or past states, are irrelevant to determining the optimal strategy. For example, in chess, if an opponent opens with the Queen's Gambit, then knowing how to play the Sicilian Defense is irrelevant. This is not true in imperfect-information games.

In imperfect-information games, it is not generally possible to reason about a part of the game in isolation because the optimal strategy in the subgame reached during play may depend on the optimal strategy of subgames not reached. We demonstrate this in a simple game we call Coin Toss, Figure S1. It is played between players P_1 and P_2. A coin is flipped and lands either Heads or Tails with equal probability; only P_1 sees the outcome. P_1 can then choose between actions "Sell" and "Play." If the coin lands Heads, it is considered lucky and P_1 can receive $\$0.50$ for choosing Sell. On the other hand, if the coin lands Tails, it is considered unlucky and P_1 must pay $\$0.50$ to get rid of it (that is, the Sell action results in P_1 receiving $-\$0.50$). If P_1 instead chooses Play, then P_2 has the opportunity to guess how the coin landed. If P_2 guesses correctly, P_1 receives a reward of $-\$1$ and P_2 receives a reward of $\$1$ (the figure shows rewards for P_1; P_2 receives the negation of P_1's reward). We now discuss what would be the optimal strategy for P_2 in the subgame S that occurs after P_1 chooses Play.

On the one hand, were P_2 to always guess Heads, P_1 would receive $\$0.50$ for choosing Sell

when the coin lands Heads, and $\$1$ for choosing Play when it lands Tails. This would result in an average of $\$0.75$ for P_1. On the other hand, were P_2 to always guess Tails, P_1 would receive $\$1$ for choosing Play when it lands Heads, and $-\$0.50$ for choosing Sell when it lands Tails. This would result in an average reward of $\$0.25$ for P_1. However, P_2 would do even better by guessing Heads with 25% probability and Tails with 75% probability. In that case, P_1 could only receive $\$0.50$ (on average) by choosing Play when the coin lands Heads—the same value received for choosing Sell. Similarly, P_1 could only receive $-\$0.50$ by choosing Play when the coin lands Tails, which is the same value received for choosing Sell. This would yield an average reward of $\$0$ for P_1. It is easy to see that this is the best P_2 could do, because P_1 can receive at least $\$0$ in expectation by always choosing Sell. Therefore, choosing Heads with 75% probability and Tails with 25% probability is a solution to the game, or optimal strategy (aka. minmax strategy), for P_2.

Now suppose the coin is considered lucky if it lands Tails and unlucky if it lands Heads. That is, the reward for selling the coin when it lands Heads is now $-\$0.50$ and $\$0.50$ when it lands Tails. It is easy to see that P_2's optimal strategy for the "Play" subgame is now to guess Heads with 75% probability and Tails with 25% probability. This shows that a player's optimal strategy in a subgame can depend on the outcomes and optimal strategies in other parts of the game. Therefore, one cannot solve a subgame using information about that subgame alone. This is a key challenge in playing imperfect-information games as opposed to perfect-information games.

Description of heads-up no-limit Texas hold'em

No-limit Texas hold'em is the most popular form of poker. It is also an extremely large game at 10^{161} decision points. The head's up (that is, two-player) variant prevents opponent collusion and kingmaker scenarios where a bad player causes a mediocre player to shine, and therefore al-

lows a clear winner to be determined. For all of these reasons, heads-up no-limit Texas hold'em (HUNL) is the primary testbed for research on solving large imperfect-information games.

In the common form of HUNL poker agreed upon by the research community, each player starts each hand with $20,000 in chips. One player is designated P_1, while the other is P_2, and this assignment alternates between hands. HUNL consists of four rounds of betting. On a round of betting, each player can choose to either fold, call, or raise. If a player folds, that player immediately surrenders the pot to the opponent and the game ends. If a player calls, that players places a number of chips in the pot equal to the opponent's contribution. If a player raises, that player adds more chips to the pot than the opponent's contribution. A round of betting ends after a player calls. Players can continue to go back and forth with raises in a round until running out of chips.

If either player chooses to raise first in a round, they must raise a minimum of $100. If a player raises after another player has raised, that raise must be greater than or equal to the last raise. The maximum amount for a bet or raise is the remainder of that player's chip stack, which in our model is $20,000 at the beginning of a game.

At the start of HUNL, both players receive two private cards from a standard 52-card deck. P_1 must place a big blind of $100 in the pot, while P_2 must place a small blind of $50 in the pot. There is then a first round of betting (called the preflop), where P_2 acts first. When the round ends, three community cards are dealt face up between the players. There is then a second round of betting (called the flop), where P_1 acts first. After that round of betting ends, another community card is dealt face up, and a third round of betting (called the turn) commences where P_1 acts first. Finally, a fifth community card is dealt face up, and a fourth betting round (called the river) occurs, again with P_1 acting first. If neither player folds before the final betting round completes, the player with the best five-card poker hand, constructed from her two private cards and the five face-up community cards, wins the pot. In the case of a tie, the pot is split evenly.

Brains vs. AI competition setup

As is common in bridge and in the ACPC, we used duplicate matches to reduce the role of luck. The four humans were matched into two pairs. Within each pair, whatever cards one human received against Libratus, Libratus received against the other human. This way neither the human team nor Libratus could get systematically lucky.

In order to try to reduce variance, we split the pot by averaging over all possible roll-outs of remaining cards—as opposed to just one roll-out—in situations where the players went all-in before the final card had been dealt.

There are many further design choices in setting up a man-machine poker competition. The 2017 Brains vs. AI competition was conservatively designed to favor the humans, so that in case the AI won, the results would be conclusive. These choices included the following.

- A large number of hands (120,000) was played. This gave the humans the best chance to find weaknesses in the AI and to exploit them.

- The humans were allowed to choose how many days they would spend to play the 120,000 hands, how many break days to have, etc. They chose to play the hands in 20 back-to-back days. This was likely the best choice for the humans in that it minimized the time the AI had available to fix its strategy as the humans found and exploited its weaknesses. The humans were also allowed to choose the times of day to play, how many hands each session had, and the lengths of the coordinated breaks. Furthermore, they were allowed to take breaks at any time. They were also allowed to stop playing for the day if they felt tired or sick; they exercised this option infrequently.

- The humans were allowed, but not required, to play two tables at once. This enabled each human to switch to the other table while the AI was thinking. Each human had the choice

of playing any number of actions and hands of the session on one table before switching to the other table, and so on.

- Per the humans' request, a 4-color deck was used in the user interface so flushes were easier for the human visual system to recognize.

- User-programmable hot keys (most importantly for making bets that are various unusual fractions of the pot) were provided for the humans in the user interface.

- The humans requested specific high-resolution monitors, and two of them were provided for each human, one per table.

- The humans were allowed to bring their preferred computer mice.

- Each human was allowed to choose whether to keep the streaming video chat (Twitch) on or off—even dynamically. The humans made different choices on this dimension. Some felt that they play better when while interacting with a supportive audience while others felt they perform better in private.

- Due to the duplicate matches, only one human in each pair could play in public and the other had to be isolated in a private room (otherwise they, or the audience, could tell each other what cards are coming next). We let the humans decide which human in each pair played in public versus in private because they had different preferences between these options. We also allowed them to alter that decision dynamically during the competition.

- Each hand started with each player having 200 big blinds in their chip stack. This is the standard in the ACPC and is consider a "deep stack". Another common stack size is 100 big blinds, but that would be more favorable for the AI due to the smaller game size.

- It is well known that sometimes a player's timing leaks information about the player's cards to the opponent. We agreed that the AI would not use timing tells while the humans were allowed to use them.

- The action history in the hand was displayed on the user interface. This helped the human in case he forgot what had happened in the hand so far.

- Both sides had access to the day's hand histories every evening, including hands that the opponent folded. The self-improver in Libratus used information about which opponent actions were taken, but did not use information about the opponents' hands.

- The humans were allowed to use computers and any software for analysis in the evenings and breaks. They did that for hours per day. The humans were also allowed to bring outsiders to help them with their analysis, and they exercised this option.

- The humans were allowed to collaborate and coordinate actions (except not within each hand). They exercised this option heavily. For example, they coordinated how they explored the AI's potential weaknesses via using different bet sizes.

- The humans were allowed to think as long as they wanted. On average, Daniel McCauley thought for 22.4 seconds per hand, Jason Les for 21.7, Dong Kim for 20.1, and Jimmy Chou for 16.4. These averages do not include any hands that took more than 10 minutes for them to decide; we are interpreting those hands as ones where the human took a break. Libratus thought for 13.0 seconds per hand on average.

  Furthermore, because the humans were allowed to take advantage of timing tells while the AI was not, Libratus had to be designed so that its timing did not depend on its private cards. In contrast, the humans could safely play quickly in obvious situations.

- The humans were allowed to claim that they accidentally clicked the wrong action button on the user interface. In each occurrence, we canceled the hand.

Hardware

Libratus ran on Bridges, the newest and most powerful supercomputer at the Pittsburgh Supercomputing Center. Libratus used between 200 and 600 nodes on Bridges before and during the Brains vs. AI competition and used approximately 12 million core hours in total (some of which was used for exploratory experiments before the production runs). Each node has 128 GB of memory and two Intel Haswell CPUs with 14 cores each. Since Libratus was bottlenecked primarily by memory access, only 14 cores were used on each node rather than the 28 available. Counting the unused 14 cores on each node would double the core hour tallies reported in this section. No GPUs were used.

The blueprint strategy was computed using 196 nodes. We computed multiple blueprint strategies which Libratus would switch between depending on whether Libratus was the first or second mover and on the opening action behavior of the humans. Roughly 3 million core hours were used to compute blueprint strategies.

Nested subgame solving was conducted in real time during the Brains vs AI competition. Each of the four humans had two copies of Libratus available to play against simultaneously, for a total of 8 simultaneous games. Each game used 50 nodes on Bridges, for a total of 400 nodes. Overall, Libratus used roughly 1.5 million core hours for nested subgame solving in the competition.

In the self-improvement component, three bet sizes were solved for in parallel at night and one during the day (because we had fewer nodes available during the day since most of the nodes were used for endgame solving in daytime). Each bet size used 196 nodes for a total of 588 nodes at night. Roughly 1.5 million core hours were used for self improvement.

Notation and background

In a two-player zero-sum imperfect-information extensive-form game there are two players, $\mathcal{P} = \{1, 2\}$. H is the set of all possible nodes in the game tree. $A(h)$ is the actions available in a node and $P(h) \in \mathcal{P} \cup c$ is the player who acts at that node, where c denotes chance. Chance plays actions with fixed probability. The node h' reached after an action is taken in h is a child of h, represented by $h \cdot a = h'$. If a sequence of actions leads from h to h' then we can write $h \prec h'$. $Z \subseteq H$ are terminal nodes from which no actions are available. For each player $i \in \mathcal{P}$, there is a payoff function $u_i : Z \to \Re$ where $u_1 = -u_2$.

Imperfect information is represented by information sets (infosets) for each player $i \in \mathcal{P}$ by a partition $\mathcal{I}_i$ of $h \in H$.[1] For any infoset $I_i \in \mathcal{I}_i$, all nodes $h, h' \in I_i$ are indistinguishable to i, so $A(h) = A(h')$ and $P(h) = P(h')$. $I_i(h)$ is the infoset I_i belonging to i where $h \in I_i$. $A(I_i)$ is the set of actions such that for all $h \in I_i$, $A(I_i) = A(h)$. $P(I_i)$ is the player such that for all $h \in I_i$, $P(I_i) = P(h)$.

A strategy $\sigma(I_i)$ is a probability vector over $A(I_i)$. $\sigma(I_i)$ is only defined if $P(I_i) = i$. The probability of a particular action a is denoted by $\sigma(I_i, a)$. Since all nodes in an infoset belonging to player i are indistinguishable, the strategies in each of them must be identical. A full-game strategy $\sigma_i \in \Sigma_i$ defines a strategy for each infoset where player i acts. A strategy profile σ is a tuple of strategies, one for each player, and σ_{-i} denotes the strategies in σ of all players other than i.

Let $\pi^\sigma(h) = \prod_{h' \cdot a \preceq h} \sigma_{P(h')}(h', a)$ denote the joint probability of reaching h if all players play according to σ. $\pi^\sigma_i(h)$ is the contribution of player i to this probability (that is, the probability of reaching h if all players other than i, and chance, always chose actions leading to h). $\pi^\sigma_{-i}(h)$ is the contribution of all players other than i, and chance. Similarly, $\pi^\sigma(h, h')$ is the

[1] This is a non-standard definition, which facilitates our—arguably more natural—definition of subgame. Traditional definitions have defined $\mathcal{I}_i$ to only include a node $h \in H$ if $P(h) = i$.

probability of reaching h' given that h has been reached, and 0 if $h \not\prec h'$. In a perfect-recall game, a player never forgets information. Thus $\forall h, h' \in I_i$, $\pi_i(h) = \pi_i(h')$. In all subgames in Libratus we used a perfect-recall abstraction. We define $\pi_i(I_i) = \pi_i(h)$ for $h \in I_i$. Moreover, $I'_i \prec I_i$ if for some $h' \in I'_i$ and some $h \in I_i$, $h' \prec h$. Similarly, $I'_i \cdot a \prec I_i$ if $h' \cdot a \prec h$. Finally, $\pi^\sigma(I_i, I'_i)$ is probability of reaching I'_i from I_i according to the strategy σ.

An imperfect-information subgame is a set of nodes $S \subseteq H$ such that for all $h \in S$, if $h \prec h'$, then $h' \in S$, and for all $h \in S$, if $h' \in I_i(h)$ for some player i then $h' \in S$.

A Nash equilibrium (*65*) is a strategy profile σ^* such that $\forall i,\ u_i(\sigma^*_i, \sigma^*_{-i}) = \max_{\sigma'_i \in \Sigma_i} u_i(\sigma'_i, \sigma^*_{-i})$. In two-player zero-sum games, all Nash equilibria give identical expected values for a player. A best response $BR(\sigma_{-i})$ is a strategy for player i such that $u_i(BR(\sigma_{-i}), \sigma_{-i}) = \max_{\sigma'_i \in \Sigma_i} u_i(\sigma'_i, \sigma_{-i})$. The exploitability *exp*(σ_{-i}) of a strategy σ_{-i} is defined as $u_i(BR(\sigma_{-i}), \sigma_{-i}) - u_i(\sigma^*)$, where σ^* is a Nash equilibrium.

The value of a node h is the value player i expects to achieve if all players play according to σ, having already reached h. Formally, $v^\sigma_i(h) = \sum_{z \in Z} \left(\pi^\sigma(h, z) u_i(z)\right)$. The value of an infoset $v^\sigma(I_i)$ is the value player i expects to achieve if all players play according to σ, having already reached infoset I_i. Formally, $v^\sigma(I_i) = \sum_{h \in I_i} \left(\pi^\sigma_{-i}(h) v^\sigma_i(h)\right) / \sum_{h \in I_i} \pi^\sigma_{-i}(h)$ and $v^\sigma(I_i, a) = \sum_{h \in I_i} \left(\pi^\sigma_{-i}(h) v^\sigma_i(h \cdot a)\right) / \sum_{h \in I_i} \pi^\sigma_{-i}(h)$.

A counterfactual best response (*15*) $CBR(\sigma_{-i})$ is similar to a best response, but additionally maximizes value at every infoset. Specifically, a counterfactual best response is a strategy σ_i that is a best response with the additional condition that if $\sigma(I_i, a) > 0$ then $v^\sigma(I_i, a) = \max_{a'} v^\sigma(I_i, a')$.

We further define counterfactual best response value $CBV^{\sigma_{-i}}(I_i)$ as the value player i expects to achieve by playing according to $CBR(\sigma_{-i})$, having already reached infoset I_i. Formally, $CBV^{\sigma_{-i}}(I_i, a) = v^{\langle CBR(\sigma_{-i}), \sigma_{-i} \rangle}(I_i, a)$ and $CBV^{\sigma_{-i}}(I_i) = v^{\langle CBR(\sigma_{-i}), \sigma_{-i} \rangle}(I_i)$. Throughout the paper, $\langle CBR(\sigma_{-i}), \sigma_{-i} \rangle$ denotes the strategy profile where player(s) $-i$ use strategy σ_{-i}

and player i uses a strategy that is a counterfactual best response to σ_{-i}.

Further details of the equilibrium-finding algorithm

As discussed in the body text, Libratus uses a modified form of External-Sampling Monte Carlo Counterfactual Regret Minimization (ES-MCCFR) (*34*) in order to determine the blueprint strategy. In this section we formally describe the algorithm and present pseudocode for its implementation.

ES-MCCFR is an iterative algorithm in which a sampled portion of the game tree is traversed on each iteration. We maintain two values for each action a in each infoset I_i where $P(I_i) = i$: a regret value $R(I_i, a)$ which roughly corresponds to how much we "regret" having not taken this action in past traversals (which we formally define later), and a count $\phi(I_i, a)$ of how many times the action is sampled during an opponent traversal.

Let $R_+(I_i, a) = \max\{0, R(I_i, a)\}$. Whenever a non-chance and non-terminal node h is encountered during a traversal on iteration t, ES-MCCFR sets the strategy at the node's infoset as $\sigma^t(I, a) = \frac{R_+^{t-1}(I,a)}{\sum_{a' \in A(I)} R_+^{t-1}(I,a')}$ where $I = I_{P(h)}(h)$. In the case where $\sum_{a' \in A(I)} R_+^{t-1}(I, a') = 0$, ES-MCCFR sets $\sigma^t(I, a) = \frac{1}{|A(I)|}$.

We assume the algorithm runs for T iterations. On each iteration t we traverse the game tree once for each player. For simplicity of explanation, we refer to the player traversing the game tree as P_i and refer to all other players (or the single other player in the case of HUNL), as P_{-i}.

The game tree is traversed in a depth-first manner. When a chance node is encountered, we sample a single action according to the fixed probability distribution for that node and explore only that action. When a P_{-i} node is encountered, we also sample and explore a single action, where the probability that action a is selected is $\sigma^t(I_{-i}, a)$. Additionally, the value $\phi(I_{-i}, a)$ of the sampled action is incremented by 1. When a P_i node is encountered, every action is explored. Eventually, a terminal node is reached and its value is passed up to the preceding

node. Chance and P_{-i} nodes pass this exact value farther up the game tree. For P_i nodes, after every action a has returned a value $v^t(I_i, a)$, the weighted average value is calculated as $v^t(I_i) = \sum_{a \in A(I_i)}(v^t(I_i, a) \cdot \sigma^t(I_i, a))$. The regret for each action is updated to be $R^t(I_i, a) = R^{t-1}(I_i, a) + v^t(I_i, a) - v^t(I_i)$. The weighted average $v^t(I_i)$ is then passed up to the preceding node.

After running ES-MCCFR for T iterations, we calculate the strategy $\bar{\sigma}_i^T$ for each player by simply normalizing $\phi(I_i, a)$ at every infoset where $P(I_i) = i$ so that $\sum_{a \in A(I_i)} \phi(I_i, a) = 1$. ES-MCCFR guarantees that as $T \to \infty$, with high probability $\bar{\sigma}^T$ will approach a Nash equilibrium in two-player zero-sum games without abstraction. In cases where imperfect-recall forms of abstraction prevent convergence to a Nash equilibrium (as is possibly the case with the blueprint strategy of Libratus), ES-MCCFR still approaches a good strategy in practice.

Libratus improves upon ES-MCCFR by sometimes pruning an action from the tree traversal if it has very negative regret. This pruning only occurs after some number of iterations have passed; prior to that point, traditional ES-MCCFR is used. Formally, on iterations $t > T_0$, in an infoset I_i where $P(I_i) = i$, an action a with regret $R(I_i, a)$ that is below a threshold C (where C is negative) is sampled with probability $K/\big(K + C - R(a)\big)$, where K is a positive constant. This sampling applies to each action independently—multiple actions may be sampled at the same decision point. We also apply a floor of 2% on the probability of an action being sampled. This floor is applied to the chained sampling probability: if the action leading to the current node had only a 10% chance of being sampled, then the floor on each action at the current node being sampled is 20%. This ensures that as the opponent's strategy changes, we will detect if a seemingly bad action starts to lead to higher payoffs. If an action is sampled, the algorithm proceeds identically to ES-MCCFR except in any descendant opponent infoset I_{-i}, $\phi(I_{-i}, a)$ is updated by $1/p$ rather than by 1, where p is the chained sampling probability so far. If an action is not sampled, then the game tree below it is not traversed on that iteration and the action's

regret is not updated on that iteration. Since an action will only be pruned if it has negative regret and therefore $\sigma^t(I_i, a) = 0$, pruning an action does not affect the calculation of $v(I_i)$.

This form of pruning empirically leads to better performance by allowing more iterations to be conducted in the same amount of time by spending less time on situations that are unlikely to be relevant to the final strategy. If an action has extremely negative regret, then the action has performed very poorly in past iterations and is unlikely to be part of the final strategy. We do not prune an action completely because it is possible that an action that has performed poorly in the past may improve as both players' strategies adjust. As discussed in the main body of the paper, this form of pruning also mitigates the problems of using an imperfect-recall abstraction.

The form of sampled pruning we use does not have theoretical guarantees of convergence to a Nash equilibrium. It was inspired by similar non-sampled algorithms that we developed, which do have theoretical guarantees, called regret-based pruning (*66*) and best-response pruning (*67*).

Further details of the nested subgame-solving algorithm

The purpose of subgame solving is to calculate a better strategy for a specific part of the game, while fitting that strategy within the overarching blueprint that has already been calculated. In this section we provide more detail about subgame solving.

Subgame solving is accomplished by solving an augmented subgame, which contains the subgame as well as additional structure. This additional structure depends on the form of subgame solving used. We define S_{top} as the set of earliest-reachable nodes in S. More formally, $h \in S_{top}$ if for $h' \prec h$, $h' \notin S$.

In unsafe subgame solving (which lacks theoretical guarantees but performs well empirically in some cases) the augmented subgame starts with a single chance node which connects to a node $h \in S_{top}$ with probability $\frac{\pi^\sigma(h)}{\sum_{h' \in S_{top}} \pi^\sigma(h')}$. The rest of the augmented subgame is iden-

Algorithm 1 ESMCCFR with Negative-Regret Pruning

During actual play, when in infoset I_i, sample action a in proportion to $\phi(I_i, a)$.

function ESMCCFR-P(T) ▷ Conduct External-Sampling Monte Carlo CFR with Pruning
  for $P_i \in \mathcal{P}$ **do**
    for $I_i \in \mathcal{I}_i$ where $P(I_i) = i$ **do**
      for $a \in A(I_i)$ **do**
        $R(I_i, a) \leftarrow 0$, $\phi(I_i, a) \leftarrow 0$
  for $t = 1$ to T **do**
    for $P_i \in \mathcal{P}$ **do**
      if $t > \frac{T}{2}$ **then**
        TRAVERSE-ESMCCFR-P($\emptyset, P_i, 1$)
      else
        TRAVERSE-ESMCCFR($\emptyset, P_i$)

function TRAVERSE-ESMCCFR(h, P_i) ▷ Traverses the game tree once for P_i
  if h is terminal **then**
    return $u_i(h)$
  else if $P(h) = P_i$ **then**
    $I_i \leftarrow I_i(h)$ ▷ The P_i infoset of this node
    $\sigma(I_i) \leftarrow$ CALCULATE-STRATEGY($R(I_i), I_i$) ▷ Determine the strategy at this infoset
    $v(h) \leftarrow 0$ ▷ Initialize expected value at zero
    for $a \in A(h)$ **do**
      $v(h, a) \leftarrow$ TRAVERSE-ESMCCFR($h \rightarrow a, P_i$) ▷ Traverse each action
      $v(h) \leftarrow v(h) + \sigma(I_i, a) \cdot v(h, a)$ ▷ Update the expected value
    for $a \in A(h)$ **do**
      $R(I_i, a) \leftarrow R(I_i, a) + v(h, a) - v(h)$ ▷ Update the regret of each action
    return $v(h)$ ▷ Return the expected value
  else if $P(h) = P_{-i}$ **then**
    $I_{-i} \leftarrow I_{-i}(h)$ ▷ The P_{-i} infoset of this node
    $\sigma(I_{-i}) \leftarrow$ CALCULATE-STRATEGY($R(I_{-i}), I_{-i}$) ▷ Determine the strategy at this infoset
    $a \sim \sigma(I_{-i})$ ▷ Sample an action from the probability distribution
    $\phi(I_{-i}, a) \leftarrow \phi(I_{-i}, a) + 1$ ▷ Increment the action counter
    return TRAVERSE-ESMCCFR($h \rightarrow a, P_i$)
  else ▷ h is a chance node
    $a \sim \sigma(h)$ ▷ Sample an action from the chance probabilities
    return TRAVERSE-ESMCCFR($h \rightarrow a, P_i$)

function TRAVERSE-ESMCCFR-P(h, P_i, p) ▷ ESMCCFR with sampled pruning
  if h is terminal **then**
    return $u_i(h)$
  else if $P(h) = P_i$ **then**
    $I_i \leftarrow I_i(h)$ ▷ The P_i infoset of this node
    $\sigma(I_i) \leftarrow$ CALCULATE-STRATEGY($R(I_i), I_i$) ▷ Determine the strategy at this infoset
    $v(h) \leftarrow 0$ ▷ Initialize expected value at zero
    for $a \in A(h)$ **do**
      if $R(I_i, a) < C$ **then**
        thresh $\leftarrow \max\{\frac{0.02}{p}, \frac{K}{K+C-R(I_i,a)}\}$
      else
        thresh $\leftarrow 1$
      $q \sim [0, 1)$
      if $q <$ *thresh* **then** ▷ $C < 0 < K$
        $v(h, a) \leftarrow$ TRAVERSE-ESMCCFR-P($h \rightarrow a, P_i, p \cdot \min\{$*thresh*$, 1\}$)
        explored$(a) \leftarrow$ *True*
        $v(h) \leftarrow v(h) + \sigma(I_i, a) \cdot v(h, a)$ ▷ Update the expected value
      else
        explored$(a) \leftarrow$ *False*
    for $a \in A(h)$ **do**
      if *explored*$(a) =$ *True* **then**
        $R(I_i, a) \leftarrow R(I_i, a) + v(h, a) - v(h)$ ▷ Update the regret for this action
    return $v(h)$ ▷ Return the expected value
  else if $P(h) = P_{-i}$ **then**
    $I_{-i} \leftarrow I_{-i}(h)$ ▷ The P_{-i} infoset of this node
    $\sigma(I_{-i}) \leftarrow$ CALCULATE-STRATEGY($R(I_{-i}), I_{-i}$) ▷ Determine the strategy at this infoset
    $a \sim \sigma(I_{-i})$ ▷ Sample an action from the probability distribution
    $\phi(I_{-i}, a) \leftarrow \phi(I_{-i}, a) + 1/p$ ▷ Increase the action counter
    return TRAVERSE-ESMCCFR-P($h \rightarrow a, P_i, p$)
  else ▷ h is a chance node
    $a \sim \sigma(h)$ ▷ Sample an action from the chance probabilities
    return TRAVERSE-ESMCCFR-P($h \rightarrow a, P_i, p$)

```
function CALCULATE-STRATEGY(R(I_i), I_i)          ▷ Calculates the strategy based on regrets
    sum ← 0
    for a ∈ A(I_i) do
        sum ← sum + R_+(I_i, a)
    for a ∈ A(I_i) do
        if sum > 0 then
            σ(I_i, a) ← R_+(I_i,a) / sum
        else
            σ(I_i, a) ← 1 / |A(I_i)|
    return σ(I_i)
```

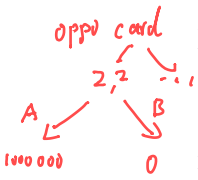
tical to S. Unsafe subgame solving assumes the opponent is playing according to the blueprint strategy, but that might not be the case and therefore the opponent may be able to exploit us by changing her strategy. This motivates safe subgame solving, discussed below.

Libratus uses unsafe subgame solving upon first reaching the third betting round, and uses safe subgame solving in response to every subsequent opponent bet or raise. Unsafe subgame solving was used because it only requires storing the strategy for the first two betting rounds. In contrast, safe subgame solving would have required storing infoset values for the third betting round (which would have increased the space used by the blueprint strategy by about a factor of 50). In medium-scale experiments, we found that unsafe subgame solving exhibited competitive head-to-head performance and typically low exploitability.

In safe subgame solving, without loss of generality, say we wish to find a strategy for player P_2 in the subgame. The opponent is P_1. We modify the augmented subgame used in unsafe subgame solving so that for every node $h \in S_{top}$, there is an additional node h_r belonging to P_1 such that the initial chance node instead leads to h_r and an action a_S leads from h_r to h. In other words, $h_r \cdot a_S = h$. The set of all such h_r nodes is represented by the set S_r. The initial chance node connects to $h_r \in S_r$ with probability $\frac{\pi^{\sigma}_{-2}(h)}{\sum_{h'_r \in S_r} \pi^{\sigma}_{-2}(h')}$. In other words, a root node h_r's probability is proportional to the probability P_1 would reach that node if P_1 always took actions that attempted to reach that node with probability 1, rather than play according to the

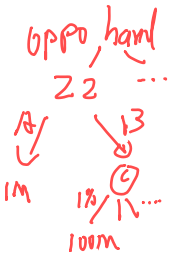
blueprint.

At each node $h_r \in S_r$ there are two actions which P_1 can choose between. The first action is a_S, which leads to $h \in S_{top}$, after which the rest of the augmented subgame is identical to S. The second action is a_{alt} which leads to a terminal reward of $v_{alt}(h) = v_{bp}(I_1(h)) + g[c]$, where $v_{bp}(I_1(h)) = \frac{\sum_{t=1}^{T} v^t(I_1)}{T}$ after playing T iterations of a CFR variant in the blueprint and $g[c]$ is a "gifts" modifier discussed in the next paragraph. In the case of nested subgame solving, in which we conduct subgame solving repeatedly as we descend the game tree, $v_{bp}(I_1(h))$ comes from the previous augmented subgame's solution; that strategy acts as the blueprint until subgame solving occurs again.



When an opponent takes some action, we may be able to gain information about what her private information is by examining the actions the opponent did not take. For example, suppose the opponent is faced with a choice between action A and action B. Action A leads to a terminal reward of \$1,000,000 only if the opponent's private cards are 2♠2♣, while Action B leads to an expected value of \$0 if her private cards are 2♠2♣ and both players play a Nash equilibrium strategy beyond that point. If the opponent chooses action B, we can be fairly confident in our future decisions that she does not hold 2♠2♣ because the only way she would have that hand is if she had made a mistake earlier. Thus, when conducting safe subgame solving following action B, we can afford to let the opponent's value for holding 2♠2♣ increase, and instead focus on decreasing the opponent's values for other hands. However, in order to ensure the opponent cannot exploit us for making this assumption, we must ensure when conducting subgame solving that we do not let the opponent's value for taking action B with 2♠2♣ increase to more than \$1,000,000. This must be true even when conducting subgame solving independently in every subgame following action B, because applying subgame solving to any subgame reached during play has the same exploitability as conducting subgame solving at every subgame prior to play beginning.

We capture this idea by maintaining a gifts vector $\vec{g}$ over possible opponent (P_1) poker hands. It shows for each pair of private cards how much of a mistake the opponent would have made to get to the current infoset with that pair. This vector is initialized to zeros either at the beginning of the game or, in the case of Libratus, immediately after the unsafe subgame solving. The vector is updated after each opponent action. When we (P_2) are in infoset I_2 and observe P_1 take some action a, we consider every node $h \in I_2$ that we might have been in. Each node $h \in I_2$ corresponds to a poker hand the opponent might be holding. For each $h \in I_2$, we compare the value $v_{bp}(I_1(h))$ that the opponent would have expected to receive with that hand to the value $v_{bp}(I_1(h), a)$ (or the equivalent augmented subgame value if we conducted subgame solving in response to the action) that the opponent would have expected to receive with that hand for the action she ultimately chose. We add this difference to $\vec{g}$, so $g_{new}[c] = g_{old}[c] + v_{bp}(I_1(h)) - v_{bp}(I_1(h), a)$, where c is the opponent poker hand that corresponds to $h \in I_2$. In future subgames, $g[c]$ is added to $v_{alt}(h)$ (where h corresponds to the node where the opponent holds poker hand c). Thus if the opponent would only have a particular hand if she previously made a "mistake" by giving up a larger expected reward and instead choosing a (that is, $v_{bp}(I_1(h)) > v_{bp}(I_1(h), a)$), then we can afford to be less concerned about that hand. This is captured by increasing v_{alt} for that hand.



If one were to solve only a single subgame before play begins, then it would be possible to scale up the gifts dramatically. For example in the 2♠2♣ example, suppose there are 100 non-overlapping subgames following the B action and for each one the opponent could only reach it with 1% probability if she tried to do so (e.g., there is a chance node immediately after the B action which leads to each subgame with 1% probability). If we were to solve just one of those subgames, then the gift for 2♠2♣ could be $100,000,000 in that subgame, because this would still only increase the opponent's value for the B action to $1,000,000. However, if we were to solve all 100 subgames independently and apply this same reasoning to each subgame,

then the opponent's value for the B action could increase to $100,000,000. Since applying subgame solving in real time to any subgame we happen to encounter is, from an exploitability standpoint, equivalent to applying subgame solving to every subgame independently, we cannot scale up gifts and still maintain safety guarantees.[2]

The augmented subgame can be solved with any equilibrium-finding algorithm. In Libratus, we used CFR+. The number of iterations of CFR+ we used varied depending on the size of the pot. If at most $1,600 were committed to the pot by both players combined, then we used 1,000 iterations of CFR+. Otherwise, we used 1,900 iterations of CFR+. This was motivated by the observation that subgames with larger pots were more important, due to the larger amounts of money at stake, and that they were also faster to iterate over, due to fewer remaining actions in the game tree.

In order to more quickly evaluate the performance of subgame solving in Libratus against Baby Tartanian8, we employed a variance reduction technique. We first measured the performance of the Libratus blueprint strategy (without subgame solving) against Baby Tartanian8, which can be determined quickly due to the lack of real-time computation. Next, we randomly chose a set of subgames starting on the third betting round from the hands played between the two AIs and measured how much nested subgame solving would improve performance, in expectation, in each of those subgames for each set of private cards the players may be holding. We then calculated the weighted average of those values, weighted by the probability of the players holding each set of private cards based on the sequence of actions leading to the subgame. This allowed us to evaluate the improvement of nested subgame solving across a large number of hands by solving a subgame only once. The results are in Table S1.

[2] In Libratus and the pseudocode in this supplementary material, the different subgames are, in effect, weighted according to the probability that the opponent could reach those subgames, that is, the product of chance's and our action probabilities on the path but not the opponent's action probabilities. In other words, each subgame gets to take advantage of 100% of the gift, and no more or less. Any other way of splitting the gift among subgames could also be employed as long as the reach-weighted sum of the gift pieces allocated to the subgames does not sum up

Algorithm 2 Safe Subgame Solving

Assume the opponent is P_1 and we are P_2

function OPPONENTACTION($S, a, \vec{g}$) ▷ Opponent chose action a. Construct and solve a subgame.
  for each node $h \in S_{top}$ **do**
    $\pi_{-i}[h] \leftarrow \pi_{-i}(h)$ ▷ The probability P_1 would reach this node, if she tried to
  for each P_1 infoset $I_1 \in S_{top}$ **do**
    $c \leftarrow$ the private cards P_1 would hold in I_1
    $v_{bp}[I_1] \leftarrow v_{bp}(I_1)$ ▷ Estimated optimal value of P_1 infoset based on last-computed strategy
    $v_{alt}[I_1] \leftarrow g[c] + v_{bp}[I_1]$ ▷ Add gifts from previous potential P_1 mistakes
  $S' \leftarrow$ CONSTRUCTSUBGAME(S, a) ▷ Construct subgame following action a
  SOLVEAUGMENTED($S', \vec{v}_{alt}, \vec{\pi}_{-i}$) ▷ Conduct safe subgame solving
  for each P_1 infoset $I_1 \in S_{top}$ **do**
    $c \leftarrow$ the private cards the opponent would hold in I_1
    $g_{new}[c] \leftarrow g[c] + v_{bp}[I_1] - v_{bp}(I_1, a)$ ▷ Update gifts based on value of newly-solved subgame
  return $\vec{g}_{new}$ ▷ Return updated gifts vector

Further details of the self-improvement algorithm

As described in the body of this paper, Libratus's third module is a self-improvement algorithm. Libratus used two versions of it, which we will call Type 1 and Type 2. They are illustrated in Figure S2. Both versions were only applied to actions in the first betting round. We now describe both techniques in detail. In all cases, bet sizes are measured as fractions of the size of the pot.

Each evening after the day's games were over, Libratus determined one opponent bet size to solve with a Type 1 self-improver and two opponent bet sizes to solve with Type 2 self-improvers.

For the Type 2 self-improvers, the two bet sizes to add were determined by scoring each "gap" between existing bet sizes in the abstraction. A gap is defined by two neighboring bet sizes A and B already in the abstraction. If, during the day, an opponent chose a bet size x such that $A < x < B$, then the gap's score would increase by the distance of x from A or

to more than the entire gift.

B. Formally, the score would increase by $\min\{x - A, B - x\}$. The two bet sizes to add (one per Type 2 self-improver) were the pseudo-harmonic midpoints (*27*) of the two highest-scoring gaps. Specifically, if a gap between bet sizes A and B was selected, the bet size to add was $(A + B + 2AB)/(A + B + 2)$.

For the Type 1 self-improver, the bet size to add was determined by simply choosing the most common opponent bet size (aggregated across the four opponents) from that day.

In the Type 1 self-improver, a subgame was solved using unsafe subgame solving, which means we assume both players play according to the blueprint strategy for all moves preceding the subgame. At the beginning of the subgame the opponent was given the choice between folding, checking, or calling (except in the first action of the game, where calling was not provided as a valid option), or betting the self-improvement bet size. After the subgame was solved, the strategy in the subgame of the response to the self-improvement bet size was used if the opponent consistently bet that particular size in the future. Specifically, if an opponent bet that particular size for each of the last eight times he bet in that particular situation, then Libratus would use the Type 1-created strategy to respond to the bet size. One can view the Type 1 self-improver as enabling Libratus to somewhat exploit an opponent in a fairly safe way if he were not playing a balanced strategy. However, during the Brains vs. AI competition the human opponents changed the bet sizes they used almost every day in order to prevent Libratus from calculating an effective response to their strategy. As a result, the Type 1 self-improver played little role in the competition.

In the Type 2 self-improver, a subgame was solved in a manner similar to the Type 1 self-improver, but with the addition of at least one default bet size that was commonly played in the blueprint strategy (which we had computed in advance of the competition using the first module of Libratus as described in the body of this paper). The sole role of including the default action was to determine a balanced strategy in the game tree following the self-improvement bet size:

we do not want to assume that the opponent uses the selected new bet size for all private cards. For computational speed, our algorithm used a significantly coarser abstraction following the default action, and its strategy was discarded after subgame solving finished, because that bet size was already in the blueprint. Using unsafe subgame solving with default actions still has theoretical bounds on exploitability when applied to the first action of the game—because there is no assumption being made about the opponent's prior play. For situations other than the first action of the game, unsafe subgame solving lacks theoretical guarantees, but empirically we found it to produce competitive strategies with generally low exploitability—as shown, for example, in the experiments discussed in the body of this paper. The strategy in the subgame of the response to the self-improvement bet size (but not the default bet size) was added to the overall blueprint strategy of Libratus. If an opponent chose an action that was close to the self-improvement bet size, then Libratus would use the self-improvement strategy as a response. This is in contrast to the Type 1 self-improvements, which were used only if the opponent played that bet size fairly consistently in that point of the game, as described above. The Type 2 self-improver played a more significant role in the competition. By the end of the competition, roughly half of all the hands played by Libratus were played using a strategy determined by it.

Post-processing

Post-processing the action probabilities before acting is a common and beneficial technique used in the AI community working on imperfect-information games. The most basic post-processing technique sets low-probability actions' probabilities to zero and renormalizes the rest of the probabilities so they sum to one. This is motivated in two ways. First, iterative algorithms like CFR always have—after any finite number of iterations—some positive probability on all actions, even ones that are not part of an equilibrium, and rounding the small probabilities to zero facilitates getting to an exact equilibrium. Second, a solution to an abstraction is not

necessarily a solution to the full unabstracted game, and empirical results suggests that high-probability actions in abstractions are more likely to do well when played in the full game (*68*).

However, setting low-probability actions to zero also potentially increases the exploitability of an AI because its strategy may no longer be balanced and may be more predictable. We actually have observed this in our 2015 AI Claudico. For example, limping (not betting in the first action in the game and instead simply calling) is an action that most HUNL AIs use, but with a small probability among all hands. Suppose an AI limps with 6% probability with bad hands and 4% probability with good hands. When faced with a limp, it would be difficult to determine whether the AI has a good hand or a bad hand. However, applying a threshold at 5% such that any action with less than 5% probability is set to zero probability would result in the AI only limping with bad hands, which would be extremely exploitable!

Libratus uses a new form of post-processing we refer to as range thresholding that mitigates the exploitability typically caused by post-processing. Rather than set any probability below some threshold to zero, range thresholding only reduces an action to zero probability if every hand has a probability below that threshold. In the limping example, using range thresholding with a threshold of 5% would be no different than not applying thresholding at all, because at least one hand limps with probability above 5%. Alternatively, if the threshold were set to 7%, then the AI would not limp in any situation. Libratus used a range threshold of 4% on the first two betting rounds during the 2017 Brains vs. AI competition.

Proof of Theorem 1

Without loss of generality, we assume that it is player P_2 who conducts subgame solving. We define a node h in a subgame S as earliest-reachable if there does not exist a node $h' \in S$ such that $h' \prec h$. For each earliest-reachable node $h \in S$, let h_r be its parent and a_S be the action leading to h such that $h_r \cdot a_S = h$. We require h_r to be a P_1 node; if it is not, then we can simply

insert a P_1 node with only a single action between h_r and h. Let S_r be the set of all h_r for S.

Applying subgame solving to subgames as they are reached during play is equivalent to applying subgame solving to every subgame before play begins, so we can phrase what follows in the context of all subgames being solved before play begins. Let σ'_2 be the P_2 strategy produced after subgame solving is applied to every subgame. We show inductively that for any P_1 infoset $I_1 \notin \mathcal{S}$ where it is P_1's turn to move (i.e., $P(I_1) = P_1$), the counterfactual best response values for P_1 satisfy

$$CBV^{\sigma'_2}(I_1) \leq CBV^{\sigma^*_2}(I_1) + 2\Delta \tag{S1}$$

Define *Succ*(I_1, a) as the set of infosets belonging to P_1 that follow action a in I_1 and where it is P_1's turn and where P_1 has not had a turn since a, as well as terminal nodes follow action a in I_1 without P_1 getting a turn. Formally, a terminal node $z \in Z$ is in *Succ*(I_1, a) if there exists a history $h \in I_1$ such that $h \cdot a \preceq z$ and there does not exist a history h' such that $P(h') = P_1$ and $h \cdot a \preceq h' \prec z$. Additionally, an infoset I'_1 belonging to P_1 is in *Succ*(I_1, a) if $P(I'_1) = P_1$ and $I_1 \cdot a \preceq I'_1$ and there does not exist an earlier infoset I''_1 belonging to P_1 such that $P(I''_1) = P_1$ and $I' \cdot a \preceq I''_1 \prec I'_1$. Define *Succ*$(I_1)$ as $\cup_{a \in A(I_1)}$*Succ*(I_1, a). Similarly, we define *Succ*(h, a) as the set of histories belonging to $P(h)$ and terminals that follow action a and where $P(h)$ has not had a turn since a. Formally, $h' \in$ *Succ*(h, a) if either $P(h') = P(h)$ or $P(h') \in Z$ and $h \cdot a \preceq h'$ and there does not exist a history h'' such that $P(h'') = P(h)$ and $h \cdot a \preceq h'' \prec h'$.

Now we define a level L for each P_1 infoset where it is P_1's turn and the infoset is not in the set of subgames $\mathcal{S}$.

- For immediate parents of subgames we define the level to be zero: for all $I_1 \in S_r$ for any subgame $S \in \mathcal{S}$, $L(I_1) = 0$.
- For infoset that are not ancestors of subgames, we define the level to be zero: $L(I_1) = 0$ for any infoset I_1 that is not an ancestor of a subgame in $\mathcal{S}$.

- For all other infosets, the level is one greater than the greatest level of its successors: $L(I_1) = \ell + 1$ where $\ell = \max_{I_1' \in Succ(I_1)} L(I_1')$ where $L(z) = 0$ for terminal nodes z.

Base case of induction

First consider infosets $I_1 \in S_r$ for some subgame $S \in \mathcal{S}$. We define $M^{\sigma_2'}(I_1) = v^{\sigma}(I_1, a_S) - CBV^{\sigma_2'}(I_1, a_S)$. Consider a subgame $S \in \mathcal{S}$. Estimated-Maxmargin subgame solving arrives at a strategy σ_2' such that $\min_{I_1 \in S_r} M^{\sigma_2'}(I_1)$ is maximized. By the assumption in the theorem statement, $|v^{\sigma}(I_1, a_S) - CBV^{\sigma_2^*}(I_1, a_S)| \leq \Delta$ for all $I_1 \in S_r$. Thus, σ_2^* satisfies $\min_{I_1 \in S_r} M^{\sigma_2^*}(I_1) \geq -\Delta$ and therefore $\min_{I_1 \in S_r} M^{\sigma_2'}(I_1) \geq -\Delta$, because Estimated-Maxmargin subgame solving could, at least, arrive at $\sigma_2' = \sigma_2^*$. From the definition of $M^{\sigma_2'}(I_1)$, this implies that for all $I_1 \in S_r$, $CBV^{\sigma_2'}(I_1, a_S) \leq v^{\sigma}(I_1, a_S) + \Delta$. Since by assumption $v^{\sigma}(I_1, a_S) \leq CBV^{\sigma_2^*}(I_1, a_S) + \Delta$, this gives us $CBV^{\sigma_2'}(I_1, a_S) \leq CBV^{\sigma_2^*}(I_1, a_S) + 2\Delta$.

Now consider infosets I_1 that are not ancestors of any subgame in $\mathcal{S}$. By definition, for all h such that $h \succeq I_1$ or $I_1 \succeq h$, and $P(h) = P_2$, $\sigma_2^*(I_2(h)) = \sigma_2(I_2(h)) = \sigma_2'(I_2(h))$. Therefore, $CBV^{\sigma_2'}(I_1) = CBV^{\sigma_2^*}(I_1)$.

So, we have shown that (S1) holds for any I_1 such that $L(I_1) = 0$.

Inductive step

Now assume that (S1) holds for any P_1 infoset I_1 where $P(I_1) = P_1$ and $I_1 \notin \mathcal{S}$ and $L(I_1) \leq \ell$. Consider an I_1 such that $P(I_1) = P_1$ and $I_1 \notin \mathcal{S}$ and $L(I_1) = \ell + 1$.

From the definition of $CBV^{\sigma_2'}(I_1, a)$, we have that for any action $a \in A(I_1)$,

$$CBV^{\sigma_2'}(I_1, a) = \Big(\sum_{h \in I_1} \Big(\big(\pi_{-1}^{\sigma_2'}(h)\big) \big(v^{\langle CBR(\sigma_2'), \sigma_2' \rangle}(h \cdot a)\big) \Big) \Big) / \sum_{h \in I_1} \pi_{-1}^{\sigma_2'}(h) \tag{S2}$$

Since for any $h \in I_1$ there is no P_1 action between a and reaching any $h' \in Succ(h, a)$, so

$\pi_1^{\sigma_2'}(h \cdot a, h') = 1$. Thus,

$$\begin{aligned} CBV^{\sigma_2'}(I_1, a) &= \Big(\sum_{h \in I_1} \Big(\pi_{-1}^{\sigma_2'}(h) \sum_{h' \in Succ(h,a)} \pi_{-1}^{\sigma_2'}(h, h') \big(v^{\langle CBR(\sigma_2'), \sigma_2' \rangle}(h') \big) \Big) \Big) / \sum_{h \in I_1} \pi_{-1}^{\sigma_2'}(h) \\ &= \Big(\sum_{h \in I_1} \sum_{h' \in Succ(h,a)} \Big(\big(\pi_{-1}^{\sigma_2'}(h') \big) v^{\langle CBR(\sigma_2'), \sigma_2' \rangle}(h') \Big) \Big) / \sum_{h \in I_1} \pi_{-1}^{\sigma_2'}(h) \end{aligned}$$

Since the game has perfect recall, $\sum_{h \in I_1} \sum_{h' \in Succ(h,a)} f(h') = \sum_{I_1' \in Succ(I_1,a)} \sum_{h' \in I_1'} f(h')$ for any function f. Thus,

$$CBV^{\sigma_2'}(I_1, a) = \Big(\sum_{I_1' \in Succ(I_1,a)} \sum_{h' \in I_1'} \Big(\big(\pi_{-1}^{\sigma_2'}(h') \big) \big(v^{\langle CBR(\sigma_2'), \sigma_2' \rangle}(h') \big) \Big) \Big) / \sum_{h \in I_1} \pi_{-1}^{\sigma_2'}(h) \tag{S3}$$

From the definition of $CBV^{\sigma_2'}(I_1')$ we get

$$CBV^{\sigma_2'}(I_1, a) = \Big(\sum_{I_1' \in Succ(I_1,a)} \big(CBV^{\sigma_2'}(I_1') \sum_{h' \in I_1'} \pi_{-1}^{\sigma_2'}(h') \big) \Big) / \sum_{h \in I_1} \pi_{-1}^{\sigma_2'}(h) \tag{S4}$$

Since (S1) holds for all $I_1' \in Succ(I_1, a)$,

$$CBV^{\sigma_2'}(I_1, a) \leq \Big(\sum_{I_1' \in Succ(I_1,a)} \big((CBV^{\sigma_2^*}(I_1') + 2\Delta) \sum_{h' \in I_1'} \pi_{-1}^{\sigma_2'}(h') \big) \Big) / \sum_{h \in I_1} \pi_{-1}^{\sigma_2'}(h) \tag{S5}$$

Since P_2's strategy is fixed according to σ_2 outside of $\mathcal{S}$, we have that for all $I_1 \notin \mathcal{S}$, $\pi_{-1}^{\sigma_2'}(I_1) = \pi_{-1}^{\sigma_2}(I_1) = \pi_{-1}^{\sigma_2^*}(I_1)$. Therefore,

$$CBV^{\sigma_2'}(I_1, a) \leq \Big(\sum_{I_1' \in Succ(I_1,a)} \big((CBV^{\sigma_2^*}(I_1') + 2\Delta) \sum_{h' \in I_1'} \pi_{-1}^{\sigma_2^*}(h') \big) \Big) / \sum_{h \in I_1} \pi_{-1}^{\sigma_2^*}(h) \tag{S6}$$

Separating out the two addends and applying equation (S4) for $CBV^{\sigma_2^*}(I_1, a)$ we get

$$CBV^{\sigma_2'}(I_1, a) \leq CBV^{\sigma_2^*}(I_1, a) + 2\Delta \Big(\Big(\sum_{I_1' \in Succ(I_1,a)} \sum_{h' \in I_1'} \pi_{-1}^{\sigma_2^*}(h') \Big) / \sum_{h \in I_1} \pi_{-1}^{\sigma_2^*}(h) \Big) \tag{S7}$$

Since $\big(\sum_{I_1' \in Succ(I_1,a)} \sum_{h' \in I_1'} \pi_{-1}^{\sigma_2^*}(h') \big) = \sum_{h \in I_1} \pi_{-1}^{\sigma_2^*}(h)$ we arrive at

$$CBV^{\sigma_2'}(I_1, a) \leq CBV^{\sigma_2^*}(I_1, a) + 2\Delta \tag{S8}$$

Thus, (S1) holds for I_1 as well. So, the inductive step is satisfied. Extending (S1) to the root of the game, we get $exp(\sigma_2') \leq exp(\sigma_2^*) + 2\Delta$. □

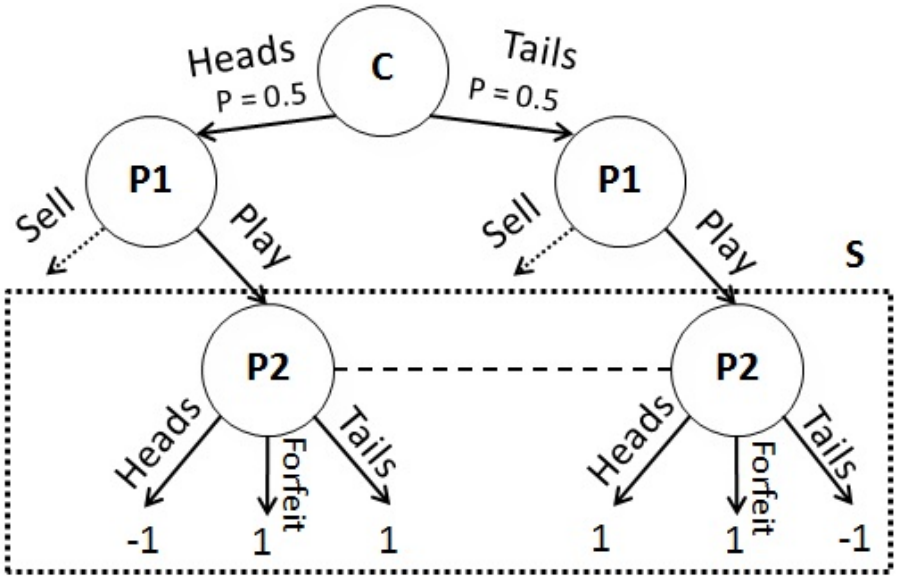


Figure S1: **The example game of Coin Toss.** "C" represents a chance node. S is a Player 2 (P_2) subgame. The dotted line between the P_2 nodes means that P_2 cannot distinguish them.

Board	Pot Size	No subgame solving	On-tree subgame solving	Libratus subgame solving
TsQhTcJh	520	-75.23	-38.55	-41.41
KsKh4dTs	1092	75.21	159.78	163.91
7s4d9d9h	520	8.28	59.01	59.06
8s9sKhTd	728	95.45	99.30	117.76
5hTd5cKh	780	-247.02	-208.57	-211.50
6s9h3cAd	520	-126.85	-92.15	-59.19
As4d2cJc	520	-151.94	-86.30	-70.30
4hQd5c3s	2256	1227.03	1038.71	1007.39
QsTd8c6d	520	-13.91	28.36	38.52
5s4d9dAc	520	-150.97	-101.26	-81.42
9sAs5c2s	2600	-389.93	-349.89	-306.15
KhQcAc2s	1352	336.16	502.44	516.27
Kh7d7cQh	8580	2338.06	2651.35	2825.91
4s6sQsTd	780	-165.98	-119.19	-116.21
KsKhQd9d	520	-133.13	-120.79	-108.70
2s5dJdQs	520	-89.03	2.03	17.39
6s6hTdTc	780	155.89	177.40	193.27
Qh3d9cKd	520	-49.39	45.82	57.01
7s5dAc8s	780	-403.14	-275.20	-274.15
3s7hAdJs	520	10.33	60.96	70.35
8sAhJdKc	780	94.97	132.76	135.92
7h7d8d3s	520	9.08	72.90	76.82
7hQd4cQc	1352	1303.02	1240.78	1247.54
9s3cAc3d	520	-217.77	-132.11	-125.55
7h7d7c8s	1092	-568.40	-571.03	-590.47
Js9d2c8d	1300	552.52	646.54	681.49
Qs7h7d2s	1092	241.72	282.08	279.79
6dJcQc2h	1352	201.72	276.59	271.48
QsKhTcAc	1352	-280.37	-167.85	-165.82
3hKdJcTh	520	-131.42	-91.74	-86.58
8hJd2cKc	520	-113.94	-34.56	-18.38
2sJs3h9c	520	-174.23	-93.54	-75.81
8h4d2c6h	520	169.69	216.60	224.91
Qh5c9cAd	1352	-813.72	-633.45	-639.38
8s5h6c2d	520	155.94	179.78	188.28
4sAsAh9d	520	-78.62	5.94	10.01
7s9s3dAc	520	-190.03	-118.76	-98.08
5h4dTc6s	520	121.58	177.83	188.94
3sThJh7s	1528	469.15	614.37	631.98
8sKs4d8d	520	-122.02	-51.29	-58.84

Board	Pot Size	No subgame solving	On-tree subgame solving	Libratus subgame solving
7d9dQd4c	520	-117.01	-61.63	-55.00
KhQd4c2c	520	18.25	52.79	64.53
Th5dKc3c	1528	106.44	28.27	27.89
5hTdKd5d	520	-112.83	-51.58	-51.14
8sAsQd3c	520	-23.29	11.88	25.58
JsQsAh8c	520	2.00	48.29	57.12
2h4c8c7h	1092	-440.96	-302.17	-300.19
KhTdJc9d	520	149.99	252.14	258.13
8dKdJc7d	520	97.12	109.29	113.36
4s5dQc9d	520	-189.63	-148.10	-154.40
KsAdTc2c	1612	289.54	310.62	317.29
7sJs3hKh	1092	-647.40	-540.64	-543.01
As9cJcAd	1612	815.01	851.62	856.96
2sTd3c3d	520	-27.02	22.20	23.99
5s7s8hAh	1612	86.36	185.57	201.80
9s6dKc7h	520	155.74	150.35	153.13
2s8d5c7h	1612	-932.11	-699.46	-696.78
2c4c8cTc	520	179.47	194.87	192.98
2s8h2cAh	1560	-178.13	-105.25	-101.20
5h4c5cJh	780	-892.96	-810.03	-803.20
4sQdKc6d	1612	-505.65	-509.44	-500.73
4h2c5cKs	520	-221.57	-130.80	-116.83
3d8cJc3c	520	-72.23	34.85	38.82
6s8hQhAd	520	-203.37	-111.32	-105.72
Js6c8c6h	4836	-2249.11	-2092.56	-2031.98
Js9h2d7c	1352	526.09	678.14	671.05
3h6hTdQc	520	11.64	74.34	84.66
5h8hTcTh	1352	47.53	186.05	181.73
3dTd4c6d	1560	917.74	888.56	916.29
TsTcQc5c	780	-212.49	-172.65	-168.73
3s4dKdTd	1560	1247.06	1280.26	1284.98
3s5s5c7s	1560	1265.86	1487.49	1517.39
8h8cAc3h	780	-162.50	-147.70	-149.50
4s9cTc2d	1352	-727.41	-630.83	-635.38
2c4cQc7s	520	-63.12	4.93	10.63
6h9h9c8h	520	144.15	234.16	250.82
8s7h2c3h	520	108.11	126.82	116.77
6h5d3c4d	520	42.51	107.51	113.62
9s8dQd2s	1040	924.84	898.25	913.95
Jh8dAc3c	520	-21.25	-8.82	-3.73

Board	Pot Size	No subgame solving	On-tree subgame solving	Libratus subgame solving
5s9h3d4d	3250	-1708.69	-1520.19	-1405.83
6s6d7cAh	780	-173.77	-191.10	-187.85
8h9d9cQh	1092	-557.55	-529.21	-514.26
3s5sAcAs	520	-175.24	-149.30	-126.65
4d8d3cTs	1528	-1820.90	-1649.05	-1656.47
Js4d8dAc	1612	-230.33	-141.24	-141.94
6d5cJcAc	520	45.69	67.99	83.00
4sKs8d6c	2600	-320.31	-252.69	-206.71
Qh3d4dAd	1092	170.19	225.92	239.47
JdAd9cJc	520	-201.31	-125.81	-126.42
As5hTcJc	520	-93.34	-85.89	-78.74
5hJh4d5d	2340	888.33	955.21	976.24
2dQd2c8d	1092	-289.68	-215.88	-213.99
Qd4cQcKd	728	-114.28	-76.71	-75.00
Js2hKh6d	520	-80.67	-57.63	-51.86
QsThQh5s	520	-39.13	-18.47	-14.80
4sQdAdQc	520	-300.33	-198.81	-196.46
7sTsAcTd	520	-178.02	-81.26	-88.38
2h3h7hTd	520	55.23	136.49	151.27
2sJs3c9c	520	-226.71	-146.81	-135.62
4dJcAc9d	2296	-1034.20	-474.41	-469.85
Ks5d7d3c	600	172.84	280.07	277.34
8h6c9cTd	840	-971.34	-777.72	-760.64
Ks5dKdAd	400	-182.61	-91.91	-95.73
3hTh2c5s	500	-34.19	16.19	17.14
6h3d8cKd	2870	-3567.41	-3081.27	-3063.21
Jh6d7d2s	400	-34.05	30.77	27.71
Qh5cAc6c	1800	-358.43	-243.44	-172.53
7hQh4cJh	840	-411.06	-289.09	-283.06
5hJd9c3c	400	-46.98	43.96	43.73
7s7hJh3c	750	428.49	499.65	512.14
7h5dKdKh	840	-722.57	-748.68	-750.67
Qh6c7c2c	500	-198.70	-106.36	-101.85
Kd2c7cJc	1500	-1037.45	-825.45	-805.20
9hAd5cQh	500	-15.98	54.23	48.47
AsAh9c4d	600	121.25	209.73	217.16
As9h4dKs	200	61.19	93.94	93.18
5hKd8c3d	500	44.38	88.45	91.13
Kh7dJdJc	500	164.96	213.16	207.76
9h7d2c2s	3720	-707.91	-111.95	-77.88

Board	Pot Size	No subgame solving	On-tree subgame solving	Libratus subgame solving
8hThAhJs	400	-51.38	26.14	33.01
Ah4d7d6c	400	-83.75	-49.52	-45.26
Qd4c7cAc	500	-129.45	-59.69	-62.48
As4d7c5c	400	57.90	109.01	118.80
5h9cJc6s	820	63.85	219.76	220.22
TsQcKc4s	600	-213.01	-133.21	-132.06
As5h2c3s	400	-41.16	-2.73	0.78
2s3d8c4s	750	790.74	825.09	832.87
2s5h9d6c	1736	-1208.04	-1009.04	-1023.91
2c6cQcKc	820	-497.50	-264.09	-261.04
KsThQhAs	500	-219.12	-132.67	-115.43
AsJd9c4h	600	-116.09	-121.14	-130.61
QsAh8d6c	500	-16.98	-8.09	-10.06
JsKhTdTh	1148	-531.94	-231.56	-233.51
Th5dAcAd	600	-43.45	58.15	59.69
Qh2cTc9h	200	18.60	13.84	25.11
5sJh4cTh	500	154.19	234.89	233.41
4s7dKcAd	400	109.78	165.07	169.29
8s7hKh9c	200	-116.57	-68.41	-61.54
8hAd2c6s	1000	43.91	202.78	275.15
Td6cQc7h	500	-15.18	4.53	-1.09
Ts6h2dAh	1200	-469.32	-158.67	-120.48
6h8c9c8s	820	190.24	208.49	211.93
8sJdKcAd	1560	-59.44	483.56	503.69
6d7c9cAs	500	-37.92	34.55	43.12
As3c6cTs	200	34.60	60.03	58.57
2h9h5c6h	500	-278.54	-201.04	-205.13
JhAhTc3h	200	14.55	30.04	26.95
3h5h5d6h	750	804.25	868.59	879.38
Js8h8dKs	800	-657.86	-422.25	-422.26
Js7d9cKh	2000	-768.82	-802.70	-729.70
4s4c5c6c	1000	289.90	463.65	492.33
4sJsAd5c	500	3.49	56.67	56.72
Qs4dTdAd	750	-50.79	47.67	51.79
7sKd4cTs	2170	-1583.66	-964.11	-973.51
2d3dJcTd	1000	248.53	389.11	401.54
9h8cAcQc	2170	72.84	364.51	364.78
3h5dKdQc	500	149.32	170.97	169.80
4s8s7h4c	600	245.88	301.93	313.03
KsJhAd3s	500	-193.76	-123.54	-128.88

Board	Pot Size	No subgame solving	On-tree subgame solving	Libratus subgame solving
3s6sKd9c	2050	-1207.35	-942.42	-950.63
Qs6hJc4h	900	595.00	688.62	703.47
5h6d7d9h	1000	-814.50	-696.74	-646.74
7sKh2c2h	500	-278.04	-169.95	-164.57
JdAd7c9c	200	13.40	49.60	50.96
2s2c3c4d	1000	388.22	554.00	558.09
Js4hJhKd	4100	-2391.94	-1760.21	-1732.97
Js8hJcAh	500	-93.74	-29.43	-18.92
8s2h7dAd	900	578.77	712.53	725.38
Ah7c8c6h	750	270.70	271.89	276.54
8sKsQc3d	700	-85.27	69.62	68.10
2h4h8dQh	1500	454.59	538.38	520.63
KhAdAcTh	1148	167.00	163.70	185.03
3s5d9dQh	820	-284.74	-164.53	-159.39
Kd7cJcTd	1000	58.34	143.92	159.36
Ks8d6c3c	500	-117.37	8.45	8.15
9s5h9c5d	1736	-1051.14	99.60	125.06
Ks2h3hAh	750	143.04	217.25	201.98
5s9sQhQc	500	8.95	27.56	32.80
2sJh7c3s	1200	403.91	672.09	706.53
3sAsKh7h	600	77.68	135.95	146.43
JhJd6c5h	800	186.89	281.78	280.71
3s6sTcKs	500	-237.53	-159.56	-161.37
6s2h9c8c	500	-165.05	-121.95	-125.86
2sQh8d2d	1148	-861.12	-464.76	-467.09
As2h6hKc	700	-201.82	-105.74	-93.06
8hKd5c4s	500	-148.49	-15.04	-14.16
7s4hJcJd	400	-76.97	-45.56	-33.83
Ad4c5c7s	750	285.43	371.72	389.62
Qh2d9d5d	750	379.64	455.14	468.81
7hJhJc8s	500	-129.49	-59.89	-58.37
As5h4d5c	400	-179.49	-116.61	-110.66
6hTh9dJh	2050	-250.22	-137.60	-152.96
4s6s6hTh	1000	203.52	315.36	313.61
8d9cQcKd	500	-197.92	-125.92	-125.19
6s5dJc6d	2050	-1265.94	-986.52	-1003.76
Kh9d4cJh	1000	-156.65	40.33	19.59
4sJdQd7s	200	81.59	100.17	108.55
7h3d3cTh	1000	524.70	652.06	657.05
5sAhQc4c	600	-237.31	-101.09	-104.96

Table S1: **Performance against Baby Tartanian8 in various subgames.** Measured in mbb/h. "No subgame solving" measures Libratus's blueprint strategy using fold-call-bet purification (*31*). "On-tree subgame" solving shows the performance of Libratus with nested subgame solving using only bet sizes in the abstraction of Baby Tartanian8. "Libratus subgame solving" measures the full strength of our final AI. No post-processing was used in the subgames in nested subgame solving.

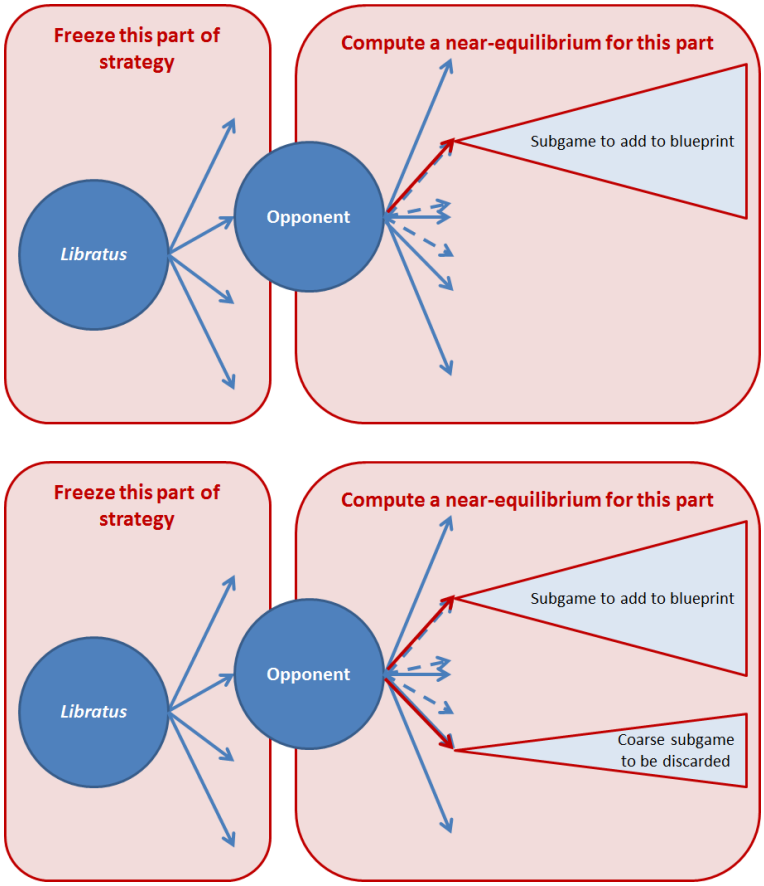


Figure S2: **Self-improvement.** The solid blue lines are existing actions in the abstraction. The dashed blue lines are bet sizes used by the opponents that are not in the abstraction. The red arrow shows the bet sizes included in the subgame solving. **Top:** Type 1 self-improvement. Here the red arrow is the most commonly played opponent bet size from the previous day. We solve a subgame following this bet size and add its solution to the blueprint strategy. **Bottom:** Type 2 self-improvement. Here the top red arrow is the highest-score opponent bet size from the previous day and the bottom red arrow is a default bet size that is already present in the blueprint abstraction. We add only the response to the top red arrow to the blueprint strategy.

References and Notes

1. J. Schaeffer, *One Jump Ahead: Challenging Human Supremacy in Checkers* (Springer, 1997).

2. M. Campbell, A. J. Hoane Jr., F.-H. Hsu, Deep Blue. *Artif. Intell.* **134**, 57–83 (2002). doi:10.1016/S0004-3702(01)00129-1

3. D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis, Mastering the game of Go with deep neural networks and tree search. *Nature* **529**, 484–489 (2016). doi:10.1038/nature16961 Medline

4. See supplementary materials for more details.

5. J. Nash, "Non-cooperative games," thesis, Princeton University (1950).

6. J. F. Nash, L. S. Shapley, *Contributions to the Theory of Games*, H. W. Kuhn, A. W. Tucker, Eds. (Princeton Univ. Press, 1950), vol. 1, pp. 105–116.

7. D. A. Waterman, Generalization learning techniques for automating the learning of heuristics. *Artif. Intell.* **1**, 121–170 (1970). doi:10.1016/0004-3702(70)90004-4

8. J. Shi, M. Littman, in *CG '00: Revised Papers from the Second International Conference on Computers and Games* (Springer, 2002), pp. 333–345.

9. D. Billings *et al*., in *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI)* (Morgan Kaufmann Publishers, San Francisco, 2003), pp. 661–668.

10. A. Gilpin, T. Sandholm, in *Proceedings of the National Conference on Artificial Intelligence (AAAI)* (AAAI Press, 2005), pp. 1684–1685.

11. M. Bowling, N. Burch, M. Johanson, O. Tammelin, Heads-up limit hold'em poker is solved. *Science* **347**, 145–149 (2015). doi:10.1126/science.1259433 Medline

12. Libratus is Latin and means balanced (for approximating Nash equilibrium) and forceful (for its powerful play style and strength).

13. An imperfect-information subgame (which we refer to simply as a subgame) is defined differently than how a subgame is usually defined in game theory. The usual definition requires that a subgame starts with the players knowing the exact state of the game, that is, no information is hidden from any player. Here, an imperfect-information subgame is determined by information that is common knowledge to the players. For example, in poker, a subgame is defined by the sequence of visible board cards and actions the players have taken so far. Every possible combination of private cards—that is, every node in the game tree which is consistent with the common knowledge—is a root of this subgame. Any node that descends from a root node is also included in the subgame. A formal definition is provided in the supplementary material.

14. N. Burch, M. Johanson, M. Bowling, in *AAAI Conference on Artificial Intelligence (AAAI)* (AAAI Press, 2014), pp. 602–608.

15. M. Moravcik, M. Schmid, K. Ha, M. Hladik, S. Gaukrodger, in *AAAI Conference on Artificial Intelligence (AAAI)* (AAAI Press, 2016).

16. E. Jackson, in *AAAI Workshop on Computer Poker and Imperfect Information* (AAAI Press, 2014).

17. M. Zinkevich, M. Johanson, M. H. Bowling, C. Piccione, in *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)* (Neural Information Processing Systems Foundation, Inc., 2007), pp. 1729–1736.

18. Y. Nesterov, Excessive gap technique in nonsmooth convex minimization. *SIAM J. Optim.* **16**, 235–249 (2005). doi:10.1137/S1052623403422285

19. S. Hoda, A. Gilpin, J. Peña, T. Sandholm, Smoothing techniques for computing Nash equilibria of sequential games. *Math. Oper. Res.* **35**, 494–512 (2010). doi:10.1287/moor.1100.0452

20. A. Gilpin, J. Peña, T. Sandholm, First-order algorithm with O(ln(1/ϵ)) convergence for ϵ-equilibrium in two-person zero-sum games. *Math. Program.* **133**, 279–298 (2012). doi:10.1007/s10107-010-0430-2.

21. C. Kroer, K. Waugh, F. Klnç-Karzan, T. Sandholm, in *Proceedings of the ACM Conference on Economics and Computation (EC)* (ACM, New York, 2017).

22. O. Tammelin, N. Burch, M. Johanson, M. Bowling, in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* (AAAI Press, 2015), pp. 645–652.

23. The version of HUNL that we refer to, which is used in the Annual Computer Poker Competition, allows bets in increments of $1, with each player having $20,000 at the beginning of a hand.

24. M. Johanson, "Measuring the size of large no-limit poker games," (Technical Report, Univ. of Alberta Libraries, 2013).

25. A. Gilpin, T. Sandholm, T. B. Sørensen, in *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems* (International Foundation for Autonomous Agents and Multiagent Systems, 2008), vol. 2, pp. 911–918.

26. D. Schnizlein, M. Bowling, D. Szafron, in *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence* (AAAI Press, 2009), pp. 278–284.

27. S. Ganzfried, T. Sandholm, in *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence* (AAAI Press, 2013), pp. 120–128.

28. Annual Computer Poker Competition; www.computerpokercompetition.org.

29. N. Brown, T. Sandholm, in *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI)* (AAAI Press, 2014), pp. 594–601.

30. N. Brown, T. Sandholm, in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)* (AAAI Press, 2016), pp. 4238–4239.

31. N. Brown, S. Ganzfried, T. Sandholm, in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems* (International Foundation for Autonomous Agents and Multiagent Systems, 2015), pp. 7–15.

32. N. Brown, S. Ganzfried, T. Sandholm, in *AAAI Conference on Artificial Intelligence (AAAI)* (AAAI Press, 2015), pp. 4270–4271.

33. M. Johanson, N. Burch, R. Valenzano, M. Bowling, in *Proceedings of the 2013 International Conference on Autonomous Agents and Multiagent Systems* (International Foundation for Autonomous Agents and Multiagent Systems, 2013), pp. 271–278.

34. M. Lanctot, K. Waugh, M. Zinkevich, M. Bowling, in *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)* (Neural Information Processing Systems Foundation, Inc., 2009), pp. 1078–1086.

35. R. Gibson, M. Lanctot, N. Burch, D. Szafron, M. Bowling, in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence* (AAAI Press, 2012), pp. 1355–1361.

36. M. Johanson, N. Bard, M. Lanctot, R. Gibson, M. Bowling, in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems* (International Foundation for Autonomous Agents and Multiagent Systems, 2012), vol. 2, pp. 837–846.

37. R. Gibson, "Regret minimization in games and the development of champion multiplayer computer poker-playing agents," thesis, University of Alberta (2014).

38. There are a number of theoretically correct ways to choose actions on the basis of their regrets. The most common is regret matching, in which an action is chosen in proportion to its positive regret (*58*). Another common choice is hedge (*59*, *60*).

39. An action a with regret $R(a)$ that is below a threshold C (where C is negative) is sampled with probability $K/[K + C – R(a)]$, where K is a positive constant. There is additionally a floor on the sample probability. This sampling is only done for about the last half of iterations to be run; the first half is conducted using traditional external-sampling MCCFR. Other formulas can also be used.

40. K. Waugh *et al*., in *Symposium on Abstraction, Reformulation, and Approximation (SARA)* (AAAI Press, 2009).

41. M. Johanson, N. Bard, N. Burch, M. Bowling, in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence* (AAAI Press, 2012), pp. 1371–1379.

42. S. Ganzfried, T. Sandholm, in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (International Foundation for Autonomous Agents and Multiagent Systems, 2015), pp. 37–45.

43. N. Brown, T. Sandholm, *Adv. Neural Inf. Process. Syst.* **30**, 689–699 (2017).

44. In Libratus, we considered "sufficiently small" to be situations where no additional bets or raises could be made.

45. Despite lacking theoretical guarantees, unsafe subgame solving empirically performs well in certain situations and requires less information to be precomputed. For this reason, Libratus uses it once upon first reaching the third betting round, while using safe subgame solving in all subsequent situations.

46. We solved augmented subgames using a heavily optimized form of the CFR+ algorithm (*22*, *61*) because of the better performance of CFR+ in small games where a precise solution is desired. The optimizations we use keep track of all possible P1 hands rather than dealing out a single one at random.

47. Note that the theorem only assumes perfect recall in the actual game, not in the abstraction that is used for computing a blueprint strategy. Furthermore, applying Estimated-Maxmargin assumes that that subroutine maximizes the minimum margin; a sufficient condition for doing so is that there is no abstraction in the subgame.

48. Indeed, the original purpose of safe subgame solving was merely to reduce space usage by reconstructing subgame strategies rather than storing them.

49. Specifically, Libratus increased or decreased all its bet sizes by a percentage chosen uniformly at random between 0 and 8%.

50. M. Moravčík, M. Schmid, N. Burch, V. Lisý, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, M. Bowling, DeepStack: Expert-level artificial intelligence in heads-up no-limit poker. *Science* **356**, 508–513 (2017). doi:10.1126/science.aam6960 Medline

51. D. Billings, D. Papp, J. Schaeffer, D. Szafron, in *Proceedings of the National Conference on Artificial Intelligence (AAAI)* (AAAI Press, 1998), pp. 493–499.

52. S. Ganzfried, T. Sandholm, in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (International Foundation for Autonomous Agents and Multiagent Systems, 2011).

53. S. Ganzfried, T. Sandholm, Safe opponent exploitation. *ACM Transaction on Economics and Computation (TEAC)* **3**, 1–28 (2015). doi:10.1145/2716322

54. Based on the available computing resources, we chose $k = 3$ so that the algorithm could typically fix three holes to reasonable accuracy in 24 hours.

55. Baby Tartanian8 and all other AIs in the ACPC are available to ACPC participants for benchmarking.

56. Baby Tartanian8 uses action translation in response to bet sizes that are not in its action abstraction. Our experiments above demonstrated that action translation performs poorly compared to subgame solving. Using only bet sizes in Baby Tartanian8's abstraction disentangles the effects of action translation from the improvement of nested subgame solving. Baby Tartanian8 still used actions that were not in Libratus's abstraction, and therefore the experiments can be considered conservative.

57. Because both the humans and the AI adapted over the course of the competition, treating the hands as independent is not entirely inappropriate. We include confidence figures to provide some intuition for the variance in HUNL. In any case, 147 mbb/hand over 120,000 hands is considered a massive and unambiguous victory in HUNL.

58. S. Hart, A. Mas-Colell, A simple adaptive procedure leading to correlated equilibrium. *Econometrica* **68**, 1127–1150 (2000). doi:10.1111/1468-0262.00153

59. N. Littlestone, M. K. Warmuth, The weighted majority algorithm. *Inf. Comput.* **108**, 212–261 (1994). doi:10.1006/inco.1994.1009

60. Y. Freund, R. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.* **55**, 119–139 (1997). doi:10.1006/jcss.1997.1504

61. M. Johanson, K. Waugh, M. Bowling, M. Zinkevich, in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)* (AAAI Press, 2011), pp. 258–265.

62. L. Kocsis, C. Szepesvári, in *European Conference on Maching Learning (ECML)* (Springer, 2006), pp. 282–293.

63. R. Coulom, *Computers and Games* (Springer, 2007), pp. 72–83.

64. D. E. Knuth, R. W. Moore, An analysis of alpha-beta pruning. *Artif. Intell.* **6**, 293–326 (1975). doi:10.1016/0004-3702(75)90019-3

65. J. F. Nash, Equilibrium points in *n*-person games. *Proc. Natl. Acad. Sci. U.S.A.* **36**, 48–49 (1950). doi:10.1073/pnas.36.1.48 Medline

66. N. Brown, T. Sandholm, in *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)* (2015), pp. 1972–1980.

67. N. Brown, T. Sandholm, in *International Conference on Machine Learning* (Proceedings of Machine Learning Research, 2017).

68. S. Ganzfried, T. Sandholm, K. Waugh, in *International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (International Foundation for Autonomous Agents and Multiagent Systems, 2012), pp. 871–878.