Supplementary Information for “Population-dependent agent performance in non-transitive games: a multi-agent rock–paper–scissors benchmark”

Ou Deng[1,*], **Jianting Xu**[1], **Shoji Nishimura**[2], **Atsushi Ogihara**[2], **and Qun Jin**[2,*]

[1]Graduate School of Human Sciences, Waseda University. Tokorozawa, Saitama 359-1192, Japan
[2]Faculty of Human Sciences, Waseda University. Tokorozawa, Saitama 359-1192, Japan
[*]dengou@toki.waseda.jp; jin@waseda.jp

ABSTRACT

This Supplementary Information accompanies the main manuscript on multi-agent Rock–Paper–Scissors tournaments, providing additional details on the agent portfolio and hyper-parameters, hyper-parameter selection and resource matching (with a focus on Transformer baselines), and robustness checks and auxiliary analyses referenced in the main text.
Section S1 provides detailed agent implementation specifications including architectures, hyperparameters, and training procedures. Section S2 defines the Core-19 validation opponent set used for hyperparameter selection (referenced in Methods and Figure 2 of the main text). Section S3 reports validation tournament results and a targeted Transformer hyperparameter sweep addressing tuning concerns (Tables S3–S4; Figure S1). Section S4 documents transformer configuration selection and traceability for the final benchmark. Section S5 describes additional robustness checks on update budgets, context lengths, and reward scaling. Section S6 provides extended analyses for the 54-agent benchmark: stability across seeds (Figure S2), cumulative score trajectories by method (Figure S3), and the complete agent ranking (Table S6). Section S7 presents analysis of non-transitive meta-game structure: pairwise payoff analysis, cycle enumeration, and α-Rank stationary distributions. Section S8 presents additional Lipschitz bound diagnostics for the RNN–LSTM matchup (Figure S5, complementing Figure 3 of the main text). Section S9 reports trained-versus-untrained ablations (Table S9; Figure S6). Section S10 presents results on a stronger nine-agent roster excluding highly exploitable baselines (Table S10). Section S11 analyzes sliding-window length sensitivity for the Lipschitz diagnostic (Figure S7; Table S11). Section S12 provides computational cost benchmarking for representative agents (Table S12).

Guide to the Supplementary Information

To facilitate reader-friendly navigation, the following guide maps each Supplementary section/subsection, figure, and table to the part of the main manuscript that it supports or is cited from.

Supplementary item	Contents	Main-manuscript linkage
Section S1	**Agent implementation details.** Provides detailed specifications for all 18 agent archetypes including network architectures, hyper-parameters, training procedures, and feature engineering. Extends the summary in Methods: *Agent portfolio*.	Methods: *Agent portfolio*.
Subsection S1.1	**Deep learning agents (GPU-accelerated).** Details architectures and hyperparameters for RNN, LSTM, Transformer, and A3C families. Includes Table S1 summarizing update strategies.	Methods: *Agent portfolio*; *Hyperparameter selection and resource matching*.
Table S1	Summary of deep learning agent architectures and online update strategies (RNN, LSTM, Transformer, A3C).	Methods: *Hyperparameter selection and resource matching*.
Subsection S1.2	**Classical machine learning (CPU-based).** Specifications for SVM, RF, and XGB agents including feature engineering and refit schedules.	Methods: *Agent portfolio*.
Subsection S1.3	**Probabilistic and custom strategies.** Documents B_v1/v2 (Dirichlet frequency tracking), M_v1/v2 (Markov), and MSA (multi-scale aggregator).	Methods: *Agent portfolio*.

Supplementary item	Contents	Main-manuscript linkage
Subsection S1.4	**Rule-based baselines.** Describes R (random), CG (counter-guesser), and WL (win–lose heuristics) agents.	Methods: *Agent portfolio*.
Subsection S1.5	**Implementation notes.** Software dependencies, hardware specifications, and code organization.	Methods: *Agent portfolio*.
Subsection S1.6	**Statistical methodology.** Documents the seed-paired Wilcoxon signed-rank test with Holm correction used for pairwise method comparisons. Justifies why paired tests are appropriate given within-seed dependencies.	Methods: *Statistical analysis*.
Section S2	**Validation opponent set (Core-19).** Defines the held-out 19-agent roster used exclusively for hyperparameter tuning; this roster is not used for final benchmark ranking. Ensures that neural configurations are selected independently of the 54-agent evaluation pool.	Results: *Validation tournament (Core-19) and transformer tuning*; Methods: *Hyperparameter selection and resource matching*.
Table S2	Core-19 roster listing all 19 agent IDs and their family categories (Baseline, Probabilistic, Classical ML, Deep predictor, RL).	Cited in Results *Validation tournament (Core-19)* and Figure 2 caption ("Supplementary Table S2").
Section S3	**Validation results and Transformer sweep.** Reports Core-19 tournament scores and a targeted Transformer hyperparameter sweep. Addresses reviewer concern that Transformer baselines may be under-tuned by showing that even the best-tuned configuration remains negative.	Results: *Validation tournament (Core-19) and transformer tuning*; Discussion: *Why do recurrent architectures win?*
Table S3	Full Core-19 validation scores aggregated by method (18 methods, 10 seeds), with mean, std, and 95% CI.	Cited in Results *Validation tournament (Core-19)* ("full statistics in Supplementary Table S3").
Table S4	Transformer hyperparameter sweep summary: 8 configurations varying context length, width, depth, dropout, and learning rate.	Cited in Results *Validation tournament (Core-19)* ("Supplementary Table S4") and Discussion (Transformer under-tuning concern).
Figure S1	Visualization of Transformer sweep results with 95% CI across seeds; confirms all configurations remain negative.	Cited in Results *Validation tournament (Core-19)* ("Supplementary Fig. S1").
Section S4	**Transformer configuration traceability.** Documents exact hyperparameters used for Tr_v1 and Tr_v2 in the final benchmark, mapping them to sweep configurations to ensure reproducibility and address "under-tuned" concerns.	Results: *Validation tournament*; Methods: *Hyperparameter selection*.
Section S5	**Additional robustness checks.** Documents sensitivity analyses for update-budget, context-length, and reward-scaling choices. Confirms that qualitative conclusions (recurrent advantage, transformer performance under our budget) are robust to reasonable protocol variations.	Supports Methods *Hyperparameter selection and resource matching* and Results/Discussion claims about robustness.
Section S6	**Full 54-agent tournament: additional analyses.** Provides extended diagnostics for the main benchmark: seed-level stability, temporal score evolution, and the complete 54-agent ranking.	Results: *Robustness across seeds and temporal profiles*; complements Figure 1 and Tables 1–2.
Subsection S6.1	**Stability across random seeds.** Plots mean score vs. coefficient of variation (CV) to identify agents that combine strong performance with low variance.	Cited in Results *Robustness across seeds and temporal profiles*.
Figure S2	Stability scatter plot (mean vs. CV). High-performing agents cluster at large positive mean with moderate CV.	Cited in Results ("Supplementary Fig. S2 reports a stability scatter plot").
Subsection S6.2	**Cumulative score trajectories.** Shows how method-level scores accumulate over the tournament schedule, confirming persistent (not transient) advantages.	Cited in Results *Robustness across seeds and temporal profiles*.
Figure S3	Cumulative score evolution over 1.43M game rounds, aggregated by method family.	Cited in Results ("Supplementary Fig. S3 shows cumulative score trajectories").
Subsection S6.3	**Full agent ranking.** Provides the complete 54-agent ranking (extending Table 1 which shows only Top-5) for transparency and reproducibility.	Extends Table 1 (Top-5 agents) and supports Results *Overall ranking in the 54-agent tournament*.

Supplementary item	Contents	Main-manuscript linkage
Table S6	Full ranking of all 54 agents by mean score, with std and 95% CI across 10 seeds.	Extends Table 1; enables independent verification of all agent-level results.
Section S7	**Non-transitive meta-game structure.** Provides direct evidence of cyclic dominance: pairwise payoff heatmaps, enumeration of 3-cycles, and α-Rank evolutionary analysis. Strengthens the paper's thematic focus on non-transitivity.	Results: *Non-transitive structure*; Related work: *Evaluation in non-transitive environments*.
Table S7	Top 10 of 177 detected 3-cycles from the Core-19 payoff matrix demonstrating non-transitive dominance relations.	Cited in Results *Non-transitive structure*.
Figure S4	α-Rank stationary distribution over Core-19 agents showing evolutionary stable mass allocation.	Cited in Results *Non-transitive structure*.
Table S8	Rank correlations across evaluation pools (Core-54, Core-19, Top-R, Pack4).	Cited in Results *Validation tournament*.
Section S8	**Additional regret-certificate diagnostics.** Extends the main-text regret-certificate analysis (Figure 3 and Table 3) to a predictor-vs.-predictor regime (pretrained RNN$\rightarrow$LSTM). Shows that the certificate can become slack when regret is near zero, leading to weaker error–regret correlation.	Results: *Regret certificate diagnostics and tightness analysis* (Table 3, Figure 3).
Figure S5	Lipschitz scatter for RNN_v2$\rightarrow$LSTM_v2 (pretrained). All points satisfy $\Delta_t \leq 2\|p_t - \hat{p}_t\|_1$; weak correlation reflects near-zero regret floor.	Complements the pretrained RNN$\rightarrow$LSTM row in Table 3.
Section S9	**Trained vs. untrained controls.** Ablation separating initialization/pretraining effects from architectural capacity. Trained recurrent and A3C agents outperform untrained counterparts; Transformer remains negative regardless, supporting that its weaker performance under our budget is not due to poor initialization.	Results: *Pretraining and opponent-pool strength*.
Table S9	Mixed 8-agent population (trained + untrained variants of A3C, RNN, LSTM, Transformer) with mean and 95% CI.	Cited in Results *Pretraining and opponent-pool strength* ("Supplementary Table S9").
Figure S6	CI plot for the trained/untrained mixed population; visualizes the pretraining benefit for recurrent families and absence thereof for Transformer.	Cited in Results *Pretraining and opponent-pool strength* ("Supplementary Fig. S6").
Section S10	**Stronger opponent pool (Top-R roster).** Removes highly exploitable baselines (RF, XGB) to test whether recurrent advantage persists in a tougher environment. RNN_v2 remains top-ranked, confirming robustness to opponent-pool strength.	Results: *Pretraining and opponent-pool strength*.
Table S10	Nine-agent Top-R tournament results (mean and 95% CI, 10 seeds).	Cited in Results *Pretraining and opponent-pool strength* ("Supplementary Table S10").
Section S11	**Sliding-window length sensitivity for regret-certificate diagnostics.** Tests robustness of the certificate diagnostics to the window length K used to estimate the empirical opponent distribution p_t. While the certificate inequality holds for all $K \in \{5, 10, 20, 50\}$ by construction, correlation and tightness statistics vary; qualitative conclusions remain stable.	Methods: definition of p_t; Results: *Regret certificate diagnostics and tightness analysis*.
Figure S7	Spearman correlation between $\|p_t - \hat{p}_t\|_1$ and regret Δ_t as a function of K.	Cited in Methods ("Supplementary Fig. S7").
Table S11	Summary Lipschitz diagnostics (mean error, mean regret, slope, violation rate) across $K \in \{5, 10, 20, 50\}$ for four representative matchups.	Cited in Methods ("Supplementary Table S11").
Section S12	**Computational cost benchmarking.** Reports end-to-end wall-clock runtime (ms/decision) and parameter counts for representative agents. Ensures budget comparability across families and addresses reviewer requests for cost transparency.	Methods: Hyper-parameter selection and resource matching.
Table S12	Parameter counts and ms/decision (CPU) for six representative agents spanning non-neural baselines, actor–critic, Transformer, RNN, and LSTM.	Cited in Methods ("Supplementary Table S12").

S1 Agent implementation details

This section provides detailed specifications for all agent archetypes implemented in our benchmark. Each archetype is instantiated with multiple variants (typically three) using different initializations and hyper-parameters, and evaluated across ten random seeds.

S1.1 Deep learning agents (GPU-accelerated)

All deep learning agents are implemented in PyTorch 1.12+ and support GPU acceleration (tested on NVIDIA RTX A6000). Table S1 summarizes the key architectural parameters and online update strategies for each family.

Table S1. Summary of deep learning agent architectures and online update strategies. The "Update strategy" column describes the default behaviour; agents supporting `set_batch_size()` can be configured via command-line arguments.

Agent	Architecture	Key hyperparameters	Online update strategy
`RNN_v2`	GRU predictor, 1 layer, 64 hidden	lr=10^{-3}, ctx=16, grad_clip=1.0	Configurable:[†] default per-step
`LSTM_v1`	LSTM, 1 layer, 64 hidden	lr=10^{-3}, ctx=24, dropout=0.1	Per-step + replay (freq=15)
`LSTM_v2`	LSTM, 2 layers, 80 hidden	lr=1.5×10^{-3}, ctx=24, dropout=0.05	Per-step + replay (freq=15)
`Tr_v1`	Transformer, 1 layer, d=64	lr=10^{-3}, ctx=32, heads=4	Mini-batch: batch=64, freq=64
`Tr_v2`	Transformer, 2 layers, d=64	lr=10^{-3}, ctx=32, dropout=0.05	Mini-batch: batch=64, freq=64
`A3C_v1`	Shared MLP (2×64), A-C	lr=10^{-3} (RMSprop), ent=0.01	Per-step (buffered, batch=64)
`A3C_v2`	Separate actor/critic, target net	lr=2×10^{-3} (Adam), τ=0.02	Per-step (immediate)

[†] `RNN_v2` exposes `set_batch_size()`, but batch size is used only in optional mini-batch mode (`online_mode=False`); the default behavior in this work is per-step online updates.

Detailed specifications:

- **RNN_v2**: GRU-based recurrent predictor with tanh activation, one hidden layer (64 units), trained with Adam optimizer (learning rate 10^{-3}), gradient clipping (max norm 1.0), and context length 16. *Update strategy:* Default online mode performs a parameter update after each round (one gradient step per new observation). The implementation also supports an optional mini-batch mode (`online_mode=False`) in which updates occur every `update_freq` rounds using the most recent `batch_size` samples; unless stated otherwise, we use the default per-step mode. The network predicts the opponent's next action distribution from encoded history features.

- **LSTM_v1**: Single-layer LSTM with 64 hidden units, dropout (rate 0.1), and layer normalization. Trained with Adam (learning rate 10^{-3}). *Update strategy:* Per-step updates with periodic experience replay. A small replay buffer (size 20) is maintained, and replay occurs every 15 steps.

- **LSTM_v2**: Deeper LSTM with 80 hidden units and two stacked layers, dropout (rate 0.05), learning rate 1.5×10^{-3} with slow decay ($\gamma = 0.99995$). *Update strategy:* Same as LSTM_v1 (per-step + replay every 15 steps). Uses hidden-state persistence across rounds within a matchup.

- **Tr_v1**: Basic transformer encoder with a single self-attention layer, model dimension $d = 64$, 4 attention heads, context length 32, and learned positional encodings. Trained with Adam (learning rate 10^{-3}). *Update strategy:* Mini-batch updates (batch_size=64, update_freq=64).

- **Tr_v2**: Deeper transformer with 2 layers, multi-head attention (4 heads), model dimension $d = 64$, residual connections, and dropout (rate 0.05). Learning rate 10^{-3}. *Update strategy:* Mini-batch updates (batch_size=64, update_freq=64). The Transformer sweep (Table S4) varied context length (16–64), depth (2–4 layers), width (d=64–128), dropout (0–0.10), and learning rate (10^{-3} to 5×10^{-4}).

- **A3C_v1**: Actor–critic architecture with a shared MLP backbone (two hidden layers, 64 units each) and separate policy and value heads. Trained with RMSprop, entropy regularization coefficient 0.01. *Update strategy:* Per-step updates with a small experience buffer; gradient updates occur when buffer fills (effective batch size 64).

- **A3C_v2**: Advanced actor–critic variant with separate actor and critic networks (no weight sharing), target network stabilization (Polyak averaging with $\tau = 0.02$), entropy regularization (coefficient 0.01), and Adam optimizer with learning rate 2×10^{-3}. *Update strategy:* Immediate per-step updates after each round (no buffering).

Note on update strategies. The different update strategies reflect the distinct inductive biases and stability requirements of each architecture under short-horizon online play. Recurrent predictors (RNN/LSTM) operate in per-round update mode (with LSTM variants additionally using a small replay buffer), which can provide fast adaptation to local non-stationarities. Transformer predictors use buffered mini-batch updates (controlled by an internal `update_freq` and a configurable `batch_size`) to reduce gradient variance under sparse online data. Actor–critic agents similarly rely on online updates with optional buffering. Because these update rhythms are not directly comparable in terms of "number of gradient steps", we report end-to-end ms/decision as a unified resource metric (Table S12) rather than forcing a single shared schedule.

S1.2 Classical machine learning (CPU-based)

- **SVM**: Support vector machine with RBF kernel ($\gamma = 0.1$, $C = 1.0$), trained on engineered features summarizing recent action histories (last 20 actions encoded as frequency vectors and transition counts). Refit every 50 rounds.
- **RF**: Random forest classifier with 100 trees, maximum depth 10, minimum samples per leaf 5. Uses the same feature engineering as SVM. Refit every 50 rounds with a sliding window of the last 200 observations.
- **XGB**: Gradient-boosted decision trees (XGBoost) with 50 estimators, maximum depth 6, learning rate 0.1, and L2 regularization ($\lambda = 1.0$). Same feature pipeline and refit schedule as RF.

S1.3 Probabilistic and custom strategies

- **B_v1**: Frequency-tracking baseline using Dirichlet–multinomial inference. Maintains a Dirichlet posterior over opponent action frequencies with concentration parameter $\alpha_0 = 1.0$ (uniform prior) and best-responds to the posterior mean.
- **B_v2**: Adaptive frequency tracker that detects non-stationarity via a change-point heuristic (comparing recent window statistics to historical statistics) and resets its concentration parameters when a shift is detected.
- **M_v1**: First-order Markov agent that estimates transition probabilities $P(b_t|b_{t-1})$ from observed opponent actions and best-responds to the predicted distribution.
- **M_v2**: Higher-order Markov agent that considers transitions $P(b_t|b_{t-1}, b_{t-2})$ with a back-off scheme: falls back to first-order or zeroth-order estimates when higher-order counts are insufficient (threshold: 5 observations).
- **MSA**: Multi-scale aggregator that maintains frequency statistics over multiple time windows (last 10, 50, 200 rounds) and combines predictions using exponentially weighted averaging, giving more weight to recent windows.

S1.4 Rule-based baselines

- **R**: Uniform random agent that selects each action (Rock, Paper, Scissors) with probability $1/3$, independent of history. Serves as a neutral baseline.
- **CG** (Counter-Guesser): Predicts that the opponent will play the action that beats the agent's most recent action, then plays the counter to that prediction. Exploits naive "beat the last move" opponents.
- **WL** (Win–Lose heuristics): Hand-crafted rules based on the last game outcome. WL_v1: win-stay/lose-shift (repeat action after win, switch after loss). WL_v2: win-shift/lose-stay. WL_v3: deterministic cycling through R$\rightarrow$P$\rightarrow$S regardless of outcome.

S1.5 Implementation notes

All implementations are contained in the `AI_RPS` directory of the code repository. Neural agents use PyTorch 1.12+ and were developed/tested with NVIDIA GPU support (RTX A6000), while classical ML agents use scikit-learn 1.0+ and XGBoost 1.6+. All agents implement the common interface expected by the tournament runner (`RPS_main.py`), in particular `punches(round_idx)` for action selection and `play(my_action, opp_action)` for online updates (with `batch_play` for batch-capable models). Code has been cross-checked for consistency between documented intent and implementation.

S1.6 Statistical methodology

This subsection documents the statistical procedures used to compare method performance across the tournament.

Rationale for seed-paired analysis. Our tournament generates scores under ten independent random seeds, but within each seed the scores of different methods are statistically dependent: all agents face the same sequence of opponents (determined by the tournament schedule) and respond to common random events (e.g., exploration noise, initial conditions). Using unpaired tests such as Mann–Whitney U would treat all 30 observations per method (3 variants $\times$ 10 seeds) as independent, violating the assumption and inflating Type-I error rates when within-seed correlations are positive.

To account for this dependency structure, we adopt a seed-paired design. For each method pair (A,B), we compute the mean score of method A within each seed (averaging over its 3 variants) and likewise for B, yielding ten paired observations $(x_1^A, x_1^B), \ldots, (x_{10}^A, x_{10}^B)$. We then apply the Wilcoxon signed-rank test to the paired differences $d_i = x_i^A - x_i^B$, which tests whether the median of the signed-rank distribution differs from zero. This approach is robust to non-normality and accounts for seed-level confounds.

Multiple comparison correction. With 18 method families, there are $\binom{18}{2} = 153$ pairwise comparisons. We apply the Holm–Bonferroni step-down procedure to control the family-wise error rate at $\alpha = 0.05$. Results are reported in the analysis output file `nonparam_wilcoxon_holm.csv`, which includes raw p-values, adjusted p-values, and significance flags for each method pair.

Implementation note. The statistical analysis is implemented in `utility/analyze_2.py`, function `nonparam_wilcoxon_holm()`. The function pivots the score data so that each row corresponds to a seed and each column to a method, then applies the Wilcoxon signed-rank test (via `scipy.stats.wilcoxon` with `zero_method="pratt"`) to each method pair.

S2 Validation opponent set (Core-19)

To reduce the risk of tuning on the final benchmark, we select neural hyper-parameters on a held-out validation tournament consisting of a compact but diverse opponent set ("Core-19"). The set contains representative instances from each major family (rule-based, probabilistic, classical ML, deep sequence models and RL) and is used only for configuration selection. This separation is designed to help ensure that the 54-agent benchmark results are not biased by hyper-parameter choices optimized on the same population.

Table S2. Core-19 validation opponent set (`Agent_seats/val_core19.csv`). This roster is reserved for hyper-parameter selection and sensitivity checks.

Seat	Agent ID	Agent family type
1	`01_R`	Baseline
2	`02_CG`	Baseline
3	`03_WL`	Baseline
4	`04_B_v1`	Probabilistic
5	`05_B_v2`	Probabilistic
6	`06_M_v1`	Probabilistic
7	`07_M_v2`	Probabilistic
8	`08_MSA_v2`	Probabilistic
9	`09_SVM`	Classical machine learning
10	`10_RF`	Classical machine learning
11	`11_XGB`	Classical machine learning
12	`12_RNN_v2`	Deep neural network
13	`13_RNN_v2`	Deep neural network
14	`14_LSTM_v1`	Deep neural network
15	`15_LSTM_v2`	Deep neural network
16	`16_Tr_v1`	Deep neural network
17	`17_Tr_v2`	Deep neural network
18	`18_A3C_v1`	Reinforcement learning
19	`19_A3C_v2`	Reinforcement learning

S3 Validation results and transformer sweep

The Core-19 validation tournament (Table S3) is used for tuning and sensitivity checks, and is not used for selecting the final 54-agent benchmark ranking. Because ranking in non-transitive settings can depend strongly on the opponent population, these validation results are reported separately from the main benchmark. This population dependence is a characteristic feature of non-transitive games and motivates our emphasis on population-level evaluation throughout the paper.

To directly address potential concerns that Transformer baselines are under-tuned, we also run a targeted Transformer hyper-parameter sweep on a validation sweep roster that includes multiple Transformer configurations. Table S4 and Figure S1 summarize the sweep: across all tested configurations, Transformer variants remain negative, and the best configuration (TrA) still has negative mean score. This provides evidence that the observed Transformer performance pattern is not solely an artifact of a single poor hyper-parameter choice.

Table S3. Core-19 validation results aggregated by method (10 seeds). Scores are total win–loss points over the full validation tournament; 95% confidence intervals are computed across seeds.

Method	n	Mean	Std	$\mathrm{CI}_{2.5\%}$	$\mathrm{CI}_{97.5\%}$
`A3C_v2`	10	3985.4	676.9	3565.8	4405.0
`RNN_v2`	20	932.8	478.8	722.9	1142.7
`M_v1`	10	354.4	178.4	243.8	465.0
`LSTM_v1`	10	338.7	180.5	226.8	450.6
`M_v2`	10	229.9	180.4	118.1	341.7
`A3C_v1`	10	38.0	199.8	-85.8	161.8
`R`	10	9.1	135.1	-74.6	92.8
`CG`	10	-20.3	68.5	-62.8	22.2
`LSTM_v2`	10	-117.6	240.9	-266.9	31.7
`RF`	10	-163.4	676.9	-582.9	256.1
`WL`	10	-309.6	276.8	-481.2	-138.0
`MSA_v2`	10	-321.6	120.5	-396.3	-246.9
`B_v2`	10	-341.3	105.8	-406.9	-275.7
`B_v1`	10	-352.1	83.3	-403.7	-300.5
`SVM`	10	-463.7	387.9	-704.1	-223.3
`Tr_v1`	10	-1083.6	288.2	-1262.2	-905.0
`Tr_v2`	10	-1797.1	451.1	-2076.7	-1517.5
`XGB`	10	-1850.8	1051.7	-2502.7	-1198.9

Table S4. Transformer hyperparameter sweep on the validation sweep roster (10 seeds). Each configuration varies context length (ctx), model width (d), depth (layers), dropout, and learning rate (lr). Scores are total win–loss points; 95% confidence intervals are computed across seeds.

Config	ctx	d	L	dropout	lr	Mean	$\mathrm{CI}_{2.5\%}$	$\mathrm{CI}_{97.5\%}$
`TrA`	16	64	2	0.05	1e-03	-882.1	-1123.1	-641.1
`TrF`	64	64	2	0.05	1e-03	-1597.5	-1913.0	-1282.0
`TrE`	32	64	2	0.00	1e-03	-1784.9	-1967.2	-1602.6
`TrB`	32	64	3	0.05	1e-03	-1933.8	-2215.6	-1652.0
`TrC`	32	96	3	0.10	5e-04	-1957.6	-2199.8	-1715.4
`TrG`	32	128	3	0.05	5e-04	-2019.9	-2275.0	-1764.8
`TrD`	64	96	4	0.10	5e-04	-2280.9	-2677.1	-1884.7
`TrH`	64	128	4	0.05	5e-04	-2671.5	-2912.7	-2430.3

Extended sweep (planned). To further reduce the risk that transformer performance is limited by an overly narrow tuning space, we also provide a ready-to-extend configuration template that can be expanded to longer contexts (e.g., 100–200) and wider/deeper models (e.g., $d \geq 128$, $L \geq 4$) under the same training budget. Results from such extended sweeps can be inserted into Table S4 if additional runs are performed.

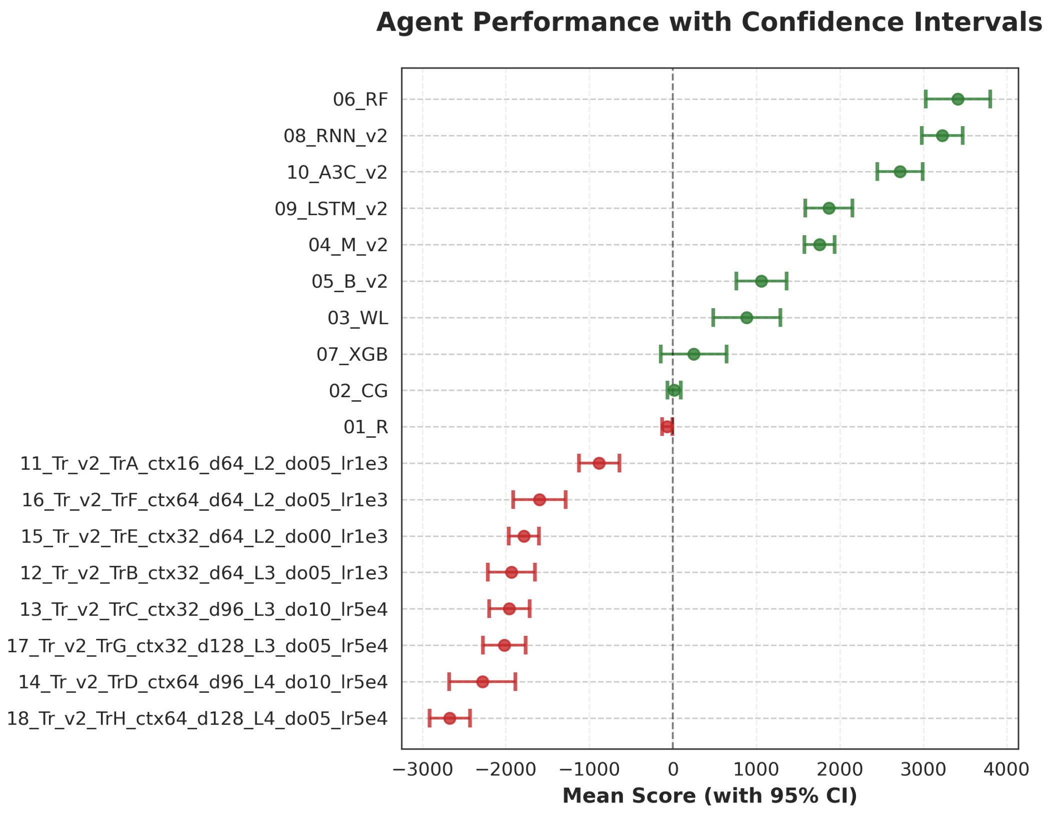

Figure S1. Transformer sweep results on the validation sweep roster (10 seeds). Points show mean score with 95% confidence intervals. Even the best configuration remains negative under our online update budget and short-context design, supporting that the observed Transformer performance pattern is not explained by a single hyperparameter choice within the explored budget.

S4 Transformer configuration traceability

To support reproducibility and address potential concerns about under-tuning, this section documents the exact Transformer configurations used in the final Core-54 benchmark and their relationship to the hyper-parameter sweep.

S4.1 Final benchmark configurations

The `Tr_v1` and `Tr_v2` agents used in the 54-agent tournament employ the following hyper-parameters:

Table S5. Transformer configurations used in the final Core-54 benchmark. These are the codebase defaults and were frozen before final evaluation.

Agent	ctx	d	Layers	Heads	Dropout	lr
`Tr_v1`	32	64	1	4	0.00	10^{-3}
`Tr_v2`	32	64	2	4	0.05	10^{-3}

S4.2 Mapping to sweep configurations

The sweep (Table S4) explored variations around these defaults:

- `Tr_v1` (L=1) was not directly included in the sweep, which focused on L$\geq$2 configurations.
- `Tr_v2` closely matches **TrE** (ctx=32, d=64, L=2, dropout=0.00, lr=10^{-3}), which achieved mean -1784.9.
- The best sweep configuration **TrA** (ctx=16, d=64, L=2, dropout=0.05) achieved mean -882.1, suggesting that *shorter* contexts may be preferable under our 500-round online update protocol.

S4.3 Implications for fairness

We emphasize that:

1. The benchmark configurations (`Tr_v1`, `Tr_v2`) were not selected post-hoc from the sweep; they are codebase defaults frozen before the final tournament.
2. The sweep indicates that within the explored space, no configuration achieves positive scores on Core-19, and the best (TrA) remains negative.
3. The finding that shorter contexts (TrA: ctx=16) outperform longer contexts may reflect an inductive-bias mismatch rather than under-tuning alone: Transformers may struggle with sample efficiency in our 500-round online setting regardless of context length.

Future work with longer interaction horizons (e.g., 5000+ rounds) or larger pretraining corpora may benefit from configurations with longer contexts and more layers; our results characterize performance specifically under the current protocol.

S5 Additional robustness checks

This section documents sensitivity analyses that support the robustness of our qualitative conclusions to reasonable variations in experimental protocol. These checks are important because multi-agent learning outcomes can be sensitive to implementation details, and we want to help ensure that our findings about recurrent architectures and transformer performance patterns are not artifacts of specific choices.

Update-budget sensitivity. We vary the online update frequency (batch size and update interval) for the neural predictors while holding the tournament protocol fixed, to examine whether conclusions are driven by an idiosyncratic training schedule.

Context-length sensitivity. We vary the history window length used by each agent (when applicable) and report score curves and variance across seeds.

Reward scaling. The stage payoff is fixed to $\{-1,0,1\}$ in all main experiments. Any affine transformation of the payoff (e.g., shifting to remove negative values) preserves best-response structure but rescales cumulative scores and the constants in Lipschitz-type bounds. Because our neural predictors are trained by predicting opponent actions (supervised) rather than directly optimizing reward magnitude, affine transformations do not change their decisions; we therefore treat payoff rescaling as a reporting convention rather than a distinct experimental condition. For RL-style agents whose gradients scale with reward magnitude, we report a small sensitivity study.

S6 Full 54-agent tournament: additional analyses

This section provides additional analyses for the main 54-agent, 500-round, 10-seed tournament reported in the main manuscript. These supplementary diagnostics extend the main results by examining stability across random seeds, temporal dynamics of score accumulation, and providing the complete agent-level ranking for full transparency.

S6.1 Stability across random seeds

Beyond average performance, stability across random seeds is an important criterion for evaluating learning algorithms in non-stationary environments. Figure S2 plots each agent's mean score against its coefficient of variation (CV), computed over seeds. High-scoring agents tend to maintain large positive mean scores with moderate relative variability, while agents with near-zero mean exhibit inflated CV due to small denominators.

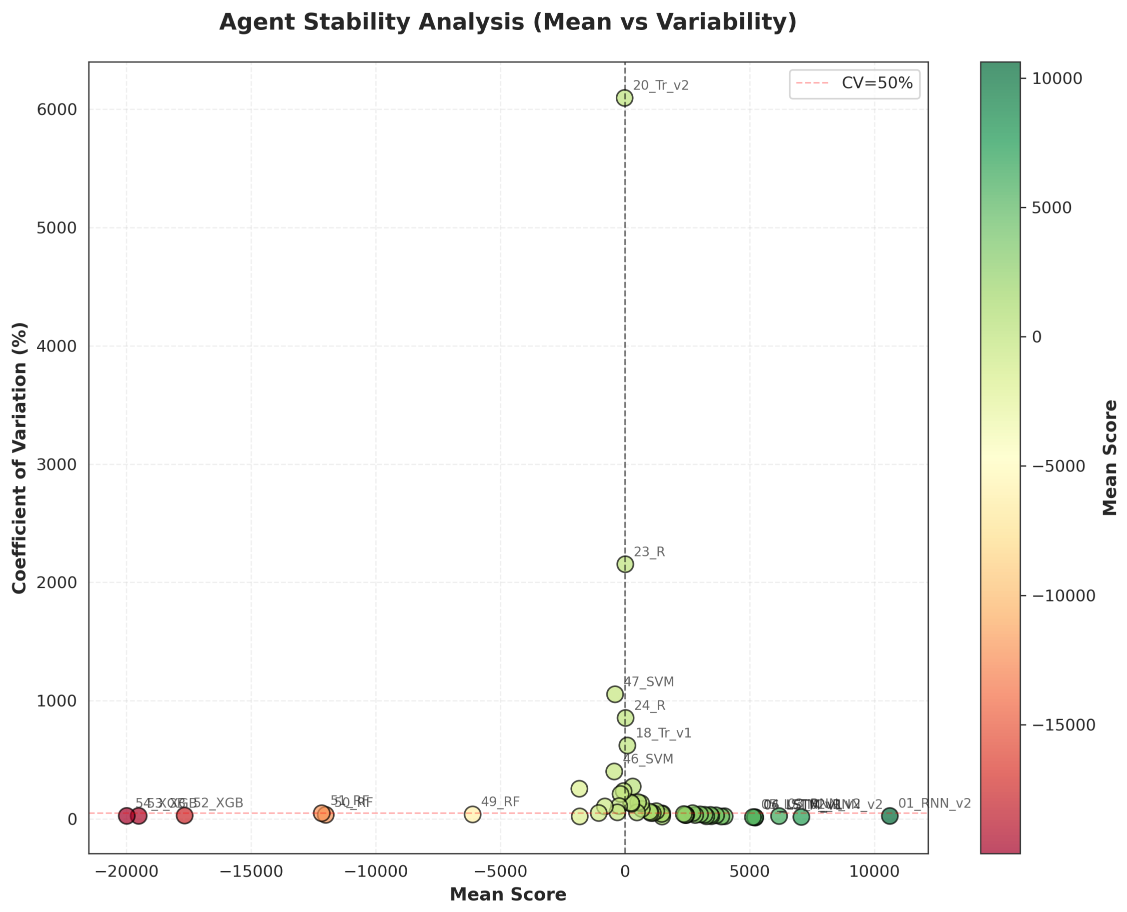

Figure S2. Stability analysis across seeds: mean score versus coefficient of variation (CV). Points far to the right correspond to agents with consistently high scores; large CV values often occur for agents with mean scores near zero.

S6.2 Cumulative score trajectories over the tournament schedule

To understand whether performance advantages are persistent throughout the tournament or concentrated in specific matchups, we examine cumulative score trajectories. Figure S3 shows cumulative score trajectories (aggregated by method) as the tournament progresses through the ordered matchups. This visualization helps interpret whether a method's advantage is persistent or concentrated in a subset of matchups.

S6.3 Full agent ranking

For complete transparency and reproducibility, we provide the full ranking of all 54 agents. Table 1 in the main text reports only the Top-5 agents; the complete ranking below enables independent verification of all agent-level results and supports detailed inspection of where specific agents fall in the distribution.

Note: The complete 54-row table is available in the supplementary data files (`outputs/full_ranking.csv`).

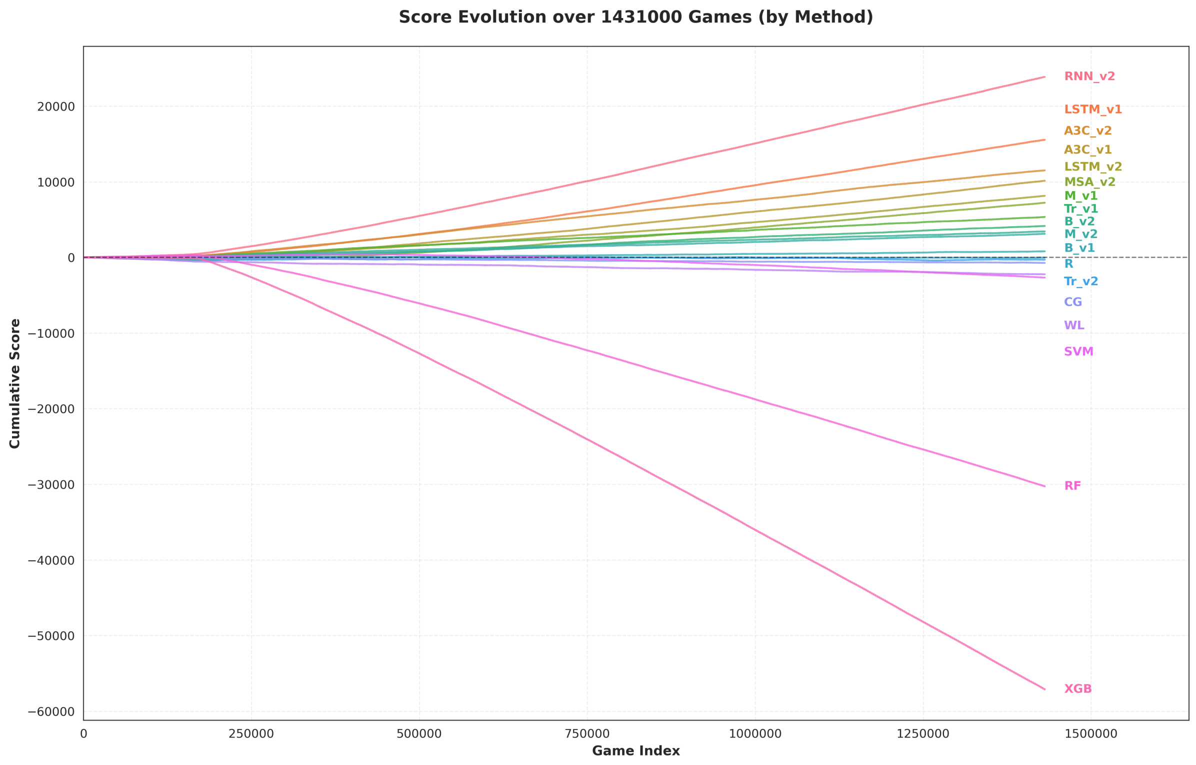

Figure S3. Cumulative score trajectories over 1,431,000 evaluated game rounds per seed (54 agents × 53 opponents × 500 rounds), aggregated by method. Top-ranked methods maintain a consistent advantage throughout the tournament schedule.

Table S6. Full ranking of all 54 agents by mean score across ten seeds. Scores are cumulative sums of the normalized payoff $(+1/0/-1)$ over all opponents and both directions in the double round-robin tournament.

Rank	Agent	Mean	Std	95% CI
1	01_RNN_v2	10618.1	2669.9	[8963.3, 12272.9]
2	02_RNN_v2	7071.0	1189.8	[6333.6, 7808.4]
3	03_RNN_v2	6177.3	1514.5	[5238.6, 7116.0]
4	06_LSTM_v1	5217.5	599.2	[4846.1, 5588.9]
5	04_LSTM_v1	5213.0	794.2	[4720.7, 5705.3]
6	05_LSTM_v1	5129.3	783.2	[4643.9, 5614.7]
7	13_A3C_v2	3995.9	908.8	[3432.6, 4559.2]
8	14_A3C_v2	3863.6	806.8	[3363.5, 4363.7]
9	15_A3C_v2	3653.0	1125.1	[2955.6, 4350.4]
10	11_A3C_v1	3459.4	792.0	[2968.5, 3950.3]
		⋮		
50	50_RF	-12003.3	4136.0	[-14566.8, -9439.8]
51	51_RF	-12162.4	5889.1	[-15812.5, -8512.3]
52	52_XGB	-17660.6	5084.5	[-20812.0, -14509.2]
53	53_XGB	-19504.8	5033.7	[-22624.7, -16384.9]
54	54_XGB	-19981.8	5160.3	[-23180.2, -16783.4]

S7 Non-transitive meta-game structure

This section provides analysis supporting the presence of non-transitive structure in our tournament, complementing the population-dependent ranking analysis in the main text. We analyze the pairwise payoff matrix, enumerate cyclic dominance relations, and compute α-Rank evolutionary dynamics.

S7.1 Pairwise payoff analysis

For the Core-19 validation roster, we construct a method-aggregated pairwise payoff matrix G. Core-19 contains 19 seat instances but only 18 distinct method families (two seats correspond to `RNN_v2`); to obtain a square meta-game over distinct methods, we aggregate seat-level outcomes by method name. Each entry $G(i,j)$ is the mean cumulative directed score of method i against method j (row player i, column player j), averaged over seeds (and averaged over duplicated seats when applicable).

In an ideal symmetric zero-sum setting the expected meta-game matrix is skew-symmetric, but in our finite-sample tournaments with online learning dynamics and direction-specific matchups, the empirical matrix need not be exactly antisymmetric. We therefore treat G as a directed meta-game and do not enforce antisymmetry. The resulting Core-19 matrix has Frobenius norm 1194.33 and rank 18 (full rank for 18 methods), indicating substantial interaction structure beyond a single transitive ordering. We use $G(i,j) > 0$ as the directed dominance relation when enumerating three-cycles.

S7.2 Enumeration of three-cycles

A three-cycle $A \succ B \succ C \succ A$ exists when $G(A,B) > 0$, $G(B,C) > 0$, and $G(C,A) > 0$. We enumerate all such cycles in the Core-19 payoff matrix. Table S7 lists the detected cycles, sorted by the minimum edge strength (the smallest G value in the cycle).

Table S7. Selected three-cycles detected in the Core-19 pairwise payoff matrix. Each row shows a cycle $A \succ B \succ C \succ A$ with the minimum edge strength (weakest link) and average edge strength. A total of 177 three-cycles were detected; the table shows the top cycles by minimum edge strength. Cycles with stronger edges provide more robust evidence of non-transitivity.

Agent A	Agent B	Agent C	Min edge	Avg edge
`B_v1`	`B_v2`	`M_v1`	37.30	54.23
`B_v2`	`M_v1`	`M_v2`	25.50	46.00
`B_v1`	`B_v2`	`RF`	24.50	34.63
`B_v1`	`XGB`	`SVM`	24.20	54.57
`LSTM_v1`	`RF`	`SVM`	18.20	65.87
`LSTM_v1`	`XGB`	`SVM`	18.20	90.33
`B_v2`	`M_v1`	`RF`	17.10	58.10
`A3C_v1`	`RF`	`SVM`	16.90	56.43
`A3C_v1`	`XGB`	`SVM`	16.90	70.63
`B_v1`	`B_v2`	`SVM`	15.90	25.80
⋮	⋮	⋮	⋮	⋮

The full list of detected cycles can be regenerated using `utility/analyze_metagame.py`, which writes `detected_3_cycles.csv`.

Interpretation. The presence of multiple three-cycles suggests that pairwise dominance relations do not form a directed acyclic graph (DAG), and hence no total ordering of agents is consistent with all pairwise outcomes. This is a defining characteristic of non-transitive competition and supports the paper's focus on population-dependent evaluation.

S7.3 α-Rank evolutionary analysis

α-Rank provides an evolutionary game-theoretic ranking by modeling a Markov chain over strategy profiles where transition probabilities depend on payoff differences and a selection intensity parameter α. Unlike Elo or TrueSkill, α-Rank does not assume transitivity and can assign non-trivial mass to multiple strategies in the stationary distribution.

We compute the α-Rank stationary distribution for the Core-19 payoff matrix with $\alpha = 0.1$ (moderate selection pressure). Figure S4 shows the resulting mass allocation across agents.

Key observations:

- The stationary distribution is not concentrated on a single agent, suggesting that no strategy dominates all others under evolutionary dynamics.

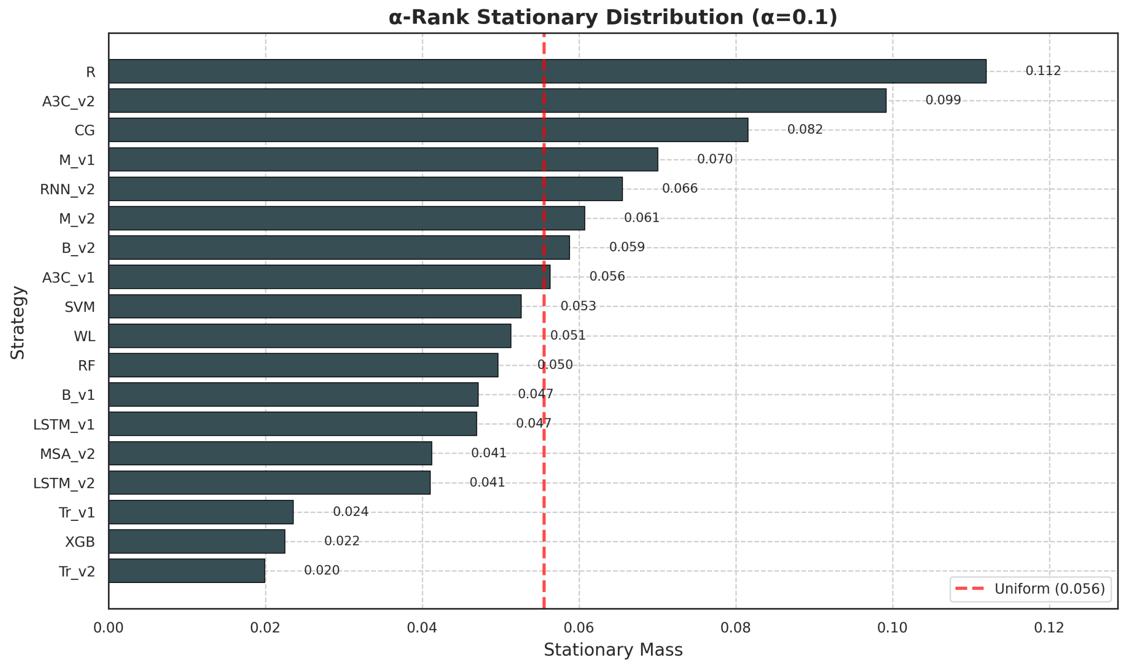

Figure S4. α-Rank stationary distribution over Core-19 agents ($\alpha = 0.1$). The uniform Random baseline (R) receives the highest mass (11.2%), followed by A3C_v2 (9.9%) and CG (8.2%). The dashed red line indicates uniform mass (5.6% per agent). This counterintuitive result—where R outranks sophisticated learners—reflects the evolutionary logic of α-Rank: in a non-transitive setting, strategies that avoid catastrophic losses may survive better under perturbed best-response dynamics.

- The uniform Random baseline (R) receives the highest mass (11.2%), a counterintuitive result explained by the evolutionary logic of α-Rank: in a non-transitive meta-game with many cycles, strategies that avoid strong losses may survive better under perturbed best-response dynamics.
- A3C_v2 receives the second-highest mass (9.9%), followed by CG (8.2%) and M_v1 (7.0%), consistent with their robustness in pairwise matchups.
- Transformer variants receive near-uniform or below-uniform mass (Tr_v1: 2.4%, Tr_v2: 2.0%), suggesting they are evolutionarily less stable under the α-Rank dynamics.

S7.4 Rank correlation across evaluation pools

Table S8 quantifies how method rankings change across different opponent pools, providing a measure of population dependence.

Table S8. Rank correlations between evaluation pools. Spearman ρ and Kendall τ are computed on the intersection of methods present in both pools.

Pool A	Pool B	# common	Spearman ρ	Kendall τ
Core-54	Core-19	18	0.651	0.503
Core-54	Top-R	8	0.52	0.29
Core-54	Pack4	4	0.80	0.67
Core-19	Top-R	8	0.738	0.571

S8 Additional Lipschitz analyses

This section extends the Lipschitz bound diagnostics presented in the main text (Figure 3 and Table 3) to an additional regime: a predictor-vs.-predictor matchup where both agents are pre-trained. While the theoretical bound $\Delta_t \leq 2\|p_t - \hat{p}_t\|_1$ continues to hold, the empirical correlation between prediction error and regret is weaker in this setting because many rounds already achieve near-zero regret. This illustrates that the worst-case bound can become loose once an opponent is effectively "solved."

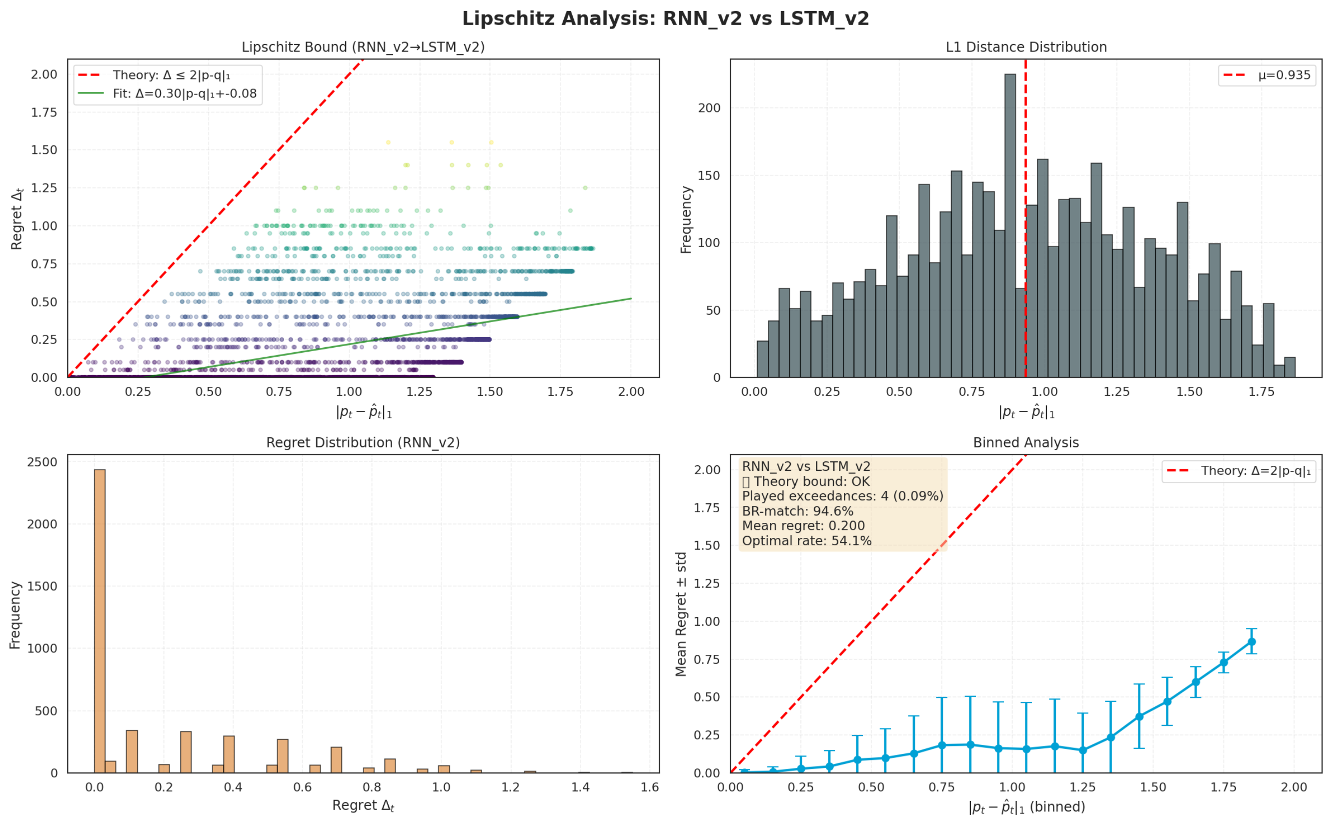

Figure S5. Lipschitz analysis for the directed matchup RNN_v2→LSTM_v2 when both agents are loaded from the core-20 training run (10 seeds, rounds 51–500, $K = 20$). The theoretical envelope $2\|p_t - \hat{p}_t\|_1$ continues to hold for regret defined via best responses, but the empirical correlation is weak because many rounds have near-zero regret despite varying prediction error.

S9 Trained vs. untrained controls

A common concern in learning-in-games benchmarks is whether architectural performance gaps are artifacts of initialization or pretraining rather than differences in online learning capacity. Here "pre-trained" refers to checkpoints produced by a separate 20-agent training tournament ("core-20", seat file `Agent_seats/test_1_20agents.csv`) run under the same 500-round interaction protocol; neural agents update online during this run and we save their final parameters at the end of each seed run, which are then loaded as initial states in subsequent controlled comparisons.

To isolate training effects, we compare these pre-trained checkpoints against untrained counterparts (fresh initialization) in both head-to-head and mixed-population settings. The results indicate that trained recurrent and A3C variants achieve notably higher scores than their untrained versions in the mixed population, while Transformer remains negative in both trained and untrained conditions—consistent with the interpretation that the Transformer's weaker performance under our budget is not solely due to poor initialization.

Table S9. Mixed population containing trained and untrained variants of four architectures (A3C, RNN, LSTM, Transformer). Scores are aggregated over a 500-round round-robin tournament and summarized across 10 seeds (mean and 95% CI).

agent	mean	95% CI
`A3C_v2`	1875.9	[839.4, 2912.4]
`RNN_v2`	1571.3	[1208.8, 1933.8]
`A3C_v2un`	1472.2	[1065.5, 1878.9]
`RNN_v2un`	371.8	[230.5, 513.1]
`LSTM_v2`	249.0	[35.2, 462.8]
`LSTM_v2un`	-468.5	[-710.9, -226.1]
`Tr_v2un`	-2517.2	[-2710.9, -2323.5]
`Tr_v2`	-2554.5	[-2752.2, -2356.8]

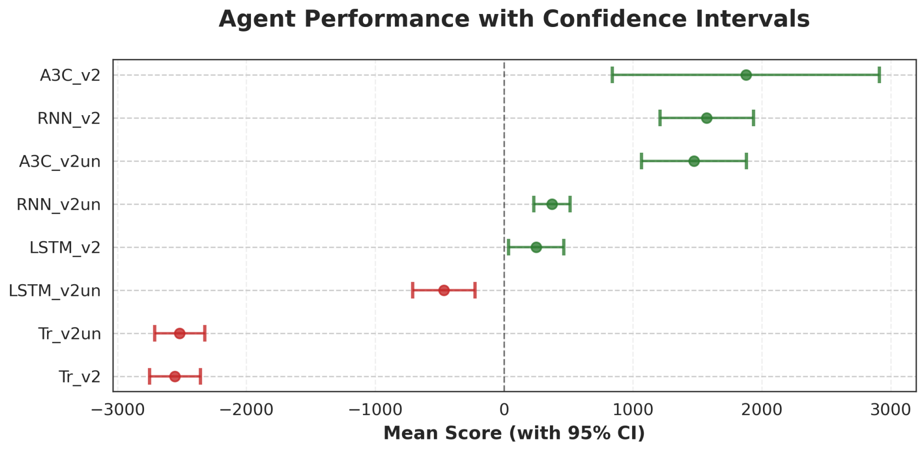

Figure S6. Mean score with 95% confidence intervals for the mixed trained/untrained eight-agent population (Table S9). Trained recurrent and A3C agents outperform their untrained counterparts, whereas Transformer remains strongly negative regardless of training.

S10 Stronger opponent pool (Top-R roster)

To probe sensitivity to opponent-pool strength, we rerun a reduced tournament restricted to a stronger nine-agent roster that excludes highly exploitable baselines (RF, XGB). This tests whether the recurrent advantage persists in a tougher environment where "easy wins" against weak opponents are removed. The ranking remains largely consistent: RNN_v2 is still the top-ranked agent, suggesting that the recurrent advantage is not solely due to exploitation of weak baselines.

Table S10. Results for a stronger nine-agent roster that excludes highly exploitable baselines. Scores are summarized across 10 seeds (mean and 95% CI).

agent	mean	95% CI
RNN_v2	825.4	[325.5, 1325.3]
RNN_v1	613.0	[98.8, 1127.2]
A3C_v2	146.2	[-820.7, 1113.1]
LSTM_v1	76.6	[-129.1, 282.3]
CG	-39.5	[-103.8, 24.8]
M_v1	-60.1	[-93.5, -26.7]
M_v2	-109.9	[-165.2, -54.6]
LSTM_v2	-673.1	[-808.9, -537.3]
A3C_v1	-778.6	[-1068.7, -488.5]

S11 Sliding-window length sensitivity for Lipschitz analysis

The Lipschitz analysis in the main text uses a sliding window of length $K = 20$ to estimate the empirical opponent distribution p_t. This section examines the sensitivity of the diagnostic to the choice of K. The theoretical bound $\Delta_t \leq 2\|p_t - \hat{p}_t\|_1$ holds for all tested values ($K \in \{5, 10, 20, 50\}$), though the empirical correlation between prediction error and regret varies with K. Smaller windows are noisier but more responsive to recent changes; larger windows are smoother but may lag behind distribution shifts.

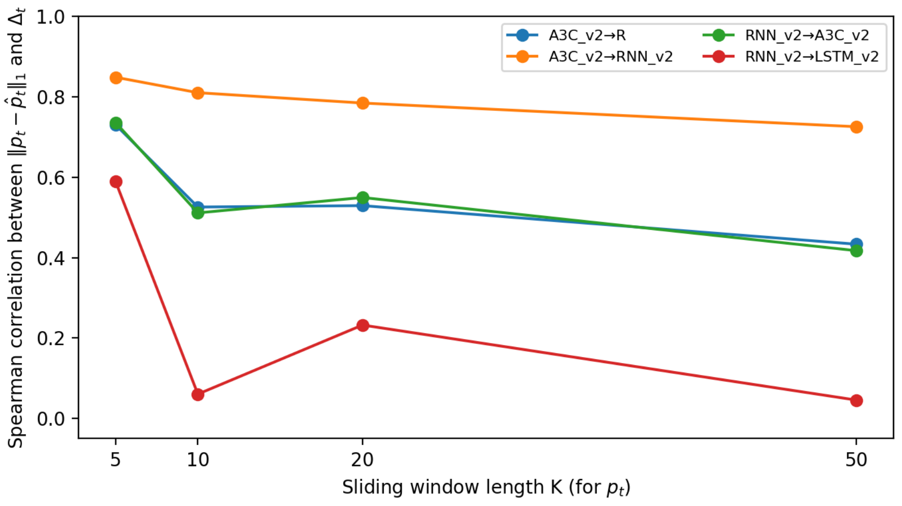

Figure S7. Sensitivity of the Lipschitz diagnostic to the sliding-window length K used to estimate the empirical opponent distribution p_t. Each curve reports Spearman correlation between $\|p_t - \hat{p}_t\|_1$ and instantaneous regret Δ_t (computed against the best response to $\hat{p}_t$).

Table S11. Summary Lipschitz diagnostics across window lengths $K \in \{5, 10, 20, 50\}$. "Slope" is the coefficient of a linear regression $\Delta_t = a\|p_t - \hat{p}_t\|_1 + b$ fit on non-zero pairs. "Viol.%" is the fraction of samples violating $\Delta_t \leq 2\|p_t - \hat{p}_t\|_1$ (0% indicates full satisfaction up to numerical tolerance).

Matchup	K	$\mu(\|p - \hat{p}\|_1)$	$\mu(\Delta)$	Spearman ρ	Slope a	Viol.%
A3C_v2→R	5	0.590	0.388	0.730	0.859	0.00
A3C_v2→R	10	0.452	0.262	0.526	0.600	0.00
A3C_v2→R	20	0.369	0.175	0.529	0.419	0.00
A3C_v2→R	50	0.312	0.100	0.433	0.240	0.00
A3C_v2→RNN_v2	5	0.785	0.636	0.848	1.112	0.00
A3C_v2→RNN_v2	10	0.675	0.524	0.810	1.091	0.00
A3C_v2→RNN_v2	20	0.579	0.416	0.784	1.061	0.00
A3C_v2→RNN_v2	50	0.488	0.294	0.725	0.895	0.00
RNN_v2→A3C_v2	5	0.517	0.232	0.736	0.760	0.00
RNN_v2→A3C_v2	10	0.444	0.177	0.511	0.632	0.00
RNN_v2→A3C_v2	20	0.425	0.148	0.549	0.535	0.00
RNN_v2→A3C_v2	50	0.463	0.121	0.417	0.330	0.00
RNN_v2→LSTM_v2	5	0.488	0.132	0.590	0.638	0.00
RNN_v2→LSTM_v2	10	0.665	0.151	0.061	0.135	0.00
RNN_v2→LSTM_v2	20	0.935	0.200	0.232	0.117	0.00
RNN_v2→LSTM_v2	50	0.961	0.102	0.046	0.003	0.00

S12 Computational cost benchmarking

To address potential concerns about computational cost comparisons and ensure budget comparability across agent families, we report an end-to-end wall-clock benchmark for representative agents. We run a minimal two-agent match (method vs. the random baseline `R`) for 500 rounds, repeated over the same ten random seeds used throughout the paper (`1,2,3,5,8,13,21,34,55,89`). We measure the total elapsed time T and report

$$\text{ms/decision} = 1000 \times \frac{T}{2 \times \text{rounds} \times n_{\text{seeds}}}, \tag{1}$$

where the factor 2 counts decisions from both players. The benchmark includes forward inference and any online updates performed by the agent in our implementation; it is therefore an end-to-end cost indicator for our framework (not a microbenchmark of model inference only). The script used to generate Table S12 is provided as `utility/benchmark_runtime_end2end.py` in the code release.

Table S12. End-to-end wall-clock runtime for representative agents (CPU). Each entry reports milliseconds per decision computed from the total elapsed time of a 2-agent match against the random baseline `R` for 500 rounds, aggregated across the 10 seeds used in the paper. Parameter counts refer to trainable `torch` parameters of the agent's neural network component (0 for non-neural baselines).

Method	# params	ms/decision (CPU)	Notes
`B_v1` (freq-tracking)	—	0.101	Frequency-tracking baseline (Dirichlet posterior).
`M_v1`	—	0.095	Simple Markov baseline.
`A3C_v2`	25 604	1.302	Actor–critic; includes online updates.
`Tr_v2`	564 291	2.018	Transformer predictor; includes online updates.
`RNN_v2`	13 443	2.256	GRU-based predictor; includes online updates.
`LSTM_v2`	79 283	5.478	LSTM predictor; includes online updates.