

BluffJAX: Adversarial Imperfect Information Games in JAX

Aryaman Reddi^{1,2,*}, Jan Peters^{1,2,3}, Carlo D’Eramo⁴

¹Department of Computer Science, TU Darmstadt, Germany

²Hessian Center for Artificial Intelligence (Hessian.ai), Germany

³German Research Center for AI (DFKI), Systems AI for Robot Learning, Germany

⁴Center for Artificial Intelligence and Data Science, University of Würzburg, Germany

We introduce BluffJAX: an open-source suite of adversarial imperfect information games in JAX. We provide canonical implementations of games designed for high simulation throughputs and parallelization on GPU accelerators. Our suite consists of well-studied benchmarks such as Texas Hold’Em Poker and Kuhn Poker, as well as games that have not been previously studied in reinforcement learning research, such as Bluff, Stud Poker, and Kemps. We hope that implementing a variety of game mechanics and difficulties will introduce new challenges and foster novel research directions in game-theoretic methods for RL. We benchmark the throughput performance and memory usage of our environments in single and multi-GPU settings, demonstrating scaling of up to hundreds of millions of samples per second, and motivating the usage of BluffJAX over related GPU and CPU-based libraries. We benchmark reinforcement learning, tree search, and game-solving algorithms in JAX in order to provide users with baseline results and facilitate future comparisons.

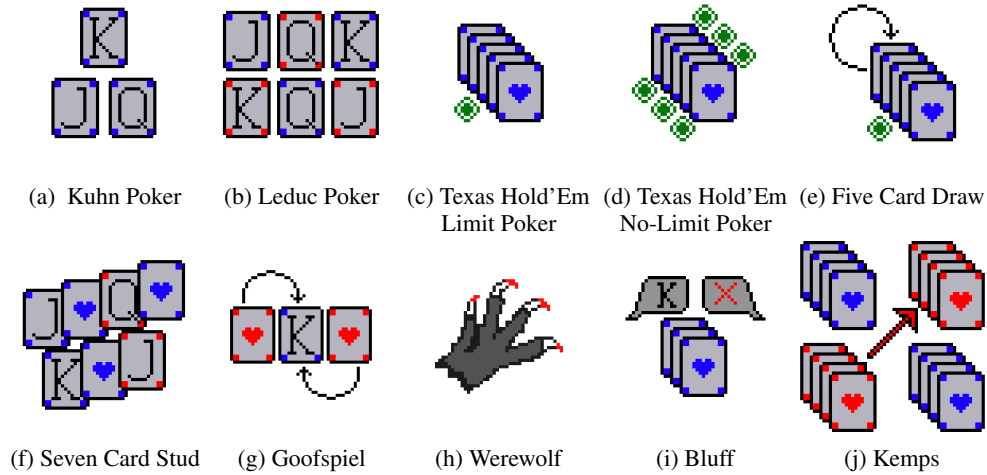


Figure 1: Games in BluffJAX.

1 Introduction

The development of algorithms that can achieve superhuman performance in games with intractably large state-action spaces has been a driving force in Artificial Intelligence (AI) research for decades [Mnih et al., 2015, Silver et al., 2016, 2017b]. Imperfect information games present unique challenges: in addition to large state and action spaces, inaccessibility to information and

*Correspondence to <aryaman.reddi@tu-darmstadt.de>

opponent strategy severely hampers standard learning methods like fictitious self-play, value estimation, and tree search - which can be seen in games such as Hold’Em Poker [Brown et al., 2020], Diplomacy [Bakhtin et al., 2022], and Dota 2 [Berner et al., 2019].

One bottleneck for scaling large-scale RL systems in such domains is the availability of data. High-speed simulators that can leverage hardware accelerators and collect diverse data from parallelized environments are a crucial ingredient in modern RL research. In particular, JAX [Frostig et al., 2019] has gained popularity among machine learning researchers in recent years by utilizing the XLA compiler infrastructure for JIT (just-in-time) compilation and automatic differentiation of Python functions.

In this work, we introduce **BluffJAX**, which is (to the best of our knowledge) the first open-source hardware-accelerated JAX suite focused primarily on imperfect information games. Our benchmark include well-studied games such as Texas Hold’Em Poker variants and Goofspiel, while also introducing games that have not been studied in RL research thus far, such as Bluff, Kemps, and multi-round Werewolf. We hope that providing high-speed implementations of well-known games will help alleviate the reproducibility crisis in RL research, as well as introduce new challenges to the RL community that will foster novel research directions in game-theoretic RL and search methods. All code is open-source and available at <https://github.com/bluffjax/bluffjax>.

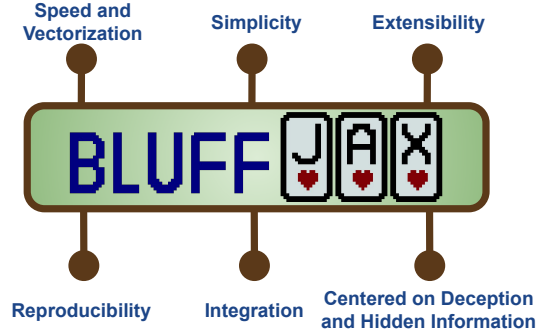


Figure 2: Our philosophy and design criteria.

To validate the functionality and scalability of our suite, we benchmark the speed and memory usage of all 10 implemented games across varying numbers of parallel environments in single-GPU and multi-GPU settings. We also analyze the scaling performance of BluffJAX against related game libraries. We provide algorithm baseline results for some well-studied games to motivate our claim that the implementations use canonical game mechanics and are compatible with existing algorithms. Additionally, we include some pre-trained baseline models that we hope will provide users of BluffJAX with a predefined standard by which they can evaluate their own experiments. We implement fully JIT-compatible algorithms for certain games, including Counterfactual Regret Minimization [Zinkevich et al., 2007] (CFR) and exploitability metrics for Kuhn Poker and Leduc Poker. In large games, it is typically intractable to expand the entire game tree in order to perform CFR; thus, we also implement JIT-compatible deep RL and tree search algorithms.

Figure 2 outlines our philosophy and design criteria for BluffJAX. The contributions of our library are summarized as follows:

- **Speed and vectorization:** BluffJAX emphasizes high environment transition throughput rates using efficient game branching logic and leverages JAX-native parallelization, JIT compilation, and on-GPU processing.
- **Simplicity:** We use a simple API and well-documented code for ease of use.
- **Extensibility:** Our library is structured to enable easy addition of new games and mechanics.
- **Reproducibility:** We provide implementations of well-known game-solving and RL algorithms as well as baseline models to give users a reproducible starting point for their research directions.
- **Integration:** We include wrappers and helper functions to ease the on-boarding of users already familiar with other RL libraries.
- **Centered on Deception and Hidden Information:** To the best of our knowledge, GPU-accelerated implementations of imperfect information games remains a mostly-unfulfilled niche within the ecosystem of RL environment libraries. We aim to fill this gap by providing implementations of both well-known and new games.

2 Related Works

Games as AI Benchmarks. The pursuit of superhuman game-playing abilities in AI systems has been a driving force in AI research for decades [Samuel, 1959, Tesauro et al., 1995, Campbell et al., 2002], and advances in deep learning have enhanced AI game-playing systems beyond the need for human expert knowledge. Mnih et al. [2015] is often attributed with reinvigorating deep RL research by demonstrating superhuman performance in several Atari games using Deep Q-Networks. AlphaGo [Silver et al., 2016], AlphaGo Zero [Silver et al., 2017b], Alpha Zero [Silver et al., 2017a], and MuZero [Schrittwieser et al., 2020] were critical milestones in exhibiting the increasing abilities of deep RL and tree search to reach increasing capabilities in unsolved games like Go, Chess, and Shogi. Systems like OpenAI Five [Berner et al., 2019], AlphaStar [Vinyals et al., 2019], and Cicero [Bakhtin et al., 2022] have demonstrated the abilities of AI systems to excel in complex multi-agent settings.

Algorithms for Imperfect Information Games. Imperfect information games fundamentally differ from perfect information games, since the existence of private information (partial observability) means that optimal performance requires reasoning over the strategies of other players. As games grow large, tabular methods such as Counterfactual Regret Minimization (CFR) [Zinkevich et al., 2007] become unfeasible. Various methods have attempted to augment standard RL and tree search algorithms with notions of belief state probability estimation. Deep CFR [Brown et al., 2019] replaces tabular CFR with Monte Carlo CFR (MCCFR) [Burch et al., 2012] for subgame sampling and neural networks to estimate instantaneous regrets. Libratus [Brown et al., 2017] and Pluribus [Brown and Sandholm, 2019] utilize MCCFR to achieve expert-level performance in 2-player and 6-player Texas Hold’Em Poker respectively. Recursive Belief-based Learning (ReBeL) [Brown et al., 2020] extended this notion further by leaving state abstraction behind and combining subgame deep CFR with value learning over public belief states. Neural Fictitious Self-Play (NSFP) [Heinrich and Silver, 2016] uses best-response policy training with supervised average-policy mimicry from experience replay in order to train RL agents in a 2-player self-play setting, achieving decent performance in model poker games without state abstraction or tree search.

The Benefits of JAX. Over the past few years, adoption of JAX-based training in deep RL research has led to a greater necessity for JAX-native benchmarks. One major benefit of JAX benchmarks is that environment logic can be run on the GPU directly, mitigating the overhead cost of transferring data between the CPU and GPU. The Python JAX function `jax.vmap` allows for efficient parallelization of environments, which has been shown to effectively scale learning [Gallici et al., 2024]. Finally, `jax.jit` allows for just-in-time compilation of JAX-compatible Python functions using the XLA compiler, leading to demonstrable speedups over PyTorch-based neural network training and array logic.

Related Benchmarks. Suites such as Gymnax [Lange, 2024] and Jumanji [Bonnet et al., 2024] implement well-known single-player environments into JAX and have provided researchers with familiar RL environments such as classic control games and MinAtar. Brax [Freeman et al., 2021] implements differentiable rigid-body simulation in JAX for physics-based control, enabling faster training in robotic domains. JAXMarl [Rutherford et al., 2024] implements well-studied multi-agent environments such as StarCraft II, Multi-Particle Environments, and Hanabi. MEAL [Tomilin et al., 2025] presents a benchmark tailored to continual MARL by procedurally generating tasks based on the Overcooked benchmark. PGX [Koyamada et al., 2023] implements adversarial games with (mostly) perfect information in JAX, such as Chess, Go, and Hex. RLCard [Zha et al., 2020] and PettingZoo [Terry et al., 2021] are CPU-based suites that contain some overlapping environments with BluffJAX (e.g. Texas Hold’Em Poker). Finally, OpenSpiel [Lanctot et al., 2019] is worth noting as it provides canonical implementations of several games in C++ and has been used across many works². In this work, we address an open niche in the benchmark ecosystem, since **BluffJAX specializes on JAX implementations of adversarial games with imperfect information, particularly those involving deception and signaling**. Table 1 summarizes some notable attributes of BluffJAX when compared to other well-known MARL benchmarks. Note that we describe BluffJAX as *mostly* adversarial because of the inclusion of team games that require collaboration within each team. Table 1 compares BluffJAX to other relevant suites, including those mentioned above as well as Ruhdorfer et al. [2024], Agapiou et al. [2022], Kurach et al. [2020].

²It should be noted that some JAX code examples (e.g. CFR, DQN) are provided in the OpenSpiel library; however, the games themselves are CPU-based.

Table 1: Comparison between BluffJAX and other Multi-Agent Benchmarks.

Benchmark	GPU-accelerated	Variable no. agents	Adversarial or cooperative	Perfect or Imperfect Info.
PGX	✓	✗	Adversarial (mostly)	Perfect (mostly)
JaxMARL	✓	✓	Both	Both
Overcooked GC	✓	✗	Cooperative	Perfect
MEAL	✓	✓	Cooperative	Both
VMAS	✗	✓	Both	Imperfect
OpenSpiel	✗	✓	Both	Both
RLCard	✗	✗	Adversarial	Imperfect
PettingZoo	✗	✓	Both	Both
Melting Pot 2	✗	✓	Both	Both
Google Football	✗	✓	Both	Both
BluffJAX	✓	✓	Adversarial (mostly)	Imperfect

3 BluffJAX

In this section, we discuss our choices of environments, API, and design elements of BluffJAX. Full details of the games, including mechanics, observation/action spaces, and rewards are available in Appendix A. A thorough discussion of the API and particular design choices is available in Appendix B.

3.1 Environments

We aim for a mixture of toy environments, well-studied games, and new games in BluffJAX. A summary of the 10 implemented games is available in Table 2, where we note the number of agents, observation size, number of actions, number of JAX branches, and the performance metric for each game. Note that the number of JAX branches refers to the number of parallel executions at each timestep in game logic, which is not necessarily equal to the branching factor of the game.

Kuhn Poker [Kuhn, 1953] and **Leduc Poker** are well-studied and theoretically solved poker variants, in that their exact exploitability and equilibrium strategies are tractable. In this paper, we benchmark some common RL and game-theoretic algorithms on Kuhn and Leduc Poker in order to validate our implementations and motivate our choices for baselines in more complex environments.

Texas Hold’Em Limit Poker and **Texas Hold’Em No-Limit Poker** are commonly-studied poker variants which present sufficient challenges to existing game-solving algorithms. They are typically implemented with certain abstractions for simplification; e.g. fixed increment sizes and simple bankroll management.

We also introduce two lesser-known poker variants that include mechanics that we believe pose interesting challenges for current RL algorithms. In **5 Card Draw Poker**, players build a hand between betting rounds by exchanging cards with the house, which moves the signaling focus (which is typically on bet sizes and public cards) to the exchanges made by players. The exchange mechanic introduces a new dimension of play, as agents must consider more sources of deception with less information. **7 Card Stud Poker** is a variant where player hands are revealed across multiple betting rounds, requiring reasoning based on public information and betting history. Stud Poker differs to Texas Hold’Em as a game with longer horizons, higher information density, and more dynamic strategies.

Goofspiel is a 2-player simultaneous-move game which simulates simple auctions; each round consists of players making a fixed bet for a random, visible prize until no prizes or bids remain. Pure strategies in Goofspiel are strictly dominated, requiring agents to model stochastic strategies in order to avoid exploitation.

Bluff (AKA Cheat, I Doubt It) is a card game where each player aims to reduce their hand to 0 by discarding cards in a constrained order - players have the option to lie about which cards they are discarding. Bluff presents a unique challenge among the BluffJAX environments, as agents must evade detection in long episodes in order to succeed.

Werewolf is a team-based social role game that has been studied in the context of reasoning and cooperation among Large Language Models. In BluffJAX, we introduce the first JAX-based implementation of multi-round werewolf with multiple roles. As a social deduction game, werewolf requires agents to cooperate with other members of their respective team in order to win. **Kemps** is a team game centered around communication where agents play in teams of two and aim to build a hand of a certain rank. To win, they must publicly signal their teammate about their complete hand at the risk of getting caught by opposing teams.

Table 2: Games in BluffJAX. Note than in several games, the observation size depends on n. Note also that Kemps contains an explicit communication channel (default size 2).

Environment	n	Obs. size	# Actions	# Branches	Metric
Kuhn Poker	2	9	2	1	exploitability
Leduc Poker	2	36	3	3	exploitability
THL Poker	2-10	72+n	4	4	chips/hand
THNL Poker	2-10	54	5	5	chips/hand
5 Card Draw	2-10	54	37	11	chips/hand
7 Card Stud	2-10	77+208(n-1)+n	4	4	chips/hand
Goofspiel	2	39	13	1	win rate
Bluff	≥ 3	263	13	10	win rate
Werewolf	6	43	7	7	win rate
Kemps	4,6,8..	52(n+1) + n(comm)	172*comm.	1	win rate

3.2 API

The API supports both parallel and agent-environment-cycle environments (AEC) with generic base classes that can be easily modified in order to fulfill specific game logic. Unlike other suites which use a dual API format [Terry et al., 2021], BluffJAX forgoes agent iterator cycles in favor of handling agent transitions and reset logic internally for ease of use. For ease of use for researchers more familiar with other libraries, wrappers for the APIs of JaxMARL and PettingZoo are available. Since most research in imperfect information games is done in self-play, we find that the most efficient pattern is to keep the agent-environment interface agent-agnostic by default, allowing specific placement of agents if required. Games such as Bluff, Goofspiel, and Kemps are easily customizable and can be run at different difficulty levels by simply changing certain environments parameters.

The following code snippet shows the basic usage of BluffJAX to run 1000 agent actions in 16 parallel environments:

```

1 import jax
2 import bluffjax
3 import agent
4
5 env=make("kuhn_poker") # Create an environment
6 rng=jax.random.key(seed=42)
7 rng_resets=jax.random.split(rng,16) # 16 parallel environments
8 state,obs=jax.vmap(env.reset)(rng_resets)
9
10 def step(carry,unused):
11     state,obs=carry
12     action=jax.vmap(agent.get_action)(obs)
13     state,obs,rew,absorbing,done,info=jax.vmap(env.step)(state,action)
14     return (state,obs),rew
15
16 def step_scan(state,obs):
17     return jax.lax.scan(step,(state,obs),None,length=1000)
18
19 (final_state,final_obs),rew=jax.jit(step_scan)(state,obs)

```

We use this snippet to explain some features of our API and functional JAX programming for those unfamiliar. The object `rng` is a pseudorandom key generated deterministically by an initial seed. Unlike NumPy, JAX requires explicitly passing pseudorandom keys to functions that require

randomness³. `jax.random.split` deterministically splits a pseudorandom key into multiple keys that may be used by separate random functions. `jax.vmap` is a vectorizing map which maps a function over vectorized arguments - in this case, `env.reset` is vectorized over an array of reset keys, producing vectorized states and obs arrays. `state` contains all the information that defines the current game state, including the current player and public/private information. `obs` is a vector observation for the current player. `env.step` takes the current state and action and generates the next state, the observations for the next player, and a vector of rewards for all agents. The function `step` is converted into a `jax.lax.scan` function with `length=1000` to indicate 1000 sequential steps within each parallel environment. Finally, `step_scan` outputs the final carry and vectorized reward outputs.

4 Performance Benchmarking

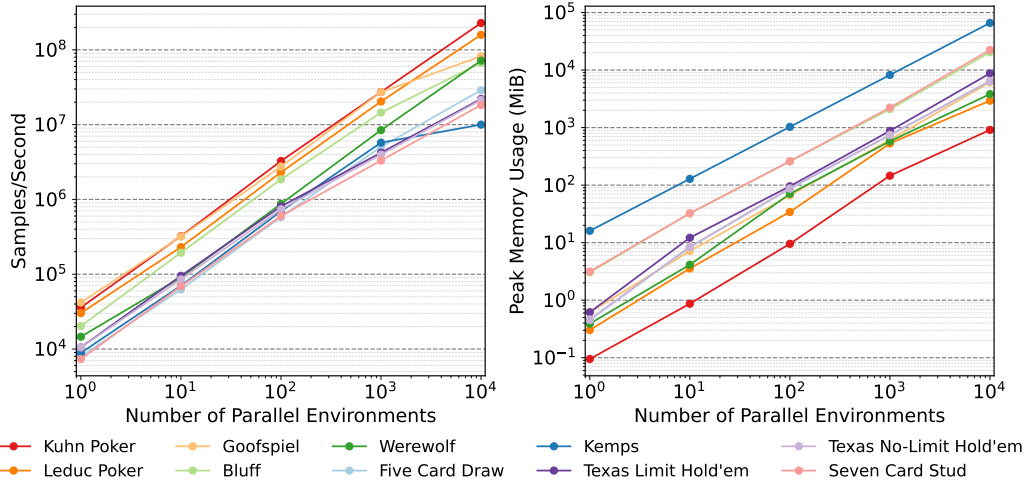


Figure 3: Speed and peak memory usage benchmarking for BluffJAX environments. All results are obtained on an RTX 6000 Ada GPU for 1000 random steps on all parallel environments. Each datapoint is a mean across 10 seeds (standard errors are not visible at this scale).

In this section, we measure the speed and memory requirements of BluffJAX environments. Figure 3 shows the simulation throughput and peak memory usage for increasing parallel environments for all BluffJAX games. For almost all games in BluffJAX, throughput scales log-linearly with the number of parallel environments. At 10,000 environments, we see that even the slowest game (Seven Card Stud) still generates on the order of $\sim 2e7$ samples/s, evincing the benefits of parallelization in addressing data bottlenecks. Certain environments scale sub-log-linearly in throughput at high parallelization, which appears to be due to GPU memory constraints, as peak memory usage scales log-linearly as well - notably, Kempts demands the highest memory and throughput suffers after 1000 environments. As expected, environments with heavier game logic and larger array operations are subject to higher memory requirements and lower throughput at all levels of parallelization. In Figure 4, we compare simulation throughput on both single and multi-GPU settings. We see that in the availability of high compute resources, BluffJAX achieves an average of 2.3x higher throughput.

Figure 5 compares the sample throughput rates of BluffJAX against those of similar GPU-based (PGX) and CPU-based (OpenSpiel, RLCard, PettingZoo) libraries. The throughputs of BluffJAX are similar to those of PGX - this is likely due to the relative simplicity of the overlapping games (Kuhn/Leduc poker) which leaves little room for further game logic optimizations. Our similarity to PGX motivates our claim that our implementations are canonical and well-optimized, since they match the results of a well-established benchmark also designed for GPU acceleration. Note that our novel contribution relative to PGX is not in the throughput rates for these 2 simple games, but rather in our JAX implementations of non-overlapping games. BluffJAX outperforms all CPU-based libraries with a gap of at least an order of magnitude at 1000 environments and beyond. Notably, efficient

³<https://docs.jax.dev/en/latest/random-numbers.html>

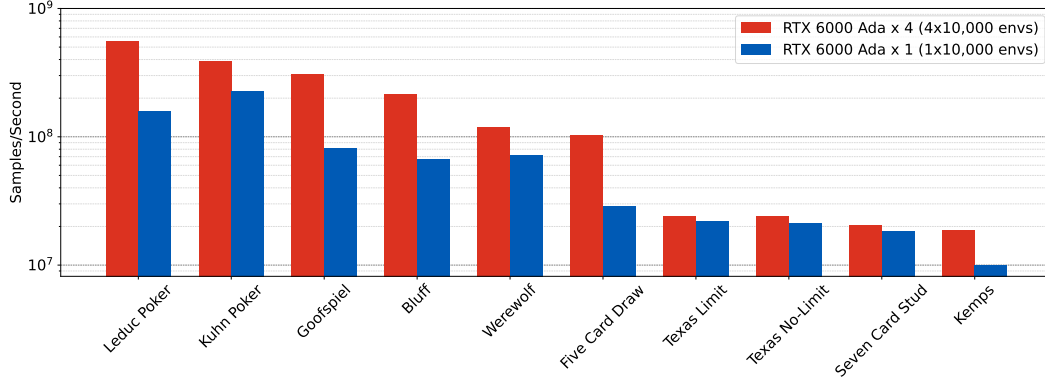


Figure 4: Speed benchmarking for BluffJAX environments on single vs multi-GPU setups. All results are obtained with RTX 6000 Ada GPUs for 1000 random steps on 10,000 parallel environments. Each bar is a mean across 10 seeds (standard errors are not visible at this scale).

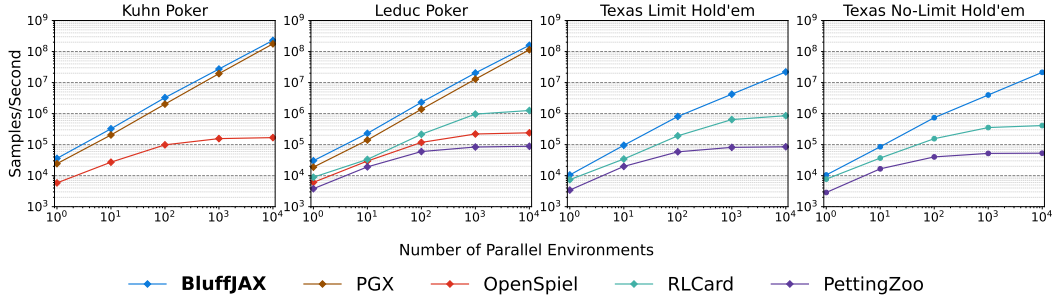


Figure 5: Speed benchmarking comparison between BluffJAX and related libraries on single-GPU and multi-CPU-core setups. We specifically chose the 4 games with the most overlapping implementations in related libraries (although only BluffJAX implements all 4). All GPU results are obtained on an RTX 6000 Ada GPU for 1000 random steps on all parallel environments. All CPU results are obtained on a computational cluster with 64GB of RAM and an AMD Ryzen 9 16-Core processor using the Python multiprocessing library. Each datapoint is a mean across 10 seeds (standard errors are not visible at this scale).

vectorization enables GPU-based libraries to continue scaling linearly as parallelization increases, while scaling stagnates on CPUs. Note also that BluffJAX outperforms CPU-based libraries even with no parallelization due to efficient game tree logic and leveraging JIT compilation.

5 Experimental Results

In this section, we validate our implementations and benchmark some RL, tree search, and model-based game solving algorithms on our implemented environments. We first validate the ability of our JAX-based implementation of CFR to find near-optimal strategies for solved games, as well as standard RL, RL augmented with NFSP, and Deep CFR algorithms. We then compare RL+NFSP with ReBeL baselines on Texas Hold’Em Poker, a well-studied medium for game-theoretic algorithms. Finally, we benchmark our remaining environments with RL+NFSP baselines.

Figure 6 shows the exploitability of various algorithms on Kuhn and Leduc Poker. The exploitability of a strategy profile is defined as net utility gained by an optimal policy playing against that strategy profile. Lower exploitability is better; an exploitability of 0 means a strategy profile is a Nash equilibrium [Nash Jr, 1950]. We note the exploitability of a uniform random policy on Kuhn (0.4583) and Leduc (2.3736), which matches values from Kawamura et al. [2017] and OpenSpiel. We also include the exploitability obtained by 100 iterations of CFR. Deep CFR achieves near-convergence on Kuhn Poker with 100 iterations, but struggles with Leduc Poker. This may be because truncating the relatively shallow game tree of Leduc Poker may exaggerate errors from function approximation.

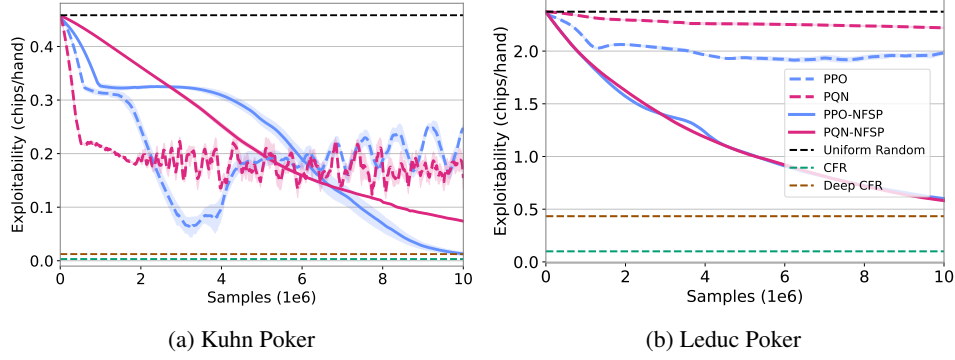


Figure 6: Exploitability of Kuhn and Leduc Poker. Each algorithm is trained in self-play with $1e7$ samples and evaluated across 10 seeds.

We benchmark the exploitability of two RL algorithms, Proximal Policy Optimization (PPO) [Schulman et al., 2017] and Parallelized Q-Network (PQN) [Gallici et al., 2024], on these environments. We choose PPO and PQN as they are simple to implement, well-known by the RL research community, and are well-suited to parallelized environment training and GPU acceleration. While PPO and PQN are able to achieve decent results in Kuhn Poker (beating Uniform Random by a fair margin), they tend to exhibit oscillatory behavior. Cycling is a well-studied phenomenon in two-player zero-sum games [Shapley, 1963, Balduzzi et al., 2018] which is worsened in self-play settings due to bootstrapping with self-produced training data [Heinrich and Silver, 2016].

We also implement PPO-NFSP and PQN-NFSP, which are versions of NFSP with PPO and PQN as the best response learners respectively alongside a supervised anticipatory network. We note that they achieve better results than vanilla PPO and PQN in both environments, but still fall short of optimality in Leduc Poker.

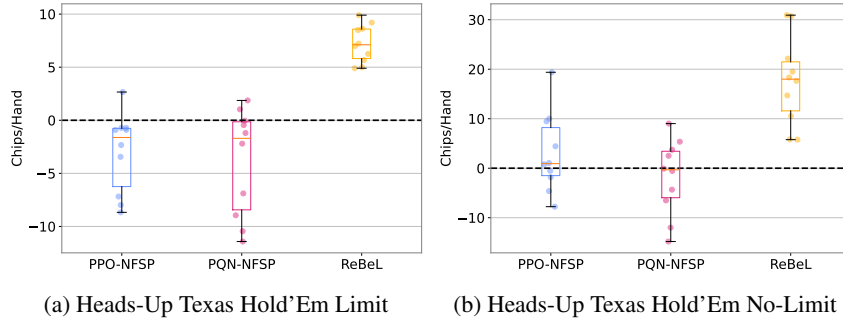


Figure 7: Final performance of PPO-NFSP, PQN-NFSP, and ReBeL against pre-trained ReBeL evaluation opponent in Heads-Up Texas Hold’Em. Each algorithm is trained in self-play with $1e7$ samples and evaluated across 10 seeds. Evaluation opponent is trained in self-play with $5e6$ samples. Dashed lines indicate break-even performance.

In Figure 7 we test the performance of baseline algorithms in Heads-Up (two-player) Texas Hold’Em Poker. In order to provide standardized performance results, we use ReBeL trained in self-play for $5e6$ steps as an evaluation opponent. As expected, ReBeL performs well when tested against itself, due to access to more training samples and exploitation strategies learned during self-play. PPO-NFSP and PQN-NFSP are typically unable to reach break-even performance despite training on double the number of samples as the ReBeL baseline. This is likely due to ReBeL’s well-tuned formulation for Hold’Em Poker, which utilizes both depth-limited CFR for subgame solving and function approximation, thereby leveraging game knowledge which is unavailable in model-free RL.

Finally, in Figure 8 we evaluate the performance of baseline algorithms in the remaining 6 games in BluffJAX. Note that we do not test ReBeL here as it utilizes model-based subgame solving, which is currently only available for Heads-Up Hold’Em variants. Therefore, we use PPO-NFSP

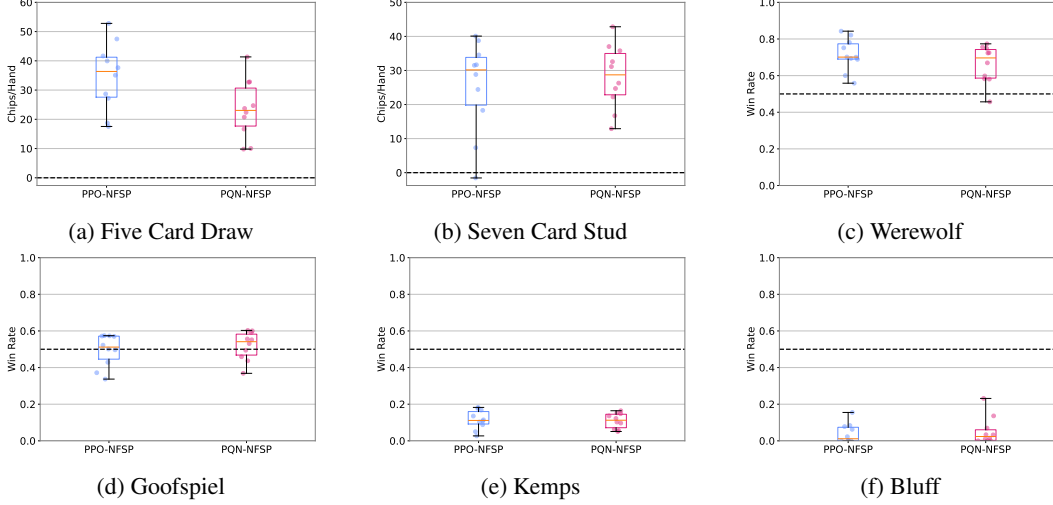


Figure 8: Final performance of PPO-NFSP and PQN-NFSP against pre-trained PPO-NFSP evaluation opponent in BluffJAX games. Each algorithm is trained in self-play with $1e7$ samples and evaluated across 10 seeds. Evaluation opponent is trained in self-play with $5e6$ samples. Dashed lines indicate break-even performance.

trained in self-play for $5e6$ steps as an evaluation opponent. Across the 6 games tested, PPO-NFSP had higher average performance in 3 with higher margins relative to PQN. Notably, the results for Five Card Draw, Seven Card Stud, and Werewolf indicate that the baselines were able to gain a noticeable advantage in certain games despite their lack of game knowledge exploitation. However, both algorithms struggled to gain an advantage in Goofspiel, likely because performing well requires a stochastic policy. Finally, both algorithms failed to break even in Kemps and Bluff. It should be noted that, by virtue of their game mechanics, these are the two environments in BluffJAX which do not enforce a game-ending condition in finite time (i.e. games can end via reaching a horizon). Thus, low win rates in these games are due to draws in horizon-timeouts. Kemps introduces difficult mechanics: not only must agents learn to signal when they have built a hand, but their partner must also correctly register that signal - certainly a strong challenge in protocol formation for RL. Similarly, Bluff requires introduces a harsh ‘theory of mind’ challenge to adversarial RL, as agents must successfully predict when opponents are lying.

6 Conclusion

We introduce BluffJAX, an open-source suite of adversarial imperfect information games implemented in JAX for GPU-accelerated RL research, including five new games which have not been studied in RL thus far. We describe our API, design choices, and game implementations. We benchmark the performance of our implementations in terms of simulation throughput and memory usage in single and multi-GPU settings and compare against other libraries. We evaluate the performance of game-theoretic solvers, model-free RL, and model-based/tree search algorithms on our environments. We hope that BluffJAX will enable RL researchers to explore new challenges by providing fast and vectorized environments.

Limitations and Future Work. A key limitation is that BluffJAX is currently not equipped with PyTorch wrappers; therefore, it mostly caters to users already familiar with JAX and vectorized training. Additionally, the suite currently has limited performance evaluation tools. While solvers and baseline algorithms are provided for all games, a wider range of baselines would enable better standardization of results, such as external poker benchmarks. The sub-log-linear scaling of certain environments at higher parallelization (e.g. Kemps) suggests further optimizations can be made. Future plans include expanding the suite with environments such as Battleship, Mahjong, and Coup.

References

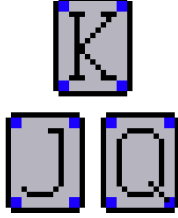
- John P Agapiou, Alexander Sasha Vezhnevets, Edgar A Duéñez-Guzmán, Jayd Matyas, Yiran Mao, Peter Sunehag, Raphael Köster, Udari Madhushani, Kavya Kopparapu, Ramona Comanescu, et al. Melting pot 2.0. *arXiv preprint arXiv:2211.13746*, 2022.
- Anton Bakhtin, Noam Brown, Emily Dinan, Gabriele Farina, Colin Flaherty, Daniel Fried, Andrew Goff, Jonathan Gray, Hengyuan Hu, et al. Human-level play in the game of diplomacy by combining language models with strategic reasoning. *Science*, 378(6624):1067–1074, 2022.
- David Balduzzi, Sebastien Racaniere, James Martens, Jakob Foerster, Karl Tuyls, and Thore Graepel. The mechanics of n-player differentiable games. In *International Conference on Machine Learning*, pages 354–363. PMLR, 2018.
- Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębniak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.
- Clément Bonnet, Daniel Luo, Donal Byrne, Shikha Surana, Sasha Abramowitz, Paul Duckworth, Vincent Coyette, Laurence I. Midgley, Elshadai Tegegn, Tristan Kalloniatis, Omayma Mahjoub, Matthew Macfarlane, Andries P. Smit, Nathan Grinsztajn, Raphael Boige, Cemlyn N. Waters, Mohamed A. Mimouni, Ulrich A. Mbou Sob, Ruan de Kock, Siddarth Singh, Daniel Furelos-Blanco, Victor Le, Arnau Pretorius, and Alexandre Laterre. Jumanji: a Diverse Suite of Scalable Reinforcement Learning Environments in JAX, March 2024. URL <http://arxiv.org/abs/2306.09884>. arXiv:2306.09884 [cs].
- Noam Brown and Tuomas Sandholm. Superhuman ai for multiplayer poker. *Science*, 365(6456): 885–890, 2019.
- Noam Brown, Tuomas Sandholm, and Strategic Machine. Libratus: The superhuman ai for no-limit poker. In *IJCAI*, pages 5226–5228, 2017.
- Noam Brown, Adam Lerer, Sam Gross, and Tuomas Sandholm. Deep counterfactual regret minimization. In *International conference on machine learning*, pages 793–802. PMLR, 2019.
- Noam Brown, Anton Bakhtin, Adam Lerer, and Qucheng Gong. Combining deep reinforcement learning and search for imperfect-information games. *Advances in neural information processing systems*, 33:17057–17069, 2020.
- Neil Burch, Marc Lanctot, Duane Szafron, and Richard Gibson. Efficient monte carlo counterfactual regret minimization in games with many player actions. *Advances in neural information processing systems*, 25, 2012.
- Murray Campbell, A Joseph Hoane Jr, and Feng-hsiung Hsu. Deep blue. *Artificial intelligence*, 134 (1-2):57–83, 2002.
- C Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax—a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021.
- Roy Frostig, Matthew James Johnson, and Chris Leary. Compiling machine learning programs via high-level tracing. In *SysML conference 2018*, 2019.
- Matteo Gallici, Mattie Fellows, Benjamin Ellis, Bartomeu Pou, Ivan Masmitja, Jakob Nicolaus Foerster, and Mario Martin. Simplifying deep temporal difference learning. *arXiv preprint arXiv:2407.04811*, 2024.
- Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*, 2016.
- Keigo Kawamura, Naoki Mizukami, and Yoshimasa Tsuruoka. Neural fictitious self-play in imperfect information games with many players. In *Workshop on Computer Games*, pages 61–74. Springer, 2017.

- Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- Sotetsu Koyamada, Shinri Okano, Soichiro Nishimori, Yu Murata, Keigo Habara, Haruka Kita, and Shin Ishii. Pgx: Hardware-Accelerated Parallel Game Simulators for Reinforcement Learning. *Advances in Neural Information Processing Systems*, 36:45716–45743, December 2023. URL https://papers.nips.cc/paper_files/paper/2023/hash/8f153093758af93861a74a1305dfdc18-Abstract-Datasets_and_Benchmarks.html.
- Harold W Kuhn. Extensive games and the problem of information. *Contributions to the Theory of Games*, 2(28):193–216, 1953.
- Karol Kurach, Anton Raichuk, Piotr Stańczyk, Michał Zajac, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, et al. Google research football: A novel reinforcement learning environment. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 4501–4510, 2020.
- Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, et al. Openspiel: A framework for reinforcement learning in games. *arXiv preprint arXiv:1908.09453*, 2019.
- Robert Tjarko Lange. gymmax, March 2024. URL <https://github.com/RobertTLange/gymmax>. [Online; accessed 3. Mar. 2026].
- Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.
- John F Nash Jr. Equilibrium points in n-person games. *Proceedings of the national academy of sciences*, 36(1):48–49, 1950.
- Constantin Ruhdorfer, Matteo Bortoletto, Anna Penzkofer, and Andreas Bulling. The overcooked generalisation challenge. *EWRL*, 2024.
- Alexander Rutherford, Benjamin Ellis, Matteo Gallici, Jonathan Cook, Andrei Lupu, Gardar Ingvarsson, Timon Willi, Ravi Hammond, Akbir Khan, Christian Schroeder de Witt, Alexandra Souly, Saptarashmi Bandyopadhyay, Mikayel Samvelyan, Minqi Jiang, Robert Tjarko Lange, Shimon Whiteson, Bruno Lacerda, Nick Hawes, Tim Rocktaschel, Chris Lu, and Jakob Nicolaus Foerster. JaxMARL: Multi-Agent RL Environments and Algorithms in JAX, November 2024. URL <http://arxiv.org/abs/2311.10090>. arXiv:2311.10090 [cs].
- Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.
- Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, 2020.
- John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.
- Lloyd S Shapley. Some topics in two-person games. Technical report, RAND Corporation, 1963.
- David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.
- David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017a.
- David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017b.

- Jordan Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34:15032–15043, 2021.
- Gerald Tesauro et al. Temporal difference learning and td-gammon. *Communications of the ACM*, 38(3):58–68, 1995.
- Tristan Tomilin, Luka van den Boogaard, Samuel Garcin, Bram Grooten, Meng Fang, Yali Du, and Mykola Pechenizkiy. MEAL: A Benchmark for Continual Multi-Agent Reinforcement Learning, September 2025. URL <http://arxiv.org/abs/2506.14990>. arXiv:2506.14990 [cs].
- Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *nature*, 575(7782):350–354, 2019.
- Daochen Zha, Kwei-Herng Lai, Yuanpu Cao, Songyi Huang, Ruzhe Wei, Junyu Guo, and Xia Hu. RLCard: A Toolkit for Reinforcement Learning in Card Games, February 2020. URL <http://arxiv.org/abs/1910.04376>. arXiv:1910.04376 [cs].
- Martin Zinkevich, Michael Johanson, Michael Bowling, and Carmelo Piccione. Regret minimization in games with incomplete information. *Advances in neural information processing systems*, 20, 2007.

A Games in BluffJAX

A.1 Kuhn Poker



Environment	Kuhn Poker
Number of players (n)	2
Observation size	9
Number of actions	2
Number of JAX branches	1
Metric	Exploitability
API Type	AEC

Objective: Kuhn Poker is a well-studied two-player zero-sum game [Kuhn, 1953]. There are 3 cards: Jack, Queen, and King. At the start of a round, players are randomly assigned one private card each, and each player antes 1. Each player then gets one chance to bet. If player 1 raises, player 2 can either call or fold. If player 1 checks, player 2 can check or raise. If player 2 raises after player 1 checked, player 1 can either call or fold. If either player folds, the other player wins the pot. If neither player folds, there is a showdown and the higher card wins.

Observations

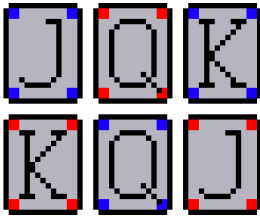
Input feature	Feature Dimension	Feature type
Agent's private card	3	One-hot
Self chips in pot	2	One-hot
Opponent chips in pot	2	One-hot
Player order (first or second to act)	2	One-hot
Total	9	

Actions

Action	Logit
Check/fold	0
Raise/call	1
Total	2

Rewards: Agents receive net chips gained as terminal rewards.

A.2 Leduc Hold'em



Environment	Leduc Hold'em
Number of players (n)	2
Observation size	36
Number of actions	3
Number of JAX branches	3
Metric	Exploitability
API Type	AEC

Objective: Two-player poker with a 6-card deck (two J, two Q, two K), one private card per player, and one public card revealed after round 1. Players bet over two rounds. Best hand (pair beats high card) wins at showdown, or a fold ends the hand early.

Observations

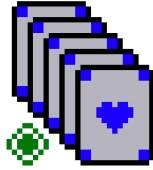
Input feature	Feature Dimension	Feature type
Agent private rank (J/Q/K)	3	One-hot
Public rank (if revealed)	3	One-hot
Self ante bucket (0–14+)	15	One-hot
Opponent ante bucket (0–14+)	15	One-hot
Total	36	

Actions

Action	Logit
Fold	0
Call/Check	1
Raise	2
Total	3

Rewards: Terminal reward is net chips relative to starting stack (non-terminal reward is 0).

A.3 Texas Limit Hold'em



Environment	Texas Limit Hold'em
Number of players (n)	2–10 (common setting)
Observation size	72 + n
Number of actions	4
Number of JAX branches	4
Metric	Chips/Hand
API Type	AEC

Objective: Multi-player limit hold'em with fixed betting amounts and blinds. Players maximize expected chip gain by betting/folding through preflop, flop, turn, and river.

Observations

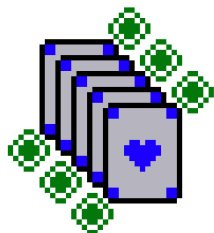
Input feature	Feature Dimension	Feature type
Private + visible public cards	52	Binary
Raise-count history (4 rounds, 0–4 raises each)	20	One-hot
Position relative to small blind	n	One-hot
Total	72 + n	

Actions

Action	Logit
Call	0
Raise	1
Fold	2
Check	3
Total	4

Rewards: Rewards are terminal chip payoffs normalized by big blind.

A.4 Texas No-Limit Hold'em



Environment	Texas No-Limit Hold'em
Number of players (n)	2-10 (common setting)
Observation size	54
Number of actions	5
Number of JAX branches	5
Metric	Chips/Hand
API Type	AEC

Objective: No-limit Hold'em variant with discrete raise abstractions (half-pot, pot, all-in). Agents optimize chip profit over each hand.

Observations

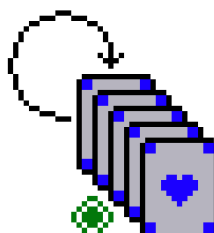
Input feature	Feature Dimension	Feature type
Private + visible public cards	52	Binary
Current player's chips in pot	1	Scalar
Max chips in pot among players	1	Scalar
Total	54	

Actions

Action	Logit
Check/Call	0
Raise half-pot	1
Raise pot	2
All-in	3
Fold	4
Total	5

Rewards: Rewards are terminal net chips gained (chips won minus chips invested).

A.5 5 Card Draw



Environment	5 Card Draw
Number of players (n)	2-10
Observation size	54
Number of actions	37
Number of JAX branches	11
Metric	Chips/Hand
API Type	AEC

Objective: Players play one draw-poker hand with a betting round, a draw phase, and a final betting round, maximizing expected chip gain.

Observations

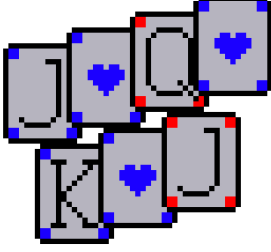
Input feature	Feature Dimension	Feature type
Current player 5-card hand (in 52-card index space)	52	Binary
Current player's chips in pot	1	Scalar
Max chips in pot among players	1	Scalar
Total	54	

Actions

Action	Logit
Check/Call, Raise 1/2 Pot, Raise Pot, All-in, Fold	0–4
Draw pattern (keep/discard over 5 cards)	5–36
Total	37

Rewards: Rewards are terminal net chips gained.

A.6 7 Card Stud



Environment	7 Card Stud
Number of players (n)	2–10
Observation size	$52 + 208(n - 1) + 25 + n$
Number of actions	4
Number of JAX branches	4
Metric	Chips/Hand
API Type	AEC

Objective: Stud poker with antes/bring-in and five betting streets. Players maximize expected chip return by betting and showing down strongest 7-card hand among active players.

Observations

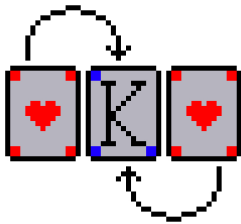
Input feature	Feature Dimension	Feature type
Own 7 cards (global deck index)	52	Binary
Opponents' visible upcards (4 slots each opponent)	$208(n - 1)$	One-hot blocks
Raise-count history (5 rounds, 0–4)	25	One-hot
Position relative to bring-in	n	One-hot
Total	$52 + 208(n - 1) + 25 + n$	

Actions

Action	Logit
Call	0
Raise	1
Fold	2
Check	3
Total	4

Rewards: Terminal payoff is net chips normalized by big bet.

A.7 Goofspiel



Environment	Goofspiel
Number of players (n)	2
Observation size	39
Number of actions	13
Number of JAX branches	1
Metric	Return/Hand (or Win Rate)
API Type	Parallel

Objective: Both players bid cards simultaneously for sequential prize cards. Highest unique bid wins prize points; ties discard the prize.

Observations

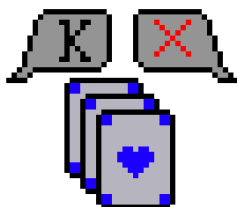
Input feature	Feature Dimension	Feature type
Current prize card	13	One-hot
Self cards already used	13	Binary
Opponent cards already used	13	Binary
Total	39	

Actions

Action	Logit
Play bid card (rank 1–13)	0–12
Total	13

Rewards: Per-round reward is prize value to the unique highest bidder; terminal return is cumulative points.

A.8 Bluff



Environment	Bluff
Number of players (n)	≥ 3
Observation size	$5 \cdot \text{deck_size} + 3$ (default 263)
Number of actions	13
Number of JAX branches	10
Metric	Win Rate
API Type	AEC

Objective: Turn-based social bluffing game (Cheat/I Doubt It). Players make claims, play hidden cards, and other players may challenge. Goal is to empty your hand first.

Observations

Input feature	Feature Dimension	Feature type
Pile claims	52	Thermometer
Pile size	52	Thermometer
Own hand counts	52	Thermometer
Claimant hand size (challenge phase)	52	Thermometer
Current claim size (challenge phase)	52	Thermometer
Phase id (claim/play/challenge)	3	One-hot
Total	$5 \cdot \text{deck_size} + 3$	

Actions

Action	Logit
Phase-dependent action index	0–12
Total	13

Rewards: Shaped rewards are used for successful/failed challenges and card outcomes; winner also receives terminal bonus.

A.9 Werewolf



Environment	Werewolf
Number of players (n)	6 (current implementation)
Observation size	$7 + 6n$
Number of actions	$n + 1$
Number of JAX branches	7
Metric	Win Rate
API Type	AEC

Objective: Hidden-role social deduction game with phases Night $\rightarrow$ Accuse $\rightarrow$ Vote. Humans try to eliminate all werewolves; werewolves try to reach parity with humans.

Observations

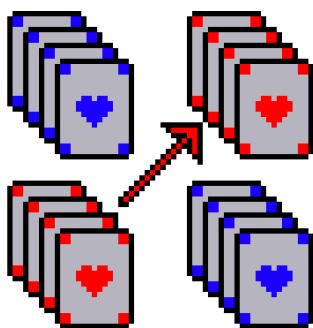
Input feature	Feature Dimension	Feature type
Own role	4	One-hot
Current phase	3	One-hot
Alive players (relative indexing)	n	Binary
Werewolf teammate info (if werewolf)	n	Binary
Seer belief/results (if seer)	$3n$	Ternary one-hot per player
Accusation targets this round	n	Binary
Total	$7 + 6n$	

Actions

Action	Logit
Relative target player index	$0 \dots n - 1$
No-op	n
Total	$n + 1$

Rewards: At terminal state, all winners get +10 and all losers get -10.

A.10 Kemps



Environment	Kemps
Number of players (n)	4, 6, 8, ... (even)
Observation size	$52(n + 1) + n \cdot \text{comm}$ (default deck)
Number of actions	$172 \cdot \text{comm}$
Number of JAX branches	1
Metric	Win Rate
API Type	Parallel

Objective: Partnership card game with simultaneous moves. Teams try to form four-of-a-kind and correctly declare KEMPS (or STOP KEMPS against opponents).

Observations

Input feature	Feature Dimension	Feature type
Private hand	52	Binary
Public center cards	52	Binary
Relative communication signals (all players)	$n \cdot \text{comm}$	One-hot
Total	$52(n + 1) + n \cdot \text{comm}$	

Actions

Action	Logit
Rank swap (lose-rank, gain-rank)	0 ... 168
No-op	169
Declare KEMPS	170
Declare STOP KEMPS	171
Each paired with communication signal	$\times \text{comm}$
Total	$172 \cdot \text{comm}$

Rewards: Terminal team reward is +1/-1 based on declaration correctness or correctness of spotting opposing team’s signal

B Design Choices

In this section, we hope to motivate some of the design choices we made in order to optimize BluffJAX for performance and usability.

B.1 Binary or One-Hot Features?

Our environments often contain features which are binary, e.g. whether the agent currently has the ability to play a certain action or not. We conduct a simple test to see whether a neural network is better able to distinguish features of this type when they are represented in binary format [0/1] or one-hot format [10/01]. We train a neural network on a randomized regression task with one input feature and two output features. We train the network on 20,000 examples over 10 epochs. In order to standardize the number of input neurons, we feed the binary features as [00/11] instead of [0/1].

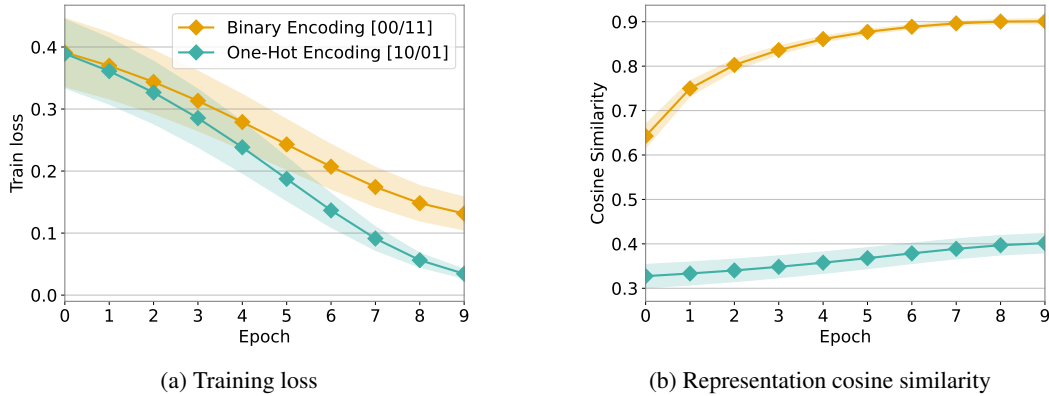


Figure 9: Training loss and cosine similarity of representations during training. Results show mean and std error over 10 seeds with 20,000 datapoints each.

Figure 9 shows the results; the training loss indicates that the network learns the regression task faster when the features are one-hot rather than binary. This makes intuitive sense, as one-hot features ‘separate’ the distinct input types, thereby saving the network from having to build an internal representation to partition ‘0’ inputs from ‘1’ inputs. However, Figure 10f demonstrates that the cosine similarity of the internal representations of the distinct inputs was *already* low at the start of training (inferring that they were already separated in feature space) and they actually become more similar over the course of training. This suggests that one-hot features can help learning when used, but their representational separation is not necessarily optimized for or maintained during learning.

B.2 Parallel or AEC API?

One consideration when designing a multi-agent benchmark is whether the environments are more amenable to a parallel agent-environment interaction interface or an agent-environment-cycle (AEC) interface. Similar to PettingZoo [Terry et al., 2021], we choose to implement both in BluffJAX. In

turn-based games where most steps only involve one agent, a parallel interface wastes significant compute by querying all other agents but forcing them to choose noop actions; on the other hand, certain environments like Goofspiel naturally follow a parallel-action game loop.

B.3 Handle resets internally or externally?

In parallelized settings, JAX requires all conditional logic to be executed regardless of whether the results are used or not⁴. Therefore, the typical RL ‘state→action→ next state’ loop must be accompanied at each instance by reset logic, and the correct ‘next state’ is chosen depending on the termination condition of the episode. The choice thus remains of whether to handle this logic inside the environment API, or to expose this logic for the user to handle. In BluffJAX, we choose to handle this logic internally, as users don’t typically need to know/worry about this, and exposing it simply adds to the confusion that befalls JAX newcomers.

B.4 Game logic datatypes: floats or ints?

This question arises as a consequence of conversions; internal game logic must often convert actions (which arrive as `jax.numpy.int32` in discrete settings), and the input to a neural network must typically be of the type `jax.numpy.float32`. Therefore, which datatype should be used for internal game logic? Does the conversion cost matter? We find that the cost is generally **negligible** within the overall training loop, and therefore we simply use whichever datatype is most convenient in each case.

B.5 Done, absorbing, truncated, terminated?

Environment suites do not generally agree on a convention on how to distinguish when agents reach absorbing states or reach a horizon. We choose to implement both: our API provides a boolean array `absorbing` to indicate the absorbing state of each agent (as some may drop out of a game earlier than others) as well as a single boolean `done` if all agents have reached absorbing states or if the global episode horizon has been reached.

B.6 Type hints?

What type hints should be given to users to understand the code? We find no universally consistent typing scheme for JAX functions and arrays. We therefore implement our own type hints that we hope are useful in distinguishing normal Python types from JAX arrays and pytrees.

B.7 Randomize the start player?

In self-play settings, this question has no bearing since the same network will be trained on all agent data. However, if distinct players are loaded into a game (e.g. if a training model is loaded as ‘player 1’ and a baseline model as ‘player 2’, then player order absolutely makes a difference, and thus we randomize the starting player in all cases for simplicity.

B.8 Give the agent an observation of its own order?

We find that this feature either does not help or actively harms learning in practice, as it causes agents to overfit to this feature rather than generalize across experiences while agnostic to the player they originated from. We therefore omit this feature in all games except where it explicitly matters (e.g. whether you are the start player or second player in Kuhn and Leduc Poker matters).

C Details on Experiments

In this section, we detail the hyperparameters and training settings for the experiments in the main paper.

⁴https://docs.jax.dev/en/latest/_autosummary/jax.lax.cond.html

C.1 CFR and Deep CFR

Our results for CFR were run with 100 iterations over the whole game tree for both games. Our results for Deep CFR were run with 50 iterations and 100 traversals per iteration. The policy network and advantage networks used in both cases are fully connected MLPs with 2 hidden layers, 64 hidden units/layer, ReLu activations, and LayerNorm after each hidden layer. The networks were trained with 200 advantage training steps and a memory capacity of 100,000. All networks were trained with the Adam [Kingma and Ba, 2014] optimizer. We ran 10 seeds per game for Deep CFR.

C.2 PPO, PQN, PPO-NFSP, PQN NFSP

We benchmarked PPO and PPO-NFSP in all environments using the following settings for the value network and policy networks of PPO: fully connected MLPs with 2 hidden layers, 128 hidden units/layer, ReLu activations. Table 3 details the hyperparameters and training settings for the PPO actor and value networks.

Table 3: PPO hyperparameters.

Hyperparameter	Value
num_timesteps	$1e7$
num_envs	1024
num_steps_per_env_per_update	64
num_epochs	4
num_minibatches	4
lr	3×10^{-5}
anneal_lr	True
gamma	0.99
gae_lambda	0.95
clip_eps	0.2
vf_clip	0.2
ent_coef	0.0
vf_coef	1.0
max_grad_norm	0.5
optimizer	adam

We also benchmarked PQN and PQN-NFSP in all environments. Here are the settings for the PQN networks in each case: fully connected MLPs with 2 hidden layers, 128 hidden units/layer, LayerNorm after each layer, ReLu activations. Table 4 details the hyperparameters and training settings for the PQN Q-network.

In experiments with PPO-NFSP and PQN-NFSP, we use the settings above for the best response PPO/PQN agents. For the NFSP anticipatory (supervised) networks, we use the following settings: 2 hidden layers, 128 hidden units/layer, ReLu activations. Table 5 details the hyperparameters for the supervised network.

C.3 ReBeL

Our experiments with ReBeL had the following settings for the value network in each case: fully connected MLP with 2 hidden layers, 128 hidden units/layer, ReLu activations. Table 6 details the hyperparameters and training settings for ReBeL.

D Head-to-Head Evaluations

Table 4: PQN hyperparameters.

Hyperparameter	Value
num_timesteps	$1e7$
num_envs	1024
num_steps_per_env_per_update	64
num_epochs	2
num_minibatches	2
lr	2.5×10^{-4}
start_e	1.0
end_e	0.05
exploration_fraction	0.2
q_lambda	0.9
max_grad_norm	0.5
gamma	0.99
fc_dim_size	128
anticipatory_eta	0.1
sl_reservoir_capacity	100000
sl_batch_size	256
sl_num_steps_per_update	2
sl_lr	3×10^{-4}
optimizer	adam

Table 5: NFSP Anticipatory Network hyperparameters.

Hyperparameter	Value
anticipatory_eta	0.1
sl_reservoir_capacity	100000
sl_batch_size	256
sl_num_steps_per_update	2
sl_lr	3×10^{-4}
optimizer	adam

Table 6: ReBeL hyperparameters.

Hyperparameter	Value
num_update_steps	1000
num_subgames_per_update	2
cfr_iterations	2
max_depth	4
random_action_prob	0.2
value_lr	3×10^{-4}
replay_capacity	10000
batch_size	256
num_value_train_steps	4

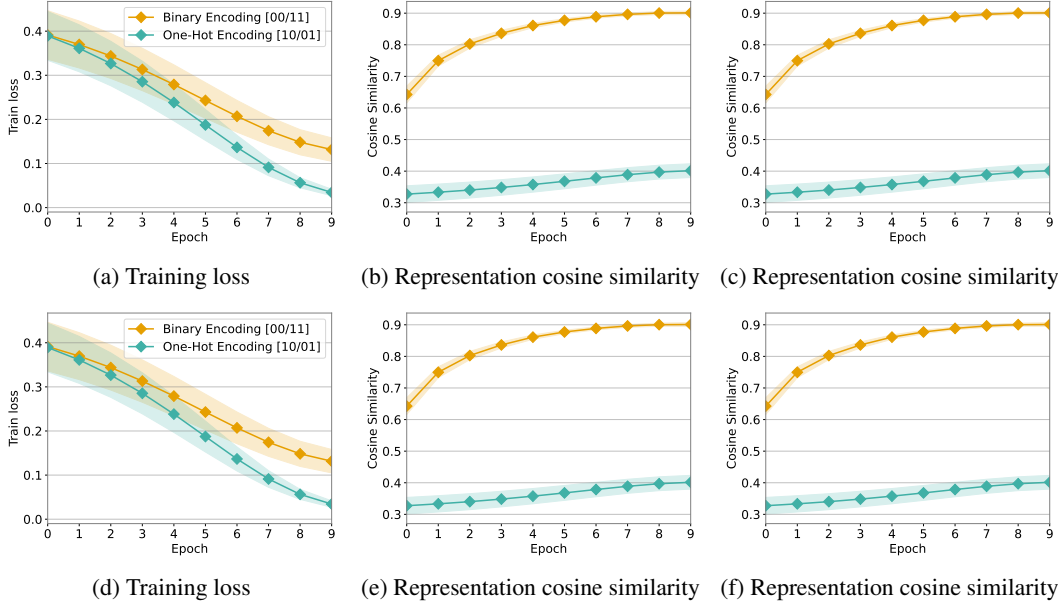


Figure 10: Training dynamics across twelve settings. Top row: training loss curves; bottom row: representation cosine similarity over training.

NeurIPS Paper Checklist

1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper’s contributions and scope?

Answer: [Yes]

Justification: Our claims include: the open-source implementation of adversarial imperfect information games (supported by the code provided), high performance throughput on GPU devices (evidenced by Section 4), and baseline algorithmic results for future comparisons (evidenced by Section 5).

Guidelines:

- The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
- The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
- The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
- It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [Yes]

Justification: We conclude our submission with a discussion on some shortcomings of our library with regards to usability, algorithmic benchmarking, and optimization.

Guidelines:

- The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate “Limitations” section in their paper.

- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. Theory assumptions and proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: Our work does not include novel theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. Experimental result reproducibility

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: Appendix C of our work includes the hyperparameters and training settings required to replicate the experimental algorithmic performance findings in our paper. Our benchmarking performance results are replicable provided a user has access to the hardware described in Section 4.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.

- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general, releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
 - (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
 - (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
 - (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
 - (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. Open access to data and code

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [\[Yes\]](#)

Justification: Our submission includes a link to a public anonymous Github (<https://github.com/bluffjax/bluffjax>) which contains didactic examples and tutorials. We also submit our code as supplementary material.

Guidelines:

- The answer [\[N/A\]](#) means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (<https://neurips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [\[No\]](#) is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (<https://neurips.cc/public/guides/CodeSubmissionPolicy>) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. Experimental setting/details

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [\[Yes\]](#)

Justification: Our training and test details are shared in Appendix C and within the attached code.

Guidelines:

- The answer [\[N/A\]](#) means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. Experiment statistical significance

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [\[Yes\]](#)

Justification: Our performance and algorithmic benchmark results include distributional information as standard errors over multiple seeds. Note that these bounds are not visible on the scale of performance benchmarking.

Guidelines:

- The answer [\[N/A\]](#) means that the paper does not include experiments.
- The authors should answer [\[Yes\]](#) if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. Experiments compute resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [\[Yes\]](#)

Justification: The exact GPU and CPU setups and memory requirements are discussed in the paper.

Guidelines:

- The answer [\[N/A\]](#) means that the paper does not include experiments.
- The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.

- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. Code of ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics <https://neurips.cc/public/EthicsGuidelines>?

Answer: [Yes]

Justification: Our work conforms to the Code of Ethics provided. We should note that as a benchmarking tool centered around games with hidden information, our library may be used for the training of AI agents with the explicit goal of deception. We hereby state that BluffJAX is solely intended as a research tool and should not be used for enabling deceptive behavior in the real world or for-profit applications.

Guidelines:

- The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
- If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. Broader impacts

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [Yes]

Justification: The previous checklist point discusses potential misuse of our library which may result in negative societal impact.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. Safeguards

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [\[Yes\]](#)

Justification: In its current state, BluffJAX is primarily suitable for research purposes and is designed to not be useful for any potential harmful applications as discussed previously.

Guidelines:

- The answer [\[N/A\]](#) means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [\[Yes\]](#)

Justification: BluffJAX is an original, open-source software library under the MIT license. No external code, data, or models were used in the creation of this work.

Guidelines:

- The answer [\[N/A\]](#) means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, paperswithcode.com/datasets has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. New assets

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [\[Yes\]](#)

Justification: Our open-source library is provided with appropriate documentation for ease of training, benchmarking, and extensibility.

Guidelines:

- The answer [\[N/A\]](#) means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. Crowdsourcing and research with human subjects

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: Our work does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. Institutional review board (IRB) approvals or equivalent for research with human subjects

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: Our work does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. Declaration of LLM usage

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: The core development of this library does not involve LLMs for any novel, important, or non-standard components.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.