Neural Algorithmic Reasoning for Nash Equilibrium

Anonymous Authors[1]

Abstract

We apply Neural Algorithmic Reasoning (NAR) to Nash Equilibrium computation in two-player zero-sum games. We represent bimatrix games as bipartite graphs and train a GNN processor to imitate Fictitious Play. The resulting model, NAR-FP, combines GATv2 attention with GRU memory and per-step trajectory supervision. Using the multi-size training protocol, NAR-FP achieves strong out-of-distribution size generalisation, with only $1.02\times$ exploitability degradation from 10×10 to 50×50 games. NAR-FP outperforms classical Fictitious Play at matched step budgets, learning to accelerate the algorithm rather than merely replicate it. It also generalises across game structures, outperforming FP on cyclic games never seen during training. Our results suggest that algorithmic alignment is an effective framework for bridging classical game-theoretic solvers and modern neural learning.

1. Introduction

Computing Nash Equilibria (NE) is a fundamental problem in game theory with applications spanning economics, multi-agent systems, and security (Li et al., 2024). For two-player zero-sum games, linear programming yields exact solutions in polynomial time (Dantzig, 1951), but this does not extend to the general-sum case, where the problem is PPAD-complete (Daskalakis et al., 2009). Iterative methods such as Fictitious Play (FP) (Brown, 1951) remain widely used due to their simplicity (Heinrich & Silver, 2016), but converge slowly, requiring hundreds or thousands of iterations per game.

The problem has also been studied from a machine learning perspective. Neural approaches to NE computation have employed equivariant networks trained to minimise exploitability directly (Marris et al., 2022), and neural warm-starting of classical solvers (Duan et al., 2023). However, these methods do not leverage the inherent algorithmic structure of the iterative solvers they aim to replace. In contrast, heuristics and solvers for NE computation, such as Fictitious Play, follow well-defined iterative procedures that construct solutions through a series of progressively improving steps.

In the context of this observation, we show that knowledge of algorithms may be useful when training neural networks to compute Nash Equilibria. Recent advances in Neural Algorithmic Reasoning (NAR) (Veličković & Blundell, 2021) have shown that neural networks can effectively learn and reproduce the execution of classical algorithms by supervising on their intermediate steps. Moreover, generalist neural algorithmic reasoners (Ibarz et al., 2022) can learn from multiple input sizes at once, achieving robust out-of-distribution (OOD) generalisation. So far, NAR has been applied to classical graph and sorting algorithms (Veličković et al., 2022) and combinatorial optimisation (Georgiev et al., 2023), but not to game-theoretic algorithms. In addition, prior NAR work supervises on exact algorithms that terminate at the correct answer in polynomial time. We instead supervise on FP, an iterative method that only approaches NE asymptotically, and show that the resulting model can match, and even surpass, the heuristic it is trained on.

We represent bimatrix games as bipartite graphs where payoffs are encoded as edge features, and train a GNN to imitate FP's intermediate reasoning steps. The resulting model, NAR-FP, combines GATv2 attention (Brody et al., 2022) with GRU memory (Cho et al., 2014) and per-step trajectory supervision. Our contributions are as follows:

1. We present the **first, to our knowledge, application of NAR to a game-theoretic algorithm**, introducing NAR-FP, a GATv2-based GNN with GRU memory and trajectory supervision over FP traces.
2. We adopt the multi-size training protocol for game-solving GNNs, achieving $1.02\times$ **OOD degradation** from 10×10 to 50×50, essentially size-invariant.
3. We show that NAR-FP **converges faster than classical FP**, demonstrating learned acceleration of the algorithm.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

2. Background

Nash Equilibrium and Fictitious Play. A two-player normal-form game is defined by payoff matrices $(A, B) \in \mathbb{R}^{n\times m}$, where n and m are the numbers of pure actions available to players 1 and 2 respectively, and in zero-sum games $B = -A$. A Nash Equilibrium (NE) is a pair of mixed strategies where neither player can improve their payoff by unilaterally deviating. An ε-NE relaxes this, permitting deviations of at most ε (formal definitions in Appendix A). For zero-sum games, NE can be computed exactly via linear programming (Dantzig, 1951). FP is an iterative method where each player best-responds to the opponent's empirical strategy distribution. FP converges in zero-sum games (Robinson, 1951), producing an ε-NE after $O(1/\varepsilon^2)$ rounds. This guarantees a usable supervision signal, while the slow rate leaves room for a learned model to converge faster.

Neural Algorithmic Reasoning. The core principle of Neural Algorithmic Reasoning (NAR, Veličković & Blundell, 2021) is algorithmic alignment, meaning that a neural network generalises better when its computation mirrors the target algorithm. Alignment can be achieved in two complementary ways. Architectural alignment structures the network so that its submodules map to submodules of the algorithm. Trajectory supervision trains the network on the algorithm's intermediate states rather than only the final output (Veličković et al., 2022). NAR adopts an encode-process-decode architecture with a weight-shared processor, combining both forms of alignment. A key objective is out-of-distribution size generalisation. Ibarz et al. (2022) show that training on a mixture of input sizes produces generalist models that transfer to unseen sizes, a protocol we adopt in this work.

Neural Approaches to NE. The Neural Equilibrium Solver (NES, Marris et al., 2022) trains an equivariant neural network to minimise exploitability directly, without algorithm traces. NES operates on payoff tensors (not graphs) and handles NE, CE, and CCE. It demonstrates a size-invariant architecture for NE computation but uses unsupervised training signals rather than algorithmic supervision. Duan et al. (2023) study PAC learnability of NE approximators and use neural networks to warm-start classical solvers.

3. Method

3.1. Game-to-Graph Representation

We consider square games with $n = m$ and represent a two-player zero-sum game $(A, -A)$ with $A \in \mathbb{R}^{n\times n}$ as a bipartite graph $G = (V, E)$. The set of nodes $V = V_1 \cup V_2$ contains n nodes per player. Features encode the player identities with $\mathbf{x}_i = [1, 0]$ for $i \in V_1$ and $\mathbf{x}_j = [0, 1]$ for $j \in V_2$. The edge set is fully connected and bidirectional between V_1 and V_2, with each edge (i, j) carrying attributes $[A_{ij}, B_{ij}]$. All game-specific information resides in edge attributes, while node features carry only player identity. The representation is therefore size-agnostic, since graphs of any n share the same node feature space and edge feature dimensionality, allowing a single processor (described in Section 3.3) to be applied across sizes. This bipartite structure directly mirrors FP's alternating player updates, as each message-passing step transmits information between the two player sets, aligning the graph computation with the algorithm's structure (Veličković & Blundell, 2021).

3.2. Selection of Relevant Algorithm

NAR requires an iterative algorithm whose intermediate states can serve as supervision targets. Linear programming, support enumeration (Porter et al., 2008), and the Lemke-Howson algorithm all compute NE but do not produce a sequence of progressively improving strategy profiles suitable for trajectory supervision.

FP is a natural fit. Its update rule (best-respond to the opponent's empirical distribution) is applied identically at each iteration, matching the weight-shared processor of NAR. It converges in zero-sum games (Robinson, 1951), and its intermediate states are mixed strategy profiles that directly serve as per-step supervision targets.

3.3. Choice of GNN Architecture

Following the processor design principles of the CLRS benchmark (Veličković et al., 2022; Ibarz et al., 2022), our architecture requires three properties: (i) edge-aware attention, since all game information resides on edges, (ii) recurrent processing with shared weights, mirroring FP's iterative structure, and (iii) a gating mechanism to maintain persistent state across steps. Standard MPNN processors aggregate neighbour messages uniformly, but FP's best-response step is inherently selective, concentrating on the highest-payoff opponent actions. We therefore adopt GATv2 (Brody et al., 2022) with edge features projected into the attention mechanism (Gilmer et al., 2017), so that attention weights are conditioned on payoff values and the model can learn this selective aggregation. For gating, we use a GRU cell (Cho et al., 2014) rather than the linear gate used in CLRS processors. These choices combine into a processor applied with shared weights for T steps:

$$\mathbf{m}_i^{(t)} = \mathrm{ELU}\left(\mathrm{GATv2}\left(\mathbf{h}_i^{(t-1)}, \left\{\mathbf{h}_j^{(t-1)}, \mathbf{e}_{ij}\right\}_{j\in\mathcal{N}(i)}\right)\right), \tag{1}$$

$$\mathbf{h}_i^{(t)} = \mathrm{GRU}\left(\left[\mathbf{m}_i^{(t)} \| \mathbf{h}_i^{(0)}\right],\ \mathbf{h}_i^{(t-1)}\right) \tag{2}$$

where $\mathbf{h}_i^{(0)} = \text{ELU}(W_{\text{enc}}\,\mathbf{x}_i + b_{\text{enc}})$ is the encoded node representation, $\mathbf{e}_{ij} = [A_{ij}, B_{ij}]$ carries both players' payoffs, and $\|$ denotes concatenation. The concatenation of $\mathbf{h}_i^{(0)}$ at every step (input re-injection) provides a stable reference to the original encoded representation, preventing the processor from drifting away from the input signal over many iterations. At each step t, the hidden state is decoded to produce per-action logits $\boldsymbol{\ell}^{(t)}$, where $\ell_i^{(t)} = W_{\text{dec}}\,\mathbf{h}_i^{(t)} + b_{\text{dec}}$.

3.4. Training and Baselines

We generate random zero-sum games with entries $A_{ij} \sim \mathcal{N}(0,1)$ and compute exact NE via linear programming. FP traces of T steps provide intermediate supervision targets. The training loss applies binary cross-entropy (BCE) between per-action logits (with implicit sigmoid) and the target probabilities, treating each action independently.

$$\mathcal{L} = \frac{1}{T}\sum_{t=1}^{T} w_t \cdot \text{BCE}(\boldsymbol{\ell}^{(t)}, \sigma_{\text{FP}}^{(t)}) + \lambda \cdot \text{BCE}(\boldsymbol{\ell}^{(T)}, \sigma^*) \quad (3)$$

where $\boldsymbol{\ell}^{(t)}$ are the decoder logits at step t, $\sigma_{\text{FP}}^{(t)}$ is the FP strategy at step t, σ^* is the ground-truth NE, and exponential step weights w_t emphasise later, more converged FP steps (details in Appendix B). At inference, per-player softmax replaces sigmoid to produce valid strategy profiles. For the generalist model, we sample training dimensions uniformly per batch and validate on the largest training dimension. Random Gaussian games may admit multiple NE. Both σ^* and the FP trace target a single equilibrium selected by their respective solvers, so the supervision signal is consistent but not unique. Exploitability is invariant to which equilibrium the model converges on, while joint accuracy can penalise valid alternative equilibria, and we therefore treat exploitability as the primary metric.

Baselines. **FP-K** runs Fictitious Play for $K \in \{30, 300\}$ iterations. All neural models are parameter-matched for fair comparison (details in Appendix B). **Recurrent GNN** uses encode-process-decode with GATv2, 8 weight-shared steps, trained with only the ground-truth term of Eq. 3 ($\text{BCE}(\boldsymbol{\ell}^{(T)}, \sigma^*)$), no GRU. **Feedforward GNN** uses two stacked GATv2 layers with no weight sharing.

4. Experiments

Setup. Single-size models (Feedforward GNN, Recurrent GNN, NAR-FP) are trained on 10,000 random 10×10 zero-sum games, making dimensions $\{15, 20, 50\}$ out-of-distribution. NAR-FP (generalist) is trained on a mixture of dimensions $\{3, 5, 10, 15\}$, making $\{20, 50\}$ out-of-distribution. We evaluate all models on $\{10, 15, 20, 50\}$. We report means over 3 seeds unless otherwise noted. The primary metric is *exploitability*, which measures the total payoff each player could gain by switching to their best response. An exploitability of 0 means neither player can improve and the strategies form an exact NE. We define *OOD degradation* as the ratio $\text{Exploit}(50\times50)/\text{Exploit}(10\times10)$.

4.1. Main Results

Table 1 presents exploitability across game sizes. NAR-FP (generalist) achieves the best OOD degradation ($1.02\times$), essentially size-invariant. All baselines exhibit substantially higher degradation. FP-300 degrades $2.36\times$ and Recurrent GNN degrades $2.56\times$. At matched computation budget ($\sim$50 steps), NAR-FP (generalist) with exploitability 0.163 outperforms FP-30 (1.001) at 50×50 by $6.1\times$. Even against FP-300, which uses $6\times$ more iterations, NAR-FP achieves $2\times$ lower exploitability at 50×50.

Table 1. Exploitability (lower = better) across game sizes. Mean and standard deviation over 3 seeds. NAR-FP (generalist) achieves the strongest size-invariance, with only $1.02\times$ degradation from 10×10 to 50×50, compared to $2.56\times$ for Recurrent GNN, $2.36\times$ for FP-300, and $1.97\times$ for FP-30.

Model	**10**	**15**	**20**	**50**
FP-300	.139	.184	.219	.329
FP-30	.508	.642	.739	1.001
Feedforward GNN	$.531_{\pm.009}$	$.510_{\pm.014}$	$.562_{\pm.021}$	$.586_{\pm.009}$
Recurrent GNN	$.166_{\pm.011}$	$.187_{\pm.006}$	$.238_{\pm.033}$	$.425_{\pm.094}$
NAR-FP	$.167_{\pm.009}$	$.157_{\pm.016}$	$.151_{\pm.015}$	$.176_{\pm.065}$
NAR-FP (gen.)	$\mathbf{.159}_{\pm.008}$	$\mathbf{.126}_{\pm.006}$	$\mathbf{.114}_{\pm.010}$	$\mathbf{.163}_{\pm.015}$

Among neural models, Feedforward GNN exhibits low degradation ($1.10\times$) but uniformly high exploitability (>0.5), indicating it learns a near-constant prediction. Recurrent GNN, which uses the same GATv2 architecture and encode-process-decode structure but lacks GRU memory, input re-injection, and trajectory supervision, degrades $2.56\times$, confirming that GRU memory and trajectory supervision drive the improvement (Section 4.4).

Inference cost. NAR-FP inference takes $\sim$51ms per game on GPU across the sizes we evaluate (10×10 to 50×50). At 50×50, FP-300 requires $\sim$16ms but achieves $2\times$ worse exploitability (0.329 vs. 0.163). NAR-FP provides better solutions at moderate extra cost.

4.2. Game-Type Generalisation

All models above are trained on random Gaussian games. To test structural generalisation, we evaluate on cyclic games (generalised rock-paper-scissors with circular dominance), a game family never seen during training. Table 2 reports exploitability across sizes. Both NAR-FP variants achieve lower exploitability than FP-300 at every game size, showing that the learned processor generalises across game structure, not only game size.

Table 2. Exploitability on cyclic games (not seen during training). Both NAR-FP variants outperform FP-300 at all sizes. Mean and standard deviation over 3 seeds.

Model	10	15	20	50
FP-300	.180	.207	.225	.277
FP-30	.596	.678	.717	.844
Feedforward GNN	$.312_{\pm.040}$	$.238_{\pm.002}$	$.224_{\pm.002}$	$.176_{\pm.004}$
Recurrent GNN	$.343_{\pm.053}$	$.322_{\pm.098}$	$.293_{\pm.147}$	$.184_{\pm.021}$
NAR-FP	$.111_{\pm.010}$	$\mathbf{.118}_{\pm.015}$	$\mathbf{.113}_{\pm.015}$	$.138_{\pm.038}$
NAR-FP (gen.)	$\mathbf{.096}_{\pm.009}$	$.122_{\pm.019}$	$.140_{\pm.022}$	$\mathbf{.127}_{\pm.008}$

Table 3. Effect of processor steps and multi-size training on OOD generalisation. "(10)" denotes single-size training on 10×10 games; "(gen.)" denotes generalist training on $\{3, 5, 10, 15\}$. Mean and standard deviation over 3 seeds.

Training	10	15	20	50
30 processor steps				
NAR-FP (10)	$.174_{\pm.020}$	$.162_{\pm.031}$	$.155_{\pm.034}$	$.236_{\pm.123}$
NAR-FP (gen.)	$.166_{\pm.006}$	$.140_{\pm.015}$	$.139_{\pm.027}$	$.218_{\pm.067}$
50 processor steps				
NAR-FP (10)	$.167_{\pm.009}$	$.157_{\pm.016}$	$.151_{\pm.015}$	$.176_{\pm.065}$
NAR-FP (gen.)	$\mathbf{.159}_{\pm.008}$	$\mathbf{.126}_{\pm.006}$	$\mathbf{.114}_{\pm.010}$	$\mathbf{.163}_{\pm.015}$

4.3. Multi-Size Training

Table 3 shows the effect of processor steps and multi-size training. Increasing steps from 30 to 50 reduces 50×50 exploitability from 0.236 to 0.176 for the single-size model and from 0.218 to 0.163 for the generalist. Multi-size training independently improves OOD performance at every step count. The combination of 50 steps and multi-size training on $\{3, 5, 10, 15\}$ achieves the best result ($1.02\times$ degradation). Multi-size training also reduces cross-seed variance at 50×50 (±.015 for the generalist vs. ±.065 for the single-size model), suggesting it regularises the model as well as broadening size coverage.

4.4. Component Ablation

To isolate which architectural components drive NAR-FP's generalisation, we ablate GRU memory, input re-injection, and trajectory supervision (single-size training, 50 steps, evaluated at 50×50). GRU memory is the most critical component, with exploitability rising by 157% when removed. Trajectory supervision is the second most important ($+26\%$), confirming that algorithmic alignment must be enforced through both architecture and supervision signal. Input re-injection is not distinguishable from the full model at this scale (Appendix C). We retain it for consistency with the standard NAR template. Crucially, parameter matching means Recurrent GNN has higher per-step capacity (h=75) than NAR-FP (h=32) yet underperforms, indicating that the gap is driven by inductive bias and supervision rather than model size.

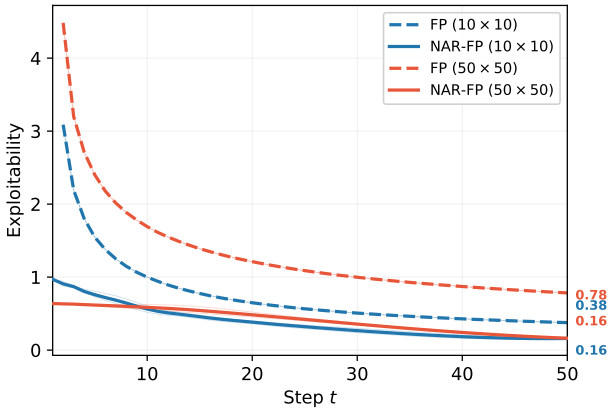

Figure 1. Per-step exploitability for NAR-FP (generalist) vs. classical FP on held-out games (solid: 10×10, dashed: 50×50). At each step, NAR-FP achieves lower exploitability, showing it converges faster than the algorithm it imitates.

4.5. Algorithmic Alignment Analysis

NAR-FP uses FP as an inductive bias through both architectural alignment and trajectory supervision. A natural question is whether this causes the model to merely replicate FP's behaviour, or whether it learns to improve upon it. To answer this, we decode the model's latent state at each intermediate processor step into strategy predictions and measure exploitability, comparing directly against classical FP run on the same held-out games.

Figure 1 plots the result. At every step t, NAR-FP achieves lower exploitability than FP at the corresponding iteration. At 10×10, NAR-FP reaches exploitability 0.16 after 50 processor steps, compared to 0.38 for FP after 50 iterations. The gap widens at 50×50 (0.16 vs. 0.78), showing that NAR-FP has learned to *accelerate* convergence rather than faithfully reproduce it. FP serves as a useful training signal, but the model discovers a faster path to equilibrium by compressing multiple updates into each processor step.

5. Conclusion

We have shown that NAR-FP, a GNN trained to imitate FP via trajectory supervision, achieves strong size invariance ($1.02\times$ OOD degradation from 10×10 to 50×50) and outperforms FP-300 at 50×50 despite using $\sim6\times$ fewer reasoning steps. The combination of GRU memory, trajectory supervision, and multi-size training (Ibarz et al., 2022) enables this generalisation. Our evaluation is restricted to two-player zero-sum games where FP is guaranteed to converge. The bipartite graph representation readily supports asymmetric (general-sum) games, but FP is not guaranteed to converge there, and multiplayer settings would require hypergraph representations, an NAR setup that has not been studied extensively.

References

Brody, S., Alon, U., and Yahav, E. How attentive are graph attention networks? In *International Conference on Learning Representations*, 2022.

Brown, G. W. Iterative solution of games by fictitious play. *Activity Analysis of Production and Allocation*, 13(1): 374–376, 1951.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Dantzig, G. B. A proof of the equivalence of the programming problem and the game problem. Technical report, RAND Corporation, 1951.

Daskalakis, C., Goldberg, P. W., and Papadimitriou, C. H. The complexity of computing a Nash equilibrium. In *SIAM Journal on Computing*, volume 39, pp. 195–259, 2009.

Duan, Z., Wen, W., Li, K., Song, W., Zhang, S., and Chen, W. Is nash equilibrium approximator learnable? *arXiv preprint arXiv:2308.10751*, 2023.

Georgiev, D., Numeroso, D., Bacciu, D., and Liò, P. Neural algorithmic reasoning for combinatorial optimisation. In *Learning on Graphs Conference*. PMLR, 2023.

Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. Neural message passing for quantum chemistry. In *International Conference on Machine Learning*, pp. 1263–1272, 2017.

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

Ibarz, B., Kurin, V., Papamakarios, G., Nikiforou, K., Bennani, M., Csordás, R., Duber, A., Graves, A., Velickovic, P., and Blundell, C. A generalist neural algorithmic learner. In *Learning on Graphs Conference*, pp. 2:1–2:23. PMLR, 2022.

Li, H., Shen, W., and Li, J. A survey on algorithms for nash equilibria in finite normal-form games. *Computer Science Review*, 51:100606, 2024.

Marris, L., Gemp, I., Anthony, T., Tacchetti, A., Liu, S., and Tuyls, K. Turbocharging solution concepts: Solving NEs, CEs, and CCEs with neural equilibrium solvers. In *Advances in Neural Information Processing Systems*, volume 35, pp. 5765–5778, 2022.

Porter, R., Nudelman, E., and Shoham, Y. Simple search methods for finding a Nash equilibrium. In *Games and Economic Behavior*, volume 63, pp. 642–662, 2008.

Robinson, J. An iterative method of solving a game. *Annals of Mathematics*, 54(2):296–301, 1951.

Veličković, P. and Blundell, C. Neural algorithmic reasoning. *Patterns*, 2(7):100273, 2021.

Veličković, P., Badia, A. P., Budden, D., Pascanu, R., Banino, A., Dashevskiy, M., Hadsell, R., and Blundell, C. The CLRS algorithmic reasoning benchmark. In *International Conference on Machine Learning*, pp. 22084–22102. PMLR, 2022.

A. Formal Definitions

A two-player normal-form game is defined by a pair of payoff matrices $(A, B) \in \mathbb{R}^{n \times m}$, where $n = |S_1|$ and $m = |S_2|$ are the number of actions available to each player. A_{ij} and B_{ij} are the payoffs to player 1 and player 2, respectively, when player 1 plays action i and player 2 plays action j. In zero-sum games, $B = -A$.

An ε-Nash Equilibrium (ε-NE) of a bimatrix game (A, B) is a pair of mixed strategies $(\sigma_1, \sigma_2) \in \Delta(S_1) \times \Delta(S_2)$ such that

$$\sigma_1^\top A \sigma_2 \geq {\sigma_1'}^\top A \sigma_2 - \varepsilon \quad \forall\, \sigma_1' \in \Delta(S_1) \tag{4}$$

$$\sigma_1^\top B \sigma_2 \geq \sigma_1^\top B \sigma_2' - \varepsilon \quad \forall\, \sigma_2' \in \Delta(S_2) \tag{5}$$

where $\Delta(S) = \{x \in \mathbb{R}^{|S|}_{\geq 0} : \mathbf{1}^\top x = 1\}$ is the probability simplex. An exact NE corresponds to $\varepsilon = 0$.

Exploitability. Given a strategy profile (σ_1, σ_2), exploitability measures how much each player could gain by deviating to their best response.

$$\text{Exploit}(\sigma_1, \sigma_2) = \sum_{p=1}^{2} \left(\max_{\sigma_p'} {\sigma_p'}^\top M_p\, \sigma_{-p} - \sigma_p^\top M_p\, \sigma_{-p} \right) \tag{6}$$

where $M_1 = A$ and $M_2 = B$. An exploitability of 0 indicates an exact Nash Equilibrium.

B. Training Hyperparameters

All models are trained for 100 epochs with Adam (lr $= 0.002$) and batch size 64 on 10,000 random zero-sum games. The trajectory step weights are $w_t = e^{\beta(t-1)/(T-1)}/\bar{w}$ with $\beta = 2$ and $\bar{w} = \frac{1}{T}\sum_t e^{\beta(t-1)/(T-1)}$, so that $\frac{1}{T}\sum_t w_t = 1$. This gives a ratio $w_T/w_1 \approx 7.4$ that emphasises later, more converged FP steps. We set $\lambda = 1$ for the final-answer loss weight.

Neural baseline hyperparameters at $\sim$182K parameters are as follows. Recurrent GNN uses h=75, heads=4, 8 steps. Feedforward GNN uses h=148, heads=4, 2 layers. NAR-FP uses h=32, heads=4, 50 steps.

C. Component Ablation

Table 4 isolates the contribution of each NAR-FP component. All variants are trained on 10×10 games with 50 processor steps and evaluated OOD at 50×50. GRU memory is the most critical component, without it exploitability increases by 157%. Trajectory supervision is the second most important (+26%). Input re-injection is not distinguishable from the full model at this scale.

Variant	Exploit. (50)	vs. Full
Full (GRU + re-injection + trace loss)	$0.248_{\pm.099}$	—
No input re-injection	$0.212_{\pm.074}$	-14%
No trajectory loss	$0.313_{\pm.025}$	$+26\%$
No GRU (direct overwrite)	$0.638_{\pm.000}$	$+157\%$

Table 4. Component ablation for NAR-FP (single-size, 50 steps). Exploitability at 50×50 (OOD). Mean and standard deviation over 3 seeds.

D. Joint Accuracy

Table 5 reports joint accuracy, defined as the fraction of games where both players' argmax strategies match the ground-truth NE simultaneously. NAR-FP achieves the highest accuracy in-distribution, while the generalist is competitive at OOD sizes (50×50) and achieves lower exploitability overall (Table 1).

Joint accuracy depends only on which action has the highest predicted probability for each player. In random Gaussian games, NE strategies often place near-equal weight on several actions, so small prediction errors can flip the selected action even when the predicted strategy closely matches the true distribution. Exploitability captures the full distance between strategies and is therefore more informative here.

Model	10	15	20	50
Feedforward GNN	$32.1_{\pm 0.2}$	$22.8_{\pm 2.3}$	$17.3_{\pm 3.1}$	$4.9_{\pm 2.5}$
Recurrent GNN	$58.3_{\pm 1.0}$	$45.9_{\pm 2.3}$	$34.4_{\pm 4.0}$	$9.1_{\pm 5.1}$
NAR-FP	$\mathbf{79.4}_{\pm 1.3}$	$\mathbf{72.4}_{\pm 2.5}$	$\mathbf{62.5}_{\pm 3.4}$	$\mathbf{35.6}_{\pm 7.1}$
NAR-FP (generalist)	$65.6_{\pm 2.6}$	$61.5_{\pm 0.3}$	$56.0_{\pm 1.3}$	$25.7_{\pm 1.5}$

Table 5. Joint accuracy (%) across game sizes. Mean and standard deviation over 3 seeds. NAR-FP wins in-distribution while the generalist is competitive OOD.

E. Computational Resources

All experiments were conducted on the CSD3 HPC cluster at the University of Cambridge, using single NVIDIA A100-80GB GPUs.