AlphaExploitem: Going Beyond the Nash Equilibrium in Poker by Learning to Exploit Suboptimal Play

Vlad Murgoci
Delft University of Technology
2628 CD Delft, The Netherlands
v.murgoci@tudelft.nl

Matthijs Spaan
Department of Intelligent Systems
Delft University of Technology
2628 CD Delft, The Netherlands
M.T.J.Spaan@tudelft.nl

Yaniv Oren
Department of Intelligent Systems
Delft University of Technology
2628 CD Delft, The Netherlands
y.oren@tudelft.nl

Abstract

Poker is an imperfect information game that has served as a long-standing benchmark for decision-making under uncertainty. To maximize utility beyond the Nash equilibrium, an agent can deviate from Nash-equilibrium policies to exploit suboptimal play. We introduce **AlphaExploitem**, which extends the competitive RL poker agent AlphaHoldem by using a hierarchical transformer encoder that enables reasoning over previously played hands and modifying the training procedure with the inclusion of a diverse pool of exploitable opponents to facilitate learning to exploit. We train and evaluate AlphaExploitem on two standard benchmarks for imperfect-information games. Empirically, AlphaExploitem successfully exploits weak play by *both in- and out-of-distribution opponents*, without losing performance against NE opponents.

1 Introduction

Poker is a family of card games whose incomplete information setting, stochastic elements, and deep strategic complexity have made it a long-standing challenge for artificial intelligence [Rubin and Watson, 2011]. Unlike perfect information games such as chess and Go, poker requires agents to reason under uncertainty about hidden cards while navigating a vast strategic decision space spanning multiple betting rounds [Brown and Sandholm, 2019].

State-of-the-art poker agents approximate the Nash Equilibrium (NE): strategies from which no player can improve by unilateral deviation [Nash, 1950]. Systems such as DeepStack [Moravcík et al., 2017], Libratus [Brown and Sandholm, 2018], Pluribus [Brown and Sandholm, 2019], and AlphaHoldem [Zhao et al., 2022] have achieved high performance in various poker formats by learning low-exploitability, approximately-NE strategies. However, while NE guarantees a non-negative expected return against any opponent, it does not maximize returns against *out-of-equilibrium*

Preprint.

play. Against suboptimal opponents, substantial additional expected value (EV) can be captured by deviating from the equilibrium to exploit observed flaws in the opponent's strategy.

Competitive poker is rarely played as a single *hand*, however. A hand is one full deal of cards from shuffling the deck to determining the winner of the wagered money, i.e., a single phase of the entire game. Sessions may consist of hundreds to thousands of hands between the same opponents. This creates a repeated-interaction structure that proficient players often leverage [Frey et al., 2018], adapting their strategy in response to patterns observed in an opponent's play. However, state-of-the-art NE-approximating agents treat every hand as an independent encounter: their decisions depend only on the current hand's cards and betting sequence, discarding any additional information available from the session so far.

Prior work on exploiting suboptimal poker opponents has largely relied on hand-crafted summary features of opponent behavior: conditional strategies based on aggregated game statistics [Billings et al., 1998, 2002, Xu et al., 2021], recurrent networks consuming pre-computed summary features [Li and Miikkulainen, 2017, 2018a,b]. Li et al. [2024] train a causal transformer via algorithm distillation [Laskin et al., 2022]. Although similar in architecture, the method has been criticized for its training technique that does not guarantee or evaluate on human and rational flawed strategies.

In this work, we introduce **AlphaExploitem**, an actor-critic agent trained through Proximal Policy Optimization (PPO) [Schulman et al., 2017] for which trajectories are generated via *K-best-league* self-play [Zhao et al., 2022], a *long-tail buffer* of past iterations of the main agent, and a set of hand-crafted exploitable opponents. To exploit weak play, AlphaExploitem extends AlphaHoldem's [Zhao et al., 2022] architecture with a hierarchical transformer encoder that processes the raw tokenized history of previously played hands within a session. By using separate learned embedding tables for different types of observations, we provide complete tokenized information over the opponent's behavior and the environment's dynamics, delegating the intrinsic task of processing game data to the model itself rather than to human-built feature aggregators.

We evaluate on two standard imperfect-information benchmarks, Kuhn Poker [Kuhn, 1950] and Leduc Hold'em [Southey et al., 2012]. AlphaExploitem captures more than twice the per-hand EV of the AlphaHoldem-style league-only baseline, while remaining close to the NE and roughly twice the EV of the same trained policy without access to hand histories. In addition, we observe that AlphaExploitem generalizes effectively to unseen exploitable opponents, exploiting out-of-distribution agents as well as in distribution. A reproducible evaluation ensemble is also provided in Appendix C, and to the best of our knowledge, this is the first work that provides means of comparison for future research on exploiting suboptimal opponents in poker.

Through this work, we hope to further research and comparison benchmarks on exploitative play in poker, and more broadly on learning how to leverage complete observable information over weak playstyles in repeated interactions in order to maximize returns. Since poker is mainly played between human players, our research can be used as a foundation for building tools that can help players find weaknesses in their game and as well as in their opponents' strategies, and to train towards effectively adapting.

2 Background

Poker is a card game which usually involves *hidden* private cards, sometimes *shared* community cards, and rounds of *betting* with chips (in-game money). A player wins either by having the best hand at *showdown* or by making all others fold (give up) their hand. Because cards are hidden, *bluffing* is an important strategic element. Poker is usually formalized through two complementary lenses: as a *partially observable decision problem* from a single player's perspective, and as a multi-agent strategic interaction problem.

We refer to the player we're interested in as the *hero* while poker convention denotes the other competing players as *villains*. Each hand begins with two forced pre-deal wagers, the *small blind* (SB) and the *big blind* (BB), which seed the central wager pool — the *pot*. The main utility metric is *big blinds per hand* (BBs/hand) where one big blind is the standard unit of wager. At each decision the active player chooses among four canonical actions: *fold* (forfeit the hand), *check* (decline to wager when no bet is pending), *call* (match a pending bet), and *bet/raise* (commit chips beyond what is already wagered). The hand ends either when one player folds — at which point the other

wins the pot uncontested — or at showdown, when the surviving players reveal their *hole cards* and the highest-ranked hand wins. Players are commonly described in terms of stylistic *archetypes* that summarize their tendencies across many hands. We use these archetypes both as training opponents and as a mental model for the kind of cross-hand information our agent is meant to exploit. More details about player archetypes can be found in Appendix B

Imperfect information games. We adopt standard extensive-form game notation: each player i has a behavioral strategy σ_i mapping their *information sets* — decision points the player cannot distinguish — to distributions over legal actions, with terminal utility u_i, and a strategy profile $\sigma = (\sigma_i)_{i \in N}$ collects all players' policies. Heads-up poker is the two-player zero-sum special case ($u_1 = -u_2$), which makes the equilibrium and exploitability concepts of the next paragraph well-defined and, for Kuhn and Leduc poker, exactly computable.

In poker, each information set $I \in \mathcal{I}_i$ corresponds to a decision node at which the acting player knows the public game history and their own private cards but cannot distinguish between histories that differ only in the opponent's private cards. Heads-up poker instantiates the two-player zero-sum setting ($N = \{1, 2\}$, $u_1(z) = -u_2(z)$), which makes equilibrium and exploitability — introduced next — well-defined and finite-dimensionally computable for the environments we study.

Nash Equilibrium and Exploitability. A Nash Equilibrium [Nash, 1950] is a strategy profile σ^* from which no player can improve their expected payoff by unilateral deviation: $u_i(\sigma^*) \geq \max_{\sigma_i'} u_i(\sigma_i', \sigma_{-i}^*)$ for all i. Exploitability [Nikaidô and Isoda, 1955] measures how much an opponent can gain by deviating against a given strategy to maximize utility. NE has zero exploitability. While NE strategies are safe, they sacrifice the additional utility available against opponents who are not at the equilibrium.

Modeling Poker as a partially observable Markov Decision Process Heads-up (two-player) poker is naturally a *partially observable stochastic game*: both players see the public betting sequence and any community cards, but each player's hole cards are hidden from the other.

Fixing a stationary opponent policy reduces this to a single-agent *partially observable Markov decision process* for the hero, with two consequences for the rest of the paper. The resulting transition and observation distributions are *opponent-specific*, so an agent that conditions on observed opponent patterns is effectively learning a per-opponent transition model.

AlphaHoldem. AlphaHoldem [Zhao et al., 2022] is a competitive end-to-end RL poker agent that reaches CFR-level play [Moravcík et al., 2017] via two design choices we inherit: architecture and training technique. The authors employ self-play training with PPO through a *K-best league*, a fixed-size pool of past checkpoints where past iterations that achieve higher ELOs survive in the league, while low-performing agents are eliminated. AlphaHoldem uses a pseudo-siamese architecture: two separate networks process information about cards and player actions of the current poker hand reflecting the fact that these observations capture different aspects of the game. The output of the two networks is fused only at the policy and value heads.

We choose AlphaHoldem as our foundation because it is a competitive end-to-end RL agent that reaches CFR-level play at a fraction of the compute. Crucially, its architecture cleanly separates the card and action streams and fuses them only at the policy and value heads, leaving a natural extension point for a cross-hand module as a third input stream.

3 Related Work

RL for Poker. Other additional techniques complementing the approaches discussed in Section 1 used for NE play in Poker are: Neural Fictitious Self-Play (NFSP) [Heinrich and Silver, 2016] which combines DQN-based best-response learning with supervised average-strategy regression to approach Nash equilibrium, Deep CFR [Brown et al., 2019] and its model-free successor DREAM [Steinberger et al., 2020] replace tabular regret with deep networks, scaling regret minimization beyond hand-crafted abstractions while ReBeL [Brown et al., 2020] further couples deep RL with search to attain strong play in large imperfect-information games. These systems aim for low exploitability rather than opponent-specific adaptation.

Exploiting suboptimal opponents. Work on exploiting suboptimal poker opponents can be broadly grouped by how opponent behavior is represented. Early approaches relied on hand-crafted statistics

accumulated over the course of play and passed them to conditional decision-making modules, including expert systems and feed-forward networks [Billings et al., 1998, 2002, Hoehn et al., 2005, Xu et al., 2021]. More recent work replaced these manually designed pipelines with learned sequence models, such as recurrent architectures, but still operated on per-hand summaries or cumulative behavioral statistics rather than raw interaction histories [Li and Miikkulainen, 2017, 2018a,b]. In contrast, [Li et al., 2024] processes raw hand histories directly with a causal transformer, avoiding explicit feature engineering. However, its training and evaluation protocol has been questioned for not clearly ensuring exposure to a sufficiently diverse set of weak opponents.

LLM-based approaches. PokerGPT [Huang et al., 2024] leverages large language models by casting the poker state as natural language prompts, a methodologically distinct direction from the specialized numerical encoders used by our work and by the AlphaHoldem family. Current results indicate that LLMs struggle to match the performance of dedicated architectures especially against NE [Provost et al., 2026].

4 AlphaExploitem - Transformer-based Exploitation

AlphaExploitem extends AlphaHoldem to enable cross-hand opponent-specific adaptation without abandoning the inductive biases that make the base architecture compute-efficient. The design follows three ideas. First, we **preserve AlphaHoldem's architecture** for current-hand information in order to preserve the already proven efficiency over singular hand decisions. Second, we **separate responsibilities between streams**: the transformer sees only tokens from previously completed hands, while decisions within the current hand remain the responsibility of the original modules, thereby following the principle of dividing different types of observations to different network modules. These principles yield an architecture in which the benefit of cross-hand context can be isolated (by fully masking the transformer input during inference). Third, we **enhance** the existing K-best league training technique with training against a fixed set of hand-crafted flawed policies, and a dynamic collection of past checkpoints of the agent itself, to ensure that the agent is exposed to a wide variety of exploitable patterns during training.

4.1 Encoding Poker Games

For each decision, AlphaExploitem assembles three input tensors from the running game state: the cards available to the agent, the action sequence of the current hand, and a data structure containing all previously completed hands against the current opponent.

Cards and Current-hand actions We replicate AlphaHoldem's encoding of card information and action information as separate one-hot encoded arrays.

Cross-hand history. The history stream is a sequence of tokens recording every observable event from previously completed hands: hero actions, opponent actions, and revealed hole and community cards. Each token carries a type tag (agent action, opponent action, private card, community card, opponent card), so that the encoder can route different token types through separate embedding tables. We choose to exclude reward signals from the history stream as they can be inferred from the sequence of actions and cards. In contrast to prior work, we do not aggregate hand information.

4.2 Architecture

Figure 1 summarizes the overall architecture.

Current-hand encoders. The card and action tensors are each projected through a fully-connected layer. We retain AlphaHoldem's separation of card and action processing to preserve representational separation.

Hierarchical history transformer. We encode the history stream with a hierarchical transformer [Vaswani et al., 2017] that mirrors poker's natural temporal granularity. A **within-hand encoder** is applied independently to each past hand: it embeds the hand's tokens via type-specific embedding tables, runs self-attention over them, and pools the result to a single summary vector h_i. An **across-hand encoder** then processes the resulting sequence of summaries $\{h_1, \ldots, h_M\}$ to produce a session-level context vector z. In contrast to prior work, we delegate the task of summarizing hand information to the model itself rather than relying on hand-crafted statistics.

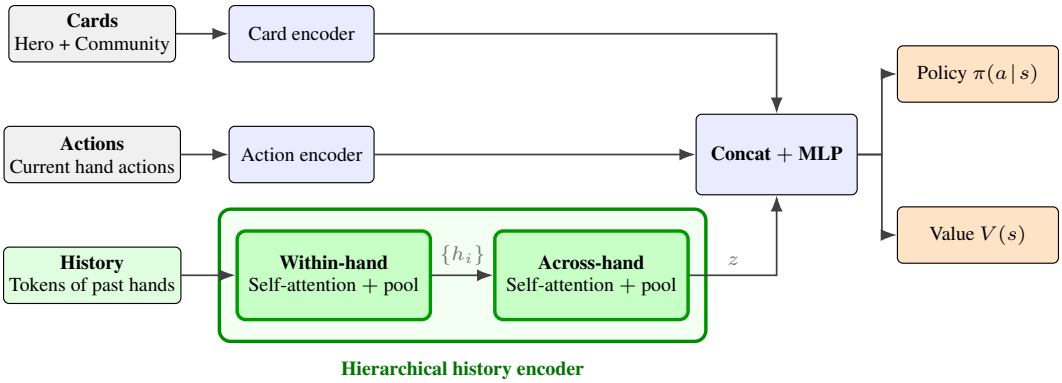

Figure 1: **AlphaExploitem architecture (left → right).** Three input streams—the cards available to the hero, the current hand's action history, and a tokenized record of all past hands against the same opponent—are fused via a shared MLP that splits into policy and value heads. Our architectural contribution is the hierarchical history encoder (highlighted): a within-hand transformer summarizes each completed past hand to a vector h_i, and an across-hand transformer attends over the per-hand summaries to produce a session-level context z.

Current-hand exclusion from the transformer. The transformer is restricted to tokens from hands already completed. The current hand's card and action information is handled exclusively by the current-hand encoders. This separation serves two purposes. First, it produces a structural decomposition — the transformer attends to *cross-hand* information while the dedicated card and action encoders handle *in-hand* reasoning. Second, it prevents the transformer from becoming the primary decision-maker within a single hand.

Fusion and output heads. The three encoded streams are concatenated, layer-normalized, and passed through a shared hidden network. The representation then splits into an actor head producing an action distribution via softmax and a critic head producing a scalar value estimate.

4.3 Training

During each training epoch, we generate trajectories of experience from 3 opponent pools: the league, the flawed toy policies, and the dynamic buffer of past versions of the agent. To generate training minibatches, we uniformly sample from the combined pool of trajectories and apply PPO updates to the agent's parameters.

Adapted K-best league self-play. Based on AlphaHoldem [Zhao et al., 2022], we train via self-play against opponents sampled from a fixed-size league pool. The league maintains ELO ratings [Elo, 1978] updated based on head-to-head match outcomes, weighted by the winning margin. At checkpoint intervals, the current agent competes against all league members. If its ELO exceeds the lowest-rated member, it replaces that member. This framework provides a diverse and progressively stronger set of training opponents and drives convergence toward a robust baseline policy.

Exposure to flawed policies. In addition to the league, we add trajectories generated against a fixed ensemble of hand-crafted weak policies that cover common human-like archetypes. The set of training toys has been crafted with domain expertise to cover a range of exploitable patterns, such as over-aggression or over-conservatism. We take the standard assumption that flawed opponents do not adapt to the agent's behavior, following a stationary policy. While this training setup has been used before [Wu et al., 2021], its combination with a transformer conditioned on session-level history is what enables the agent to learn *adaptive* exploitation.

Long-tail buffer self-play. In order to enable the agent to generalize to diverse policies, we use a long-tail buffer of previous checkpoints of the agent. Unlike the toy set, where opponents have stationary policies, the snapshot opponent is a frozen past version of the main agent that does have access to its own cross-hand history channel. As the agent cycles through different strategy profiles throughout training, playing against past, possibly flawed iterations exposes the main agent to various

play patterns that are not contained within the set of hand-crafted policies. The buffer acts as a bridge between the toy set and the league, providing a dynamic source of diverse opponents that possibly present various exploitable patterns and counter-adaptive pressure without the risk of overfitting.

5 Experiments

AlphaExploitem is evaluated on Leduc and Kuhn Poker against diverse opponent archetypes. Our results support four claims: (i) the agent exploits in-distribution suboptimal opponents, with gains attributable to cross-hand context; (ii) the exploitation demonstrates generalization to out-of-distribution opponents; (iii) the agent identifies strong policies and play accordingly against them; (iv) the exploitative behavior is indeed caused by the cross-hand encoder.

5.1 Experimental setup

AlphaExploitem is implemented in JAX [Bradbury et al., 2018] using Flax and Optax. The Kuhn and Leduc environments are provided by PGX [Koyamada et al., 2024], a JAX-native library of game simulators enabling fully vectorized, GPU-accelerated trajectory generation. Training uses standard PPO [Schulman et al., 2017] with AdamW, gradient clipping, KL-based early stopping. Full architectural and hyperparameter details are given in Appendix F.

Each trained agent is evaluated in two modes: an exploiter mode with the full tokenized session history provided to the transformer, and a non-exploiter mode with the history masked so only the current-hand card and action encoders inform decisions. Rewards are reported in big-blinds per hand (BBs/hand). Two opponent pools are used for evaluation: the *in-distribution* toy strategies seen during training and a held-out *out-of-distribution* set of diverse hand-crafted policies. Information about individual toy policies can be found in Appendix C.

5.2 Training dynamics: exploitation ability

Our main claim is that the cross-hand encoder confers the main agent with the ability to extract additional EV (compared to the baseline) from exploitable opponents and that this advantage translates as well to the out-of-distribution (OOD) set. We compare the main agent against two no-encoder baselines on both Kuhn and Leduc Hold'em: a pure league-self-play baseline (the AlphaHoldem-style setup with no toys and no snapshot buffer) and an ablation that isolates the contribution of the encoder by training the original AlphaHoldem architecture (no cross-hand encoder) using the new training paradigm.

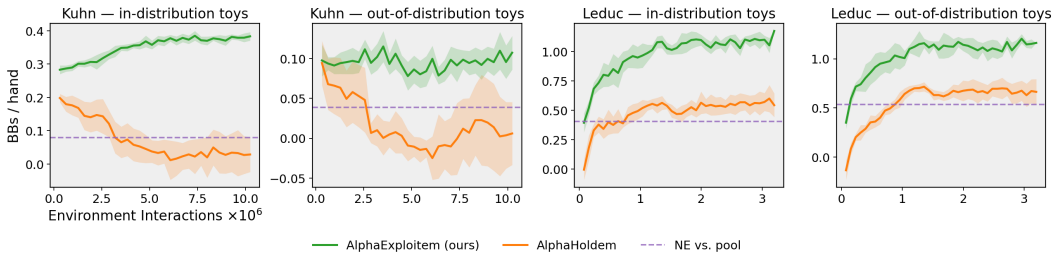

Figure 2: Reward evolution against the in-distribution and out-of-distribution toy pools, on Kuhn Poker (left two panels) and Leduc Hold'em (right two panels). 95% confidence intervals with 8 seeds. Horizontal axis: environment interactions (hands of play). The dashed line is the mean reward NE obtains against the corresponding pool (Appendix D.3).

Figure 2 reports the seed-mean reward/hand against the in-distribution toy pool and a held-out out-of-distribution pool. On Leduc the main agent dominates on both pools: it reaches an in-distribution (ID) reward of roughly $+1.0$ BBs/hand vs. $+0.5$ for AlphaHoldem. The OOD ordering is identical. The ID and OOD curves track one another closely throughout, indicating that the encoder learns class-level opponent features that transfer to held-out variants rather than overfitting per-toy fingerprints. For

Kuhn the in-distribution evolution shows the same qualitative pattern as Leduc: the main agent climbs to roughly $+0.38$ BBs/hand while AlphaHoldem stays close to zero ($\approx +0.03$). The Kuhn OOD raw-reward curve, however, looks visually flat, which would naively suggest no improvement on held-out opponents. This impression is misleading: per-toy reward divided by the toy's best-response (BR) ceiling rises monotonically with training (Figure 14 in Appendix E). The discrepancy is a result of unit choice in the averaging: a small absolute BB gain against a low-BR-ceiling toy registers strongly in BR-fraction terms but contributes negligibly to the chip-denominated mean, so improvements on the most exploitable Kuhn OOD toys are visible only after the per-toy normalisation.

5.3 Baseline play vs. Nash

We verify that the exploitation gains do not come at the cost of fundamental strength: a competent poker agent should hold its own against a Nash-equilibrium policy rather than be exploited by it. Figure 3 reports per-checkpoint mean BBs/hand against the (computed) Nash equilibrium.

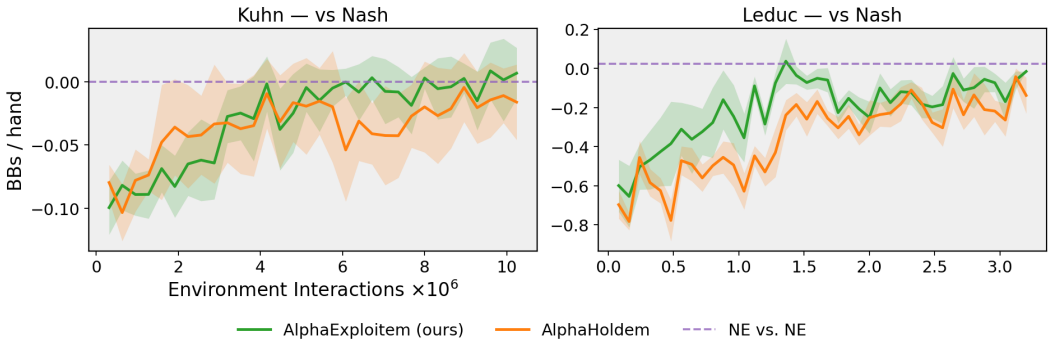

Figure 3: Reward / hand vs. the Nash equilibrium policy on Kuhn (left) and Leduc Hold'em (right). 95% confidence over 8 seeds. The dashed zero-line is the NE-vs-NE expectation, position-averaged.

On Leduc, both the baseline and AlphaExploitem improve steadily over training and converge close to the NE expectation. None of the two models reaches exact NE, but the gap shrinks monotonically with training and is meaningfully smaller than the gap to weak toy opponents.

5.4 Per-opponent results

The pool-averaged curves of Section 5.2 do not show whether a single dominant opponent is responsible for the gap. Figures 4, 5, 6, 7 decompose the rewards toy by toy, for both Kuhn and Leduc. The Appendix E variants (Figures 9, 10, 11, 12) normalize the rewards by their Best Response (BR) ceiling in order to reveal exploit *efficiency* rather than absolute reward.

The encoder advantage is broad rather than concentrated, and visible on both games. AlphaExploitem achieves better rewards on all toys. On Leduc the largest absolute gains line up with the most exploitable opponents — maniac, loose aggressive (LAG), and the OOD bluffer toys. Cautious opponents like `nit` and `rock` leave little room to extract reward and the three groups bunch together near zero, again as expected from their small BR ceilings.

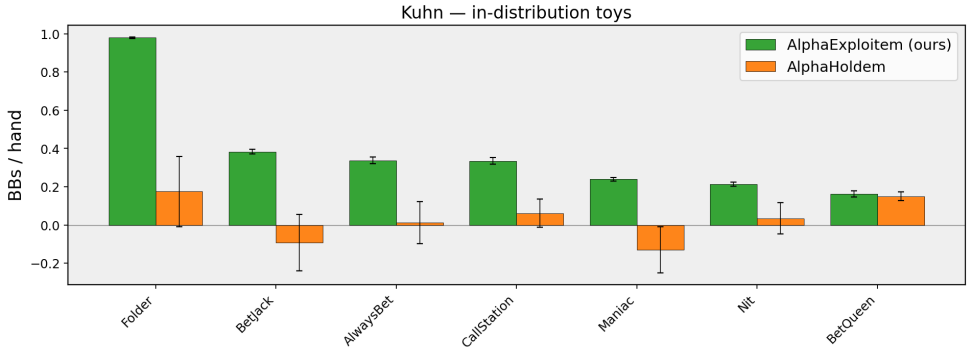

Figure 4: Kuhn in-distribution final-checkpoint reward/hand against each toy opponent. Bars are seed-means over the last 5 logged checkpoints with 95% confidence error bars. N = 8 seeds per group. Toys are sorted left-to-right by descending average reward.

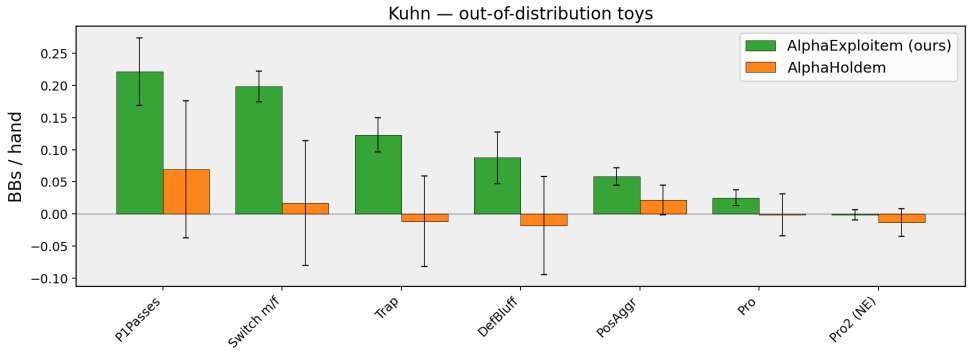

Figure 5: Kuhn — out-of-distribution toys. Same conventions as Figure 4

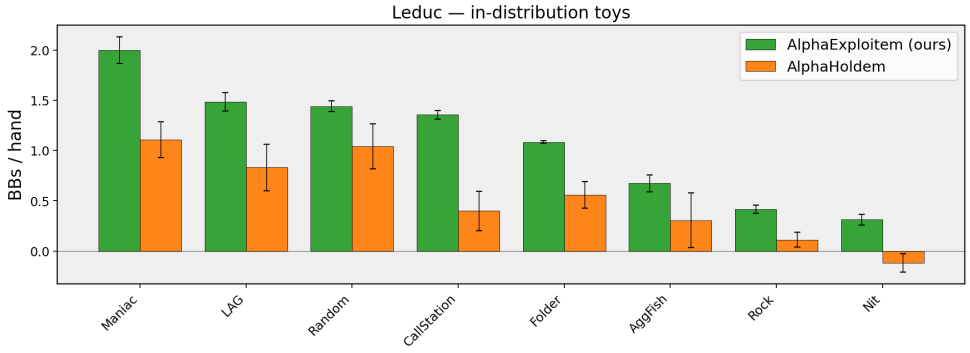

Figure 6: Leduc — in-distribution toys. Same conventions as Figure 4

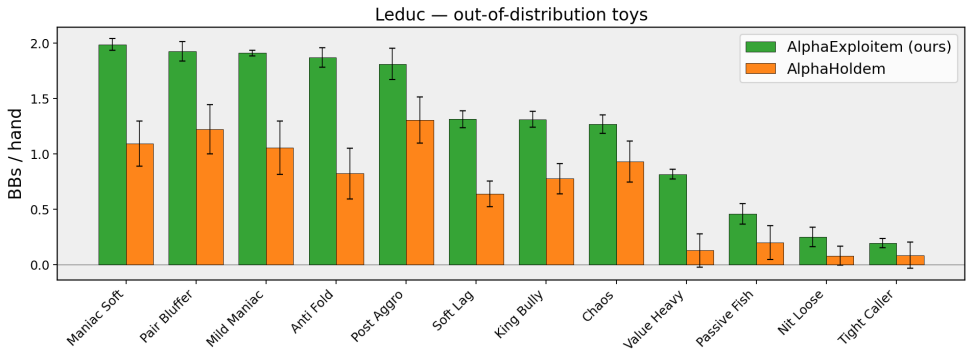

Figure 7: Leduc — out-of-distribution toys. Same conventions as Figure 4

5.5 Cross-hand context: masking ablation

To isolate the contribution of context specifically, we evaluate AlphaExploitem against the same model but with the history completely masked during evaluation time so the policy must act on the current-hand card and action observations alone. Architecture, optimization, and league exposure are by construction identical between the two. The only difference is whether the transformer encoder sees the prior-hand tokens at decision time.

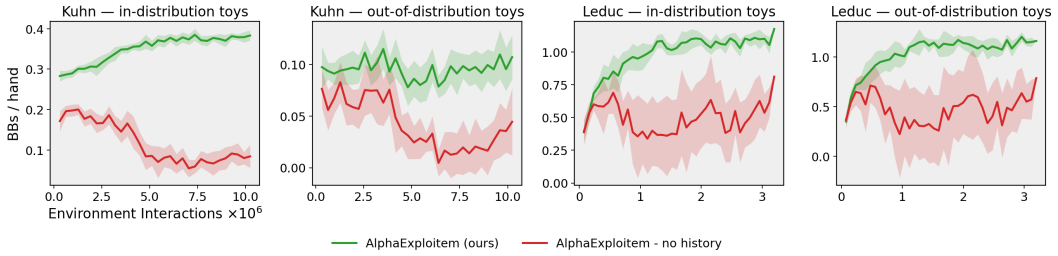

Figure 8: Effect of masking the cross-hand history channel at evaluation time on the same trained AlphaExploitem policy. Four panels: Kuhn (left two) and Leduc (right two), each split into in-distribution and out-of-distribution toy pools. Per-group seed-mean is plotted with 95% confidence over 8 seeds.

Figure 8 reports the evolution of mean reward in both evaluation modes. The two lines start nearly together early in training, when the agent has not yet learned to use cross-hand context, and then diverge as the encoder learns to extract opponent-specific signal from the prior hands. The gap between the green and red curves is the contribution attributable specifically to in-context adaptation, with architecture, training, and parameters held fixed.

The size of the gap differs sharply between the two games. On **Leduc** the unmasked main agent earns $+1.10$ BBs/hand on the ID pool and $+1.16$ on the OOD pool (seed-means, n=8). Masking the same policy collapses these to $+0.54$ and $+0.52$ respectively. Roughly half of the agent's exploitation reward is attributable specifically to cross-hand context, and the OOD pool actually benefits slightly more from context than the ID pool ($\Delta_{\text{OOD}} = +0.64$ vs. $\Delta_{\text{ID}} = +0.55$ chips/hand). This is consistent with the encoder learning class-level opponent features rather than per-toy fingerprints. On **Kuhn** the gap is proportionally even larger: ID main reaches $+0.38$ vs. $+0.09$ masked, and OOD reaches $+0.10$ vs. $+0.03$ masked. The masked policy is barely better than zero on Kuhn OOD. This is intuitive given the game: a single Kuhn hand exposes very little hidden information about an opponent's style (only one card and a short betting line), so once context is removed the policy has almost nothing to specialise on.

A natural concern with masked-vs-unmasked attribution is that the masked variant is not constrained to play near-Nash. Nothing in training forces the same parameters to behave as a safe NE-approximator when context is removed, so the masked level is simply "what this trained policy produces from card

and current-hand action observations alone". Two empirical observations bound this. First, on Leduc the masked aggregate sits noticeably above the (computed) NE level on both pools (Appendix D.3), so the masked policy is on average more exploitative than safe NE play. Second, the masked aggregate is comparable to or below the no-encoder league-only baseline on the same pools, which suggests the masked policy is being overfit on the training set. We therefore read the masked-vs-unmasked gap as a *lower bound* on the exploitation gain attributable to cross-hand context.

6 Conclusion

We presented AlphaExploitem, a transformer-augmented poker agent that learns to exploit suboptimal opponents by processing multi-hand game histories through a lightweight transformer encoder. Built on AlphaHoldem's [Zhao et al., 2022] architectural foundation and trained via PPO with K-best league self-play, the agent simultaneously learns exploitative adaptation and robust equilibrium-approximating play. We demonstrate the approach across Kuhn Poker and Leduc Hold'em, and present our findings regarding both exploitative decisions against flawed hand-crafted policies and robust play against Nash Equilibria. Our results show that AlphaExploitem can learn to exploit a variety of opponent archetypes, achieving significant gains against hand-crafted policies while maintaining competitive performance against equilibrium strategies.

References

Darse Billings, Denis Papp, Jonathan Schaeffer, and Duane Szafron. Opponent modeling in poker. *AAAI/IAAI*, 493(499):105, 1998.

Darse Billings, Aaron Davidson, Jonathan Schaeffer, and Duane Szafron. The challenge of poker. *Artificial Intelligence*, 134(1):201–240, 2002.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/jax-ml/jax`.

Noam Brown and Tuomas Sandholm. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science*, 359(6374):418–424, 2018.

Noam Brown and Tuomas Sandholm. Superhuman AI for multiplayer poker. *Science*, 365(6456):885–890, 2019.

Noam Brown, Adam Lerer, Sam Gross, and Tuomas Sandholm. Deep counterfactual regret minimization. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 793–802. PMLR, 09–15 Jun 2019. URL `https://proceedings.mlr.press/v97/brown19b.html`.

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.

Arpad E. Elo. The rating of chessplayers, past and present. 1978. URL `https://api.semanticscholar.org/CorpusID:142610973`.

Seth Frey, Dominic K. Albino, and Paul L. Williams. Synergistic information processing encrypts strategic reasoning in poker. *Cognitive Science*, 42(5):1457–1476, 2018. doi: https://doi.org/10.1111/cogs.12632. URL `https://onlinelibrary.wiley.com/doi/abs/10.1111/cogs.12632`.

Johannes Heinrich and David Silver. Deep reinforcement learning from self-play in imperfect-information games. *arXiv preprint arXiv:1603.01121*, 2016.

Bret Hoehn, Finnegan Southey, Robert C. Holte, and Valeriy Bulitko. Effective short-term opponent exploitation in simplified poker. In *AAAI*, volume 5, pages 783–788, 2005.

Chenghao Huang, Yanbo Cao, Yinlong Wen, Tao Zhou, and Yanru Zhang. PokerGPT: An end-to-end lightweight solver for multi-player Texas Hold'em via large language model. *arXiv preprint arXiv:2401.06781*, 2024.

Sotetsu Koyamada, Shinri Okano, Soichiro Nishimori, Yu Murata, Keigo Habara, Haruka Kita, and Shin Ishii. Pgx: Hardware-accelerated parallel game simulators for reinforcement learning, 2024. URL `https://arxiv.org/abs/2303.17503`.

Harold W. Kuhn. A simplified two-person poker. *Contributions to the Theory of Games*, 1:97–103, 1950.

Michael Laskin, Luyu Wang, Junhyuk Oh, Emilio Parisotto, Stephen Spencer, Richie Steigerwald, DJ Strouse, Steven Hansen, Angelos Filos, Ethan Brooks, Maxime Gazeau, Himanshu Sahni, Satinder Singh, and Volodymyr Mnih. In-context reinforcement learning with algorithm distillation, 2022. URL https://arxiv.org/abs/2210.14215.

Shuxin Li, Chang Yang, Youzhi Zhang, Pengdeng Li, Xinrun Wang, Xiao Huang, Hau Chan, and Bo An. In-context exploiter for extensive-form games, 2024. URL https://arxiv.org/abs/2408.05575.

Xun Li and Risto Miikkulainen. Evolving adaptive LSTM poker players for effective opponent exploitation. In *AAAI Workshops*, 2017.

Xun Li and Risto Miikkulainen. Dynamic adaptation and opponent exploitation in computer poker. 2018a. URL http://www.cs.utexas.edu/users/ai-lab?li:aaaiws2018.

Xun Li and Risto Miikkulainen. Opponent modeling and exploitation in poker using evolved recurrent neural networks. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 189–196, 2018b.

Matej Moravcík, Martin Schmid, Neil Burch, Viliam Lisý, Dustin Morrill, Nolan Bard, Trevor Davis, Kevin Waugh, Michael Johanson, and Michael Bowling. DeepStack: Expert-level artificial intelligence in heads-up no-limit poker. *Science*, 356(6337):508–513, 2017.

John F. Nash. Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1): 48–49, 1950.

Hukukane Nikaidô and Kazuo Isoda. Note on non-cooperative convex games. *Pacific Journal of Mathematics*, 1955.

Marc-Antoine Provost, Nejc Ilenic, Christopher Solinas, and Philippe Beardsell. Gto wizard benchmark, 2026. URL https://arxiv.org/abs/2603.23660.

Jonathan Rubin and Ian Watson. Computer poker: A review. *Artificial Intelligence*, 175(5):958–987, 2011. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2010.12.005. URL https://www.sciencedirect.com/science/article/pii/S0004370211000191. Special Review Issue.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

Finnegan Southey, Michael P. Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and Chris Rayner. Bayes' bluff: Opponent modelling in poker. *arXiv preprint arXiv:1207.1411*, 2012.

Eric Steinberger, Adam Lerer, and Noam Brown. Dream: Deep regret minimization with advantage baselines and model-free learning. *arXiv preprint arXiv:2006.10410*, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

Zhe Wu, Kai Li, Enmin Zhao, Hang Xu, Meng Zhang, Haobo Fu, Bo An, and Junliang Xing. L2E: Learning to exploit your opponent, 2021.

Jiahui Xu, Jing Chen, and Shaofei Chen. Efficient opponent exploitation in no-limit Texas Hold'em poker: A neuroevolutionary method combined with reinforcement learning. *Electronics*, 10(17), 2021.

Enmin Zhao, Renye Yan, Jinqiu Li, Kai Li, and Junliang Xing. AlphaHoldem: High-performance artificial intelligence for heads-up no-limit poker via end-to-end reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 4689–4697, 2022.

Appendix Contents

A Limitations

This section discusses the limitations and the assumptions of our work, and what are the possible solutions that can improve our approach.

Strong assumptions. All the evaluation toy policies present stationary policies. The model is exposed to non-stationary policies during the league and the buffer play stages, but we do not integrate evaluation metrics for non-stationarity conditions. Additionally, we assume that AlphaExploitem plays matches against single opponents (heads-up play), with an unlimited supply of money but with a fixed amount of hands per game.

Limitations. While Leduc Hold'em presents the same characteristics as Texas Hold'em, it does not reach the same complexity. The toy crafting component of our approach may prove harder to reliably scale to such an environment. Another flaw of our model is its finite time-horizon. Alphaexploitem is not able to play infinitely without losing information about old hands. A possible solution for this would be to use a recurrent network with a hidden state that is updated by the latent dimensions of the cross-hand encoder.

Computational Efficiency. The proposed training algorithm inherits the parallelizability traits of the transformer [Vaswani et al., 2017] architecture but also the data inefficiency of PPO [Schulman et al., 2017]. For our experiments, we used a single NVIDIA A40 with a training time of 12 hours per Leduc seed and 4 hours per Kuhn seed.

B Player Types

Real-world poker styles are conventionally placed on two axes: how often a player enters a hand (*tight* vs. *loose*) and how often they raise versus call (*passive* vs. *aggressive*). The four resulting quadrants name the colloquial archetypes used throughout the poker community: *tight-aggressive* (TAG, the canonical strong style), *loose-aggressive* (LAG, profitable but high-variance), *tight-passive* (the *nit* or *rock*, cautious to a fault), and *loose-passive* (the *calling station*, which calls down indiscriminately). Two further endpoints sit at the extremes of the action distribution: the *maniac* (raises and re-raises near-uniformly) and the *folder* (folds whenever legal). Except the TAG — balanced play that an NE-approximating agent already handles — each of these styles corresponds to a stationary mixed policy with predictable, exploitable structure: precisely the kind of suboptimal play our agent is trained to detect and exploit. We instantiate the exploitable archetypes as fixed *toy opponents* with explicit action-probability tables (Appendix C), use them both during training and for in-distribution evaluation, and reserve unseen variants for the out-of-distribution evaluation pool.

C Toy strategies

This appendix specifies every toy opponent used in training and evaluation. All toys are stationary, history-(in)dependent mixed policies. A toy never learns or adapts. We give each toy's full action distribution as fixed probability tables.

C.1 Kuhn poker toys

Kuhn poker has three cards (J, Q, K) and two actions (BET, PASS). Each toy is a 3×4 matrix giving *the probability of BET* given (hero card, betting history). Columns are the four possible histories at the hero's decision nodes: *first to act*, *after pass*, *after bet*, *after pass-bet*. "After bet"/"After pass-bet" rows are interpreted as the probability of betting (= calling). Subtracting from 1 gives the probability of passing (= folding).

In-distribution toys (training pool). These seven toys are part of the training curriculum.

Table 1: Kuhn ID toys: probability of BET. Cards: J=jack, Q=queen, K=king. Histories: *(start)* = hero acts first. *p* = opponent passed. *b* = opponent bet. *pb* = pass-bet line.

Toy	$P(\text{BET} \mid J, \cdot)$	$P(\text{BET} \mid Q, \cdot)$	$P(\text{BET} \mid K, \cdot)$
histories: (start, p, b, pb)			
`f` (Folder)	$(0.0, 0.0, 0.0, 0.0)$	$(0.0, 0.0, 0.0, 0.0)$	$(0.0, 0.0, 0.0, 0.0)$
`abj` (AlwaysBetJacks)	$(1.0, 1.0, 0.0, 0.0)$	$(0.0, 0.0, 0.5, 0.0)$	$(0.0, 0.0, 1.0, 1.0)$
`ab` (AlwaysBet)	$(1.0, 1.0, 1.0, 0.0)$	$(1.0, 1.0, 1.0, 0.0)$	$(1.0, 1.0, 1.0, 0.0)$
`cs` (Calling Station)	$(0.0, 0.0, 0.5, 1.0)$	$(0.0, 0.0, 1.0, 1.0)$	$(1.0, 1.0, 1.0, 1.0)$
`m` (Maniac)	$(1.0, 1.0, 0.0, 0.0)$	$(0.0, 0.0, 1.0, 1.0)$	$(1.0, 1.0, 1.0, 1.0)$
`n` (Nit)	$(0.0, 0.0, 0.0, 0.0)$	$(0.0, 0.0, 0.0, 0.0)$	$(1.0, 1.0, 1.0, 1.0)$
`abq` (AlwaysBetQueens)	$(0.0, 0.0, 0.0, 0.0)$	$(1.0, 1.0, 1.0, 1.0)$	$(0.0, 0.0, 0.6, 0.6)$

Out-of-distribution toys (evaluation only). These seven toys are never sampled during training.

Table 2: Kuhn OOD toys: probability of BET. Layout matches Table 1. `ood_switch_mf` is non-stationary: it alternates between Maniac and Folder every 50 hands within a session, so the entries below summarise its two component policies.

Toy	$P(\text{BET} \mid J, \cdot)$	$P(\text{BET} \mid Q, \cdot)$	$P(\text{BET} \mid K, \cdot)$
`ood_p1p` (P1Passes)	$(0.1, 0.33, 0.0, 0.0)$	$(0.1, 0.0, 0.33, 0.0)$	$(0.1, 1.0, 1.0, 0.0)$
`ood_switch_mf` (Maniac half)	$(1.0, 1.0, 0.0, 0.0)$	$(0.0, 0.0, 1.0, 1.0)$	$(1.0, 1.0, 1.0, 1.0)$
`ood_switch_mf` (Folder half)	$(0.0, 0.0, 0.0, 0.0)$	$(0.0, 0.0, 0.0, 0.0)$	$(0.0, 0.0, 0.0, 0.0)$
`ood_trap` (Trap)	$(0.5, 0.5, 0.0, 0.0)$	$(0.7, 0.5, 0.4, 0.3)$	$(0.0, 1.0, 1.0, 1.0)$
`ood_def_bluff`(DefBluff)	$(0.0, 0.6, 0.0, 0.0)$	$(0.0, 0.4, 0.0, 0.0)$	$(0.7, 1.0, 1.0, 1.0)$
`ood_pos_aggr` (PosAggr)	$(0.4, 0.0, 0.0, 0.0)$	$(0.5, 0.1, 0.5, 0.4)$	$(0.9, 0.5, 1.0, 1.0)$
`ood_p` (Pro)	$(0.1, 0.33, 0.0, 0.0)$	$(0.33, 0.0, 0.33, 0.33)$	$(1.0, 1.0, 1.0, 1.0)$
`ood_p2` (Pro2 / NE)	$(0.1, 0.33, 0.0, 0.0)$	$(0.0, 0.0, 0.33, 0.43)$	$(0.3, 1.0, 1.0, 1.0)$

C.2 Leduc Hold'em toys

Leduc has six cards (J, Q, K $\times$ 2 suits but suits are irrelevant for resolution), two betting rounds, and three actions (CALL/CHECK, BET/RAISE, FOLD). Toys condition on *(hero rank, community rank, round)* but not on the hero's preceding actions. Some additionally condition on whether a bet is pending (i.e. on the legal-action mask, which gates FOLD). All probabilities below are renormalized over legal actions at runtime.

In-distribution toys (training and evaluation pool). Eight toys, sampled uniformly within each training batch.

Table 3: Leduc ID toys. Action probabilities are ordered as (call/check, bet/raise, fold). Where the policy varies with state we list each branch separately. Otherwise the toy plays a single fixed mixture.

`maniac`	$(0.05, 0.95, 0.0)$ everywhere — aggressive toy.
`lag`	Pre-flop: $(0.25, 0.7, 0.05)$. Post-flop pair: $(0.05, 0.95, 0.0)$. Post-flop no-pair: $(0.3, 0.55, 0.15)$.
`random`	Uniform random over the legal actions at every decision.
`calling_station`	$(1.0, 0.0, 0.0)$ everywhere — pure call.
`folder`	$(0.05, 0.0, 0.95)$ everywhere — cautious toy.
`aggfish`	Pre-flop: $(0.1, 0.9, 0.0)$. Post-flop pair: $(0.05, 0.95, 0.0)$. Post-flop no-pair: $(0.1, 0.05, 0.85)$.
`rock`	Pre-flop: J $(0.3, 0.0, 0.7)$, Q/K $(0.95, 0.0, 0.05)$. Post-flop pair $(0.8, 0.15, 0.05)$, K-no-pair $(0.6, 0.0, 0.4)$, otherwise $(0.1, 0.0, 0.9)$.
`nit`	Pre-flop: J $(0.0, 0.0, 1.0)$, Q $(0.8, 0.0, 0.2)$, K $(0.3, 0.7, 0.0)$. Post-flop pair $(0.3, 0.7, 0.0)$, K-no-pair $(0.7, 0.1, 0.2)$, otherwise $(0.0, 0.0, 1.0)$.

Out-of-distribution toys (evaluation only). Twelve toys never sampled during training. They are novel combinations or perturbations of the styles seen in training.

Table 4: Leduc OOD toys. Layout matches Table 3. Action probabilities (call/check, bet/raise, fold).

`ood_maniac_soft`	$(0.2, 0.75, 0.05)$ — softer maniac.
`ood_pair_bluffer`	Pre-flop $(0.4, 0.5, 0.1)$. Post-flop pair $(0.85, 0.1, 0.05)$ (trap), no-pair $(0.1, 0.8, 0.1)$ (bluff). Inverted value-betting.
`ood_mild_maniac`	$(0.3, 0.6, 0.1)$ — between LAG and Maniac.
`ood_anti_fold`	$(0.5, 0.5, 0.0)$ — never folds.
`ood_post_aggro`	Pre-flop $(0.9, 0.05, 0.05)$. Post-flop $(0.15, 0.7, 0.15)$ regardless of card.
`ood_soft_lag`	Pre-flop $(0.45, 0.45, 0.1)$. Post-flop pair $(0.15, 0.85, 0.0)$, no-pair $(0.45, 0.35, 0.2)$.
`ood_king_bully`	K $(0.05, 0.9, 0.05)$, J/Q $(0.8, 0.05, 0.15)$.
`ood_chaos`	Per-decision: with prob $1/3$ commit to all-call, $1/3$ all-raise, $1/3$ all-fold.
`ood_value_heavy`	Pre-flop $(0.7, 0.2, 0.1)$. Post-flop pair $(0.1, 0.9, 0.0)$, no-pair $(0.8, 0.0, 0.2)$.
`ood_passive_fish`	Pre-flop $(0.85, 0.1, 0.05)$. Post-flop pair $(0.4, 0.6, 0.0)$, no-pair $(0.15, 0.0, 0.85)$.
`ood_nit_loose`	Pre-flop: J $(0.2, 0.0, 0.8)$, Q/K $(0.5, 0.4, 0.1)$. Post-flop pair $(0.2, 0.8, 0.0)$, Q/K no-pair $(0.5, 0.1, 0.4)$, otherwise $(0.0, 0.0, 1.0)$.
`ood_tight_caller`	Pre-flop: J $(0.0, 0.0, 1.0)$, Q/K $(0.95, 0.0, 0.05)$. Post-flop pair $(0.9, 0.05, 0.05)$, J $(0.0, 0.0, 1.0)$, Q/K no-pair $(0.8, 0.0, 0.2)$.

D Nash equilibria

This appendix records the Nash-equilibrium strategies used as the reference policy in the vs-Nash evaluation of Section 5.3. For Kuhn the equilibrium is a one-parameter family with a closed form. For Leduc Hold'em it is a sampled finite table over the game's information sets.

D.1 Kuhn Nash equilibrium

Kuhn poker admits a one-parameter family of Nash-equilibrium strategies indexed by $\alpha \in [0, 1/3]$ [Kuhn, 1950]. Both players' strategies are listed below. Cards are J=jack, Q=queen, K=king. Actions are BET and PASS. The betting trees considered are *(start)* (the first to act), *after pass*, *after bet*, and *after pass-bet*.

Table 5: Player 1 (first to act).

history	J	Q	K
(start)	α	0	3α
p	0	0	1
b	0	$1/3 + \alpha$	1
pb	0	$1/3 + \alpha$	1

Table 6: Player 2 (second to act).

history	J	Q	K
p	$1/3$	0	1
b	0	$1/3$	1

Game value. Under any Nash strategy in this family the game value is $V_1 = -1/18$ chips per hand for Player 1 (in position. First to act preflop) and $+1/18$ for Player 2. The vs-Nash evaluation in Section 5.3 reports rewards averaged across positions (since the dealer is randomized per hand by the simulator), so the relevant zero-line is 0 chips/hand and not $-1/18$.

For our evaluations we instantiate $\alpha = 1/3$ as a canonical choice.

D.2 Leduc Hold'em Nash equilibrium

Leduc Hold'em has a much larger but still finite information-set space. Our reference Nash strategy is a tabulated equilibrium with one row per information set, computed via counterfactual regret minimisation to convergence. We expose two slices below.

Round-1 information sets. Round 1 (pre-community card) has 18 reachable information sets keyed by the player's own card and the action history so far. Action letters: $x =$ check, $b =$ bet, $r =$ raise, $c =$ call, $f =$ fold. Each row sums to 1.

Table 7: Leduc Nash strategy, round-1 slice. Rows are info sets (*card history*). Columns are action probabilities. Entries below 10^{-4} are reported as 0.000. Game value: $V_1 \approx -0.0856$ chips per hand for Player 1 (first to act preflop), $+0.0856$ for Player 2.

info set	x (check)	b (bet)	r (raise)	c (call)	f (fold)
J	0.926	0.074	-	-	-
J b	-	-	0.055	0.126	0.819
J br	-	-	-	1.000	0.000
J x	0.704	0.296	-	-	-
J xb	-	-	0.018	0.035	0.947
J xbr	-	-	-	1.000	0.000
Q	0.256	0.744	-	-	-
Q b	-	-	0.369	0.631	0.000
Q br	-	-	-	1.000	0.000
Q x	0.151	0.849	-	-	-
Q xb	-	-	0.160	0.840	0.000
Q xbr	-	-	-	1.000	0.000
K	0.246	0.754	-	-	-
K b	-	-	0.580	0.420	0.000
K br	-	-	-	1.000	0.000
K x	0.000	1.000	-	-	-
K xb	-	-	0.688	0.312	0.000
K xbr	-	-	-	1.000	0.000

Round-2 information sets. After the community card is dealt, each round-1 reachable history can be extended by a community card $\in \{J, Q, K\}$ and a fresh round-2 betting line. With three card configurations $\times$ six round-1 histories $\times$ six round-2 betting trees, the round-2 table has roughly 100 rows. We omit the full listing here for space. The qualitative pattern is intuitive: with a pair the policy bets aggressively (e.g. $P(b \mid \texttt{JJ xxd}) \approx 1.0$), with no pair and against a bet it folds or bluff-raises depending on hand strength relative to the community card.

D.3 NE evaluation against the toy pools

The NE policies of Sections D.1–D.2 themselves obtain a non-zero reward against the suboptimal toys, since NE play is safe-but-not-exploitative. This provides the reference value used in the dashed lines in Figure 2. Tables 8 and 9 report the per-toy and aggregate NE reward, split by which seat NE occupies. Kuhn values are exact (closed-form EV over the 12-info-set tree). Leduc values are Monte-Carlo over $20{,}000$ hands per matchup, split evenly across the two seat assignments, against the NE policy.

Table 8: NE vs. toy reward on Kuhn (chips/hand). Columns are NE's reward when it acts as P0 (first to act), as P1 (second to act), and the seat-averaged mean used as the dashed purple reference in Figure 2. Toys within each pool are sorted by descending mean.

Opponent	NE-is-P0	NE-is-P1	Mean
In-distribution toys			
`abq`	$+0.1533$	$+0.1778$	$+0.1656$
`f`	$+0.0667$	$+0.2222$	$+0.1444$
`ab`	$+0.1111$	$+0.1111$	$+0.1111$
`cs`	-0.0306	$+0.2222$	$+0.0958$
`abj`	$+0.0167$	$+0.0556$	$+0.0361$
`m`	-0.0556	$+0.0556$	0.0000
`n`	-0.0556	$+0.0556$	0.0000
ID aggregate	-	-	$\mathbf{+0.0790}$
Out-of-distribution toys			
`ood_p1p`	-0.0556	$+0.2111$	$+0.0778$
`ood_switch_mf`	$+0.0056$	$+0.1389$	$+0.0722$
`ood_trap`	$+0.0028$	$+0.0944$	$+0.0486$
`ood_pos_aggr`	-0.0078	$+0.0833$	$+0.0378$
`ood_def_bluff`	-0.0089	$+0.0556$	$+0.0233$
`ood_p`	-0.0556	$+0.0741$	$+0.0093$
`ood_p2`	-0.0556	$+0.0556$	0.0000
OOD aggregate	-	-	$\mathbf{+0.0384}$

Table 9: NE vs. toy reward on Leduc Hold'em (chips/hand). Columns are NE's reward when it acts as P1 (first to act), as P2 (second to act), and the seat-averaged mean. Monte-Carlo over $20{,}000$ hands per matchup.

Opponent	NE-is-P1	NE-is-P2	Mean
In-distribution toys			
`random`	$+0.5270$	$+0.7896$	$+0.6583$
`calling_station`	$+0.5254$	$+0.7535$	$+0.6394$
`folder`	$+0.5004$	$+0.6583$	$+0.5794$
`lag`	$+0.4665$	$+0.6083$	$+0.5374$
`maniac`	$+0.1242$	$+0.6082$	$+0.3662$
`aggfish`	$+0.1847$	$+0.2584$	$+0.2215$
`rock`	$+0.1127$	$+0.3251$	$+0.2189$
`nit`	-0.0463	$+0.1096$	$+0.0316$
ID aggregate	-	-	$\mathbf{+0.4066}$
Out-of-distribution toys			
`ood_chaos`	$+0.6344$	$+0.8816$	$+0.7580$
`ood_anti_fold`	$+0.5585$	$+0.8463$	$+0.7024$
`ood_post_aggro`	$+0.6531$	$+0.6355$	$+0.6443$
`ood_mild_maniac`	$+0.6070$	$+0.6279$	$+0.6175$
`ood_soft_lag`	$+0.4753$	$+0.7461$	$+0.6107$
`ood_king_bully`	$+0.4257$	$+0.7619$	$+0.5938$
`ood_pair_bluffer`	$+0.5842$	$+0.6020$	$+0.5931$
`ood_maniac_soft`	$+0.4312$	$+0.6871$	$+0.5592$
`ood_value_heavy`	$+0.2817$	$+0.8176$	$+0.5496$
`ood_passive_fish`	$+0.1790$	$+0.4847$	$+0.3318$
`ood_tight_caller`	$+0.1293$	$+0.4064$	$+0.2678$
`ood_nit_loose`	$+0.0692$	$+0.3123$	$+0.1908$
OOD aggregate	-	-	$\mathbf{+0.5349}$

Reading the tables. The per-seat split exposes a substantial asymmetry: Leduc NE extracts noticeably more reward when it acts second (the Maniac column is the extreme: $+0.124$ as P1, $+0.608$ as P2, since acting after a maniac's bet is much more profitable than acting before). The aggregate values — $+0.0790$ (Kuhn ID), $+0.0384$ (Kuhn OOD), $+0.4066$ (Leduc ID), $+0.5349$ (Leduc OOD) — are the dashed references in Figure 2 and bound the exploitation regime: a trained agent below the line is being out-exploited by a non-adaptive NE policy on the same pool.

E Best-response ceilings

The **best-response (BR) ceiling** of a (deterministic or mixed) opponent policy π^o is the value of the BR policy against π^o:

$$R_{\text{BR}}(\pi^o) \;=\; \max_{\pi^h} \mathbb{E}\big[\text{hero reward per hand} \,\big|\, \pi^h \text{ vs. } \pi^o\big].$$

This is the "oracle" upper bound on what *any* hero policy with full knowledge of π^o can extract per hand, and it lets us report a unit-free **BR-fraction** $R/R_{\text{BR}}(\pi^o) \in [0, 1]$ alongside the raw reward, which is comparable across opponents of very different exploitability.

Leduc per-opponent values. Table 10 reports $R_{\text{BR}}(\pi^o)$ in chips per hand for every Leduc toy used in the paper. Toys are sorted within each pool by descending ceiling — i.e. how much exploit potential a perfect adversary could extract. This ordering is the "T_n" index used in the per-opponent figures of Section 5.2.

Table 10: Best-response ceilings $R_{\text{BR}}(\pi^o)$ for the Leduc toy pools, in BBs/hand. Higher = more exploitable. ID toys are present in the training curriculum. OOD toys are evaluation-only.

In-distribution (ID)		**Out-of-distribution (OOD)**	
toy	R_{BR}	toy	R_{BR}
`random`	2.375	`ood_pair_bluffer`	3.208
`maniac`	2.366	`ood_post_aggro`	3.139
`aggfish`	1.946	`ood_maniac_soft`	2.423
`lag`	1.829	`ood_mild_maniac`	2.285
`calling_station`	1.467	`ood_king_bully`	2.213
`folder`	1.093	`ood_anti_fold`	2.149
`nit`	0.604	`ood_chaos`	1.844
`rock`	0.520	`ood_soft_lag`	1.561
		`ood_value_heavy`	1.368
		`ood_passive_fish`	1.250
		`ood_tight_caller`	0.842
		`ood_nit_loose`	0.754

Kuhn per-opponent values. Table 11 reports the analogous BR ceilings for the Kuhn toy pools. Values are exact (closed-form expectimax over the 12-info-set hero tree). The folder `f` sits at the absolute exploitation ceiling of 1 BB/hand because a perfect BR can simply bet every hand and win the starting pot uncontested the inverted-value `abj` and the indiscriminate `cs` carry the next-largest ceilings. The two near-Nash baselines (`ood_p`, `ood_p2`) sit near 0 as expected.

Table 11: Best-response ceilings $R_{\mathrm{BR}}(\pi^o)$ for the Kuhn toy pools, in chips per hand, computed exactly by closed-form expectimax. Higher = more exploitable. ID toys are present in the training curriculum. OOD toys are evaluation-only.

In-distribution (ID)		**Out-of-distribution (OOD)**	
toy	R_{BR}	toy	R_{BR}
`f`	1.000	`ood_p1p`	0.439
`abj`	0.417	`ood_switch_mf`	0.333
`cs`	0.375	`ood_def_bluff`	0.275
`ab`	0.333	`ood_trap`	0.191
`m`	0.250	`ood_pos_aggr`	0.117
`n`	0.250	`ood_p`	0.066
`abq`	0.217	`ood_p2`	0.001

BR-normalised per-opponent results. Figures 9, 10, 11, 12 restate the per-toy bars of Section 5.4 in BR-fraction units $R/R_{\mathrm{BR}}(\pi^o)$. This makes opponents with very different exploit potential directly comparable: a value of 1.0 means saturating the oracle ceiling, 0.5 means realising half of the available exploit, and 0 means baseline play. The main agent saturates a noticeably larger fraction of the ceiling than either no-encoder group on every comparable opponent; the gap to the league-only baseline is largest, in BR-fraction terms, on the moderately-exploitable opponents where there is the most room for context-conditional play to pay off.

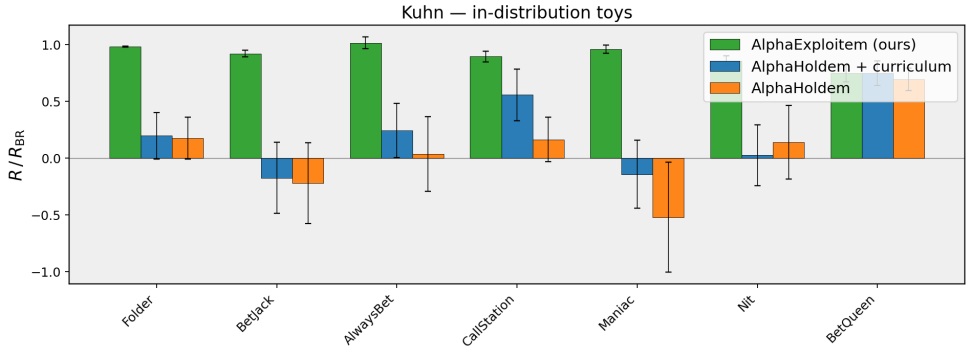

Figure 9: Kuhn in-distribution final-checkpoint R/R_{BR} against each toy opponent. Bars are seed-means over the last 5 logged checkpoints with 95% confidence error bars. N = 8 seeds per group. Toys are sorted left-to-right by descending average reward.

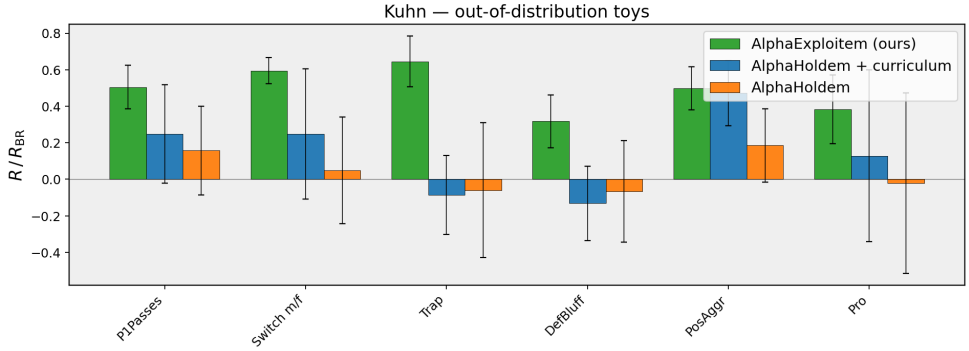

Figure 10: Kuhn — out-of-distribution toys. Same conventions as Figure 9. `ood_p2` (Pro2 / Nash) is omitted: its BR ceiling is ≈ 0.001 BB/hand by construction, so the BR-fraction is dominated by noise rather than agent quality.

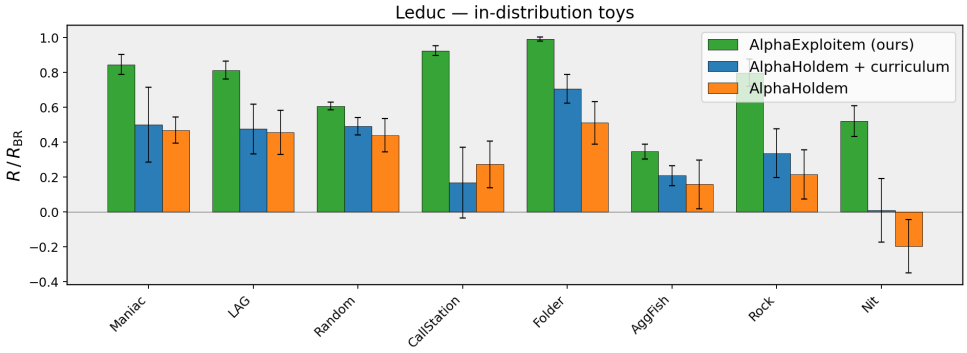

Figure 11: Leduc — in-distribution toys. Same conventions as Figure 9

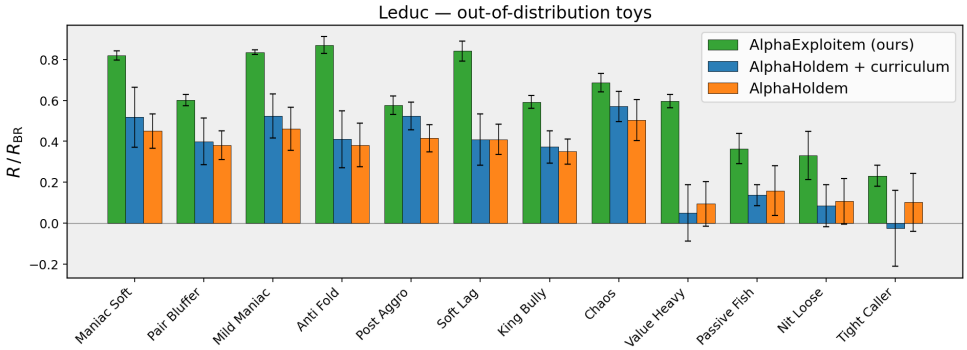

Figure 12: Leduc — out-of-distribution toys. Same conventions as Figure 9

BR-fraction evolution. The bars above are read at the final checkpoint, but the BR-fraction also varies smoothly over training. Figures 13, 14, 15, 16 report the per-epoch toy-averaged R/R_{BR} for AlphaExploitem on each game-pool combination, alongside the same trained policy with the cross-hand context masked at evaluation. `ood_p2` (Pro2 / Nash on Kuhn) is omitted from the toy-average for the same near-zero-ceiling reason given for the histograms.

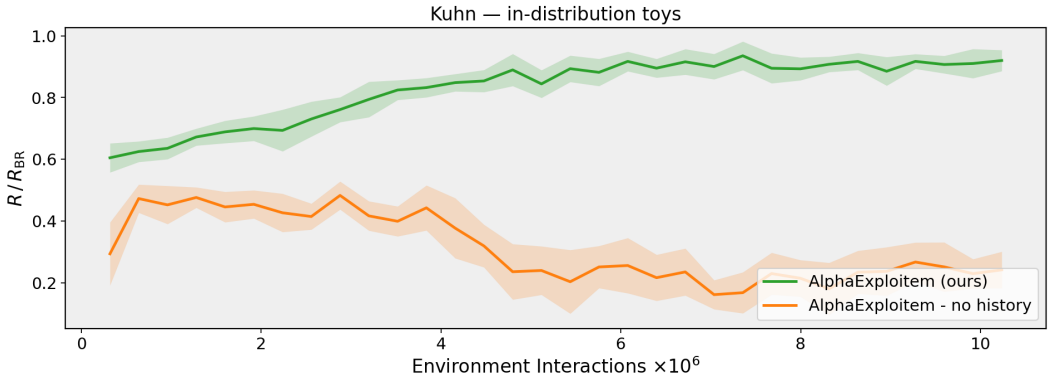

Figure 13: Kuhn in-distribution BR-fraction evolution. AlphaExploitem (green) and AlphaExploitem with the cross-hand context masked at evaluation (orange). Per-epoch seed-mean of the toy-averaged R/R_{BR} with a 95% confidence shaded band. N = 8 seeds.

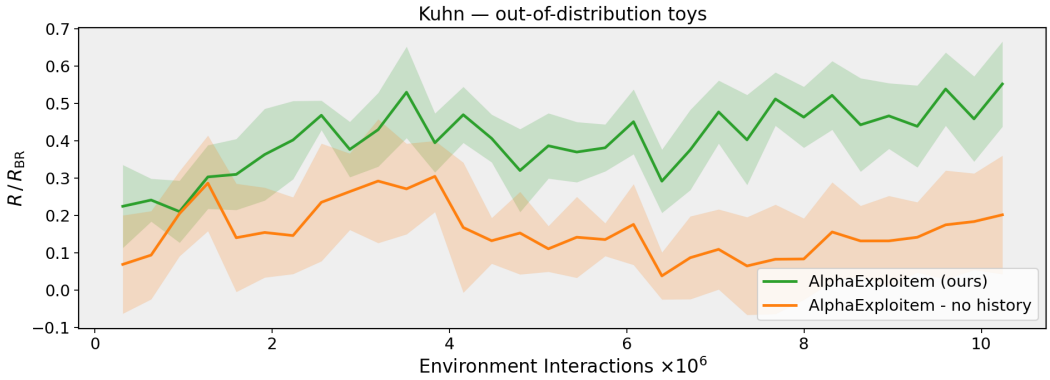

Figure 14: Kuhn — out-of-distribution. Same conventions as Figure 13

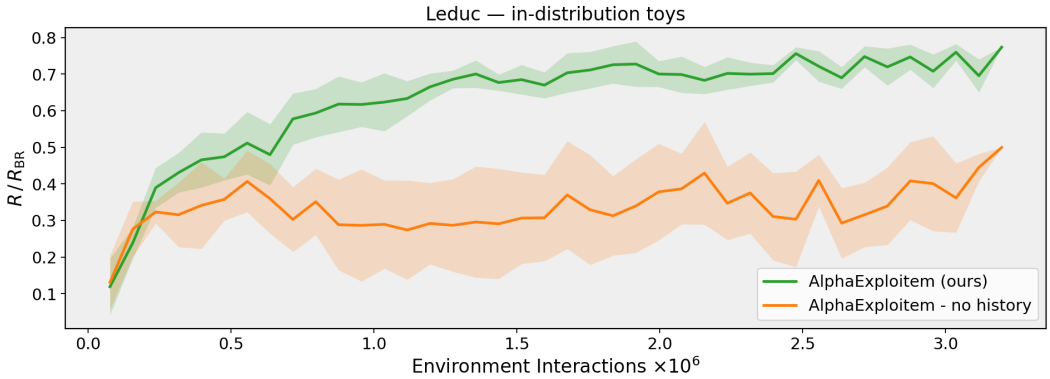

Figure 15: Leduc — in-distribution. Same conventions as Figure 13

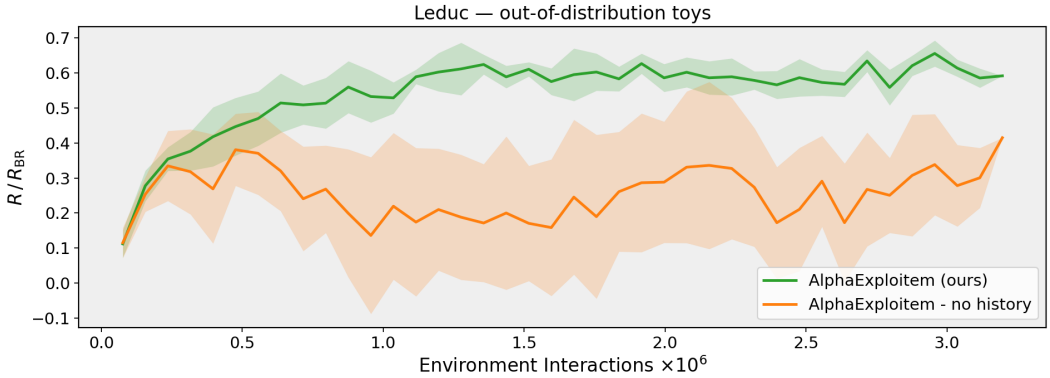

Figure 16: Leduc — out-of-distribution. Same conventions as Figure 13

F Hyperparameters

F.1 Hyperparameter tuning

The training pipeline inherits most hyperparameters from AlphaHoldem [Zhao et al., 2022]: K-best league self-play, the actor-critic backbone, and the AdamW + KL-based early-stopping schedule. The PPO objective is the standard clipped surrogate [Schulman et al., 2017]. The novel choices in this work — the hierarchical history transformer, the toy curriculum, and the snapshot buffer — introduced a small number of additional hyperparameters whose values we set as follows. For **Kuhn**, we ran a four-cell sweep over batch size, learning rate, and episode length and selected the cell with the best out-of-distribution reward and a complete training budget within the wall-clock cap, see Section 5.2 for the comparison. For **Leduc**, the league-only baseline collapses under the main agent's hyperparameters and was retuned via a small entropy / learning-rate / train-steps sweep. The main agent and the curriculum-matched ablation share a single hyperparameter set.

F.2 Network architecture

The architecture and its three input streams are described in Section 4 and depicted in Figure 1. The card and current-hand action streams are processed by lightweight per-stream linear encoders. The cross-hand history stream is processed by a hierarchical transformer (within-hand encoder followed by an across-hand encoder), and the three streams are fused by a 2-layer MLP that splits into a softmax policy head and a scalar value head. For the no-encoder ablation groups in both games we set `encoder_type=none`, which short-circuits the cross-hand stream while keeping the rest of the architecture identical.

F.3 Hyperparameters list

Tables 12 and 13 list the hyperparameters used for the main agent on Kuhn Poker and Leduc Hold'em respectively. We use 8 seeds per experiment.

Table 12: Hyperparameters used for Kuhn Poker training.

Category	Hyperparameter	Value
Architecture	encoder layers	1
	token embedding dim	8
	attention heads	4
	FFN width	64
	dropout prob	0.1
	cards-out / actions-out width	8 / 8
	MLP hidden width / layers	128 / 2
	policy head	2-way softmax
	dtype / param dtype	bf16 / fp32
Trajectory generation	envs per opponent	4
	league size	8
	episode length	100
PPO / optimizer	optimizer	AdamW
	learning rate	1×10^{-4}
	entropy coeff	0.025
	value-loss coeff	0.01
	PPO clip ϵ	0.1
	gradient clipping	global-norm 1.0
	train steps per epoch	5
	train batch size	64
	minibatches per epoch	5
	checkpoint frequency (epochs)	100

Table 13: Hyperparameters used for Leduc Hold'em training.

Category	**Hyperparameter**	**Value**
Architecture	encoder layers	1
	token embedding dim	16
	attention heads	2
	FFN width	1024
	dropout prob	0.1
	cards-out / actions-out width	64 / 128
	MLP hidden width / layers	128 / 2
	policy head	3-way softmax
	dtype / param dtype	bf16 / fp32
Trajectory generation	envs per opponent	8
	league size	5
	long-tail buffer size	25
	episode length	100
PPO / optimizer	optimizer	AdamW
	learning rate	1×10^{-4}
	entropy coeff	0.005–0.01
	value-loss coeff	0.01
	PPO clip ϵ	0.1
	gradient clipping	global-norm 1.0
	train steps per epoch	10
	train batch size	8
	minibatches per epoch	4
	checkpoint frequency (epochs)	20

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: We prove that the agent actively uses the game history to deviate from it's baseline strategy since the performance against weak opponents improves when the transformer history is not masked. Concurrently, AlphaExploitem is able to discern strong policies and play robustly against them, proven by its performance against the NE policy and the other strong toy policies.

   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 present 2 limitations: finite time horizon and game scale. The assumptions made were stationary evaluation policies, heads-up play (one opponent) and infinite money. As for computational efficiency, the algorithm is highly parallelizable, but data inefficient.

   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: No theoretical results are presented in the paper.

   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: Section 4 and Appendix F include the necessary implementation details and hyperparameters required to reproduce the results.

   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: [No]

   Justification: The code will be publicly released.

   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: The experimental settings are provided in Section 4 and in Appendix F.

   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: All the plots include statistical significance visualizations.

   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 experimental details are presented in Section 4 and in Appendix F.

   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: The research conforms in every respect with the NeurIPS Code of Ethics.

   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: [N/A]

    Justification: No significant risk,

    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: [N/A]

    Justification: The paper does not pose such risks.

    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: [N/A]

    Justification: No used assets.

    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: [N/A]

    Justification: No new assets are released.

    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: The paper does not involve crowdsourcing or 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: The paper 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 method development in this research does not involve LLMs.

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.