LEARNING REPRESENTATIONS OF POLICIES IN TWO-PLAYER ZERO-SUM GAMES

Anonymous authors
Paper under double-blind review

ABSTRACT

Search-based methods for two-player zero-sum imperfect-information games reason over public belief states, whose evolution depends on the players' policies. Scaling these methods therefore calls for compact representations of policies, not just of states. We study self-supervised learning of *policy embeddings*: low-dimensional vectors that summarize a policy's strategic behavior and, ideally, can be decoded back into a playable policy.

We make three contributions. First, we introduce three ways to generate diverse policy populations. Second, we compare six representation methods: weight-space autoencoders, behavioral encoders trained with contrastive or imitation objectives, and the conditioning embeddings of a NeuPL population network. Third, we propose a suite of downstream tasks. They range from linear probes of payoff and exploitability to zero-shot best response and equilibrium search carried out directly in embedding space.

On Kuhn and Leduc poker, behavioral encoders cut exploitability-prediction error by up to 87% while weight-based ones fail, and policy optimization in latent space improves exploitability on Leduc. **Our results point to behavior-grounded, decodable embeddings as a building block for latent-space planning in imperfect-information games.**

1 INTRODUCTION

AI research has achieved phenomenal success solving two-player zero-sum games of perfect information, such as Chess and Go Silver et al. (2016; 2017). Success has been more modest, however, in two-player zero-sum games of imperfect information, like Poker, where players may possess private information. But many real-world challenges, e.g., in cybersecurity, defense, and finance, are more aptly modeled as imperfect-information games. Thus, multiple researchers are working to scale beyond perfect-information game solvers to imperfect-information ones (Brown & Sandholm, 2019; Sokota et al., 2025; Zhang & Sandholm, 2026; Schmid et al., 2023). Many of these approaches aim to employ lookahead search at test time (e.g., MCTS (Coulom, 2006)), a method whose success was showcased by AlphaGo (Silver et al., 2016; 2017).

Two-player zero-sum imperfect-information games (2p0s IIGs) generalize partially observable Markov decision processes (POMDPs). One means of solving POMDPs is via their reduction to belief-state MDPs (Åström, 1965). Because belief states are a sufficient statistic for a POMDP's history, the transitions in belief-state MDPs are defined as usual: from belief states and actions to new belief states (given observations). Analogously, a common approach to solving IIGs involves representing them via public-belief-state games (PuB-AMGs (Brown et al., 2020; Sokota et al., 2023)), where a state is a *public belief state* (a probability distribution over all possible world states compatible with the public information). For example, in Poker, a PBS would assign probabilities to all possible hands for all players. An agent can perform lookahead search in the PuB-AMG by imagining that they and their opponent play some **policies** and end up in a new PBS (analogous to playing some *actions* in a typical perfect-information game). This approach has been successful in IIGs like Texas hold'em poker (Brown et al., 2020; Schmid et al., 2023). However, in games with larger PBSs, the PBSs and the policies become intractable to represent.

Work is ongoing to develop methods for approximating belief states in single-player and 2p0s settings (Solinas et al., 2025a; Moss et al., 2024; Šustr et al., 2021), including as latent vectors (Solinas

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

et al., 2025b). But to then predict the outcome of playing a policy at a latent PBS, the PBS must still either be sampled as discrete particles (Solinas et al., 2025a), or a latent policy representation is needed as well. This work initiates the study of policy embeddings, a missing piece towards the latter.

More specifically, we seek to learn compact (i.e., low-dimensional) representations of policies, or *policy embeddings* for IIGs. Not only can embedded policies save on memory requirements, they facilitate a shared policy structure that can naturally capture correlations so that similar inputs generate similar: e.g., if similar hands of cards suggest similar strategic play.

But like all embeddings, embedded policies are not necessarily a silver bullet. Crucial strategic information may be lost if the dimensionality is insufficient. We are thus interested not only in methods of encoding policies; we are interested in methods for decoding them as well, so that an agent can query the representation based on its private information. By decoding policies we can evaluate whether or not the learned latent structure captures an underlying policy's intent.

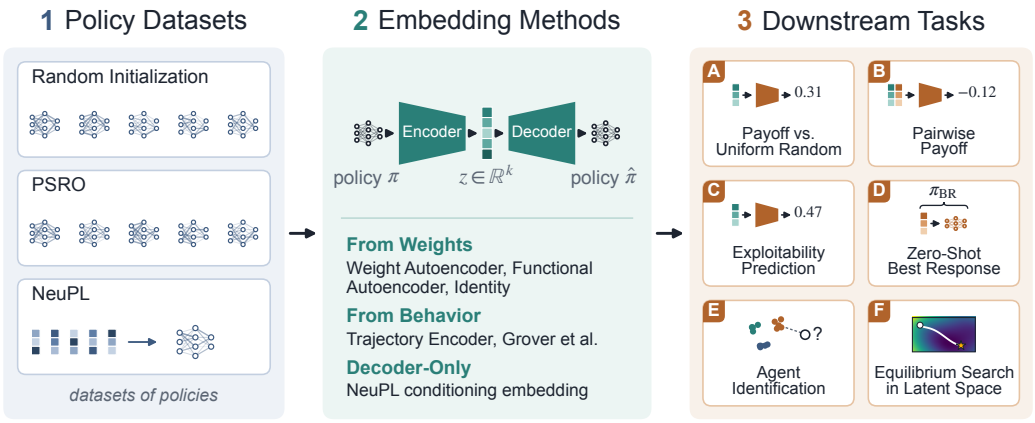


Figure 1: Overview of our approach.

Outline There are few existing methods by which to learn policy embeddings or compare their utility, especially in the context IIGs (Grover et al., 2018). In this work, we initiate an extensive, systematic approach to this learning problem. We do this in three parts (see Figure 1):

1. We introduce three methods of creating diverse datasets of policies for a game (Section 3.1)
2. We propose several methods of varying complexity for learning representations of policies. We also reimplement an existing method (Section 3.2)
3. We introduce several downstream tasks to evaluate the usefulness of the policy representations, and we use them to evaluate the methods (Section 3.3)

Code for all three parts will be made publicly available.

2 Preliminaries

Imperfect Information Games An imperfect-information game (IIG) is one in which players may not know the true state of the world. We consider finite-horizon, two-player zero-sum IIGs. Such games involve a space of game states $\mathbb{S}$, a space of actions $\mathbb{A}$, and a space of observations about the world $\mathbb{O}$, with select elements $\mathbf{s}$, $\mathbf{a}$, and $\mathbf{o}$, respectively. An *information state* records a player's observations and own actions. We denote the space of information states by $\mathbb{I}$.

Public States and Public Belief States Informally, a *public state* describes the set of game states consistent with the information that is common knowledge among the players.[1] Here, common

[1]We use public states in the sense of common-knowledge closures (Kovařík et al., 2022). Another common definition groups states by their sequences of public observations (Brown et al., 2020).

knowledge means that everyone knows the information, everyone knows that everyone knows it, and so on. A *public belief state* additionally specifies a common-knowledge probability distribution over those game states (Brown et al., 2020).

Policy Representations and Self-Supervised Learning A *policy representation*, or *embedding*, is a vector $\mathbf{z} \in \mathbb{R}^k$ that encodes information about a policy. An encoder computes an embedding from inputs, such as policy parameters or sampled behavior (i.e., trajectories). With sampled inputs, the same policy can yield different embeddings. Some methods also provide a decoder that maps an embedding back to a policy; encoding does not require such a decoder.

Related Work *Opponent modeling* (sometimes called *agent modeling*) is a closely related area of research, covered by the comprehensive surveys of Nashed & Zilberstein (2022) and Albrecht & Stone (2018). Opponent modeling is concerned with predicting the behavior of the opponent an agent is playing against based only on its observations, and using these predictions to play better against this particular opponent (Nashed & Zilberstein, 2022, Assumption A.5). Low-dimensional representations of policies are frequently used for this purpose.

Our motivation differs in that we are interested in learning policy representations in service of better approximating a Nash equilibrium of an IIG, and we are not interested in building opponent models from observations only. Likewise, our setup also differs: We consider multiple downstream tasks, intended to suggest whether or not an embedding may be useful strategically. While for opponent modeling, it is most useful to have methods that can *encode* opponent policies (or observations thereof) into embeddings, we are particularly interested in methods that can *decode* embeddings to policies, so that the policies can actually be played. Despite these key differences, there is overlap between our work and opponent modeling, and we include an opponent-modeling downstream task (best responding to a given opponent).

Self-supervised learning constructs training targets from the data themselves rather than from labels for downstream tasks (Balestriero et al., 2023). Our self-supervised methods learn representations using objectives such as reconstruction or contrastive learning. We assess their usefulness on separate downstream tasks, including by training predictors on frozen embeddings. Thus, a useful representation should retain information that transfers beyond its training objective.

We describe further related works in detail in Appendix I.

3 METHODS

In this section, we describe our methods.

3.1 METHODS FOR MAKING DATASETS OF POLICIES

In order to learn and evaluate representations of policies, we need a dataset of policies. Our aim is to generate generic representations that are capable of representing a wide range of policies. So, we must first generate a diverse set of policies from which we can learn such representations. We propose three methods of generating such datasets:

In our first method, we simply initialize policy neural networks randomly, producing a diverse population of behaviorally distinct agents. We use a custom initialization for the parameters of the neural networks, as the PyTorch default initialization produces behaviorally homogenous policies (Appendix A).

Our second method of producing a set of diverse policies for a game is to run the **Policy Space Response Oracle (PSRO)** algorithm (Lanctot et al., 2017), a method that iteratively trains approximate best-responses via deep reinforcement learning to mixtures of other policies to produce an expanding pool of policies.

Our third method of producing a set of diverse policies for a game is to run a variant of **neural population learning (NeuPL)** (Liu et al., 2022; 2024), which is the same as PSRO, except that all policies share the same conditional neural network. By "conditional neural network" we mean that the policy network takes as input both the information state (as is usual in an IIG) and a policy

Method	Encodes (black-box)	Encodes (neural networks)	Decodes
Weight autoencoders (reconstruction, functional)		✓	✓
Trajectory	✓	✓	
Grover et al. (2018)	✓	✓	✓
NeuPL			✓

Table 1: Capabilities of each policy representation approach.

embedding. Each policy embedding $\mathbf{z}$ defines a policy through the mapping $I \mapsto f(I, \mathbf{z})$, where f is the shared conditional policy network and I is an information state. After training the network (which is multiple agents [Figure 4]), we sample from its latent embedding space to generate new populations of policies.

3.2 Methods for Learning Embeddings

Here, we present several methods for learning vector embeddings that represent policies. Some methods we present are simple and naive, and others are comparatively more complex. We assume that policies are neural networks. In this work, we use neural networks which are MLPs with 3 hidden layers of size 256, and we train them (when needed) via PPO (Schulman et al., 2017).

A method for learning embeddings can encode policies to embeddings, decode embeddings to policies, or both. An overview of the methods we consider is in Table 1.

Weight Autoencoder (Reconstruction Loss) The first naive method is to autoencode the weights of the policy neural network.[2] The weight autoencoder (Figure 2) takes as input the flattened vector of policy weights and reconstructs the weights from a bottleneck layer. Given an encoder $f_\phi : \mathbb{R}^d \to \mathbb{R}^k$ and decoder $g_\psi : \mathbb{R}^k \to \mathbb{R}^d$, we obtain embeddings $\mathbf{z} = f_\phi(\mathbf{W})$ and minimize the reconstruction loss:

$$\mathcal{L}_{\text{WAE}} = \|\mathbf{W} - g_\psi(f_\phi(\mathbf{W}))\|_2^2.$$

While a policy *is* (in a sense) its parameter vector, making weight autoencoders a natural baseline, we hypothesize this approach will perform poorly: behavior is difficult to predict from raw weights, and the large input/output dimension forces either an intractably large autoencoder or an overly compressed bottleneck.

We note that the decoder is a hypernetwork (Ha et al., 2016) (a neural network that outputs a neural network). In this work, we simply use an MLP for the autoencoder. However, it is plausible that state-of-the-art hypernetwork architectures may make this type of method more feasible (Chauhan et al., 2024).

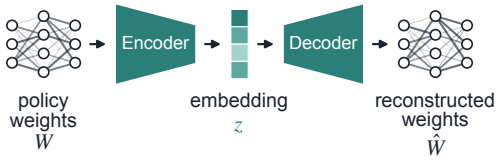


Figure 2: Weight autoencoder

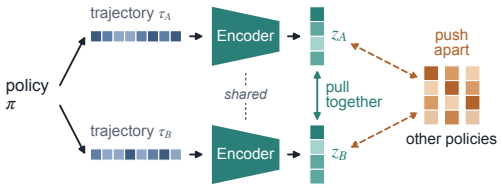
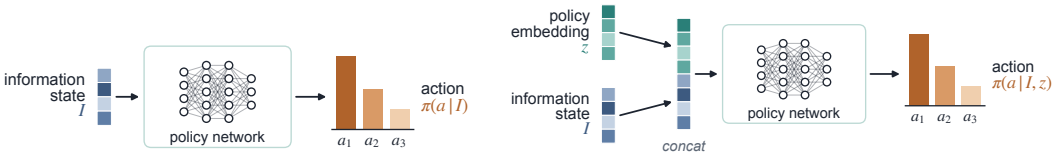


Figure 3: Trajectory encoder

Functional Autoencoder The functional autoencoder[3] uses the same encoder-decoder architecture as the weight autoencoder, but instead of reconstructing weights directly, it evaluates both the original and reconstructed networks on randomly sampled information states $I \sim \mathbb{I}$. Let $\hat{\mathbf{W}} =$

[2]We colloquially refer to all the parameters of the neural network (both weights and biases) as "weights".

[3]Pictured in Appendix B



(a) Policy network (b) Conditional policy network

Figure 4: A standard policy network (left) maps an information state I to a distribution over actions. A conditional policy network (right) additionally takes a policy embedding z, so a single network can represent many policies.

$g_\psi(f_\phi(\mathbf{W}))$ denote the reconstructed weights. The loss minimizes the KL divergence between the action distributions:

$$\mathcal{L}_{\text{FAE}} = \mathbb{E}_{I\sim\mathbb{I}}\Big[D_{\text{KL}}\big(\pi_{\mathbf{W}}(\cdot \mid I) \parallel \pi_{\hat{\mathbf{W}}}(\cdot \mid I)\big)\Big].$$

In this way, the autoencoder reconstructs the *behavior* of the original network rather than the weights themselves.

Trajectory Encoder To capture the behavioral semantics of policies rather than their internal network architecture, we use a trajectory encoder (Figure 3) trained with contrastive learning inspired by SimCLR (Chen et al., 2020). A trajectory $\tau = (o_0, a_0, r_0, \ldots, o_T, a_T, r_T)$ is collected by rolling out a policy against a randomly sampled opponent from the population. An encoder h_ϕ maps variable-length trajectories to a fixed-dimensional latent space: $\mathbf{z} = h_\phi(\tau)$.

For a mini-batch of N policies, we sample two trajectories per policy (against different opponents), yielding $2N$ embeddings. Let $\mathbf{z}_i$ and $\mathbf{z}_j$ be a positive pair (two trajectories from the same policy). We optimize the normalized temperature-scaled cross-entropy (NT-Xent) loss:

$$\mathcal{L}_{\text{NT-Xent}} = -\frac{1}{2N}\sum_{(i,j)}\log\frac{\exp\big(\text{sim}(\mathbf{z}_i,\mathbf{z}_j)/t\big)}{\sum_{\substack{k=1\\k\neq i}}^{2N}\exp\big(\text{sim}(\mathbf{z}_i,\mathbf{z}_k)/t\big)},$$

where $\text{sim}(\cdot,\cdot)$ denotes cosine similarity and t is a temperature parameter. This objective ensures embeddings are invariant to stochastic environmental noise yet discriminative of strategic intent.

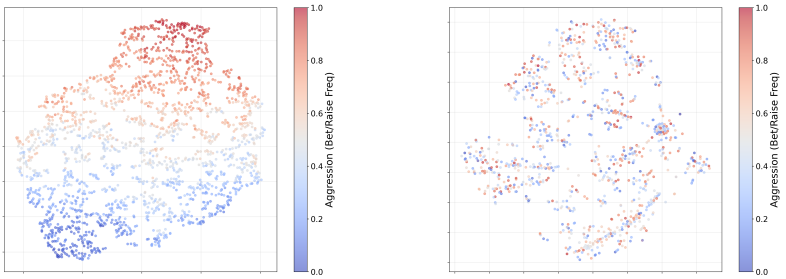
NeuPL-style Conditional Embeddings Training NeuPL induces an embedding space in which any vector can be decoded into a policy via the conditional network (Figure 4b). [4] Unlike the other methods, this generates the population and embeddings jointly rather than post-hoc.

Grover et al. Encoder Following Grover et al. (2018), we implement a hybrid encoder-decoder that learns policy embeddings from episode data. An MLP encoder f_θ processes individual (observation, action) pairs and averages the resulting vectors across the episode to form the embedding $\mathbf{z} = \frac{1}{T}\sum_{t=1}^{T} f_\theta(o_t, a_t)$. A conditional policy network $\pi_{\phi,\theta}(\mathbf{a}|o,\mathbf{z})$ predicts the agent's actions given an observation and the embedding. The model is trained with a hybrid loss combining an imitation (behavioral cloning) term and a triplet-based agent identification term. It differs from the trajectory encoder in both architecture (Transformer vs. MLP with mean pooling) and training objective (contrastive NT-Xent vs. hybrid imitation and triplet identification).

Tabular As a comparison, we compute each agent's complete action distribution at every information state in the game by querying the policy network. Concatenating these distributions yields a fixed-dimensional vector that fully describes the policy's behavior. This requires no training.

Identity We can also test the usefulness of using the weights of the neural network directly, without embedding them to a lower-dimensional representation. For some tasks, this very high-dimensional representation is too expensive to be tractable.

[4] Architecturally, this is similar to the imitation learning embeddings (`Emb-Im`) from Grover et al. (2018), but the embeddings and policy parameters are learned from reinforcement learning instead of imitation learning.



(a) Trajectory-encoder embeddings (b) Weight autoencoder embeddings

Figure 5: t-SNE visualization of embeddings in Kuhn Poker for 1000 random agents, colored by aggression level (bet/raise frequency). With the trajectory encoder, aggressive strategies (red, top) clearly separated from passive strategies (blue, bottom). With the weight autoencoder, no such patterns emerge.

3.3 Downstream Tasks

In this section, we introduce a suite of downstream tasks to evaluate the usefulness of policy representations.

Task A (Payoff Prediction vs. Uniform Random). We test how well a linear predictor, given the embedding of a policy $\mathbf{z}_1$, can predict the policy's expected payoff against a static opponent that plays uniformly random (π_{rand}):

$$\hat{y}_A = \mathbf{w}_A^\top \mathbf{z}_1 + b_A \approx \mathbb{E}_{\pi_1, \pi_{\text{rand}}}\left[\sum_{t=0}^{t_{\max}} \mathcal{R}(\mathbf{s}^t, \mathbf{a}^t)\right].$$

Task B (Payoff Prediction: policy vs. policy).

How well can a simple predictor, given the embeddings of a player 1 policy and a player 2 policy, predict the expected payoff of the two policies?

The predictor is a bilinear probe on the two agents' embeddings:

$$\hat{y}_B = \mathbf{z}_1^\top W_B \, \mathbf{z}_2 + \mathbf{u}_1^\top \mathbf{z}_1 + \mathbf{u}_2^\top \mathbf{z}_2 + b_B \approx \mathbb{E}_{\pi_1, \pi_2}\left[\sum_{t=0}^{t_{\max}} \mathcal{R}(\mathbf{s}^t, \mathbf{a}^t)\right],$$

where $\mathbf{z}_1, \mathbf{z}_2 \in \mathbb{R}^d$, and the learned parameters are the interaction matrix $W_B = P^\top Q$ with $P, Q \in \mathbb{R}^{r \times d}$ (so $\operatorname{rank}(W_B) \leq r$; we use $r = 32$), the per-player weight vectors $\mathbf{u}_1, \mathbf{u}_2 \in \mathbb{R}^d$, and the scalar bias b_B. We use a bilinear probe instead of a linear probe on the concatenated embeddings $[\mathbf{z}_1; \mathbf{z}_2]$ (the special case $W_B = 0$) because a linear probe can only express how strong each policy is on its own, not how the two policies fare against each other.

This task tests whether the latent spaces capture information about how specific strategies interact, which can be used to find good strategies (see Task F), or as a part of lookahead search (e.g. with multi-valued states (Brown et al., 2018; Kubicek & Lisý, 2026; Milec et al., 2025)).

Task C (Exploitability Prediction). How well can a linear predictor, given the embedding of a policy, predict the payoff of an opponent's best-response to that policy? We compute ground-truth best-response values exactly, which is feasible since the games we use are small enough.

For a given Player 1 policy π_1, the best-response value is defined as $\text{BRV}(\pi_1) = \max_{\pi_2} \mathbb{E}_{\pi_1, \pi_2}[-\sum_t \mathcal{R}(\mathbf{s}^t, \mathbf{a}^t)]$. A linear probe predicts this from the embedding:

$$\hat{y}_C = \mathbf{w}_C^\top \mathbf{z}_1 + b_C \approx \text{BRV}(\pi_1).$$

Task D (Zero-shot best-response). For a given Player 1 policy π_1 with embedding $\mathbf{z}_1$, we train a zero-shot best-responder which, given $\mathbf{z}_1$, approximates a best-response against π_1. The zero-shot best-responder is itself a conditional policy neural network (Figure 4b), which we train via

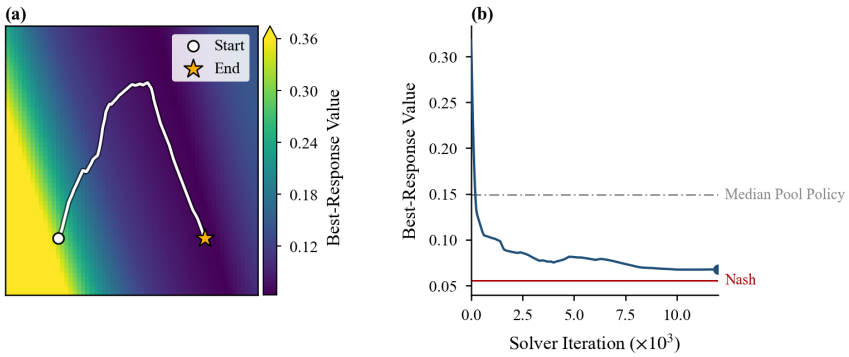


Figure 6: Task F: An example run on Kuhn poker with NeuPL embeddings. Player 1's embedding descends predicted payoffs in latent space towards an equilibrium. **Left**: the path taken by Player 1's embedding, projected onto a 2D slice colored by the exact exploitability (BRV) of the decoded policies. **Right**: the BRV decreases over training.

reinforcement learning to maximize its expected return, while receiving the opponent's embedding as input. Conditioning on the opponent's embedding was previously explored in opponent modeling research such as Grover et al. (2018, Figure 2).

Task E (Agent identification). Following Grover et al. (2018), we evaluate whether the embeddings are informative enough to identify which policy generated a given trajectory. For each of N policies, we collect multiple held-out trajectories and embed them independently. A classifier (k-nearest neighbors or linear model) is trained to predict the policy ID from the embedding.

Task F (Equilibrium-finding in latent space). After training a value function like in Task B, we can use it to try to find an approximate equilibrium in the embedding space.[5] Specifically, we use gradient ascent-descent in the embedding space by moving each embedding in the direction of its (positive or negative) gradient with respect to the predicted payoff. Although the payoff space induced by the value function is non-concave and it is thus difficult to find an equilibrium (Daskalakis et al., 2020; Cai et al., 2025), this task (A) showcases the practical application of having good policy representations, and (B) allows for an apples-to-apples comparison of (policy pool, embedding method) pairs.

Figure 6 illustrates an example run of Task F.

4 Experimental Results

We test our methods on two zero-sum IIGs: Kuhn Poker (12 information states) and Leduc Poker (936). In Appendix E.2 we report preliminary results on two larger games: Liar's Dice (1 die, 4 sides; ∼1024), and Phantom Tic-Tac-Toe (∼500K). Rules for all games can be found in Appendix H.

We report results on randomly initialized policies (except NeuPL results, which use NeuPL policies). Results on PSRO policies are in Appendix D.

We evaluate the quality of our learned representations by training a simple predictor on the embeddings for the downstream payoff prediction tasks defined in Section 3.3. A qualitative visualization of the latent space is presented in Figure 5.

Task A and B Results. We evaluate the learned embeddings from the embedding methods on the payoff prediction tasks (A and B). The results are in Table 2. Neither the identity nor the weight encoders improve over simply predicting the mean target of the training set.

When using the identity, where the representation is the raw policy weights (∼135K dims for Kuhn, ∼140K for Leduc), the bilinear probe has millions of parameters and overfits badly, illustrating why a compressed representation is needed.

[5]Unlike Task B, we use an MLP (not bilinear) value function. More details on Task F in Appendix K

		Kuhn Poker			Leduc Poker		
Task	**Encoder**	**MSE**	**Baseline**	**Imp. (%)**	**MSE**	**Baseline**	**Imp. (%)**
Task A (Fixed)	Identity	0.097 ±0.006	0.083 ±0.005	-16.9%	0.374 ±0.042	0.339 ±0.050	-10.3%
	Weight	0.081 ±0.015	0.081 ±0.016	0.0%	0.350 ±0.096	0.350 ±0.091	0.0%
	Functional	0.080 ±0.010	0.080 ±0.010	0.0%	0.462 ±0.009	0.462 ±0.009	0.0%
	Tabular	0.022 ±0.003	0.081 ±0.016	**71.9%**	0.266 ±0.037	0.350 ±0.091	**22.0%**
	Trajectory	0.016 ±0.003	0.074 ±0.009	**78.3%**	0.197 ±0.009	0.316 ±0.067	**37.6%**
	Grover	0.015 ±0.004	0.074 ±0.009	**80.0%**	0.227 ±0.031	0.336 ±0.035	**32.3%**
	NeuPL	0.0183 ±0.0012	0.0230 ±0.0012	**20.4%**	0.117 ±0.019	0.142 ±0.023	**17.9%**
Task B (Pair)	Identity	2.795 ±2.223	0.123 ±0.004	-2169.5%	0.793 ±0.131	0.764 ±0.105	-3.6%
	Weight	0.143 ±0.015	0.131 ±0.015	-9.3%	0.650 ±0.092	0.586 ±0.064	-11.0%
	Functional	0.151 ±0.008	0.128 ±0.005	-17.2%	0.576 ±0.089	0.525 ±0.042	-9.6%
	Tabular	0.0030 ±0.0003	0.1322 ±0.0083	**97.8%**	0.115 ±0.007	0.554 ±0.038	**79.2%**
	Trajectory	0.064 ±0.006	0.137 ±0.010	**53.1%**	0.453 ±0.026	0.617 ±0.026	**26.6%**
	Grover	0.013 ±0.001	0.134 ±0.008	**90.5%**	0.203 ±0.032	0.603 ±0.049	**66.3%**
	NeuPL	0.0033 ±0.0006	0.0042 ±0.0008	**21.3%**	0.020 ±0.003	0.039 ±0.009	**49.3%**
Task C	Identity	0.304 ±0.007	0.052 ±0.002	-485.7%	10.559 ±0.326	1.149 ±0.147	-818.9%
	Weight	0.058 ±0.010	0.056 ±0.007	-4.5%	1.511 ±0.086	1.440 ±0.006	-4.9%
	Functional	0.056 ±0.002	0.056 ±0.002	1.0%	1.245 ±0.224	1.246 ±0.306	0.1%
	Tabular	0.008 ±0.001	0.057 ±0.002	**86.4%**	0.179 ±0.031	1.024 ±0.104	**82.5%**
	Trajectory	0.007 ±0.001	0.056 ±0.002	**86.7%**	0.486 ±0.028	1.246 ±0.306	**61.0%**
	Grover	0.009 ±0.001	0.056 ±0.002	**84.0%**	0.415 ±0.005	1.246 ±0.306	**66.7%**
	NeuPL	0.010 ±0.001	0.010 ±0.001	**3.2%**	0.060 ±0.006	0.138 ±0.014	**56.2%**

Table 2: Performance on payoff prediction. Each method was evaluated on randomly initialized policies, except NeuPL, which used NeuPL policies. *Imp.* (improvement) denotes reduction in MSE from baseline to the method. Negative improvement occurs when the method does *worse* than the baseline (e.g. due to overfitting). ± denotes std over 3 random seeds.

		Kuhn Poker				Leduc Poker			
Encoder	**Task**	**Payoff**	**Baseline**	**Imp.**	**Upp.**	**Payoff**	**Baseline**	**Imp.**	**Upp.**
Identity	Task D	0.253 ±0.041	0.256 ±0.092	-0.003	0.497 ±0.015	OOM	OOM	OOM	OOM
Weight	Task D	0.472 ±0.029	0.412 ±0.127	+0.060	0.485 ±0.034	2.124 ±0.153	2.292 ±0.099	-0.168	3.172 ±0.182
NeuPL	Task D	0.085 ±0.012	0.038 ±0.043	+0.047	0.179 ±0.010	0.260 ±0.022	0.080 ±0.113	+0.180	0.924 ±0.015

Table 3: Performance on Task D. *Payoff* is the empirical payoff of the best-responder to embedded policies. *Baseline* is the empirical payoff of the best-responder when it is trained with random embeddings. *Imp.* denotes the improvement from Baseline to Payoff (positive improvement is good). *Upp.* denotes the upper bound: the average of the true best-response value for each embedded policy.

Task C and D results. In Task C, we embed Player 1 policies, and learn to predict the payoffs of Player 2's best-response to each Player 1 policy. Neither the identity baseline nor the weight reconstruction autoencoder perform better than predicting the mean on the dataset of random policies. NeuPL does improve over the baseline on the dataset of NeuPL policies.

In Task D (Table 3), we similarly embed Player 1 policies and train a Player 2 zero-shot best-responder with PPO to best-respond to any given embedding. NeuPL outperforms the naive baseline.[6]

Task E: Trajectory Encoder vs. Grover. We compare against Grover et al. (2018), the most closely related prior work, which learns embeddings from episode data using an MLP encoder with a hybrid generative-discriminative objective (imitation + triplet identification). Compared to our trajectory

Encoder	**Classifier**	**Kuhn Poker Top-1 (%)**	**Leduc Poker Top-1 (%)**
Trajectory	k-NN	**39.9 ± 1.3**	**45.3 ± 1.8**
Trajectory	Linear	**47.9 ± 0.6**	**57.8 ± 0.9**
Grover	k-NN	24.8 ± 0.1	34.0 ± 2.0
Grover	Linear	25.7 ± 0.5	38.9 ± 0.7

Table 4: Task E: Agent identification top-1 accuracy for 500 eval agents using 5 train / 5 test trajectories per agent. ± denotes std over 3 seeds. Choosing at random achieves 0.2%.

[6]It is difficult to compare NeuPL to the other methods, since the population of policies used for training and evaluation is different than for the other methods.

Task	Encoder	Kuhn Poker		Leduc Poker	
		BRV	Pool median	BRV	Pool median
Task F	Weight (random)	0.415 ±0.006	0.407 ±0.005	2.587 ±0.026	2.946 ±0.031
	Weight (PSRO)	0.414 ±0.000	0.289 ±0.004	2.631 ±0.001	1.170 ±0.000
	Functional (random)	0.389 ±0.089	0.407 ±0.005	2.862 ±0.038	2.946 ±0.031
	Functional (PSRO)	0.263 ±0.003	0.289 ±0.004	1.113 ±0.020	1.170 ±0.000
	NeuPL	0.127 ±0.063	0.150 ±0.002	0.661 ±0.016	1.263 ±0.025

Table 5: Task F results for Player 1. Lower BRV (best-response value) is better. Nash equilibrium BRV is 0.056 for Kuhn, and 0.086 for Leduc. Pool median denotes the median BRV of a sampled embedding. NeuPL definitively finds better policies than the median policy in Leduc poker.

encoder, the two approaches show complementary strengths. On payoff prediction (Table 2), Grover's hybrid objective is better at or is competitive in Task A. However, on Task E agent identification, our contrastive Transformer encoder substantially outperforms Grover, achieving 47.9% vs. 25.7% top-1 accuracy on Kuhn and 57.8% vs. 38.9% on Leduc with a linear classifier (Table 4).[7]

Task F: equilibrium finding in latent space. The results for Task F are presented in Table 5. As opposed to the other tasks, Task F is not predicting an aspect of an embedding from the pool – the goal is simply to produce the lowest-exploitability policy. Thus, unlike the other tasks, the results can be interpreted as apples-to-apples from one (dataset, method) pair to another.

The results show that decodable policy embeddings can support equilibrium finding directly in latent space: on Leduc poker, gradient-based search over NeuPL embeddings reduces exploitability from the starting median of 1.263 to 0.661. Although this is far from the Nash, it suggests that the embeddings do contain strategic information. Search over weight- or functional-autoencoder embeddings improve only marginally or not at all, suggesting that behavior-grounded latent structure is necessary.

5 Conclusion

We studied self-supervised learning of policy embeddings for two-player zero-sum imperfect-information games. We compared weight-space, behavioral, and NeuPL conditioning embeddings on a suite of downstream tasks, ranging from linear probes of payoff and exploitability to zero-shot best response and equilibrium search in embedding space.

What an embedding is trained on matters more than its size. Representations built from raw or autoencoded weights don't beat the predict-the-mean baseline. Behavioral encoders cut exploitability-prediction error by up to 87% on Kuhn poker and 67% on Leduc poker.

Different self-supervised objectives capture different information. The contrastive trajectory encoder is best at identifying which agent produced a trajectory (57.8% vs. 38.9% top-1 on Leduc). The imitation-based objective of Grover et al. (2018) better captures pairwise payoff structure (66.3% vs. 26.6% improvement over baseline on Leduc).

Decodable embeddings support search in latent space. On Leduc poker, gradient-based equilibrium search over NeuPL embeddings descends exploitability, demonstrating the viability of embeddings for strategy in games.

Limitations. Our experiments use small games. NeuPL embeddings are evaluated on NeuPL's own population, so they are not directly comparable to the other methods. Latent equilibrium search remains far from Nash.

Future work. The clearest gap is encoders that are both payoff-aware and decodable, so that the representations that best predict payoffs can also support search. Integrating such embeddings with learned belief-state representations is a natural next step toward decision-time planning in larger games.

[7] Results on larger games in Appendix E.3

6 AI USE DISCLOSURE

We used generative AI tools to help with writing code, running experiments, creating figures, drafting the research paper, creating tools and visualizations, and hyperparameter tuning.

REFERENCES

Stefano V. Albrecht and Peter Stone. Autonomous Agents Modelling Other Agents: A Comprehensive Survey and Open Problems. *Artificial Intelligence*, 258:66–95, May 2018. ISSN 00043702. doi: 10.1016/j.artint.2018.01.002. URL http://arxiv.org/abs/1709.08071. arXiv:1709.08071 [cs].

Karl Johan Åström. Optimal control of markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10(1):174–205, 1965. doi: 10.1016/0022-247X(65)90154-X.

Randall Balestriero, Mark Ibrahim, Vlad Sobal, Ari Morcos, Shashank Shekhar, Tom Goldstein, Florian Bordes, Adrien Bardes, Gregoire Mialon, Yuandong Tian, et al. A cookbook of self-supervised learning. *arXiv preprint arXiv:2304.12210*, 2023.

Noam Brown and Tuomas Sandholm. Superhuman AI for multiplayer poker. *Science*, 365(6456):885–890, August 2019. doi: 10.1126/science.aay2400. URL https://www.science.org/doi/full/10.1126/science.aay2400.

Noam Brown, Tuomas Sandholm, and Brandon Amos. Depth-Limited Solving for Imperfect-Information Games, May 2018. URL http://arxiv.org/abs/1805.08195. arXiv:1805.08195 [cs].

Noam Brown, Anton Bakhtin, Adam Lerer, and Qucheng Gong. Combining Deep Reinforcement Learning and Search for Imperfect-Information Games, November 2020. URL http://arxiv.org/abs/2007.13544. arXiv:2007.13544 [cs].

Yang Cai, Constantinos Daskalakis, Haipeng Luo, Chen-Yu Wei, and Weiqiang Zheng. On Tractable $$-Equilibria in Non-Concave Games, April 2025. URL http://arxiv.org/abs/2403.08171. arXiv:2403.08171 [cs.GT].

Vinod Kumar Chauhan, Jiandong Zhou, Ping Lu, Soheila Molaei, and David A. Clifton. A Brief Review of Hypernetworks in Deep Learning. *Artificial Intelligence Review*, 57(9):250, August 2024. ISSN 1573-7462. doi: 10.1007/s10462-024-10862-8. URL http://arxiv.org/abs/2306.06955. arXiv:2306.06955 [cs].

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pp. 1597–1607. PmLR, 2020.

Anne Condon. *On Algorithms for Simple Stochastic Games*, volume 13 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, Providence, Rhode Island, 1990. ISBN 978-0-8218-6597-2 978-1-4704-3971-2. doi: 10.1090/dimacs/013. URL http://www.ams.org/dimacs/013.

Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In *Computers and Games, 5th International Conference, CG 2006*, pp. 72–83. Springer, 2006.

Constantinos Daskalakis, Stratis Skoulakis, and Manolis Zampetakis. The Complexity of Constrained Min-Max Optimization, September 2020. URL http://arxiv.org/abs/2009.09623. arXiv:2009.09623 [cs.CC].

Constantinos Daskalakis, Dylan J. Foster, and Noah Golowich. Independent Policy Gradient Methods for Competitive Reinforcement Learning, January 2021. URL http://arxiv.org/abs/2101.04233. arXiv:2101.04233 [cs].

Ian Gemp, Rahul Savani, Marc Lanctot, Yoram Bachrach, Thomas Anthony, Richard Everett, Andrea Tacchetti, Tom Eccles, and János Kramár. Sample-based Approximation of Nash in Large Many-Player Games via Gradient Descent, February 2022. URL http://arxiv.org/abs/2106.01285. arXiv:2106.01285 [cs.GT].

Denizalp Goktas and Amy Greenwald. Exploitability Minimization in Games and Beyond. In *Advances in Neural Information Processing Systems*, volume 35, pp. 4857–4873. Curran Associates, Inc., 2022. doi: 10.52202/068431-0351. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/1f3b0b15d6bb860dcfa6e5c8ba7d3d96-Abstract-Conference.html.

Aditya Grover, Maruan Al-Shedivat, Jayesh K Gupta, Yuri Burda, and Harrison Edwards. Learning policy representations in multiagent systems. In *International Conference on Machine Learning*, pp. 1802–1811. PMLR, 2018.

David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

He He, Jordan Boyd-Graber, Kevin Kwok, and Hal Daumé III. Opponent modeling in deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1804–1813. PMLR, 2016.

Vojtěch Kovařík, Dominik Seitz, Viliam Lisý, Jan Rudolf, Shuo Sun, and Karel Ha. Value Functions for Depth-Limited Solving in Zero-Sum Imperfect-Information Games, March 2022. URL http://arxiv.org/abs/1906.06412. arXiv:1906.06412 [cs].

Ondrej Kubicek and Viliam Lisý. Look-ahead Reasoning with a Learned Model in Imperfect Information Games. *International Conference on Learning Representations*, 2026:53533–53547, April 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/57d7e7e1593ad1ab6818c258fa5654ce-Abstract-Conference.html.

Harold W Kuhn. A simplified two-person poker. *Contributions to the Theory of Games*, 1:97–103, 2016.

Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Marc Lanctot, Edward Lockhart, Jean-Baptiste Lespiau, Vinicius Zambaldi, Satyaki Upadhyay, Julien Pérolat, Sriram Srinivasan, Finbarr Timbers, Karl Tuyls, Shayegan Omidshafiei, Daniel Hennes, Dustin Morrill, Paul Muller, Timo Ewalds, Ryan Faulkner, János Kramár, Bart De Vylder, Brennan Saeta, James Bradbury, David Ding, Sebastian Borgeaud, Matthew Lai, Julian Schrittwieser, Thomas Anthony, Edward Hughes, Ivo Danihelka, and Jonah Ryan-Davis. OpenSpiel: A Framework for Reinforcement Learning in Games. *CoRR*, abs/1908.09453, 2019. URL http://arxiv.org/abs/1908.09453. _eprint: 1908.09453.

Benjamin Li, Shuyang Shi, Lucia Romero, Huao Li, Yaqi Xie, Woojun Kim, Stefanos Nikolaidis, Michael Lewis, Katia Sycara, and Simon Stepputtis. Adaptively Coordinating with Novel Partners via Learned Latent Strategies, November 2025. URL http://arxiv.org/abs/2511.12754. arXiv:2511.12754 [cs.AI] version: 1.

Siqi Liu, Luke Marris, Daniel Hennes, Josh Merel, Nicolas Heess, and Thore Graepel. Neupl: Neural population learning. *arXiv preprint arXiv:2202.07415*, 2022.

Siqi Liu, Luke Marris, Marc Lanctot, Georgios Piliouras, Joel Z. Leibo, and Nicolas Heess. Neural Population Learning beyond Symmetric Zero-sum Games, January 2024. URL https://arxiv.org/abs/2401.05133v1.

Edward Lockhart, Marc Lanctot, Julien Pérolat, Jean-Baptiste Lespiau, Dustin Morrill, Finbarr Timbers, and Karl Tuyls. Computing Approximate Equilibria in Sequential Adversarial Games by Exploitability Descent, June 2020. URL http://arxiv.org/abs/1903.05614. arXiv:1903.05614 [cs].

Andrei Lupu, Brandon Cui, Hengyuan Hu, and Jakob Foerster. Trajectory diversity for zero-shot coordination. In *International Conference on Machine Learning*, pp. 7204–7213. PMLR, 2021.

Wenhao Ma, Yu-Cheng Chang, Jie Yang, Yu-Kai Wang, and Chin-Teng Lin. Contrastive learning-based agent modeling for deep reinforcement learning. *arXiv preprint arXiv:2401.00132*, 2024.

Carlos Martin and Tuomas Sandholm. ApproxED: Approximate exploitability descent via learned best responses, June 2024. URL http://arxiv.org/abs/2301.08830. arXiv:2301.08830 [cs.GT].

Reid McIlroy-Young, Russell Wang, Siddhartha Sen, Jon Kleinberg, and Ashton Anderson. Learning Models of Individual Behavior in Chess. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 1253–1263, August 2022. doi: 10.1145/3534678.3539367. URL http://arxiv.org/abs/2008.10086. arXiv:2008.10086 [cs].

David Milec, Vojtěch Kovařík, and Viliam Lisý. Adapting Beyond the Depth Limit: Counter Strategies in Large Imperfect Information Games, February 2025. URL http://arxiv.org/abs/2501.10464. arXiv:2501.10464 [cs.GT] version: 3.

Robert J. Moss, Anthony Corso, Jef Caers, and Mykel J. Kochenderfer. BetaZero: Belief-State Planning for Long-Horizon POMDPs using Learned Approximations, July 2024. URL http://arxiv.org/abs/2306.00249. arXiv:2306.00249 [cs] version: 4.

Samer Nashed and Shlomo Zilberstein. A Survey of Opponent Modeling in Adversarial Domains. *Journal of Artificial Intelligence Research*, 73:277–327, January 2022. ISSN 1076-9757. doi: 10.1613/jair.1.12889. URL https://www.jair.org/index.php/jair/article/view/12889.

Georgios Papoudakis and Stefano V. Albrecht. Variational Autoencoders for Opponent Modeling in Multi-Agent Systems, January 2020. URL http://arxiv.org/abs/2001.10829. arXiv:2001.10829 [cs.LG].

Max Rudolph, Nathan Lichtle, Sobhan Mohammadpour, Alexandre Bayen, J Zico Kolter, Amy Zhang, Gabriele Farina, Eugene Vinitsky, and Samuel Sokota. Reevaluating policy gradient methods for imperfect-information games. *arXiv preprint arXiv:2502.08938*, 2025.

Martin Schmid, Matej Moravcik, Neil Burch, Rudolf Kadlec, Josh Davidson, Kevin Waugh, Nolan Bard, Finbarr Timbers, Marc Lanctot, G Zacharias Holland, Elnaz Davoodi, Alden Christianson, and Michael Bowling. Student of games: A unified learning algorithm for both perfect and imperfect information games. *Science Advances*, 9(46), 2023.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, August 2017. URL http://arxiv.org/abs/1707.06347. arXiv:1707.06347 [cs].

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, January 2016. ISSN 1476-4687. doi: 10.1038/nature16961. URL https://www.nature.com/articles/nature16961%7D.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550(7676):354–359, October 2017. ISSN 1476-4687. doi: 10.1038/nature24270. URL https://www.nature.com/articles/nature24270. Number: 7676.

Samuel Sokota, Ryan D'Orazio, Chun Kai Ling, David J. Wu, J. Zico Kolter, and Noam Brown. Abstracting Imperfect Information Away from Two-Player Zero-Sum Games, July 2023. URL http://arxiv.org/abs/2301.09159. arXiv:2301.09159 [cs].

Samuel Sokota, Eugene Vinitsky, Hengyuan Hu, J. Zico Kolter, and Gabriele Farina. Superhuman AI for Stratego Using Self-Play Reinforcement Learning and Test-Time Search, November 2025. URL http://arxiv.org/abs/2511.07312. arXiv:2511.07312 [cs.LG].

Christopher Solinas, Radovan Haluska, David Sychrovsky, Finbarr Timbers, Nolan Bard, Michael Buro, Martin Schmid, Nathan R. Sturtevant, and Michael Bowling. Neural Bayesian Filtering, October 2025a. URL http://arxiv.org/abs/2510.03614. arXiv:2510.03614 [cs.LG].

Christopher Solinas, Radovan Haluška, Pavel Yanushonak, David Sychrovský, Michael Buro, Martin Schmid, Nathan R. Sturtevant, and Michael Bowling. Belief Embedding Tree Search. July 2025b. URL https://openreview.net/forum?id=ejeaOWLqbo.

Thomas Unterthiner, Daniel Keysers, Sylvain Gelly, Olivier Bousquet, and Ilya Tolstikhin. Predicting neural network accuracy from weights. In *arXiv preprint arXiv:2002.11448*, 2020.

Conor Wallace, Umer Siddique, and Yongcan Cao. TransAM: Transformer-based agent modeling for multi-agent systems via local trajectory encoding. In *Reinforcement Learning Conference*, 2025.

Annie Xie, Dylan P Losey, Ryan Tolsma, Chelsea Finn, and Dorsa Sadigh. Learning latent representations to influence multi-agent interaction. In *Conference on Robot Learning*, pp. 575–588. PMLR, 2020.

Brian Hu Zhang and Tuomas Sandholm. General search techniques without common knowledge for imperfect-information games, and application to superhuman Fog of War chess, February 2026. URL http://arxiv.org/abs/2506.01242. arXiv:2506.01242 [cs.GT].

Luisa Zintgraf, Sam Devlin, Kamil Ciosek, Shimon Whiteson, and Katja Hofmann. Deep Interactive Bayesian Reinforcement Learning via Meta-Learning, April 2022. URL http://arxiv.org/abs/2101.03864. arXiv:2101.03864 [cs.LG].

Michal Šustr, Vojtěch Kovařík, and Viliam Lisý. Particle Value Functions in Imperfect Information Games. 2021.

A Details on Randomly Initializing Policy Neural Networks

Our layer initialization initializes weights orthogonally with a scaling factor of 2.2. It initializes the biases to 0, except for the last layer (whose outputs are the action logits), for which each bias is randomly initialized between -1 and 1.

This is in contrast to PyTorch's default layer initialization for linear layers, where weights are initialized according to a Kaiming uniform distribution, and biases are initialized randomly between $\sqrt{-b}, \sqrt{b}$, where b is the fan-in of the neuron.

Table 6 shows the mean and standard deviation of policy sets generated via different methods. we used 1000 policies randomly initialized via PyTorch default, 1000 policies randomly initialized with our method, 1000 policies generated by interpolating NeuPL embeddings, and 611 policies generated by running PSRO multiple times.

We train NeuPL with a population size of 22 per player. To generate each of the 1000 random policies, we randomly pick 2 of the agents, and interpolate uniformly at random to a convex combination of their embeddings. In the main experiments, however, we fit a multivariate Gaussian to the embeddings and then sample from the Gaussian.

Initialization	Payoff	$P(\text{check} \mid J)$
PyTorch default	0.124 ± 0.022	0.500 ± 0.023
Our init	$0.081 \pm \mathbf{0.258}$	$0.505 \pm \mathbf{0.311}$
PSRO	0.253 ± 0.096	0.386 ± 0.297
NeuPL	0.145 ± 0.096	0.832 ± 0.247

Table 6: Policy datasets generated by various methods. Values are: payoffs vs. a uniform random strategy in Kuhn poker, and probability of taking the "check" action as the first player in Kuhn poker with a jack. Values are mean $\pm$ standard deviation. **Policies initialized with the default method are less diverse (they have much lower standard deviation in both values)**

B Baseline Encoder Diagrams

Illustration of functional encoder is in Fig 7.

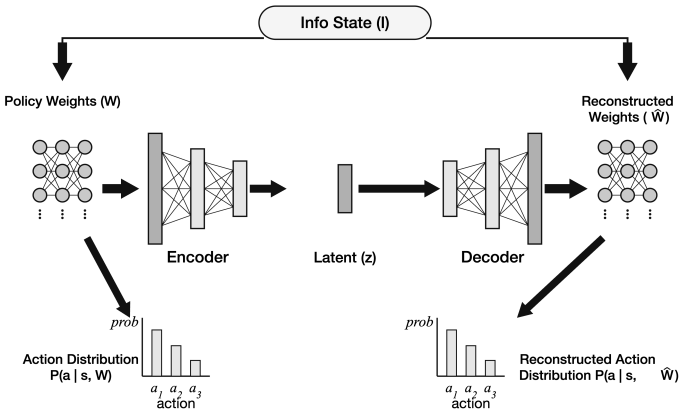


Figure 7: Functional encoder

C Learned Similarity Metric

To test whether a better similarity metric could close the opponent adaptation gap, we trained a *learned Mahalanobis projection* on Liar's Dice: a linear map $W \in \mathbb{R}^{64\times128}$ optimized on library

agents so that $\cos(Wz_i, Wz_j)$ predicts payoff correlation between agents i and j. The projection achieves r=0.70 correlation with ground-truth payoff similarity during training, confirming that it learns a meaningful transformation. However, kernel regression with the learned similarity (0.347 $\pm$ 0.030) does not outperform the simpler temperature-scaled cosine (0.351 $\pm$ 0.024) for either encoder, indicating that the bottleneck is not the similarity metric but the information content of the embeddings themselves.

D PSRO Population Results

We additionally evaluated encoders on populations generated by the Policy Space Response Oracle algorithm (PSRO) Lanctot et al. (2017). PSRO iteratively expands a population by training best-response policies via deep RL against Nash equilibrium mixtures of the current population. This produces strategically structured agents, unlike the random populations used in the main experiments.

Results on Kuhn Poker are shown in Table 7. The Trajectory Encoder maintains its advantage over weight-space methods on PSRO populations, consistent with the random population findings.

Table 7: Downstream task performance on PSRO-generated populations. $\pm$ denotes std over 3 random seeds where available.

		Kuhn Poker			Leduc Poker		
Task	**Encoder**	**MSE**	**Baseline**	**Imp. (%)**	**MSE**	**Baseline**	**Imp. (%)**
Task A (Fixed)	Identity	0.096 ±0.007	0.030 ±0.003	-217.6%	–	–	–
	Weight	0.030 ±0.003	0.029 ±0.002	-1.4%	–	–	–
	Trajectory	0.020 ±0.004	0.033 ±0.001	39.4%	0.159 ±0.050	0.213 ±0.120	25.6%
Task B (Pair)	Trajectory	0.0281	0.0348	19.4%	–	–	–
Task C	Identity	0.077 ±0.005	0.0031 ±0.0002	-2384.5%	–	–	–
	Weight	0.003609 ±0.000007	0.003276 ±0.000268	-10.2%	–	–	–
	Trajectory	0.0004 ±0.0001	0.0034 ±0.0004	87.2%	0.147 ±0.050	0.186 ±0.121	21.3%

E Additional Results

E.1 Additional Results on Tasks A, B, and C with NeuPL

Here, we show additional results in Table 8.

In the main paper, we use a linear (or bilinear) predictor to predict values from the embeddings. Here, we also experiment with using a shallow random forest. To choose the number of decision trees in the random forest, we increase the number of trees until validation loss stops improving. Also different from the main paper, here we show results with a linear predictor instead of a bilinear predictor for Task B.

In the main paper, where tasks are player-dependent, we report results on Player 1 only. Here, we also report results for Player 2.

E.2 Results on larger games (Tasks A and B)

We performed preliminary experiments on the larger games of Liar's Dice and Phantom Tic-Tac-Toe. Results are reported in Table 9.

E.3 Results on larger games (Task E: Agent Identification)

Following Grover et al. (2018), we evaluate whether the trajectory encoder embeddings can identify which policy generated a held-out trajectory. For 500 random agents, we collect 5 train and 5 test trajectories each, embed them independently, and train classifiers on the embeddings. Here, we report additional results on 2 larger games.

Table 8: NeuPL embeddings on Task A (payoff vs. a uniform-random opponent), Task B (pairwise payoff of a P_1 policy against a P_2 policy) and Task C (exploitability) with a random-forest vs. a linear downstream predictor, for each player's policy population. MSE and baseline MSE (predicting the training-set mean) are validation errors $\times 10^2$; Ratio = MSE / Baseline (lower is better; < 1 beats the baseline), with the better predictor in bold (neither when tied at two decimals). Mean $\pm$ std over 3 seeds; both predictors use the same NeuPL checkpoint per game, N=1000 sampled policies per player. Task B targets are exact payoffs for $50,000$ policy pairs, split by policy so that no validation policy appears in training.

			Random forest			Linear		
Game	Task	Player	MSE	Baseline	Ratio	MSE	Baseline	Ratio
Kuhn	A	P_1	1.57±0.23	2.00±0.28	**0.79±0.01**	1.83±0.12	2.30±0.12	0.80±0.03
		P_2	1.49±0.13	1.52±0.11	**0.98±0.01**	1.43±0.04	1.37±0.08	1.05±0.04
	B	P_1 vs. P_2	0.33±0.04	0.42±0.08	0.80±0.07	0.33±0.06	0.42±0.08	0.80±0.05
	C	P_1	0.75±0.20	0.95±0.19	**0.79±0.05**	1.00±0.15	1.04±0.14	0.97±0.02
		P_2	0.23±0.02	0.39±0.03	**0.60±0.03**	0.18±0.01	0.29±0.03	0.63±0.05
Leduc	A	P_1	12.97±0.56	15.24±0.86	0.85±0.07	11.69±1.85	14.25±2.28	**0.82±0.05**
		P_2	13.82±2.09	16.66±2.59	**0.83±0.05**	17.83±3.01	21.27±1.44	0.84±0.09
	B	P_1 vs. P_2	2.48±0.90	3.96±0.98	0.62±0.08	2.45±0.49	3.96±0.98	0.62±0.04
	C	P_1	6.52±1.29	13.83±2.14	0.47±0.02	6.03±0.55	13.77±1.43	**0.44±0.07**
		P_2	3.01±0.61	8.08±1.20	**0.37±0.02**	2.96±0.43	7.68±0.69	0.38±0.02

		Kuhn Poker			Leduc Poker			Liar's Dice			Phantom TTT		
Task	**Encoder**	**MSE**	**Baseline**	**Imp. (%)**	**MSE**	**Baseline**	**Imp. (%)**	**MSE**	**Baseline**	**Imp. (%)**	**MSE**	**Baseline**	**Imp. (%)**
Task A (Fixed)	Tabular	0.022 ±0.003	0.081 ±0.016	71.9%	0.266 ±0.037	0.350 ±0.091	22.0%	0.016 ±0.003	0.045 ±0.008	63.7%		OOM	
	Trajectory	0.016 ±0.003	0.074 ±0.009	78.3%	0.197 ±0.009	0.316 ±0.067	**37.6%**	0.019 ±0.002	0.045 ±0.008	56.6%	0.019 ±0.002	0.021 ±0.002	9.0%
	Grover	0.015 ±0.004	0.074 ±0.009	**80.0%**	0.227 ±0.031	0.336 ±0.035	32.3%	0.015 ±0.002	0.045 ±0.008	**67.2%**	0.011 ±0.001	0.021 ±0.002	**48.5%**
Task B (Pair)	Tabular	0.0030 ±0.0003	0.1322 ±0.0083	**97.8%**	0.115 ±0.007	0.554 ±0.038	**79.2%**		OOM (>32GB)			OOM	
	Trajectory	0.064 ±0.006	0.137 ±0.010	53.1%	0.453 ±0.026	0.617 ±0.026	26.6%	0.047 ±0.007	0.081 ±0.005	41.9%	0.025 ±0.001	0.044 ±0.002	43.0%
	Grover	0.013 ±0.001	0.134 ±0.008	90.5%	0.203 ±0.032	0.603 ±0.049	66.3%	0.041 ±0.005	0.081 ±0.005	**49.6%**	0.024 ±0.001	0.044 ±0.002	**45.3%**

Table 9: Payoff prediction performance for encoders evaluated on the same shared agent pools. Tabular uses the complete action distribution at every information state as a fixed-dimensional representation (24-dim for Kuhn, 2808-dim for Leduc, 9216-dim for Liar's Dice). In Kuhn, the lossless tabular representation dominates on Task B. As game complexity grows, learned encoders catch up to the tabular representation despite compressing into 64 dimensions, and in PTTT, the tabular representation is intractably large (OOM). $\pm$ denotes std over 3 random seeds.

Table 10: Task E: Agent identification top-1 accuracy for 500 eval agents using 5 train / 5 test trajectories per agent. $\pm$ denotes std over 3 random seeds. Random baseline is 0.2%.

Encoder	**Classifier**	**Kuhn Poker Top-1 (%)**	**Leduc Poker Top-1 (%)**	**Liar's Dice Top-1 (%)**	**Phantom TTT Top-1 (%)**
Trajectory	k-NN	**39.9 ± 1.3**	**45.3 ± 1.8**	**83.8 ± 0.7**	96.4 ± 0.5
Trajectory	Linear	**47.9 ± 0.6**	**57.8 ± 0.9**	**89.5 ± 0.6**	**97.5 ± 0.3**
Grover	k-NN	24.8 ± 0.1	34.0 ± 2.0	79.6 ± 2.6	96.5 ± 0.2
Grover	Linear	25.7 ± 0.5	38.9 ± 0.7	76.6 ± 2.5	95.8 ± 0.4

E.4 Our Trajectory Encoder vs. Grover

On the larger games (Liar's Dice and Phantom TTT), both encoders achieve near-perfect identification and retrieval, with diminishing gaps. On strategy classification, both encoders perform comparably. These results indicate that the contrastive objective produces more fine-grained, discriminative representations suited to identification tasks, while the hybrid imitation objective better captures the smooth payoff structure needed for regression. We hypothesize that this gap arises because Grover's imitation loss directly optimizes for encoding action distributions that determine payoffs whereas the contrastive NT-Xent objective optimizes for inter-policy discrimination on a normalized embedding

sphere, discarding magnitude information that may be relevant for regression. An ablation over training trajectories per policy (Appendix F) shows that the contrastive encoder is sample-efficient: downstream performance is stable across a $5\times$ range (10 to 50 trajectories), indicating that only modest behavioral data per agent is needed during pre-training.

Additional Identification and Retrieval Tasks. To further characterize how the two training objectives shape the embedding space, we evaluate on three additional downstream tasks: 1-shot agent identification (Task G), policy retrieval (Task H), and strategy classification (Task J). These tasks probe different aspects of representation quality, from fine-grained individual discrimination to coarse behavioral categorization. Results are shown in Table 11.

Task G (1-shot Agent Identification). A more challenging variant of Task E that tests whether the embeddings support identification from minimal behavioral evidence. Each agent provides only a single trajectory at train time (1-shot), and the classifier must identify the correct agent from a pool of 500 at test time. While Task E provides 5 training trajectories per agent, allowing the classifier to average out stochastic noise, the 1-shot setting requires the encoder to produce embeddings that are individually discriminative. This is practically motivated by settings where an opponent is encountered only briefly before a decision must be made.

Task H (Policy Retrieval). Given a query trajectory from an agent, we retrieve the most similar agents from the evaluation set using cosine similarity in embedding space. We report Recall@k for $k \in \{1, 5, 10\}$: the fraction of queries for which the correct agent appears among the top-k nearest neighbors. Unlike the classification-based Tasks E and G, retrieval evaluates the global geometric structure of the embedding space to see whether behaviorally similar policies are consistently placed near each other. This is relevant for applications such as opponent lookup, where a player wishes to find the closest match to a previously encountered opponent in a database of known strategies.

Task J (Strategy Classification). We compute a scalar aggression score for each agent, defined as the frequency of bet/raise actions across all decision points in the agent's trajectories, and discretize agents into 5 equal-frequency buckets ranging from passive to aggressive. A linear probe trained on the embeddings predicts the aggression bucket. Unlike the identity-based tasks (E and G), which require distinguishing between individual agents, strategy classification tests whether the representations capture coarse-grained behavioral categories. This complements the fine-grained tasks by asking whether the embedding space organizes policies along semantically meaningful axes.

The contrastive Trajectory Encoder consistently outperforms the Grover baseline on identification and retrieval tasks across all four games, though the gap narrows substantially on larger games. On Task E (Table 10), the contrastive encoder achieves 89.5% top-1 accuracy on Liar's Dice (Linear), compared to 76.6% for Grover. On Phantom TTT, both encoders achieve near-perfect identification ($>$95%), with the contrastive encoder maintaining a slight edge (97.5% vs. 95.8% Linear). On policy retrieval (Table 11), both encoders achieve Recall@10 $>$99% on Liar's Dice and Phantom TTT, indicating that the correct agent appears in the top 10 nearest neighbors virtually every time. The gap is particularly striking in the 1-shot setting on Kuhn and Leduc, where the contrastive encoder achieves 33.1% and 35.1% (k-NN) compared to 18.2% and 17.2% for Grover, though on Phantom TTT both converge (91.5% vs. 91.1%).

On strategy classification (Task J), both encoders perform comparably: 79.7% vs. 75.3% on Kuhn and 56.0% vs. 56.3% on Leduc. This suggests that coarse-grained strategic style (aggression level) is captured by both training objectives. The substantially higher accuracy on Kuhn compared to Leduc for both encoders reflects the structural simplicity of Kuhn Poker, where aggression (bet frequency) is a near-complete description of a policy's behavior due to the minimal game tree (3 cards, one betting round). In Leduc, with 6 cards, a community card, and two betting rounds, policies vary along many more behavioral dimensions, making a single aggression scalar a coarser summary and the classification task correspondingly harder.

Taken together, these results reveal that different SSL objectives produce representations with different epistemic content: the contrastive objective captures fine-grained identity (knowing *who*), while the hybrid objective better captures payoff structure (knowing *how much*), and both capture broad behavioral categories. This decomposition, which form of knowledge about an opponent each objective prioritizes, is consistent with their designs: NT-Xent maximizes separation between distinct policies, while the Grover hybrid loss optimizes for action-distribution fidelity.

Table 11: Additional downstream tasks comparing Trajectory and Grover encoders. 1-shot ID uses 1 train trajectory per agent (Top-1 accuracy). Retrieval reports Recall@k. Strategy classification reports accuracy over 5 aggression buckets (random baseline: 20%). $\pm$ denotes std over 3 seeds. "–" indicates tasks not applicable (strategy classification requires a game-specific style label not defined for dice/board games).

Task	Encoder	Kuhn Poker	Leduc Poker	Liar's Dice	Phantom TTT
1-shot ID (k-NN)	Trajectory	**33.1 ± 0.3**	**35.1 ± 1.0**	–	91.5 ± 1.3
	Grover	18.2 ± 1.5	17.2 ± 1.8	–	91.1 ± 0.8
1-shot ID (Linear)	Trajectory	**31.4 ± 0.7**	**34.0 ± 0.8**	–	**89.9 ± 1.4**
	Grover	16.2 ± 0.6	21.8 ± 0.7	–	88.9 ± 0.7
Retrieval R@1	Trajectory	**39.3 ± 0.5**	**45.3 ± 0.3**	**83.7 ± 1.2**	96.7 ± 0.8
	Grover	25.0 ± 0.8	30.7 ± 2.4	79.5 ± 1.0	96.5 ± 0.4
Retrieval R@5	Trajectory	**75.8 ± 1.1**	**78.1 ± 0.5**	**97.8 ± 0.2**	99.8 ± 0.1
	Grover	56.7 ± 0.5	57.6 ± 2.5	96.2 ± 0.5	99.6 ± 0.1
Retrieval R@10	Trajectory	**87.9 ± 0.2**	**87.9 ± 0.6**	**99.4 ± 0.1**	100.0 ± 0.1
	Grover	70.4 ± 1.0	70.2 ± 1.3	98.7 ± 0.4	99.9 ± 0.1
Strategy Class.	Trajectory	**79.7 ± 3.8**	56.0 ± 2.0	–	–
	Grover	75.3 ± 1.2	56.3 ± 5.9	–	–

Opponent-Adaptive Strategy Selection. To evaluate whether the learned embeddings have practical utility beyond probe-based evaluation, we design a downstream application: embedding-based opponent-adaptive strategy selection. Given a novel opponent, the task is to select the best counter-strategy from a library of agents with known pairwise payoffs.

We split the 500 evaluation agents into 50 held-out test opponents and a 450-agent response library. We use *kernel regression* to predict each library agent's payoff against the test opponent as a similarity-weighted combination of its known payoffs against all library agents, then select the agent with the highest predicted payoff. We compare against three baselines: *Oracle* (best library agent per test opponent using ground-truth payoffs), *Random* (uniformly random library agent), and *Best-Avg* (library agent with highest mean payoff across the library, equivalent to kernel regression with uniform weights). Results on Liar's Dice and Leduc Poker are shown in Table 12.

Table 12: Opponent-adaptive strategy selection. Mean payoff from a 450-agent library against 50 held-out opponents. $\pm$ std over 10 seeds.

	Liar's Dice		Leduc Poker	
Method	**Traj.**	**Grov.**	**Traj.**	**Grov.**
Oracle	0.538 ± .026		1.899 ± .077	
Best-Avg	0.343 ± .024		0.923 ± .157	
Random	−0.006 ± .030		−0.221 ± .056	
Kernel (τ=1)	0.348 ± .020	0.343 ± .025	0.903 ± .123	0.888 ± .151
Kernel (τ=5)	**0.351 ± .024**	0.344 ± .024	**0.935 ± .132**	0.897 ± .123

On Liar's Dice, kernel regression with Trajectory embeddings and temperature scaling (τ=5) achieves 0.351 vs. 0.343 for Best-Avg, winning on 9 of 10 seeds (sign test $p\approx$0.02). We observe the same pattern on Leduc Poker: Trajectory kernel (τ=5) achieves 0.935 vs. 0.923 for Best-Avg, winning on 8 of 10 seeds, but the improvement is not statistically significant (paired t-test p=0.68). We additionally trained a learned Mahalanobis similarity metric on Liar's Dice, which did not outperform temperature-scaled cosine despite achieving r=0.70 correlation with ground-truth payoff similarity (Section C), indicating that the bottleneck is the information content of the embeddings themselves.

The Grover encoder's kernel regression degenerates to Best-Avg on both games, consistent with its near-uniform cosine similarities (μ=0.90, σ=0.06 on Liar's Dice; Figure 8). The large Oracle–Best-Avg gaps on both games confirm that opponent-specific information is exploitable in principle. These results, replicated across two games, multiple similarity metrics, and a supervised learned metric, reveal an epistemic blind spot in current SSL representations: they encode enough to recognize an opponent but not enough to know how to exploit them, and post-hoc correction via learned metrics cannot compensate for information that was never captured. Whether payoff-aware training objectives or uncertainty-aware representations that explicitly model what is *not yet known* about an opponent could close this gap remains an open question.

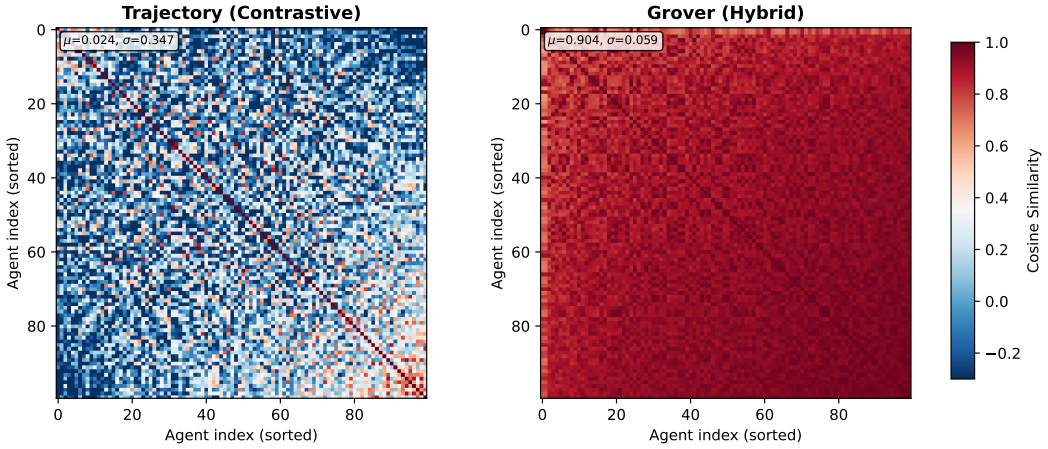


Figure 8: Cosine similarity matrices for 100 evaluation agents under Trajectory (contrastive) and Grover (hybrid) encoders on Liar's Dice. Agents are sorted by mean similarity. The Trajectory encoder produces structured similarity patterns, while the Grover encoder's similarities collapse to near-uniform values, explaining why its kernel regression degenerates to the Best-Avg baseline.

F Training-Time Trajectory Ablation

We ablate the number of trajectories per policy used during contrastive pre-training on Liar's Dice. Table 13 shows that downstream performance is remarkably stable across a 5× range of training trajectories (10 to 50 per policy). Even with only 10 trajectories per policy, the encoder achieves 55.5% improvement on Task A, comparable to the default setting of 50 trajectories (55.6%). This indicates that the contrastive encoder extracts discriminative policy representations efficiently, requiring only a modest number of behavioral observations per agent during pre-training.

Table 13: Trajectory encoder ablation on Liar's Dice (seed 42): downstream payoff prediction as a function of training trajectories per policy.

Traj./Policy	Task A MSE	Imp.	Task B MSE	Imp.
10	0.0169	55.5%	0.0447	43.7%
25	0.0183	51.8%	0.0444	44.1%
50	0.0168	55.6%	0.0458	42.4%

G Experimental Details and Hyperparameters

G.1 Code Implementation

We use the OpenSpiel Lanctot et al. (2019) library. We extensively use a modified version of IIG-RL-Benchmark Rudolph et al. (2025) for PPO training, PSRO, and NeuPL.

G.2 Standard Training Configuration

In all experiments, we use 64-dimension embeddings.

All experiments were conducted using three random seeds and results were averaged, except for the opponent-adaptive strategy selection experiment which uses 10 seeds for statistical testing. For the Trajectory Encoder, we utilized 50 trajectories per policy during the training phase. Qualitative visualizations using t-SNE were generated with a perplexity value of 30.

For Task B, we sample 50,000 policy pairs.

For Tasks A and B, the targets can be computed either exactly (with a full game-tree traversal) or empirically (by sampling several episodes). In this paper, we sample for Task A and use exact values for Task B.

Table 14: Hyperparameters for Trajectory and Functional Encoders

Parameter	Traj. (Kuhn)	Traj. (Leduc)	Traj. (LD)	Traj. (PTTT)	Func. (Kuhn)	Func. (Leduc)
Num Agents	500	500	500	500	500	500
Batch Size	16	16	16	8	16	16
Epochs	200	200	200	50	100	50
Learning Rate	1×10^{-4}	1×10^{-4}	1×10^{-4}	1×10^{-4}	Default	Default
LR Scheduler	Cosine	Cosine	Cosine	Cosine	N/A	N/A
Val Split	0.2	0.2	0.2	0.2	N/A	N/A
Dataset Fraction	1.0	1.0	1.0	1.0	1.0	0.02

The Grover encoder uses the same hyperparameters as described in the original work, except that we use 64-dimensional embeddings rather than 128-dimensional(Grover et al., 2018): a 2-layer MLP encoder, trained with a hybrid imitation and triplet loss objective. All Grover models across the four games use the same architecture and training configuration.

G.3 Hardware and Infrastructure

Experiments were executed on an **NVIDIA Quadro RTX 6000** (24GB VRAM), and **Intel(R) Xeon(R) Platinum 8268 CPU @ 2.90GHz**, and a **Macbook M4 Pro**. For Task B, we evaluate 50,000 sampled agent pairs.

H Game Rules

Kuhn Poker is the simplest possible version of poker. It was developed by game theorist Harold Kuhn in 1950 to study simplified zero-sum games Kuhn (2016). It strips poker down to its absolute mathematical core: one card, one bet and two players (P1 and P2). The deck consists of 3 cards one King K, one Queen Q and a Jack J. Each player enters a nominal ante into the pot before being dealt one card. There is one betting round. The first player can Check or Bet. If P1 checks, P2 can check (followed by an immediate showdown), or Bet. If P1 Bets, player 2 can fold (P1 wins) or call (showdown). If P2 bets (following P1 check), P1 can fold or call. If neither player folds, both players reveal their cards in the showdown where $K > Q > J$. The player with the highest card wins the pot. The game is fully solved. We know the exact Nash Equilibrium strategy (e.g., if holding a Jack, Player 1 should never bet, but if holding a King, they should bet 3 times as often as they bet with a Queen to balance their bluffing range).

Leduc Poker adds a community card and multiple rounds of betting making it significantly more complicated and a better proxy for Texas Hold'em. In this game there are two players and 6 cards: two Kings, Queens and Jacks. To start, both players pay an ante and a then dealt one private card. **Round 1:** Players bet based on their private card. There is a fixed bet size. **The Flop:** One community card is revealed from the remaining deck. **Round 2:** Players bet again now knowing their card and the community card. **The Showdown:** Both players reveal their cards; A pair is a winning hand, otherwise the player with the highest card wins. If both players have the same card, the pot is split.

Liar's Dice (1 die, 4 sides) is a bluffing game structurally distinct from poker. Each of the two players privately rolls a single four-sided die. Players alternate making claims about the total count

of a particular face value across *both* dice combined (e.g., "there are at least two 3s"). Each claim must be strictly higher than the previous one (either a higher count or a higher face value). A player may instead challenge the opponent's last claim. If challenged, both dice are revealed: if the claim is true, the challenger loses; otherwise, the claimant loses. The game requires reasoning about hidden information (the opponent's die) and incentivizes bluffing where players can make inflated claims to pressure opponents into folding or to set up a trap. With 1 die and 4 sides, the game has approximately 1024 information states and 9 possible actions per state.

Phantom Tic-Tac-Toe is a partially observable variant of Tic-Tac-Toe where players cannot see their opponent's moves. Each player places marks on a 3×3 board, but only observes their own marks and whether an attempted move was rejected (because the cell was already occupied). If a player attempts to place on an occupied cell, they are informed the move is invalid and must choose again, but they do not learn who occupies that cell. The game ends when a player completes a row, column, or diagonal, or when the board is full (draw). With the "reveal-nothing" observation type, the game has approximately 500,000 information states, making it the largest game in our benchmark and a test of scalability beyond card games.

I Extended Related Works

Opponent Modeling and Agent Modeling Early work by He et al. (2016) introduced DRON for opponent modeling via deep RL, and Grover et al. (2018) learn low-dimensional policy embeddings using variational inference, evaluating on agent identification and reward prediction (our Task E follows their identification protocol). While Grover et al. (2018) desire "simulating the agent's policy" and "distinguishing the agent's policy" as the two desiderata of good representations, we focus more on predicting payoffs in two-player zero-sum games as a measure of good representations. More recently, Papoudakis & Albrecht (2020) and Zintgraf et al. (2022) learn latent representations of policies using a VAE over trajectories. Li et al. (2025) applies the VAE method to cooperative games. Xie et al. (2020) learn latent representations of other agents' strategies to influence multi-agent interactions. Most closely related to our trajectory encoder, CLAM Ma et al. (2024) uses contrastive learning on local observations to produce real-time policy representations of other agents, and TransAM Wallace et al. (2025) uses a transformer to encode local trajectories into embeddings that capture other agents' policies. Our work differs from these approaches in that we (i) don't restrict our policy embeddings to come from observation data only, (ii) systematically compare multiple types of policy embeddings, and (iii) evaluate on 2p0s IIG game-theoretic downstream tasks such as payoff and exploitability prediction and latent-space equilibrium finding rather than opponent modeling.

Behavioral Diversity and Trajectory Representations Several works have used trajectory-level representations to measure or encourage policy diversity. TrajeDi Lupu et al. (2021) defines behavioral distances between policies in trajectory space for zero-shot coordination. Our trajectory encoder uses contrastive learning to learn embeddings that are invariant to stochastic noise but discriminative of strategic intent.

Chess style learning Other relevant research directions include learning individual players' styles in chess. This was done by McIlroy-Young et al. (2022). Their approach fine-tunes policy networks to represent styles of different players. The methods in this paper could be used to represent styles of different players instead as low-dimensional embeddings.

Weight-Space Learning Several works have explored learning from or about neural network weights directly. Unterthiner et al. (2020) showed that network properties can be predicted from weight statistics alone, while hypernetworks Ha et al. (2016) generate weights for target networks. Our weight and functional autoencoders draw on this line of work, but our results suggest that weight-space proximity is a poor proxy for behavioral similarity in games.

I.1 Grover

Details on Grover et al. (2018):

The hybrid loss is defined as:

$$\mathcal{L}_{\text{Grover}} = \mathcal{L}_{\text{imit}} + \lambda \mathcal{L}_{\text{id}}$$
$$\mathcal{L}_{\text{id}} = \left(1 + \exp(\|r_e - n_e\|_2 - \|r_e - p_e\|_2)\right)^{-2},$$

where p_e, r_e, n_e are embeddings of a positive, reference, and negative episode respectively.

J Details on NeuPL

Our training setup for NeuPL involved training 22 policies per player. Our training setup is more similar to NeuPL (Liu et al., 2022) rather than NeuPL-JPSRO (Liu et al., 2024) in that we do not strictly train one new policy at a time while keeping the other policies fixed by using distillation and regularization, for simplicity's sake. Initially, each player's policy population contains only the uniform random policy (which is fixed over the course of training – it is not a neural network). At each outer-loop iteration, we allow a new policy into each player's population, but we continue to train each policy i in the population against the opponents' population of policies $j < i$.

We attempt to use significant entropy to stabilize training, with entropy coefficients of 0.009 and 0.011 for Kuhn and Leduc poker, respectively.

In contrast to NeuPL and NeuPL-JPSRO, the input to the embedding is simply a one-hot vector denoting the policy's ID instead of the distribution of opponents it plays against.

To train agents, we use PPO where the critic receives its own embedding, plus the infostate of both the player and the opponent.

We propose two methods of sampling from NeuPL's embedding space: (A) interpolating between two of the original policies, and (B) fitting a multivariate Gaussian to the embeddings and then sampling from it. In the main results, we use method B.

K Details on Task F

We set one player as the optimizing player and set the other player as the exploiting player, where the exploiting player takes several gradient steps for every gradient step that the optimizing player takes. This is similar to exploitability descent (Lockhart et al., 2020) or two-timescale methods (Daskalakis et al., 2021; Condon, 1990). We also tried using the extragradient (Goktas & Greenwald, 2022) (which is related to exploitability descent (Gemp et al., 2022, Section K.2)) or optimistic gradient rather than the gradients. We did not try using an ensemble of opponents like in Martin & Sandholm (2024), but this could help eliminate a source of instability.

L Future Work

While our experiments span games from $\sim$12 to $\sim$500K information states, scaling to even larger games such as 3x3 dark hex (millions of information states) and games with continuous action spaces remains an important direction.

Our opponent-adaptive strategy selection results highlight a key gap: representations trained for behavioral discrimination do not encode payoff-structural similarity in a linearly recoverable form, as demonstrated by our Mahalanobis projection experiment. A natural next step is to define training objectives that explicitly optimize for payoff-relevant structure, for example contrastive losses where positive pairs are agents with correlated payoff profiles rather than same-agent trajectories, or auxiliary losses that directly predict pairwise payoffs from embedding distances.