PROMISES AS A COMMON LANGUAGE FOR ZERO-SHOT COORDINATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Zero-shot coordination (ZSC) considers cooperation with unseen partners without additional joint training. Existing approaches typically seek behavioral diversity by varying training partners, environments, or action distributions, but such diversity does not necessarily yield meaningful behaviors for coordination. To address this limitation, we propose *Promise-based Partner Coordination* (PPC), which structures partner diversity through *promises* that encode distinct coordination patterns. Inspired by Promise Theory, PPC leverages an LLM to design executable promises from task descriptions and iteratively refine them through interaction-based feedback, inducing diverse coordination patterns without manually designing partner behaviors. We train a population of *Promisers* to follow different promises and a *Promisee* to coordinate with them based on their inferred reliability. Across diverse coordination environments, PPC substantially improves both self-play and cross-play performance over existing approaches and generalizes effectively to human partners. These results demonstrate the effectiveness of promises as a common language for coordination with unfamiliar partners.

1 INTRODUCTION

Cooperation is fundamental to multi-agent systems, where agents must coordinate with partners beyond those encountered during training. Zero-shot coordination (ZSC) considers the setting where agents must cooperate with diverse, independently trained partner policies without additional joint training (Hu et al., 2020). A prevailing approach is to expose agents to diverse behaviors during training, for example through diverse partner populations or behavioral variations (Strouse et al., 2021; Yan et al., 2023). Yet greater behavioral diversity does not necessarily yield more meaningful behaviors for coordination. Different behaviors may reflect similar coordination strategies, making diversity alone insufficient to characterize how agents coordinate. The key is therefore not only *how diverse* the training partners are, but also *what coordination behaviors* their diversity represents. This raises a central question: *How can we create diverse partner behaviors that remain meaningful for coordination?*

To address this question, we seek an answer through *promises*, motivated by how humans coordinate with one another. Figure 1 (left) shows that human cooperation often relies on different patterns that establish expectations about how each participant will act. In collaborative cooking, for example, one may promise to hand off ingredients, serve completed dishes, or yield a shared passage. We use such promises to represent distinct ways of coordinating toward a shared goal, as illustrated in Figure 1 (middle). Motivated by Promise Theory (Burgess & Bergstra, 2014), which describes coordination among autonomous agents through explicit promises about their behaviors, we formulate promises as state–action specifications that represent distinct coordination patterns. Training agents to follow different promises can then induce *structured behavioral diversity*, where policies differ in how they coordinate rather than merely in how they act. Such diversity can facilitate coordination with unseen AI and human partners, as shown in Figure 1 (right). The key remaining challenges are to construct useful promises and train agents to effectively follow and respond to them.

We address these challenges with *Promise-based Partner Coordination* (PPC). PPC leverages the reasoning capabilities of a Large Language Model (LLM) to identify useful coordination behaviors from task descriptions and express them as executable promises through action constraints. It iteratively refines these promises using interaction-based feedback to increase behavioral diversity and

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.

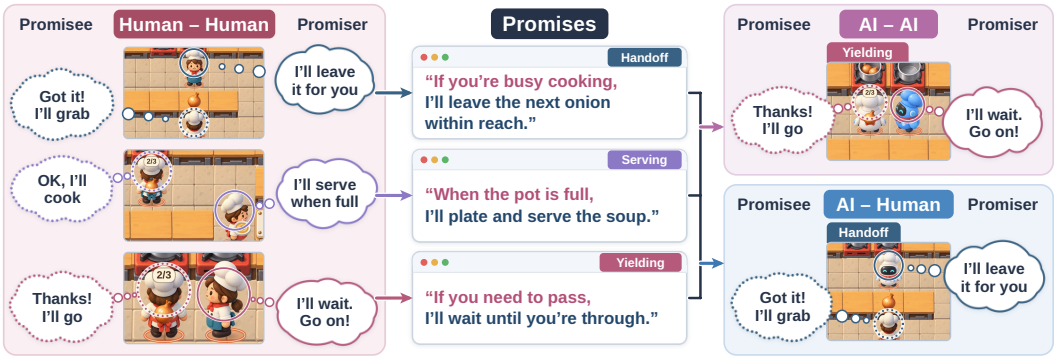


Figure 1: Overview of the motivation for promise-based coordination. Left: Human–human interactions in collaborative cooking. Middle: Promise statements corresponding to the coordination situations shown on the left. Right: AI–AI and AI–human interactions after training with the promises.

improve task performance. We then train a population of *Promisers* under different combinations of promises, yielding partner policies that follow different coordination patterns. A *Promisee* learns to coordinate with these Promisers by estimating how reliably they follow their promises from interaction history, while choosing its own actions autonomously. Across representative ZSC environments, including Overcooked and Hanabi, PPC achieves strong self-play and unseen-partner coordination with both AI and human partners. Further analyses provide a theoretical performance-gap bound and examine the design and refinement of promises, supporting their role in producing meaningful behavioral diversity.

2 Related Work

Zero-Shot Coordination. Prior ZSC methods commonly improve generalization by exposing agents to diverse partner behaviors during training. Population-based approaches broaden partner coverage through self-play runs and checkpoints (Strouse et al., 2021), while other methods promote diversity through trajectory objectives (Lupu et al., 2021), population entropy (Zhao et al., 2023), heterogeneous rewards (Yu et al., 2023), distinct conventions (Sarkar et al., 2023), and differing best-response requirements (Rahman et al., 2023). Other approaches address convention mismatch through symmetry-aware learning (Hu et al., 2020; Muglich et al., 2022), belief-based reasoning (Hu et al., 2021; Cui et al., 2021), policy mixtures (Yan et al., 2023), and partner-context inference (Lou et al., 2023). Recent work further diversifies coordination through latent partner models (Liang et al., 2024), adversarial search (Chaudhary et al., 2026), cross-environment training (Jha et al., 2025), and state-blocking penalties (Kang et al., 2026). In contrast, we use promises to structure partner behaviors around distinct, meaningful coordination patterns.

LLMs for Reinforcement Learning and Coordination. LLMs have been used in RL to generate executable control code (Liang et al., 2023), select skills (Ichter et al., 2023), guide exploration (Du et al., 2023), and design rewards (Kwon et al., 2023; Xie et al., 2024; Ma et al., 2024). In multi-agent settings, they support teammate reasoning and planning (Zhang et al., 2024; Agashe et al., 2025; Hwang et al., 2026), communication (Bae et al., 2026), and language-guided human–AI coordination (Hu & Sadigh, 2023). For ZSC, LLMs have been used to generate rewards for diverse teammates and select ego policy heads at test time (Li et al., 2025). Our approach instead uses an LLM to design executable promises that capture meaningful coordination behaviors and refine them through interaction-based feedback, without requiring LLM calls at test time.

Constrained Reinforcement Learning. Constrained RL optimizes policies under safety, cost, or behavioral constraints (Altman, 1999). Prior methods enforce constraints through trust-region updates (Achiam et al., 2017), adaptive penalties (Tessler et al., 2019), action shielding (Alshiekh et al., 2018), and action correction layers (Dalal et al., 2018; Pham et al., 2018). Related work considers stochastic action availability (Boutilier et al., 2018; Chandak et al., 2020), invalid-action elimination (Zahavy et al., 2018), and state-dependent action masking (Huang & Ontañón, 2020). We formulate each promise as a constrained MDP and use probabilistic action masking to train Promisers to follow the corresponding promise, inducing diverse promise-based coordination behaviors.

3 BACKGROUND

3.1 DEC-POMDP AND ZERO-SHOT COORDINATION

We model an environment in which multiple agents act on local information toward a shared goal as a decentralized partially observable Markov decision process (Dec-POMDP) (Oliehoek & Amato, 2016), denoted by $\mathcal{M} = \langle \mathcal{N}, \mathcal{S}, \mathcal{O}, \mathcal{A}, T, Z, R, \gamma \rangle$. Here, $\mathcal{N} = \{1, \dots, n\}$ is the set of agents, $\mathcal{S}$ is the state space, $\mathcal{O} = \prod_{i \in \mathcal{N}} \mathcal{O}^i$ and $\mathcal{A} = \prod_{i \in \mathcal{N}} \mathcal{A}^i$ are the joint observation and action spaces, $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition kernel, $Z : \mathcal{S} \times \mathcal{A} \to \mathcal{O}$ is the observation kernel, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the shared reward function, and $\gamma \in [0, 1)$ is the discount factor.

Within this Dec-POMDP, the ZSC setting distinguishes an ego agent $e \in \mathcal{N}$ with policy π^e and its partners $\mathcal{N} \setminus \{e\}$ with joint policy $\boldsymbol{\pi}^{-e}$. At timestep t, each agent selects an action $a_t^i \sim \pi^i(\cdot \mid \tau_t^i)$ from its local history $\tau_t^i = (o_0^i, a_0^i, \dots, o_t^i)$. The resulting joint action $\boldsymbol{a}_t = (a_t^i)_{i \in \mathcal{N}}$ yields $s_{t+1} \sim T(\cdot \mid s_t, \boldsymbol{a}_t)$ and the shared reward $r_t = R(s_t, \boldsymbol{a}_t)$. The return for the ego and partner policies is defined as $J(\pi^e, \boldsymbol{\pi}^{-e}) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t r_t]$. The ego policy is trained using RL with a set of training partners to maximize this return, with the goal of coordinating effectively with unseen partners at test time without additional joint training. ZSC performance is measured by $\mathbb{E}_{\boldsymbol{\pi}^{-e} \sim \Pi_{\text{test}}}[J(\pi^e, \boldsymbol{\pi}^{-e})]$, where Π_{test} is the distribution of unseen partner policies, including human partners.

3.2 LARGE LANGUAGE MODELS FOR GENERATION AND REFINEMENT

LLMs are used across diverse domains to generate desired outputs y from input prompts x according to $p_\theta(y \mid x)$, where θ denotes the model parameters. In RL, prompts can incorporate task descriptions, environment interfaces, and implementation requirements depending on the design objective, allowing LLMs to generate natural-language descriptions, structured specifications, or executable code, including policies and reward functions (Liang et al., 2023; Xie et al., 2024). LLM outputs can also be iteratively refined through feedback (Madaan et al., 2023) as $y_{\ell+1} \sim p_\theta(\cdot \mid x, y_\ell, f_\ell)$, where y_ℓ is the output at iteration ℓ and f_ℓ denotes feedback from the LLM itself, human evaluation, or environment execution (Shinn et al., 2023; Xie et al., 2024; Ma et al., 2024). In our framework, the LLM designs multiple categories of promises and their corresponding constraints for coordination and iteratively revises them using numerical feedback collected during training, removing ineffective promises and designing more useful ones.

4 METHOD

4.1 PROMISE-BASED DEC-POMDP FORMULATION

To represent meaningful coordination patterns as explicit behavioral rules, we draw on the concepts of Promise Theory (Burgess & Bergstra, 2014). In Promise Theory, a *promiser* makes a promise about its intended behavior known to a *promisee*, which evaluates its fulfillment and forms *trust*. For ZSC, we instead use promises to structure Promiser behaviors without explicitly communicating the promises to the Promisee. By training with diverse promise-conditioned Promisers, the Promisee learns to infer trust from observed behaviors and autonomously respond to the resulting coordination patterns. We use Pr and Pe to denote the Promiser and Promisee, respectively, as roles rather than fixed agent identities that can be assigned to different agents or policies depending on the interaction. We formalize each promise through state and action conditions in the Dec-POMDP.

Definition 1 (State-Action Promise). *Let $\mathcal{P} = \{p_1, \dots, p_K\}$ denote a finite set of K promises. Each promise $p \in \mathcal{P}$ is defined as*

$$p = (b,\, \mathcal{S}^p,\, \mathcal{A}^p)\,, \tag{1}$$

where b is the promise body specifying the intended coordination behavior, $\mathcal{S}^p \subseteq \mathcal{S}$ is the set of states in which the promise applies, and $\mathcal{A}^p(s)$ is the promise-conditioned joint action set defined as

$$\mathcal{A}^p(s) = \begin{cases} \mathcal{A}^p_{\text{avail}}(s), & s \in \mathcal{S}^p, \\ \mathcal{A}, & s \notin \mathcal{S}^p, \end{cases} \tag{2}$$

where $\mathcal{A}^p_{\text{avail}}(s) \subseteq \mathcal{A}$ denotes the available joint actions consistent with promise p.

A state-action promise thus restricts the joint action space only when $s \in \mathcal{S}^p$. We incorporate a set of such promises into a Dec-POMDP as follows.

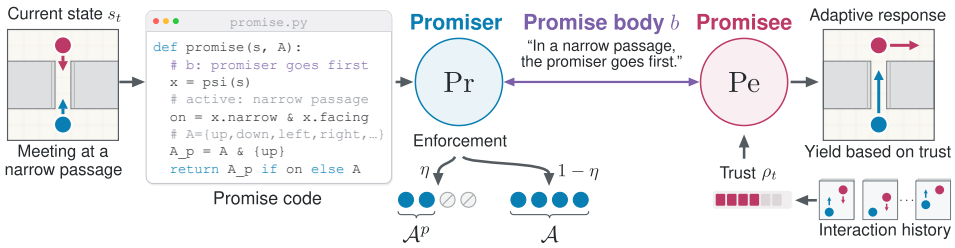


Figure 2: A simple example of promise execution in a PB-Dec-POMDP. The current state activates a promise specifying that the Promiser goes first in a narrow passage. Its action constraint is enforced on the Promiser with probability η, while the Promisee infers trust and adapts its response.

Definition 2 (Promise-Based Dec-POMDP)**.** *Given a Dec-POMDP $\mathcal{M}$ and a finite promise set $\mathcal{P} = \{p_1, \ldots, p_K\}$, we define the Promise-Based Dec-POMDP (PB-Dec-POMDP) as*

$$\mathcal{M}_{\mathcal{P}} = \langle \mathcal{N}, \mathcal{S}, \mathcal{O}, \mathcal{A}, T, Z, R, \gamma, \mathcal{P} \rangle, \tag{3}$$

where each $p_k \in \mathcal{P}$ specifies the promise-conditioned joint action set $\mathcal{A}^{p_k}(s)$.

Under the PB-Dec-POMDP, ZSC is realized through the complementary roles of the Promiser and Promisee. An agent i acting as a Promiser follows $\pi^{\mathrm{Pr},i}(\cdot \mid \tau_t^i, \boldsymbol{\eta})$, conditioned on *enforcement* probabilities $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_K)$, where η_k determines the probability of enforcing promise p_k through $\mathcal{A}^{p_k}(s_t)$. Varying these probabilities induces diverse coordination behaviors across promises. In contrast, an agent i acting as a Promisee follows $\pi^{\mathrm{Pe},i}(\cdot \mid \tau_t^i, \boldsymbol{\rho}_t)$ without direct promise constraints, conditioned on *trust* $\boldsymbol{\rho}_t = (\rho_{t,1}, \ldots, \rho_{t,K})$, where $\rho_{t,k}$ represents how likely the partner is to fulfill promise p_k. This allows the Promisee to adapt to different promise-based behaviors without knowing how an unseen partner enforces the promises.

Theoretical analysis. We analyze how promise enforcement and trust estimation affect coordination performance. We assume full observability, bounded rewards with span at most $r_b > 0$, and fixed stationary policies for the other agents. For the Promiser, we first characterize the performance gap induced by promise enforcement as follows.

Proposition 1 (Performance Gap under Promiser Enforcement)**.** *Consider a promise set $\mathcal{P}$ and define its compliance value gap as*

$$\Delta_{\mathcal{P}}(s) := V^*(s) - V^*_{\mathcal{P}}(s) \geq 0, \tag{4}$$

where V^* and $V^*_{\mathcal{P}}$ are the optimal value functions without and with all active constraints induced by $\mathcal{P}$, respectively. If the promises in $\mathcal{P}$ are independently enforced with probability $\boldsymbol{\eta} \in [0,1]^K$, then

$$J^* - J^*_{\mathcal{P},\boldsymbol{\eta}} \leq \frac{1 - \prod_{k=1}^{K}(1-\eta_k)}{1 - \gamma \prod_{k=1}^{K}(1-\eta_k)} \|\Delta_{\mathcal{P}}\|_\infty. \tag{5}$$

where J^* and $J^*_{\mathcal{P},\boldsymbol{\eta}}$ denote the unconstrained and promise-enforced optimal returns, respectively.

The proof is provided in Appendix B.1. The bound shows that enforcement incurs little cost when the promise preserves actions important for optimal behavior. Moreover, although the bound concerns optimal returns, restricting the action search space can simplify learning and improve empirical performance. For the Promisee, however, the partner's enforcement probabilities are unobserved and must be inferred through trust. We therefore characterize how trust-estimation error affects coordination performance.

Proposition 2 (Performance Gap under Promisee Trust Estimation)**.** *Assume the Promisee's action distribution is uniformly L-Lipschitz in trust from ℓ_∞ to total variation, with $\sup_s \|\boldsymbol{\rho}(s) - \boldsymbol{\eta}\|_\infty \leq \varepsilon_{\boldsymbol{\rho}}$. Then*

$$|J_{\mathcal{P},\boldsymbol{\rho}} - J_{\mathcal{P},\boldsymbol{\eta}}| \leq \frac{r_b L \varepsilon_{\boldsymbol{\rho}}}{(1-\gamma)^2}, \tag{6}$$

where $J_{\mathcal{P},\boldsymbol{\rho}}$ and $J_{\mathcal{P},\boldsymbol{\eta}}$ denote the returns under estimated trust $\boldsymbol{\rho}$ and true enforcement probabilities $\boldsymbol{\eta}$, respectively.

The proof is provided in Appendix B.2. This result indicates that smaller trust-estimation error yields a tighter performance bound. Together, these results motivate designing promises that preserve optimal behavior and accurately estimating their enforcement through trust. We next turn to designing useful promises and training the Promiser and Promisee through RL.

4.2 LLM-GUIDED PROMISE DESIGN AND REFINEMENT

Promise design with state summaries. Useful promises should capture meaningful coordination behaviors without manual specification. We therefore use LLMs to design them from the task specification I_{task}, including the task description, environment dynamics, and action specifications. Given a state-summary prompt x_ψ and I_{task}, the LLM uses Chain-of-Thought reasoning (Wei et al., 2022) to construct a state summary function $\psi \sim p_\theta(\cdot \mid x_\psi, I_{\text{task}})$ mapping states to named, coordination-relevant features that determine when a promise is feasible and which actions can fulfill it. For visual inputs, ψ can be instantiated with a vision–language model (Liu et al., 2023).

Given a promise-design prompt x_p, I_{task}, and ψ, the LLM generates initial promise specifications and executable code as $y_0 \sim p_\theta(\cdot \mid x_p, I_{\text{task}}, \psi)$. Rather than generating all promises independently, the LLM identifies M task-relevant coordination categories, such as handoff, serving, and yielding in Figure 1, to separate distinct aspects of coordination and reduce conflicts among promises. For each category $\mathcal{P}_m$, it generates K/M promises, forming $\mathcal{P} = \bigcup_{m=1}^{M} \mathcal{P}_m$ with $|\mathcal{P}_m| = K/M$ and $|\mathcal{P}| = K$. Each promise specifies its natural-language body b and executable code that uses $\psi(s)$ to determine its activation condition $s \in \mathcal{S}^p$ and the available-action constraint $\mathcal{A}^p_{\text{avail}}(s)$, thereby defining $\mathcal{A}^p(s)$ (Xie et al., 2024). The resulting constraint is enforced according to its enforcement probability only when $s \in \mathcal{S}^p$, allowing promises to guide coordination only where their intended behaviors can be performed.

Promise refinement. During Promiser training, we refine promises at regularly spaced checkpoints using the LLM (Ma et al., 2024; Bae et al., 2026). At each round ℓ, self-play rollouts with fixed policy weights evaluate three properties of each promise: *activation*, *performance*, and *diversity*. For each promise p, let $\mathcal{D}$ denote its Promiser's rollout buffer and $\mathcal{D}_p \subseteq \mathcal{D}$ its active state–action pairs. We measure activation by $|\mathcal{D}_p|/|\mathcal{D}|$, the fraction of steps at which p applies, since a rarely activated promise has little effect on behavior. Performance is an empirical estimate of the compliance value gap in Eq. (4), which Proposition 1 motivates keeping small. For diversity, we define *Promise Mode Separation* (PMS) to measure how distinctly same-category promises p, q constrain actions:

$$
\begin{aligned}
c_{p\to q} &= \mathbb{E}_{(s,\boldsymbol{a})\sim\mathcal{D}_p}\left[\mathbf{1}\{\boldsymbol{a} \in \mathcal{A}^p(s)\} - \mathbf{1}\{\boldsymbol{a} \in \mathcal{A}^q(s)\}\right], \\
\mathrm{PMS}(p,q) &= \max\left\{0, \frac{c_{p\to q} + c_{q\to p}}{2}\right\}.
\end{aligned}
\tag{7}
$$

Here, $c_{p\to q}$ measures the excess compatibility of actions in $\mathcal{D}_p$ with p over q, and higher PMS indicates stronger empirical separation. These three metrics form the refinement feedback f_ℓ. Promises with insufficient activation or a large value gap are labeled REPAIR, while PMS guides IMPROVE revisions that increase separation. The LLM then generates $y_{\ell+1} \sim p_\theta(\cdot \mid x_p, I_{\text{task}}, \psi, y_\ell, f_\ell)$ while keeping θ fixed, and Promiser training continues without resetting policy weights. After the final round, we fix the promise set and train the trust estimator on subsequent interactions. Full prompts, generated code, and refinement details are in Appendix C.

4.3 PROMISER–PROMISEE LEARNING FOR ZERO-SHOT COORDINATION

Promiser learning. To leverage the designed promises for ZSC, we first construct a Promiser population $\Pi^{\text{Pr}} = \{\boldsymbol{\pi}_j^{\text{Pr}}\}_{j=1}^{K/M}$, where $\boldsymbol{\pi}_j^{\text{Pr}} = (\pi_j^{\text{Pr},i})_{i\in\mathcal{N}}$ denotes the j-th joint Promiser policy. Each Promiser j is assigned one promise $p_{m,j} \in \mathcal{P}_m$ from each of the M categories, yielding $\mathcal{P}_j = \{p_{1,j}, \ldots, p_{M,j}\}$, and is trained via RL self-play. Its enforcement vector $\boldsymbol{\eta}_j \in [0,1]^K$ has nonzero entries only for the selected M promises. These entries are independently sampled from $\mathcal{U}(0,1)$ and fixed throughout each episode, while the remaining $K - M$ entries are zero. Varying both promise combinations and enforcement probabilities induces diverse promise-based coordination behaviors.

Promisee learning. Given this diverse Promiser population, we freeze Π^{Pr} and train the Promisee ego policy $\pi^{\text{Pe},e}$ with $\boldsymbol{\pi}_j^{\text{Pr},-e}$ sampled as its partner joint policy. Since the partner promises and enforcement probabilities are unobserved, we learn a trust estimator $\boldsymbol{\rho}_\phi(\tau_t^e)$ from the ego's interaction history τ_t^e. We first pretrain $\boldsymbol{\rho}_\phi$ on Promiser self-play trajectories using the corresponding enforcement probabilities $\boldsymbol{\eta}_j$ as supervision, and then fine-tune it during Promisee–Promiser interactions. The Promisee policy $\pi^{\text{Pe},e}(\cdot \mid \tau_t^e, \boldsymbol{\rho}_\phi(\tau_t^e))$ maximizes task return without promise-based action constraints, allowing it to autonomously respond based on the inferred trust. We additionally perform Promisee–Promisee self-play so that the Promisee also learns to coordinate with other Promisees.

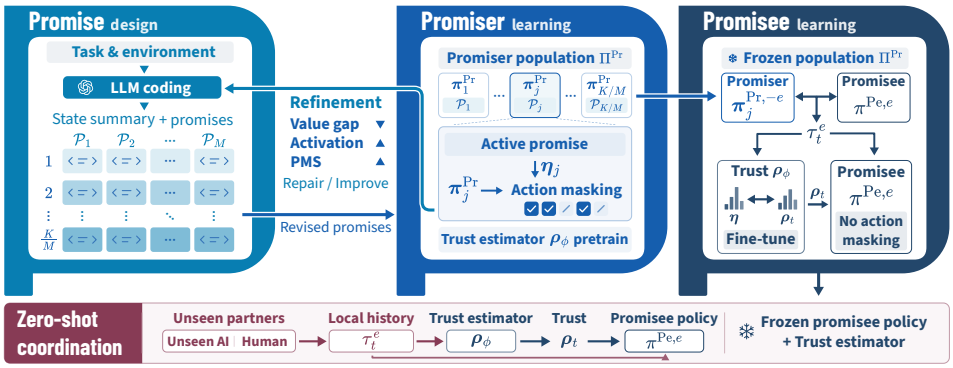


Figure 3: Overview of PPC: LLM-guided promise design and refinement, Promiser population learning, and trust-conditioned Promisee learning for ZSC with unseen AI and human partners.

At test time, the Promisee serves as the ego policy for ZSC and coordinates with diverse unseen partners using trust inferred solely from its local interaction history. Figure 3 illustrates the overall framework, with the full algorithm and implementation details provided in Appendix C.

5 Experiments

5.1 Experimental Setup

Benchmarks and baselines. We evaluate PPC across three cooperative multi-agent benchmarks. *Overcooked-v1* (Carroll et al., 2019) and *Overcooked-v2* (Gessler et al., 2025) are two-agent cooking environments, evaluated on four and six layouts, respectively. Overcooked-v1 tests task allocation and adaptation to different coordination modes, while Overcooked-v2 tests coordination under partial observability, with limited access to recipe and partner information. *Hanabi* (Bard et al., 2020) is a cooperative card game, evaluated in four settings with two to five players. It tests coordination with diverse partner strategies under partial observability across different team sizes.

We compare against **IPPO** (de Witt et al., 2020) as the base MARL algorithm, together with a range of IPPO-based ZSC methods: **OP** (Hu et al., 2020), which enforces symmetry-invariant conventions; **FCP** (Strouse et al., 2021), which trains a best response to a population of self-play partners and their past checkpoints; **MEP** (Zhao et al., 2023), which trains against a maximum-entropy partner population; **E3T** (Yan et al., 2023), which trains against a mixture of ego and random policies with partner modeling; **HSP** (Yu et al., 2023), which builds biased partners from hidden reward functions; and **GAMMA** (Liang et al., 2024), which samples diverse partners from a generative model of partner policies. Environment and baseline details are in Appendices D and E.1.

Training and evaluation. PPC uses an IPPO backbone and designs 25 promises over five categories, with LLM-guided refinement by GPT-5.6 Sol (OpenAI, 2026) for two rounds. All methods are trained for 30M environment steps in Overcooked and 3B steps in Hanabi, except GAMMA, which follows its original setting of 100M environment steps in Overcooked. For the baselines, we use their publicly available implementations and otherwise follow the training configurations specified in the original papers. For PPC, we select the training configuration through hyperparameter search. We evaluate self-play (SP) with copies of the same policy and cross-play (XP) with independently trained partners from different seeds of the same method. For Hanabi, one agent serves as the ego and the others as its partners. All PPC evaluations use the Promisee conditioned only on inferred trust, without action masking. We report mean return and standard deviation over 10 seeds.

Human study protocol. With institutional IRB approval, we conduct real-time human–AI evaluations with 40 participants across the three benchmarks. Participants collaborate with PPC and baseline agents, and we measure episodic team return. Human–AI returns are first averaged within each participant and then aggregated across participants, with 95% bootstrap confidence intervals. We additionally collect human–human trajectories in the same environments to analyze human coordination patterns and their coverage by PPC and baseline agents. Further details on training and human evaluation are provided in Appendix E.

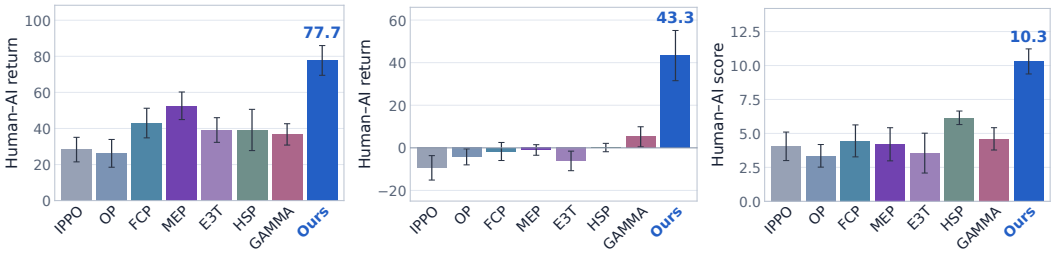
Table 1: Performance comparison on Overcooked-v1, Overcooked-v2, and Hanabi. Results are reported as mean return ± standard deviation. Best values are in **bold**, and our method is shaded.

Environment	Eval.	IPPO	OP	FCP	MEP	E3T	HSP	GAMMA	Ours
Overcooked-v1									
Counter Circuit	SP	158±27	153±21	103±43	87±24	121±31	112±12	66±29	**260±22**
	XP	32±12	13±18	107±17	83±11	38±9	114±8	74±14	**231±29**
Coordination Ring	SP	287±23	244±49	159±29	142±28	181±58	152±7	143±64	**325±9**
	XP	41±11	34±38	166±11	143±11	98±24	154±6	142±23	**311±9**
Cramped Room	SP	253±9	253±11	224±50	203±17	234±6	212±23	181±24	**260±4**
	XP	187±36	194±62	227±19	205±7	165±27	211±23	182±7	**252±7**
Forced Coordination	SP	200±29	198±10	128±37	38±21	176±10	39±12	108±35	**320±12**
	XP	10±11	13±32	140±9	43±10	105±22	39±10	129±11	**317±28**
Overcooked-v2									
Grounded Coord. Simple	SP	138±17	31±36	27±47	9±12	64±59	0±1	18±12	**156±23**
	XP	−60±39	−20±8	14±11	14±6	−18±16	1±1	20±8	**117±14**
Grounded Coord. Ring	SP	162±9	133±61	59±51	5±5	64±73	0±1	62±29	**167±12**
	XP	−18±29	−7±14	56±21	4±2	−13±10	0±1	30±11	**120±58**
Test Time Simple	SP	149±32	21±17	66±46	6±12	50±35	1±1	22±21	**165±18**
	XP	−69±46	−13±11	56±20	7±4	−20±15	1±1	29±11	**140±13**
Test Time Wide	SP	178±30	138±78	135±50	−1±2	85±100	3±5	41±25	**186±9**
	XP	−51±12	−12±11	72±23	0±1	−24±11	2±3	30±11	**167±25**
Demo Cook Simple	SP	142±35	114±26	70±28	13±11	100±68	9±5	21±17	**173±55**
	XP	12±17	51±22	65±14	12±4	−14±16	6±5	23±5	**150±59**
Demo Cook Wide	SP	152±41	159±24	43±54	12±9	129±76	11±5	0±3	**195±12**
	XP	9±6	54±16	40±22	12±2	4±16	11±5	0±1	**149±34**
Hanabi									
2 Players	SP	**23.5±0.5**	23.3±0.6	21.0±1.4	4.3±1.2	16.5±0.9	5.4±0.6	7.1±0.9	23.0±0.2
	XP	7.3±4.4	7.5±5.5	20.3±0.6	4.4±0.6	13.9±0.9	5.5±0.2	7.1±0.4	**22.9±0.3**
3 Players	SP	**23.0±0.2**	22.7±0.7	17.9±1.8	10.9±0.4	15.8±0.6	4.9±0.3	5.1±2.8	22.7±1.0
	XP	8.6±1.5	10.7±3.0	17.3±0.3	9.5±0.2	14.1±0.2	4.9±0.1	4.6±0.1	**21.9±1.2**
4 Players	SP	22.5±0.6	**22.9±0.6**	14.5±2.0	10.5±0.4	15.4±1.0	3.4±0.2	5.1±4.0	22.7±0.5
	XP	4.2±1.9	8.8±3.6	13.2±0.4	9.1±0.2	13.5±0.4	3.4±0.0	4.1±0.3	**21.1±0.4**
5 Players	SP	21.7±0.9	21.6±0.4	15.1±1.9	9.9±0.3	15.1±0.6	3.3±0.4	5.1±1.3	**22.0±0.3**
	XP	6.0±2.1	7.9±1.0	14.4±0.0	9.0±0.2	13.5±0.2	3.3±0.1	5.0±0.1	**20.5±0.6**

5.2 Performance Comparison

Coordination with AI partners. Table 1 reports SP and XP returns across the three benchmarks. PPC achieves the highest SP and XP on all ten Overcooked layouts and the highest XP across all Hanabi team sizes, while maintaining competitive SP performance. In Overcooked, PPC substantially outperforms all baselines in SP, showing that promises induce effective coordination behaviors that improve task performance rather than merely increasing behavioral diversity. Importantly, these strong SP returns are largely preserved in XP, indicating that the learned coordination behaviors transfer to unseen partners. This effect is particularly pronounced under partial observability: in Overcooked-v2, PPC attains XP returns of 117–167 across all six layouts, while no baseline exceeds 72. Similarly, in Hanabi, PPC maintains an SP–XP gap of at most 1.6 points across two to five players, whereas high-SP baselines such as IPPO and OP degrade substantially in XP. Together, these results show that PPC achieves strong task performance while generalizing effectively to unseen AI partners. We additionally provide full learning curves, held-out XP with partners trained by other algorithms, and best-response proximity against biased unseen partners in Appendix F. In both settings, PPC maintains strong performance without any additional joint training or adaptation to these partners, further supporting its generalization under ZSC.

Coordination with human partners. We further evaluate whether PPC generalizes beyond its AI training population to unseen human partners. Figure 4 reports human–AI performance averaged over all layouts in the two Overcooked benchmarks and in 2-player Hanabi. PPC achieves the highest return in all three environments (**77.7**, **43.3**, and **10.3**, respectively), demonstrating consistent generalization from AI training partners to humans across different coordination settings. The advantage is particularly pronounced in partially observable Overcooked-v2, where several baselines achieve near-zero or negative returns, while PPC maintains strong performance despite limited in-



(a) Overcooked-v1 (b) Overcooked-v2 (c) Hanabi, 2 players

Figure 4: Human evaluation. Mean human–AI return with 95% confidence intervals, pooled over layouts (per-layout results in Appendix F.3). All comparisons with baselines yield $p < 0.001$ (Holm-corrected Wilcoxon signed-rank test).

p_1: Right agent adds the left agent's onions to the least-filled accepting pot.

p_2: Left agent stages plates at free handoff counters for cooking pots.

p_3: Left agent stages needed onions at a free handoff counter.

Set	SP	XP	Div_{fix}	Div_{η}
Null (IPPO)	200	10	0.073	
p_1	300	270	0.433	0.570
p_2	200	190	0.150	0.180
p_3	240	220	0.276	0.330
$p_1 + p_2$	220	240	0.446	0.617
$p_1 + p_3$	280	280	0.478	0.714
$p_2 + p_3$	240	240	0.290	0.383
$p_1 + p_2 + p_3$	**320**	**320**	**0.567**	**0.740**

Table 2: Analysis of promise combinations.

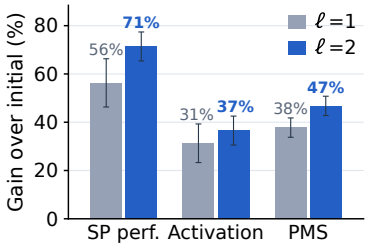


Figure 5: Gains from refinement.

formation about its human partner. These results show that the coordination behaviors induced by promises support effective collaboration with unseen humans.

5.3 Analysis Results

We further analyze how promises improve coordination, whether they induce meaningful coordination behaviors, and how individual components contribute to performance. We conduct all analyses in Overcooked-v1, where the induced coordination behaviors are more clearly distinguished, and report additional analyses on Overcooked-v2 in Appendix G, which show similar trends.

Q1. Why do promises improve coordination with unseen partners? We focus on the `forced_coord` layout, where successful task completion critically depends on cooperation between physically separated agents. Table 2 evaluates all subsets of three representative promises in this layout. Here, Div_{fix} and $\text{Div}_{\boldsymbol{\eta}}$ measure the Jensen–Shannon divergence between trajectory distributions (Lupu et al., 2021) when the enforcement probability η of each selected promise is fixed to 1 or independently randomized, respectively; randomization further diversifies behavior. The results show that individual promises induce distinct, task-useful behaviors and can substantially improve coordination; for example, p_1 increases SP from 200 to 300 while also increasing behavioral diversity. Combining promises further improves both coordination and diversity, with the full set achieving the highest SP, XP, and diversity under both enforcement settings.

Figure 5 further shows gains over the initial promises in SP performance, activation, and PMS across refinement rounds. The gains are substantial at $\ell = 1$, with smaller additional gains at $\ell = 2$, motivating our use of two refinement rounds. In particular, the increased PMS indicates better promise separation, consistent with greater behavioral diversity. Overall, these results show that promises improve coordination by inducing *distinct yet task-useful coordination modes*, while combining and refining these promises further improves both coordination performance and behavioral diversity.

Q2. Do promises capture meaningful human coordination behaviors? We examine how well the behaviors induced by each method cover coordination patterns observed in human play. To assess this, Figure 6 measures overall human behavior coverage, while Figure 7 visualizes human–human and agent cross-play trajectories in a shared UMAP space (McInnes et al., 2018), with different behavior types identified in the space. PPC covers **74.4%** of the human behavior space, more than

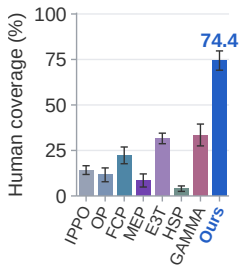


Figure 6: Human behavior coverage.

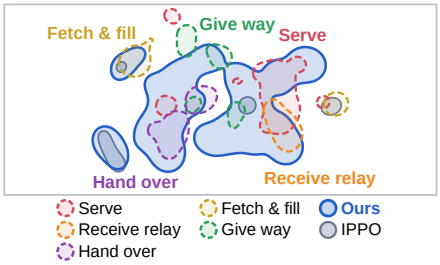


Figure 7: UMAP visualization of human and agent behavior types.

Behavior	Ours	IPPO
Serve	**61.5**	5.7
Receive relay	**88.0**	0.2
Hand over	**81.7**	2.7
Fetch & fill	**54.5**	31.7
Give way	**51.8**	49.4
All	**74.4**	14.2

Table 3: Coverage by behavior type (%).

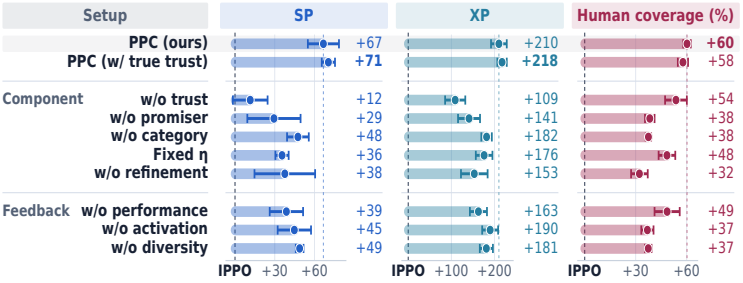


Figure 8: Ablation of PPC components and feedback.

LLM	SP	XP
Qwen3.5-9B	219±10	214±8
Qwen3.5-27B	256±22	245±26
Claude Opus 4.8	276±35	276±26
GPT-4o	274±27	264±25
GPT-5.6 Sol	**291±11**	**277±18**

Table 4: Comparison across different LLMs.

twice that of any baseline, and spans diverse human behavior patterns across common coordination behaviors in the cooking task, such as serving and fetching. Table 3 further quantifies coverage for each behavior type. While IPPO covers only a limited subset of these behaviors, PPC achieves consistently high coverage across diverse behavior types, including those requiring direct coordination between agents. Importantly, PPC achieves this broad coverage without using human behavior data for promise design or agent training. These results show that *promises can capture meaningful human coordination patterns without direct supervision from human behavior*, helping PPC coordinate effectively with unseen human partners in ZSC.

Q3. How do the key components of PPC contribute to its coordination performance? To assess the contribution of PPC's components, Figure 8 compares PPC with true trust and evaluates 'w/o trust,' 'w/o Promiser,' 'w/o promise categories,' 'w/o refinement,' 'w/o enforcement randomization' (fixed η), and variants without each refinement feedback metric. The small gap with true trust indicates that the learned trust estimator approaches oracle performance. Among the components, removing trust or Promisers causes the largest performance drops, while excluding promise categories or refinement and fixing η reduce both performance and human coverage. For refinement, all feedback metrics contribute to performance or coverage, highlighting their complementary roles in producing effective promises. Table 4 further shows consistent performance across LLMs, with only moderate degradation for smaller models. Given the few LLM calls required, stronger models incur limited additional token cost. Overall, PPC substantially benefits from its components and refinement feedback while remaining relatively robust to the choice of LLM.

We further analyze hyperparameter sensitivity in Appendix G.4 and computational and LLM costs in Appendix G.5, showing that PPC is robust to the numbers of categories M and promises K, while introducing only modest overhead. Appendix A discusses limitations and future directions.

6 Conclusion

We presented PPC, a framework that structures partner diversity through explicit promises for meaningful zero-shot coordination. PPC generates and refines executable promises with an LLM, trains diverse Promisers to fulfill them, and enables a Promisee to coordinate through inferred trust. Across diverse ZSC environments, PPC consistently improves coordination with unseen AI and human partners over existing methods. Moreover, its promise-induced behaviors broadly align with human coordination patterns without using human behavior data. Together, these results demonstrate the potential of promises as a common language for coordination with unseen partners.

AI USE STATEMENT

Generative AI tools were used to assist with writing, including grammar, wording, and clarity. A large language model is also part of the proposed method: it writes the state-summary function and the executable promise sets and revises them from training feedback (Section 4.2, Appendix C). All generated code was validated automatically and checked by the authors, and the final promise sets are reproduced in Appendix C.4. The problem formulation, method, experiments, and conclusions are the authors', who take full responsibility for this work.

ETHICS STATEMENT

This work studies zero-shot coordination in cooperative games and includes a human–AI evaluation. The study was approved by the institutional review board of the authors' institution. Participants gave informed consent, could withdraw at any time, and were compensated. They played cooperative games with AI partners and with other participants. No personal or identifying information was collected; trajectories are stored without identifiers and reported in aggregate. The agents are trained in game environments and are not deployed elsewhere. All research followed the ICLR Code of Ethics.

REPRODUCIBILITY STATEMENT

Appendix C describes the promise-generation pipeline, including the prompt templates, the validation stages, and the feedback used for refinement, and Appendix C.5 gives the architecture, optimization settings, training budgets, and seeds for every trained component; the final promise sets are in Appendix C.4. Appendix D specifies the environments, Appendix E describes the baselines, the evaluation protocols, and the human study, and Appendix G.5 reports computational costs. Proofs and the assumptions they require are in Appendix B. Since LLM outputs can vary across calls, we will release the generated state-summary and promise modules together with the implementation and trained checkpoints where the benchmark licenses permit.

REFERENCES

Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. Constrained policy optimization. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 22–31. PMLR, 2017.

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C. Courville, and Marc G. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Saaket Agashe, Yue Fan, Anthony Reyna, and Xin Eric Wang. LLM-Coordination: Evaluating and analyzing multi-agent coordination abilities in large language models. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 8053–8072, 2025.

Mohammed Alshiekh, Roderick Bloem, Rüdiger Ehlers, Bettina Könighofer, Scott Niekum, and Ufuk Topcu. Safe reinforcement learning via shielding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Eitan Altman. *Constrained Markov Decision Processes*. Chapman and Hall/CRC, Boca Raton, FL, 1999. ISBN 9780849303821.

Anthropic. Introducing Claude Opus 4.8. `https://www.anthropic.com/news/claude-opus-4-8`, 2026.

Sangjun Bae, Yisak Park, Sanghyeon Lee, and Seungyul Han. LLM-guided communication for cooperative multi-agent reinforcement learning. In *Proceedings of the 43rd International Conference on Machine Learning*, 2026. URL `https://arxiv.org/abs/2605.18077`.

Nolan Bard, Jakob N. Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H. Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, Iain Dunning, Shibl Mourad, Hugo Larochelle, Marc G. Bellemare, and Michael Bowling. The Hanabi challenge: A new

frontier for AI research. *Artificial Intelligence*, 280:103216, 2020. doi: 10.1016/j.artint.2019.103216.

Craig Boutilier, Alon Cohen, Avinatan Hassidim, Yishay Mansour, Ofer Meshi, Martin Mladenov, and Dale Schuurmans. Planning and learning with stochastic action sets. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence*, pp. 4674–4682, 2018. doi: 10.24963/ijcai.2018/650.

Mark Burgess and Jan A Bergstra. *Promise Theory: Principles and Applications (Volume 1)*. CreateSpace Independent Publishing Platform, 2014.

Micah Carroll, Rohin Shah, Mark K. Ho, Tom Griffiths, Sanjit Seshia, Pieter Abbeel, and Anca Dragan. On the utility of learning about humans for human-AI coordination. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

Yash Chandak, Georgios Theocharous, Blossom Metevier, and Philip S. Thomas. Reinforcement learning when all actions are not always available. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 3381–3388, 2020. doi: 10.1609/aaai.v34i04.5740.

Paresh Chaudhary, Yancheng Liang, Daphne Chen, Simon S. Du, and Natasha Jaques. Improving human-AI coordination through online adversarial training and generative models. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=AeehNfbHqD.

Brandon Cui, Hengyuan Hu, Luis Pineda, and Jakob Foerster. K-level reasoning for zero-shot coordination in Hanabi. In *Advances in Neural Information Processing Systems*, volume 34, pp. 8215–8228, 2021.

Gal Dalal, Krishnamurthy Dvijotham, Matej Večerík, Todd Hester, Cosmin Paduraru, and Yuval Tassa. Safe exploration in continuous action spaces. *arXiv preprint arXiv:1801.08757*, 2018.

Christian Schroeder de Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip H. S. Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the StarCraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

Yuqing Du, Olivia Watkins, Zihan Wang, Cédric Colas, Trevor Darrell, Pieter Abbeel, Abhishek Gupta, and Jacob Andreas. Guiding pretraining in reinforcement learning with large language models. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 8657–8677. PMLR, 2023.

Tobias Gessler, Tin Dizdarevic, Ani Calinescu, Benjamin Ellis, Andrei Lupu, and Jakob Foerster. OvercookedV2: Rethinking Overcooked for zero-shot coordination. In *The Thirteenth International Conference on Learning Representations*, 2025.

Hengyuan Hu and Dorsa Sadigh. Language instructed reinforcement learning for human-AI coordination. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 13584–13598. PMLR, 2023.

Hengyuan Hu, Adam Lerer, Alex Peysakhovich, and Jakob Foerster. "Other-Play" for zero-shot coordination. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 4399–4410. PMLR, 2020.

Hengyuan Hu, Adam Lerer, Brandon Cui, Luis Pineda, Noam Brown, and Jakob Foerster. Off-belief learning. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 4369–4379. PMLR, 2021.

Shengyi Huang and Santiago Ontañón. A closer look at invalid action masking in policy gradient algorithms. *arXiv preprint arXiv:2006.14171*, 2020.

Yechan Hwang, Sangjun Bae, Jeongmo Kim, Sangwoo Bang, and Seungyul Han. MARBO: Relational belief grounding for LLM agents in social deduction games. In *Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing*, 2026.

Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander T. Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. Do as I can, not as I say: Grounding language in robotic affordances. In *Proceedings of the 6th Conference on Robot Learning*, volume 205 of *Proceedings of Machine Learning Research*, pp. 287–318. PMLR, 2023.

Kunal Jha, Wilka Carvalho, Yancheng Liang, Simon Shaolei Du, Max Kleiman-Weiner, and Natasha Jaques. Cross-environment cooperation enables zero-shot multi-agent coordination. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 27198–27220. PMLR, 2025. URL https://proceedings.mlr.press/v267/jha25b.html.

Mingu Kang, Sunwoo Lee, Yonghyeon Jo, and Seungyul Han. Shaping zero-shot coordination via state blocking. *arXiv preprint arXiv:2605.11688*, 2026.

Minae Kwon, Sang Michael Xie, Kalesha Bullard, and Dorsa Sadigh. Reward design with language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=10uNUgI5Kl.

Lihe Li, Lei Yuan, Pengsen Liu, Tao Jiang, and Yang Yu. LLM-assisted semantically diverse teammate generation for efficient multi-agent coordination. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 36743–36764. PMLR, 2025. URL https://proceedings.mlr.press/v267/li25dq.html.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 9493–9500, 2023.

Yancheng Liang, Daphne Chen, Abhishek Gupta, Simon S. Du, and Natasha Jaques. Learning to cooperate with humans using generative agents. In *Advances in Neural Information Processing Systems*, volume 37, pp. 60061–60087, 2024.

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Xingzhou Lou, Jiaxian Guo, Junge Zhang, Jun Wang, Kaiqi Huang, and Yali Du. PECAN: Leveraging policy ensemble for context-aware zero-shot human-AI coordination. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2023.

Andrei Lupu, Brandon Cui, Hengyuan Hu, and Jakob Foerster. Trajectory diversity for zero-shot coordination. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 7204–7213. PMLR, 2021.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *The Twelfth International Conference on Learning Representations*, 2024.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-Refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

Leland McInnes, John Healy, and James Melville. UMAP: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

Darius Muglich, Christian Schroeder de Witt, Elise van der Pol, Shimon Whiteson, and Jakob Foerster. Equivariant networks for zero-shot coordination. In *Advances in Neural Information Processing Systems*, volume 35, pp. 6410–6423, 2022.

Frans A. Oliehoek and Christopher Amato. *A Concise Introduction to Decentralized POMDPs*. Springer, 2016.

OpenAI. GPT-4o system card. *arXiv preprint arXiv:2410.21276*, 2024.

OpenAI. GPT-5.6: Frontier intelligence that scales with your ambition. `https://openai.com/index/gpt-5-6/`, 2026.

Tu-Hoa Pham, Giovanni De Magistris, and Ryuki Tachibana. OptLayer – practical constrained optimization for deep reinforcement learning in the real world. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6236–6243, 2018. doi: 10.1109/ICRA.2018.8460547.

Qwen Team. Qwen3.5: Towards native multimodal agents. `https://qwen.ai/blog?id=qwen3.5`, 2026.

Arrasy Rahman, Elliot Fosong, Ignacio Carlucho, and Stefano V. Albrecht. Generating teammates for training robust ad hoc teamwork agents via best-response diversity. *Transactions on Machine Learning Research*, 2023.

Alexander Rutherford, Benjamin Ellis, Matteo Gallici, Jonathan Cook, Andrei Lupu, Gardar Ingvarsson, Timon Willi, Ravi Hammond, Akbir Khan, Christian Schroeder de Witt, Alexandra Souly, Saptarashmi Bandyopadhyay, Mikayel Samvelyan, Minqi Jiang, Robert Tjarko Lange, Shimon Whiteson, Bruno Lacerda, Nick Hawes, Tim Rocktäschel, Chris Lu, and Jakob Foerster. JaxMARL: Multi-agent RL environments and algorithms in JAX. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, volume 37, 2024.

Bidipta Sarkar, Andy Shih, and Dorsa Sadigh. Diverse conventions for human-AI collaboration. In *Advances in Neural Information Processing Systems*, volume 36, pp. 23115–23139, 2023.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

DJ Strouse, Kevin R. McKee, Matt Botvinick, Edward Hughes, and Richard Everett. Collaborating with humans without human data. In *Advances in Neural Information Processing Systems*, volume 34, 2021.

Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. Reward constrained policy optimization. In *International Conference on Learning Representations*, 2019.

Xihuai Wang, Shao Zhang, Wenhao Zhang, Wentao Dong, Jingxiao Chen, Ying Wen, and Weinan Zhang. ZSC-Eval: An evaluation toolkit and benchmark for multi-agent zero-shot coordination. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2024.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V. Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24824–24837, 2022.

Tianbao Xie, Siheng Zhao, Chen Henry Wu, Yitao Liu, Qian Luo, Victor Zhong, Yanchao Yang, and Tao Yu. Text2Reward: Reward shaping with language models for reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=tUM39YTRxH`.

Xue Yan, Jiaxian Guo, Xingzhou Lou, Jun Wang, Haifeng Zhang, and Yali Du. An efficient end-to-end training approach for zero-shot human-AI coordination. In *Advances in Neural Information Processing Systems*, volume 36, pp. 2636–2658, 2023. URL `https://proceedings.neurips.cc/paper_files/paper/2023/hash/07a363fd2263091c2063998e0034999c-Abstract-Conference.html`.

Chao Yu, Jiaxuan Gao, Weilin Liu, Botian Xu, Hao Tang, Jiaqi Yang, Yu Wang, and Yi Wu. Learning zero-shot cooperation with humans, assuming humans are biased. In *The Eleventh International Conference on Learning Representations*, 2023.

Tom Zahavy, Matan Haroush, Nadav Merlis, Daniel J. Mankowitz, and Shie Mannor. Learn what not to learn: Action elimination with deep reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 31, 2018.

Ceyao Zhang, Kaijie Yang, Siyi Hu, Zihao Wang, Guanghe Li, Yihang Sun, Cheng Zhang, Zhaowei Zhang, Anji Liu, Song-Chun Zhu, et al. ProAgent: Building proactive cooperative agents with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17591–17599, 2024.

Rui Zhao, Jinming Song, Yufeng Yuan, Haifeng Hu, Yang Gao, Yi Wu, Zhongqian Sun, and Wei Yang. Maximum entropy population-based training for zero-shot human-AI coordination. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 6145–6153, 2023.

APPENDIX CONTENTS

A. Limitations and Future Directions

B. Theoretical Analysis

B.1. Proof of Proposition 1

B.2. Proof of Proposition 2

C. Implementation Details

C.1. Pipeline and Prompts

C.2. Executable Promises and Validation

C.3. Refinement Feedback and Labels

C.4. Final Promise Catalogs

C.5. Training Details

D. Environment Details

D.1. Overcooked-v1

D.2. Overcooked-v2

D.3. Hanabi

E. Experimental Details

E.1. Baselines

E.2. Evaluation Protocol

E.3. Interface for Human Evaluation

F. Additional Experiments

F.1. Learning Curves

F.2. Coordination with Held-Out Partners

F.3. Per-Layout Human Evaluation

G. Additional Analyses

G.1. Promise Composition

G.2. Human Behavior Coverage

G.3. Component Ablations

G.4. Hyperparameter Sensitivity

G.5. Computational Cost

A Limitations and Future Directions

Despite its effectiveness in structuring partner diversity for zero-shot coordination, PPC has several limitations and corresponding directions for future work. First, PPC designs promises from task descriptions alone, so their quality depends on the generation model, and human trajectories are used only for evaluation. A natural direction is to ground promise generation in human behavior, for example by mining recurring coordination modes from human–human data, and to fine-tune open LLMs for promise generation using the activation, performance, and separation feedback of refinement as a training signal. Second, promises are state-conditioned action constraints in a promise set, which suits recurrent, locally identifiable decisions but less naturally captures conventions that rely on explicit communication. Since each promise carries a natural-language body, communicating promises explicitly to partners and extending PPC to more open-ended multi-agent settings would help assess how broadly promises can serve as a common language for coordination.

B Theoretical Analysis

B.1 Proof of Proposition 1

Proposition 1 (Performance Gap under Promiser Enforcement)**.** *Consider a promise set $\mathcal{P}$ and define its compliance value gap as*

$$\Delta_{\mathcal{P}}(s) := V^*(s) - V^*_{\mathcal{P}}(s) \geq 0, \tag{8}$$

where V^* and $V^*_{\mathcal{P}}$ are the optimal value functions without and with all active constraints induced by $\mathcal{P}$, respectively. If the promises in $\mathcal{P}$ are independently enforced with probability $\boldsymbol{\eta} \in [0,1]^K$, then

$$J^* - J^*_{\mathcal{P},\boldsymbol{\eta}} \leq \frac{1 - \prod_{k=1}^K (1-\eta_k)}{1 - \gamma \prod_{k=1}^K (1-\eta_k)} \|\Delta_{\mathcal{P}}\|_\infty. \tag{9}$$

where J^* and $J^*_{\mathcal{P},\boldsymbol{\eta}}$ denote the unconstrained and promise-enforced optimal returns, respectively.

Proof. Let $\mathcal{T}_{\boldsymbol{\eta}}$ denote the Bellman optimality operator under enforcement probabilities $\boldsymbol{\eta}$, with $\mathcal{T}_{\mathbf{0}}$ and $\mathcal{T}_{\mathbf{1}}$ corresponding to no enforcement and full enforcement, respectively. These operators are monotone γ-contractions in the supremum norm, with unique fixed points satisfying

$$V^* = \mathcal{T}_{\mathbf{0}} V^*, \qquad V^*_{\mathcal{P}} = \mathcal{T}_{\mathbf{1}} V^*_{\mathcal{P}}, \qquad V^*_{\mathcal{P},\boldsymbol{\eta}} = \mathcal{T}_{\boldsymbol{\eta}} V^*_{\mathcal{P},\boldsymbol{\eta}}.$$

Because enforcement only restricts the available actions and the active constraints are jointly feasible, $\mathcal{T}_{\mathbf{1}} v \leq \mathcal{T}_{\boldsymbol{\eta}} v \leq \mathcal{T}_{\mathbf{0}} v$ for every bounded v. Monotone iteration therefore yields $V^*_{\mathcal{P}} \leq V^*_{\mathcal{P},\boldsymbol{\eta}} \leq V^*$. Since actions are selected after observing the available set, we also have

$$\mathcal{T}_{\boldsymbol{\eta}} v \geq \left[\prod_{k=1}^K (1-\eta_k)\right] \mathcal{T}_{\mathbf{0}} v + \left[1 - \prod_{k=1}^K (1-\eta_k)\right] \mathcal{T}_{\mathbf{1}} v.$$

Indeed, by independence, the probability that no active promise is enforced is at least $\prod_{k=1}^K (1-\eta_k)$. On this event, the Bellman update is $\mathcal{T}_{\mathbf{0}} v$; every other realization has an action set containing the fully compliant set. Together with $\mathcal{T}_{\mathbf{0}} v \geq \mathcal{T}_{\mathbf{1}} v$, this proves the operator bound.

Set $\delta_{\boldsymbol{\eta}} := \|V^* - V^*_{\mathcal{P},\boldsymbol{\eta}}\|_\infty$. For every state s,

$$\begin{aligned}
V^*(s) - V^*_{\mathcal{P},\boldsymbol{\eta}}(s) &= V^*(s) - (\mathcal{T}_{\boldsymbol{\eta}} V^*_{\mathcal{P},\boldsymbol{\eta}})(s) \\
&\underset{(1)}{\leq} \left[\prod_{k=1}^K (1-\eta_k)\right] \left[V^*(s) - (\mathcal{T}_{\mathbf{0}} V^*_{\mathcal{P},\boldsymbol{\eta}})(s)\right] \\
&\quad + \left[1 - \prod_{k=1}^K (1-\eta_k)\right] \left[V^*(s) - (\mathcal{T}_{\mathbf{1}} V^*_{\mathcal{P},\boldsymbol{\eta}})(s)\right] \\
&\underset{(2)}{\leq} \left[\prod_{k=1}^K (1-\eta_k)\right] \left[V^*(s) - (\mathcal{T}_{\mathbf{0}} V^*_{\mathcal{P},\boldsymbol{\eta}})(s)\right] + \left[1 - \prod_{k=1}^K (1-\eta_k)\right] \Delta_{\mathcal{P}}(s) \\
&\underset{(3)}{\leq} \gamma \left[\prod_{k=1}^K (1-\eta_k)\right] \delta_{\boldsymbol{\eta}} + \left[1 - \prod_{k=1}^K (1-\eta_k)\right] \Delta_{\mathcal{P}}(s).
\end{aligned}$$

Inequality (1) uses the preceding operator bound. For (2), monotonicity gives $\mathcal{T}_{\mathbf{1}} V^*_{\mathcal{P},\boldsymbol{\eta}} \geq \mathcal{T}_{\mathbf{1}} V^*_{\mathcal{P}} = V^*_{\mathcal{P}}$. Inequality (3) uses $V^* = \mathcal{T}_{\mathbf{0}} V^*$ and the γ-contraction property.

Since $V^* - V^*_{\mathcal{P},\boldsymbol{\eta}} \geq 0$, taking the supremum over states in (3) yields

$$\delta_{\boldsymbol{\eta}} \leq \gamma \left[\prod_{k=1}^{K}(1-\eta_k)\right] \delta_{\boldsymbol{\eta}} + \left[1 - \prod_{k=1}^{K}(1-\eta_k)\right] \|\Delta_{\mathcal{P}}\|_\infty.$$

Finally, under the common initial-state distribution d_0,

$$\begin{aligned} J^* - J^*_{\mathcal{P},\boldsymbol{\eta}} &= \mathbb{E}_{s_0 \sim d_0}\left[V^*(s_0) - V^*_{\mathcal{P},\boldsymbol{\eta}}(s_0)\right] \\ &\leq \delta_{\boldsymbol{\eta}} \\ &\leq \frac{1 - \prod_{k=1}^{K}(1-\eta_k)}{1 - \gamma \prod_{k=1}^{K}(1-\eta_k)} \|\Delta_{\mathcal{P}}\|_\infty. \end{aligned}$$

□

B.2 Proof of Proposition 2

Proposition 2 (Performance Gap under Promisee Trust Estimation)**.** *Assume the Promisee's action distribution is uniformly L-Lipschitz in trust from ℓ_∞ to total variation, with $\sup_s \|\boldsymbol{\rho}(s) - \boldsymbol{\eta}\|_\infty \le \varepsilon_{\boldsymbol{\rho}}$. Then*

$$|J_{\mathcal{P},\boldsymbol{\rho}} - J_{\mathcal{P},\boldsymbol{\eta}}| \le \frac{r_b L \varepsilon_{\boldsymbol{\rho}}}{(1-\gamma)^2}, \tag{10}$$

where $J_{\mathcal{P},\boldsymbol{\rho}}$ and $J_{\mathcal{P},\boldsymbol{\eta}}$ denote the returns under estimated trust $\boldsymbol{\rho}$ and true enforcement probabilities $\boldsymbol{\eta}$, respectively.

Proof. Consider the MDP induced by the fixed Promiser, and write $\pi := \pi_{\mathcal{P},\boldsymbol{\rho}}$ and $\pi^\circ := \pi_{\mathcal{P},\boldsymbol{\eta}}$ for the Promisee policies under estimated trust and true enforcement probabilities, respectively. By the uniform Lipschitz assumption, for every state s,

$$D_{\mathrm{TV}}\big(\pi(\cdot \mid s), \pi^\circ(\cdot \mid s)\big) \le L\|\boldsymbol{\rho}(s) - \boldsymbol{\eta}\|_\infty \le L\varepsilon_{\boldsymbol{\rho}}.$$

For trajectories initialized from the common distribution d_0, define the normalized discounted state distribution

$$d^\pi(s) := (1-\gamma)\sum_{t=0}^{\infty} \gamma^t \Pr_\pi(s_t = s).$$

Let V^{π° and Q^{π° denote the value and action-value functions under π°. For each state s, define the midpoint of the action-value range by

$$\zeta_s := \frac{1}{2}\left(\sup_a Q^{\pi^\circ}(s,a) + \inf_a Q^{\pi^\circ}(s,a)\right).$$

Because the one-step reward range is at most r_b, for every state–action pair (s,a),

$$\big|Q^{\pi^\circ}(s,a) - \zeta_s\big| \le \frac{1}{2}\left(\sup_{a'} Q^{\pi^\circ}(s,a') - \inf_{a'} Q^{\pi^\circ}(s,a')\right) \le \frac{r_b}{2(1-\gamma)}.$$

We first express the return difference as

$$\begin{aligned}
J_{\mathcal{P},\boldsymbol{\rho}} - J_{\mathcal{P},\boldsymbol{\eta}} &\underset{(1)}{=} \mathbb{E}_\pi\left[\sum_{t=0}^{\infty}\gamma^t\big(r(s_t,a_t) + \gamma V^{\pi^\circ}(s_{t+1}) - V^{\pi^\circ}(s_t)\big)\right] \\
&\underset{(2)}{=} \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^\pi}\left[\mathbb{E}_{a\sim\pi(\cdot|s)}[Q^{\pi^\circ}(s,a)] - V^{\pi^\circ}(s)\right] \\
&\underset{(3)}{=} \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^\pi}\left[\mathbb{E}_{a\sim\pi(\cdot|s)}[Q^{\pi^\circ}(s,a)] - \mathbb{E}_{a\sim\pi^\circ(\cdot|s)}[Q^{\pi^\circ}(s,a)]\right].
\end{aligned}$$

For (1), the discounted value terms telescope to $-V^{\pi^\circ}(s_0)$, with the terminal term vanishing because V^{π° is bounded and $\gamma < 1$. The common initial distribution gives $\mathbb{E}_{s_0\sim d_0}[V^{\pi^\circ}(s_0)] = J_{\mathcal{P},\boldsymbol{\eta}}$. Equality (2) uses the definition of d^π and $Q^{\pi^\circ}(s,a) = r(s,a) + \gamma\,\mathbb{E}_{s'\sim T(\cdot|s,a)}[V^{\pi^\circ}(s')]$, and (3) uses $V^{\pi^\circ}(s) = \mathbb{E}_{a\sim\pi^\circ(\cdot|s)}[Q^{\pi^\circ}(s,a)]$.

Next, for every state s,

$$\begin{aligned}
&\Big|\mathbb{E}_{a\sim\pi(\cdot|s)}[Q^{\pi^\circ}(s,a)] - \mathbb{E}_{a\sim\pi^\circ(\cdot|s)}[Q^{\pi^\circ}(s,a)]\Big| \\
&\underset{(4)}{=} \left|\sum_a \big(\pi(a \mid s) - \pi^\circ(a \mid s)\big)\big(Q^{\pi^\circ}(s,a) - \zeta_s\big)\right| \\
&\underset{(5)}{\le} \frac{r_b}{1-\gamma} D_{\mathrm{TV}}\big(\pi(\cdot \mid s), \pi^\circ(\cdot \mid s)\big) \\
&\underset{(6)}{\le} \frac{r_b L \varepsilon_{\boldsymbol{\rho}}}{1-\gamma}.
\end{aligned}$$

Equality (4) holds because both action distributions sum to one, so subtracting ζ_s leaves their expectation difference unchanged. Inequality (5) follows from the triangle inequality, the bound on $|Q^{\pi^\circ}(s,a) - \zeta_s|$, and $D_{\mathrm{TV}}(q,q') = \frac{1}{2}\sum_a |q(a) - q'(a)|$. Inequality (6) uses the Lipschitz bound.

Finally, taking absolute values in (3), applying the triangle inequality and (6), and using the normalization of d^π gives

$$\begin{aligned} |J_{\mathcal{P},\boldsymbol{\rho}} - J_{\mathcal{P},\boldsymbol{\eta}}| &\le \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^\pi}\left[\frac{r_b L \varepsilon_{\boldsymbol{\rho}}}{1-\gamma}\right] \\ &= \frac{r_b L \varepsilon_{\boldsymbol{\rho}}}{(1-\gamma)^2}. \end{aligned}$$

□

On the Lipschitz assumption. In the Markov setting of Proposition 2, the assumption holds for the stochastic softmax policy $\pi(\cdot \mid s, \boldsymbol{\rho}) = \mathrm{softmax}(f(s,\boldsymbol{\rho}))$ trained by PPC. Since the softmax Jacobian has $\ell_\infty \to \ell_1$ norm at most one, $D_{\mathrm{TV}}\big(\pi(\cdot \mid s,\boldsymbol{\rho}), \pi(\cdot \mid s,\boldsymbol{\rho}')\big) \le \frac{1}{2}\|f(s,\boldsymbol{\rho}) - f(s,\boldsymbol{\rho}')\|_\infty$. As $\boldsymbol{\rho}$ enters after the state encoder and passes only through Lipschitz layers, $f(s,\cdot)$ is L_f-Lipschitz uniformly over s, and the assumption holds with $L = L_f/2$.

C Implementation Details

This appendix details how PPC is implemented. Appendix C.1 describes how the LLM designs and refines promises (Section 4.2) and the prompts it receives, Appendix C.2 how a generated promise is executed and validated, Appendix C.3 the feedback and labels that drive each refinement round, and Appendix C.4 the resulting promise sets. Appendix C.5 then describes how the Promisers, the trust estimator, and the Promisee are trained. Algorithm 1 summarizes the overall procedure of PPC.

Algorithm 1 Promise-based Partner Coordination (PPC)

Require: Dec-POMDP $\mathcal{M}$, task specification I_{task}, prompts x_ψ and x_p, LLM p_θ, numbers of categories M and promises K, number of refinement rounds $\ell_{\max}$

// Promise design

$\psi \sim p_\theta(\cdot \mid x_\psi, I_{\text{task}})$ ▷ state summary

$y_0 \sim p_\theta(\cdot \mid x_p, I_{\text{task}}, \psi)$ ▷ $\mathcal{P} = \bigcup_{m=1}^{M} \mathcal{P}_m$, $|\mathcal{P}_m| = K/M$

Initialize $\Pi^{\text{Pr}} = \{\boldsymbol{\pi}_j^{\text{Pr}}\}_{j=1}^{K/M}$ with $\mathcal{P}_j = \{p_{1,j}, \ldots, p_{M,j}\}$, $p_{m,j} \in \mathcal{P}_m$

// Promiser learning with promise refinement

for $\ell = 0, \ldots, \ell_{\max}$ **do**

  Train each $\boldsymbol{\pi}_j^{\text{Pr}}$ via RL self-play in $\mathcal{M}_{\mathcal{P}}$ with $\boldsymbol{\eta}_j$ resampled each episode

  if $\ell < \ell_{\max}$ **then**

    Form f_ℓ from activation, performance, and PMS on self-play rollouts

    Label REPAIR from activation and value gap, and IMPROVE from PMS

    $y_{\ell+1} \sim p_\theta(\cdot \mid x_p, I_{\text{task}}, \psi, y_\ell, f_\ell)$ ▷ θ and policy weights kept

  else

    Concurrently pretrain trust estimator $\boldsymbol{\rho}_\phi$ on Promiser self-play with $\boldsymbol{\eta}_j$ as targets

  end if

end for

// Promisee learning

Freeze Π^{Pr}

repeat

  Sample a partner $\boldsymbol{\pi}_j^{\text{Pr},-e} \sim \Pi^{\text{Pr}}$, or a Promisee copy for Promisee–Promisee self-play

  Roll out $\pi^{\text{Pe},e}(\cdot \mid \tau_t^e, \boldsymbol{\rho}_\phi(\tau_t^e))$ without promise-based action constraints

  Update $\pi^{\text{Pe},e}$ via RL, and fine-tune trust estimator $\boldsymbol{\rho}_\phi$ on Promisee–Promiser episodes

until the training budget is exhausted

return Promisee $\pi^{\text{Pe},e}$ and trust estimator $\boldsymbol{\rho}_\phi$

C.1 Pipeline and Prompts

For each condition, the LLM is called offline four times: once for the state summary ψ, once for the initial promise set y_0, and once per refinement round for y_ℓ ($\ell = 1, 2$), with retries when a response fails validation (Appendix C.2). It is never called during training or evaluation. Unless stated otherwise (Section 5.3), all calls use GPT-5.6 Sol (OpenAI, 2026) (Codex command-line client, medium reasoning effort, JSON-schema output, default sampling), using 6–9 $\times$ 10^4 tokens per condition in total (Appendix G.5). For the LLM comparison in Table 4, we replace it with Qwen3.5 (Qwen Team, 2026) at two model sizes (9B and 27B), GPT-4o (OpenAI, 2024), or Claude Opus 4.8 (Anthropic, 2026) under the same prompts and validation pipeline.

Variable	Symbol	Content
{task_description}	I_{task}	Goal of the cooperative task; no promise or target behavior is given.
{environment_context}		State, observation, and action definitions, transition rules, and what each role observes.
{implementation_spec}		Function signatures, tensor shapes, allowed dependencies, and the JSON output schema.
{num_categories}, {num_variants}	M, K/M	Number of categories and of variants per category.
{state_summary}	ψ	Summary code from the first call, with its field definitions.
{current_promises}	y_ℓ	Current promise set: IDs, specifications, and code.
{evaluation_feedback}	f_ℓ	Activation, performance, and PMS of each promise, with its label (Appendix C.3).
{refinement_history}	$(y_{\ell'}, f_{\ell'})_{\ell' < \ell}$	Earlier promise sets, their edits, and their feedback; empty in the first round.

Table 5: Prompt variables and their notation in Section 4.2. Burgundy: fixed per condition; gray: produced during the run; black: shared by all conditions.

C.2 Executable Promises and Validation

Each promise is returned as a body b (`Promiser ...; Promisee ...`) and code. A single generated function returns, for every promise and agent, the activation flag ($s \in \mathcal{S}^p$) and the action mask $\mathcal{A}^p_{\text{avail}}(s)$; at inactive states the mask is the legal action set. At runtime, the masks of the promises enforced at that step are intersected, falling back to the legal set if the intersection is empty, and only the Promiser row is applied (Eq. 2). The same enforcement extends to continuous action spaces, where the available set is a region of $\mathcal{A}$ and masking is realized by projecting or bounding actions (Dalal et al., 2018; Pham et al., 2018). A response is accepted only if it passes, in order:

1. JSON parsing and identifier checks, with each ID mapped to the previous round;
2. a static scan rejecting file, process, network, and dynamic-execution calls;
3. import and JIT compilation against a real environment reset, with the required tensor shapes;
4. for refinement calls, a change to every promise labeled Repair.

On failure, the model receives the traceback and returns a full replacement.

C.3 Refinement Feedback and Labels

Promiser training runs in three segments of 10^7 (Overcooked) or 10^9 (Hanabi) steps per Promiser without resetting parameters, and the two refinement rounds take place at the segment boundaries. At each boundary, the policies are frozen and every Promiser j is evaluated on complete episodes for 102 400 environment steps per measurement, with fixed seeds reused across rounds. We denote by $\hat{J}_j$ its return without enforcement and, for each $p \in \mathcal{P}_j$, by $\hat{J}_{j,p}$ its return with only p enforced. As in Section 4.2, $\mathcal{D}$ is the rollout buffer of the Promiser that holds p and $\mathcal{D}_p \subseteq \mathcal{D}$ its active state–action pairs, collected from these evaluation rollouts. The feedback f_ℓ reports three metrics per promise.

State Summary Prompt (x_ψ)

Build a compact state summary for promise execution and evaluation.

- Extract named features for agents, legal actions, resources, public history, and relevant relations.
- Preserve decision-relevant identities, and distinguish global from role-observable information.
- Use only state-derived facts and static metadata.

Output A deterministic summary function $\psi(s)$, required helpers, and field descriptions, in the specified format.

Task: {task_description} Environment: {environment_context}
Implementation: {implementation_spec}

Promise Generation Prompt (x_p)

Generate promises that induce diverse coordination behaviors and support cooperation with unseen partners, including humans.

- Define {num_categories} categories, each a recurrent coordination decision, with {num_variants} behaviorally distinct variants for shared situations.
- For each promise, specify the roles, body, activation, and action sets using the fixed summary and role-observable information.
- Keep masks legal, nonempty, and compatible across categories; inactive promises add no restrictions.
- Each promise combination contains one variant per category, and each variant is used in exactly one combination.
- Ensure sufficient activation and task viability while preserving Promisee autonomy.

Output Complete promise specifications and executable code, in the specified format.

Task: {task_description} Environment: {environment_context}
State summary: {state_summary} Implementation: {implementation_spec}

Promise Refinement Prompt (each round)

Improve promise viability and behavioral separation using self-play feedback. Review the refinement history, then follow the label of each promise:

- REPAIR: fix insufficient activation or a large estimated compliance gap.
- IMPROVE: separate overlapping behaviors using within-category PMS; higher PMS indicates greater separation.

Preserve task return, sufficient activation, role observability, and legal, nonempty action sets compatible across categories. Keep the summary, IDs, and unaffected promises unchanged, and make only necessary edits, since policy training continues without resetting.

Output Complete revised specifications and code with a brief reason for each change, in the specified format.

Task: {task_description} Environment: {environment_context}
State summary: {state_summary} Current promises: {current_promises}
Feedback: {evaluation_feedback} History: {refinement_history}
Implementation: {implementation_spec}

Figure 9: Prompts for promise generation and refinement, shared by all environments. Braced variables are listed in Table 5.

Activation. The fraction of steps at which p applies,

$$a_p = \frac{|\mathcal{D}_p|}{|\mathcal{D}|}. \tag{11}$$

Performance. The empirical return loss from enforcing p alone, normalized by the unenforced return,

$$\widehat{\Delta}_p = \frac{\hat{J}_j - \hat{J}_{j,p}}{\hat{J}_j + \varepsilon}, \qquad p \in \mathcal{P}_j, \tag{12}$$

where $\varepsilon > 0$ is a small constant for numerical stability. It is the empirical counterpart of $J^* - J^*_{\mathcal{P},\boldsymbol{\eta}}$ in Eq. 5, which Proposition 1 bounds by the compliance value gap $\|\Delta_{\mathcal{P}}\|_\infty$; a small $\widehat{\Delta}_p$ thus indicates a promise that preserves task return.

PMS. For two variants $p, q \in \mathcal{P}_m$, Eq. 7 evaluated on the buffers above,

$$\mathrm{PMS}(p, q) = \max\Big\{0,\ \tfrac{1}{2}\big(c_{p\to q} + c_{q\to p}\big)\Big\}, \tag{13}$$

where $c_{p\to q}$ is the mean over $(s, \boldsymbol{a}) \in \mathcal{D}_p$ of $\mathbf{1}\{\boldsymbol{a} \in \mathcal{A}^p(s)\} - \mathbf{1}\{\boldsymbol{a} \in \mathcal{A}^q(s)\}$. It is computed for promises with a nonempty $\mathcal{D}_p$.

Before the LLM sees the feedback, a fixed rule labels every promise (Table 6). Promises with insufficient activation or a large compliance gap are labeled REPAIR and must be changed (check 4 in Appendix C.2). All other promises are labeled IMPROVE, and the LLM uses their PMS to decide which variants to separate further.

Metric	Eq.	Label	Condition
Activation	11	REPAIR	$a_p \le 10^{-3}$
Performance	12	REPAIR	$\widehat{\Delta}_p > 0.20$
PMS	13	IMPROVE	Neither REPAIR condition holds; no threshold

Table 6: Labeling rule for refinement feedback. A promise meeting either REPAIR condition must be changed; the others are revised at the LLM's discretion.

C.4 Final Promise Sets

Each condition has its own promise set of $M = 5$ categories with $K/M = 5$ variants each, giving $K = 25$ promises and five promise combinations, one per Promiser. Table 7 lists the categories of all promise sets. Figure 10 shows one category as generated, and Table 8 summarizes one category for each convention in Figure 1 and for signaling in Hanabi, chosen for coverage rather than performance. Variants of a category share one activation condition and differ in a single choice, which makes them behaviorally distinct and their trust inferable. All promise sets and their code will be released. Our results can thus be reproduced without further language-model calls.

Condition	Categories
Overcooked-v1	
Counter circuit	close-quarters joint motion; productive service ownership; pickup ownership; pot routing; counter staging zones
Coordination ring	pot routing; supply assignment; interaction ownership; close-quarters passing; counter staging
Cramped room	ingredient-source sharing; pot-access precedence; plate-runner allocation; ready-soup service lead; close-quarters traffic flow
Forced coordination	work-flow initiative; onion pot allocation; plate pot allocation; empty-hand procurement split; soup delivery and handoff
Overcooked-v2	
Grounded simple	useful-ingredient pickup reservation; ingredient-to-pot supply reservation; plate and cooked-dish service reservation; counter transfer reservation; near-agent passing convention
Grounded ring	nearby-agent traffic protocol; ingredient-pile access allocation; pot-fill turn allocation; dish-pipeline ownership; counter handoff territory
Test time simple	recipe-signal lexicon; post-delivery ingredient hypothesis; counter-lane convention; service-station cadence; loaded-travel formation
Test time wide	open-floor travel cadence; relative-position passing etiquette; empty-hand supply-bank allocation; pot approach allocation; effective-interaction turn allocation
Demo cook simple	pot-loading access cadence; loose-resource acquisition cadence; counter handoff cadence; cooked-dish extraction cadence; cross-pot acknowledgment gesture
Demo cook wide	ingredient sourcing ownership; pot-loading turn allocation; plate and cooked-dish ownership; delivery ownership; close-quarters traffic convention
Hanabi	
2 players	certain-play ordering; certain-discard ordering; playable-card hinting; critical-card protection hinting; quiet-state discard ordering
3 players	playable relay priority; discard relay priority; critical-aware disposal; playable clue encoding; low-token tempo
4 players	playable recipient routing; playable card priority; playable clue encoding; safe-discard priority; endgame play ordering
5 players	playable-clue dialect; critical-card protection dialect; safe-discard permission dialect; playable-clue recipient routing; hint-value response channel

Table 7: Final promise sets, listing categories 0–4 of each condition.

Generated Promises: OV2 Grounded Ring, Category 0

Nearby-agent traffic protocol. All variants activate within Manhattan distance 3.

0	**Promiser yields**	Promiser stays while Promisee moves directly away if possible; Promisee stays only when that move is blocked.
1	**Promisee yields**	Promisee stays while Promiser moves directly away if possible; Promiser stays only when that move is blocked.
2	**Clockwise peel**	Both take the locally clockwise perpendicular move relative to the other, falling back to stay if blocked.
3	**Counterclockwise peel**	Both take the locally counterclockwise perpendicular move relative to the other, falling back to stay if blocked.
4	**Mutual retreat**	Both move directly away from the other if possible and otherwise stay.

Code activation $\mathcal{S}^p$ and constraint $\mathcal{A}^p_{\text{avail}}(s)$ for all five slots

```
for slot in range(5):
    for role in range(2):              # 0: Promiser, 1: Promisee
        dx, dy = delta[role, 1 - role]
        away = feasible_or_stay(role, vector_action(-dx, -dy))
        cw   = feasible_or_stay(role, vector_action(-dy, dx))
        ccw  = feasible_or_stay(role, vector_action(dy, -dx))
        chosen = [STAY if role == 0 else away,    # 0: Pr yields
                  away if role == 0 else STAY,    # 1: Pe yields
                  cw, ccw, away][slot]            # 2, 3, 4
        dist = manhattan[role, 1 - role]
        trigger = (dist > 0) & (dist <= 3)        # S^p
        masks[slot, role] = where(trigger, one_hot(chosen), ones)
```

Figure 10: One category as generated: the five promise bodies, with their shared activation condition stated once, and the code that implements them, condensed only by merging branches. Only the Promiser row of each mask is applied.

Convention	Category	Promises
Yielding Overcooked-v2 Grounded ring	Nearby-agent traffic	Near the partner, one agent yields, both sidestep, or both retreat.
Handoff Overcooked-v1 Forced coordination	Soup delivery and handoff	With a finished dish, one agent stages it on a counter and the other delivers it.
Serving Overcooked-v2 Test time simple	Station cadence	At a station, both agents interact together or take turns in fixed step cycles.
Signaling Hanabi 5 players	Clue routing	With a playable clue, the Promiser clues a designated player, and the Promisee plays a card known to be playable.

Table 8: Representative categories for the conventions of Figure 1 (yielding, handoff, serving) and for signaling in Hanabi. The five variants of each category differ in who acts, where, or when; Figure 10 lists all variants of the first one.

C.5 Training Details

This subsection details how PPC is trained. We describe the enforcement schedule of the Promisers, the architecture and training objective of the trust estimator, the training of the Promisee, and the configuration shared with the baselines. All conditions follow the same procedure, with optimization settings in Table 10 and baseline settings in Table 9.

Promisers. Each category's K/M promises are distributed one per Promiser, so every promise is held by exactly one Promiser. With probability 0.1, an episode sets $\boldsymbol{\eta}_j = \mathbf{0}$, so that the Promiser also learns the unenforced behavior whose return $\hat{J}_j$ enters Eq. 12. Otherwise, at every step, each active promise p_k is enforced by an independent Bernoulli draw with probability $\eta_{j,k}$, as assumed in Proposition 1, and the masks of the enforced promises are composed as in Appendix C.2.

Trust estimator. $\boldsymbol{\rho}_\phi$ consists of an observation encoder, a 128-unit GRU reset at every episode, and a linear head with sigmoid outputs $\boldsymbol{\rho}_\phi(\tau_t^e) \in [0,1]^K$. Its input is limited to the ego's own observations and actions; the partner's identity, promise combination, constraints, and $\boldsymbol{\eta}$ are never given. It is trained with the partner's enforcement vector $\boldsymbol{\eta}_j$ as the target, which is zero outside the partner's promise combination $\mathcal{P}_j$:

$$\mathcal{L}(\phi) = -\,\mathbb{E}_{(\tau^e,\boldsymbol{\eta}_j)}\left[\frac{1}{t_{\max}}\sum_{t=1}^{t_{\max}}\Big(\boldsymbol{\eta}_j^\top \log \boldsymbol{\rho}_\phi(\tau_t^e) + (\mathbf{1}-\boldsymbol{\eta}_j)^\top \log\big(\mathbf{1}-\boldsymbol{\rho}_\phi(\tau_t^e)\big)\Big)\right], \tag{14}$$

where $t_{\max}$ is the episode length and $\log$ acts elementwise. Its minimizer is $\boldsymbol{\rho}_\phi(\tau_t^e) = \mathbb{E}[\boldsymbol{\eta}_j \mid \tau_t^e]$, which approaches $\boldsymbol{\eta}_j$ as evidence accumulates, reducing $\varepsilon_{\boldsymbol{\rho}}$ in Proposition 2. Pretraining samples Promiser self-play trajectories equally from the early, middle, and late thirds of the third segment. Fine-tuning uses only Promisee–Promiser episodes, since Promisee self-play provides no target.

Promisee. Each partner is drawn uniformly from the K/M Promisers and a Promisee copy, giving a self-play share of $1/(K/M+1)$. The trust $\boldsymbol{\rho}_\phi(\tau_t^e)$ enters the Promisee after its encoder through a stop-gradient, so the RL loss does not update the estimator.

Configuration. Table 9 lists the baseline-specific settings; every baseline is trained with 10 seeds and, except GAMMA, with the same ego-training budget as PPC. All methods, including PPC, share the IPPO backbone in Table 10, which also lists the PPC-specific settings. All models are trained with JAX and JaxMARL (Rutherford et al., 2024) on NVIDIA RTX 3090 GPUs; computational costs are reported in Appendix G.5.

	Overcooked-v1 / v2	**Hanabi**
Shared budget (all baselines)		
Seeds	10	
Ego steps per seed (all but GAMMA)	3×10^7	3×10^9
Population-stage steps per partner	10^7	1.5×10^9
Method-specific settings		
OP: symmetry permutations	ingredients	colors
FCP: partner population	10 seeds $\times$ 3 checkpoints	
MEP: population / entropy coef.	4 / 0.1	
E3T: random-action mixing / prediction loss	0.3 / 1.0	
HSP: candidate / selected partners	36 / 18	
GAMMA: VAE latent / hidden	16 / 256	
GAMMA: total steps	10^8	3×10^9

Table 9: Baseline settings. Values spanning both columns are shared across environment families.

	Overcooked-v1 / v2	**Hanabi**
Backbone (IPPO, all methods)		
Encoder	CNN (32 features)	dense, 512
Core	FC 128, LayerNorm, GRU 128	FC 512
Optimizer / learning rate	Adam, 2.5×10^{-4} (5% warm-up)	Adam, 5×10^{-4}
γ / GAE λ / PPO clip	0.99 / 0.95 / 0.2	0.99 / 0.95 / 0.2
Entropy / value / grad. norm	0.01 / 0.5 / 0.25	0.01 / 0.5 / 0.5
Parallel envs / rollout length	256 / 256	1024, 1024, 512, 512 / 128
Minibatches / epochs	64 / 4	4 / 4
Reward shaping horizon	1.5×10^{7}	none
PPC		
Promise set		
Categories M	5	5
Promises K	25	25
Promisers and Promisee		
Promisers	5	5
Promisee seeds	10	10
Steps per Promiser (3 segments)	3×10^{7}	3×10^{9}
Steps per Promisee	3×10^{7}	3×10^{9}
Promisee LR / entropy / λ	7×10^{-4} / 0.04 / 0.9	5×10^{-4} / 0.01 / 0.95
Trust estimator		
Core	encoder, GRU 128, K sigmoid outputs	
Pretraining LR / epochs per update	2.5×10^{-4} / 4	
Fine-tuning LR / epochs per update	5×10^{-5} / 1	

Table 10: Optimization settings of PPC, identical across the conditions of an environment family. The backbone is shared by all methods, and the Promisee row overrides its values during Promisee training. Hanabi environment counts are given for 2–5 players.

D Environment Details

This appendix describes the three benchmarks, summarized in Table 11. Appendices D.1, D.2, and D.3 cover Overcooked-v1, Overcooked-v2, and Hanabi, each giving the task, observation, actions, and reward.

Environment	Agents	Observability	Observation	Actions	Horizon
Overcooked-v1	2	Full	$H \times W \times 26$	6	400
Overcooked-v2	2	Egocentric 5×5 view	$5 \times 5 \times C$	6	400
Hanabi	2–5	All hands but one's own	658–1280	21–49	Game end

Table 11: Environments. The Overcooked-v2 depth C depends on the layout, and the Hanabi sizes span two to five players (Appendices D.2 and D.3).

D.1 Overcooked-v1

Overcooked-v1 (Carroll et al., 2019) is a two-agent cooking task: agents place three onions in a pot, wait for the soup to cook, plate it, and deliver it. The four layouts (Figure 11) probe distinct coordination demands: handoffs over a divider in `Forced Coordination`, collision avoidance in `Cramped Room`, and directional conventions on a loop in `Coordination Ring` and `Counter Circuit`.

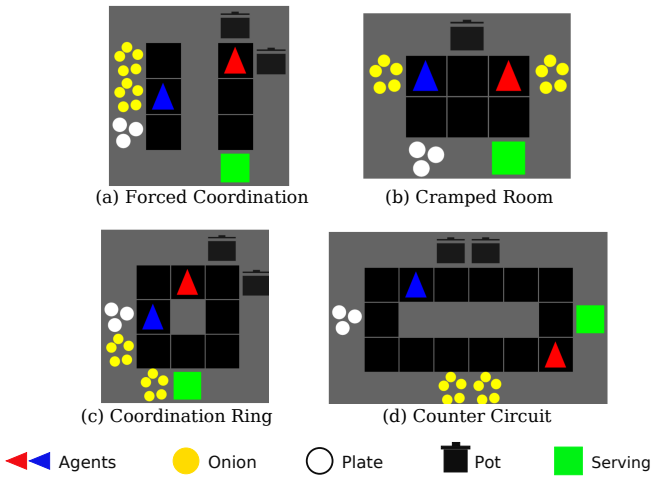


Figure 11: Overcooked-v1 layouts.

Observation. The kitchen is fully observable. Each agent receives an $H \times W \times 26$ grid of the whole layout (4×5 for `Cramped Room`, 5×5 for `Coordination Ring` and `Forced Coordination`, and 5×8 for `Counter Circuit`), encoding both agents' positions, orientations, and held objects, all objects in the kitchen, pot contents, and cooking progress.

Actions. Four cardinal moves, stay, and interact. Depending on what the agent faces and holds, interacting picks up or places an object, adds an onion to a pot, collects a finished soup with a plate, or delivers it.

Reward. Each delivered soup gives both agents $+20$. Throughout the paper, a return is the undiscounted sum of environment rewards over an episode. The shaping rewards used during training (Table 10) are excluded from all reported results and from the refinement feedback.

D.2 OVERCOOKED-V2

Overcooked-v2 (Gessler et al., 2025) adds several ingredient types, recipes, and partial observability, so coordination requires inferring what the partner knows. The target recipe is resampled after every delivery, and start positions are randomized. Among the six layouts (Figure 12), the grounded-coordination layouts require one agent to reveal the recipe to the other, and the demo-cook layouts require one agent to demonstrate it through its actions. The test-time layouts also tell both agents whether the last delivery was correct, so a wrong guess can be corrected.

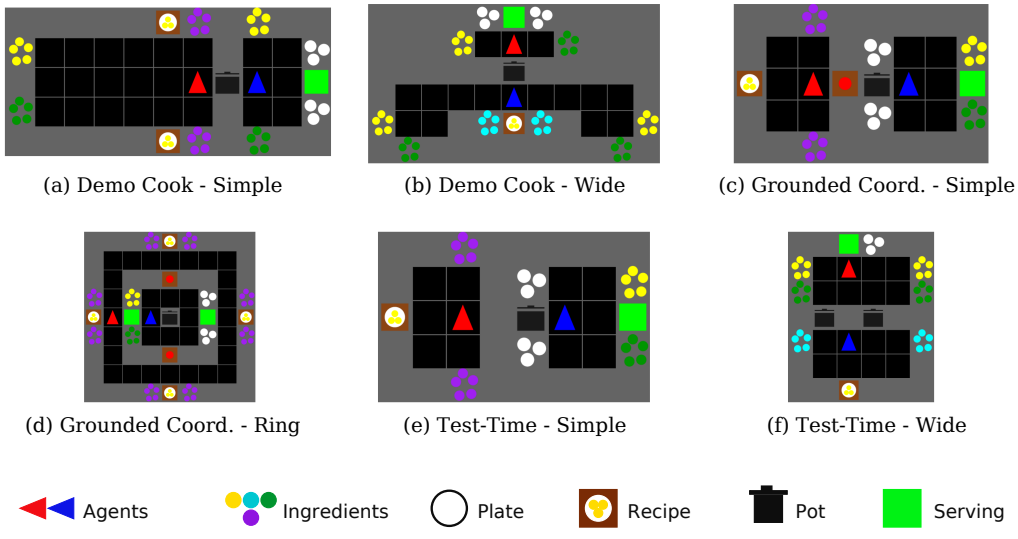


Figure 12: Overcooked-v2 layouts. Brown tiles display the current recipe, which an agent observes only when the tile lies within its view.

Observation. Each agent sees only a 5×5 window around itself, encoded with $C = 5N + 25$ channels for N ingredient types: 40 in the three-ingredient layouts and 45 in `Demo Cook Wide`, plus one delivery-feedback channel in the test-time layouts (41 and 46). The recipe is shown only on indicator tiles, so an agent that cannot see one must infer the target from its partner's behavior. In the grounded-coordination layouts, the indicator must first be switched on with a button.

Actions. The same six actions as in Overcooked-v1.

Reward. A correct delivery gives $+20$, an incorrect one -20, and pressing the recipe button -5. Returns are defined as in Overcooked-v1.

D.3 HANABI

Hanabi (Bard et al., 2020) is a cooperative card game in which players build five color stacks from 1 to 5 by playing cards in order. The 50-card deck holds three 1s, two each of 2–4, and one 5 per color. Players hold five cards in two- and three-player games and four in four- and five-player games. Figure 13 shows a two-player example; larger games differ only in the number of hands and hint targets.

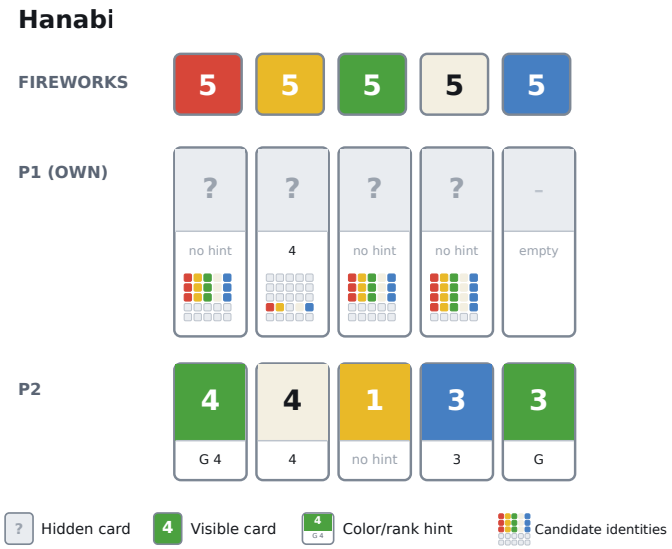


Figure 13: A two-player Hanabi state at the end of a perfect game (score 25). From P1's view, its own cards are hidden and known only through hints; the grids show the identities still consistent with those hints.

Observation. Each player sees every hand except its own, the stacks, the discard pile, and the remaining tokens and deck. About its own cards, it knows only what past hints reveal. The observation vector has 658, 956, 1041, and 1280 entries for two to five players.

Actions. On its turn, a player plays or discards one of its cards, or spends one of 8 information tokens to reveal every card of a chosen color or rank in another player's hand. Discarding returns a token.

Reward. The score is the total height of the stacks, up to 25. A misplay costs one of 3 life tokens, and losing all three ends the game with a score of zero; otherwise, the game ends one round after the deck runs out. The return is the final score, without shaping.

E EXPERIMENTAL DETAILS

This appendix specifies how PPC is compared with the baselines and with human partners. Appendix E.1 describes the baselines and the source of partner diversity in each, Appendix E.2 the AI–AI and human–AI evaluation protocols, and Appendix E.3 the interface used in the human study. Environments are described in Appendix D and training configurations in Appendix C.5.

E.1 BASELINES

All seven baselines share the IPPO backbone and ego budget of PPC, except for the GAMMA budget noted in Table 9. They differ in where partner diversity comes from and how the ego policy is trained against it (Table 12).

Method	Partner diversity	Training mechanism
IPPO	None	Each agent is trained with independent PPO on the shared team reward in self-play (de Witt et al., 2020). The resulting policy follows the single convention that emerges in training, which serves as a reference for ZSC without explicit diversity.
OP	Environment symmetries	Self-play is performed under random task-preserving relabelings, such as ingredient or color permutations, applied to one agent (Hu et al., 2020). The policy therefore cannot rely on arbitrary symmetry-breaking conventions.
FCP	Seeds and checkpoints	Independent self-play runs are frozen at early, middle, and final checkpoints, forming a population with varied conventions and skill levels (Strouse et al., 2021). The ego policy is then trained as a best response to partners sampled from this population.
MEP	Population entropy	A population is trained with a bonus on the entropy of its mixed policy, encouraging members to behave differently (Zhao et al., 2023). The ego policy is then trained against it, with prioritized sampling that favors partners on which the ego performs poorly.
E3T	Random actions	A single end-to-end policy is trained against a partner that mixes its own actions with uniformly random ones (Yan et al., 2023). An auxiliary module predicts the partner's actions from recent context and conditions the ego policy.
HSP	Hidden rewards	Candidate partners are trained on randomly weighted rewards over event features, modeling biased human preferences (Yu et al., 2023). A diverse subset is selected, and the ego policy is trained against it together with self-play partners.
GAMMA	Generative model	A variational autoencoder is fit to trajectories of a partner population to learn a latent space of partner strategies (Liang et al., 2024). The ego policy is trained against new partners decoded from sampled latent vectors.

Table 12: Baselines, the source of their partner diversity, and their training mechanisms. Method-specific settings are listed in Table 9.

E.2 EVALUATION PROTOCOL

AI–AI evaluation. All evaluations use greedy actions, fixed evaluation seeds, and frozen policy and trust-estimator weights. In multi-player Hanabi, the ego occupies one seat and one partner policy fills the others.

- **Self-play (SP)** pairs a policy with itself for 32 episodes per seed.
- **Cross-play (XP)** pairs independently trained seeds of the same method for 32 episodes per ordered pair. We report the mean over ordered pairs and the standard deviation over seed-level scores, each averaging the pairs that include that seed.

- **Inter-algorithm XP** pairs the ego policy of each method with those of every other method (Appendix F.2).
- **BR-Prox** (Wang et al., 2024) compares a policy's return with that of a best response trained for each of 20 held-out hidden-utility partners per Overcooked-v1 layout (Appendix F.2). Overcooked-v2 is omitted, since such fixed partners cannot follow its recipe-dependent interactions and their best responses almost always score zero.

Human–AI evaluation.

- **Participants and design.** All 40 participants gave informed consent, could withdraw at any time, and were compensated. Each played 88 episodes across Overcooked-v1, Overcooked-v2, and two-player Hanabi with every evaluated agent, in an order of games, layouts, and partners randomized per participant; partner identities were never revealed.
- **Measures.** We record the raw team return of every episode.
- **Human–human data.** Participants also played in pairs, yielding 20 human–human episodes per environment, used only for the behavior-coverage analysis of Section 5.3.
- **Statistical analysis.** Participants are the independent units (N=40), with returns averaged over layouts within each participant. PPC is compared with each baseline by two-sided Wilcoxon signed-rank tests with Holm correction; all comparisons give $p < 0.001$ (Figure 4).

E.3 Interface for Human Evaluation

Participants play in a web browser (Figure 14). After a short tutorial on the controls and goal of each game, they play in real time with the agent, and the interface logs the trajectories for analysis.

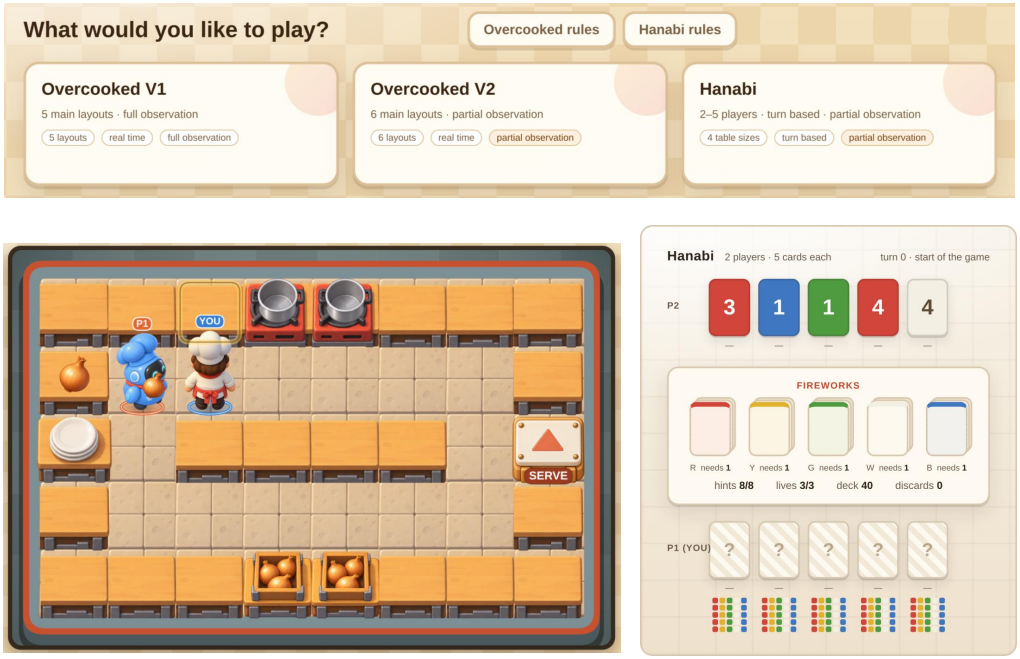


Figure 14: Human evaluation interface: game selection and real-time gameplay.

F Additional Experiments

Self-play and same-method cross-play for every condition are reported in Table 1 of the main paper, so this section covers what the table cannot show. Appendix F.1 reports training behavior, Appendix F.2 coordination with partners that PPC never trained with, Appendix F.3 per-layout human–AI coordination. Further analyses and computational costs are given in Appendix G. The figures use the training budgets and evaluation procedures of Appendices C.5 and E.2, with uncertainty over independent training seeds.

F.1 Learning Curves

Figures 15–17 report cross-play return throughout ego-policy training, so the curves show how coordination between independently trained seeds develops rather than task competence under a shared convention. PPC reaches a higher final return than every baseline in all conditions, and its cross-play return rises from the start of training instead of plateauing near the level of self-play-only methods. In Overcooked-v2 several baselines settle below zero, since an incorrect delivery is penalized and independently trained seeds disagree on the recipe to cook.

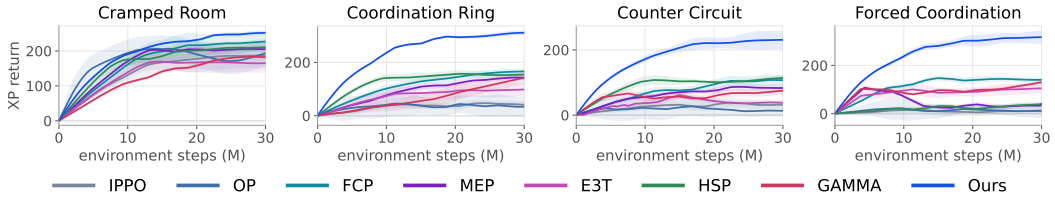


Figure 15: Learning curves on Overcooked-v1. Curves show the mean cross-play return across training seeds, with shaded standard deviation.

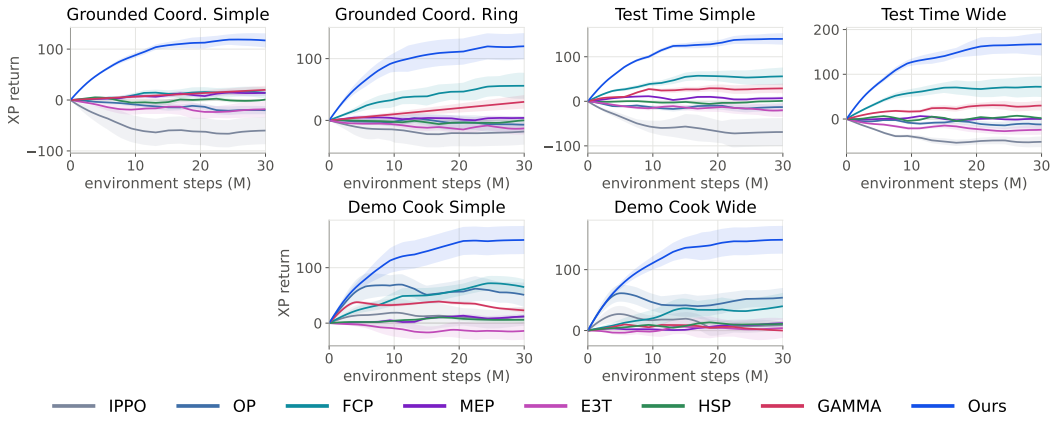


Figure 16: Learning curves on Overcooked-v2, for all six layouts.

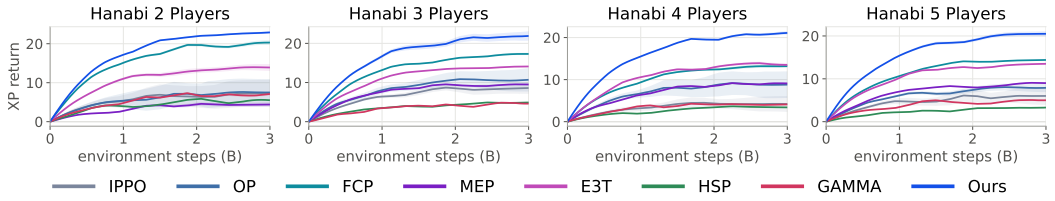
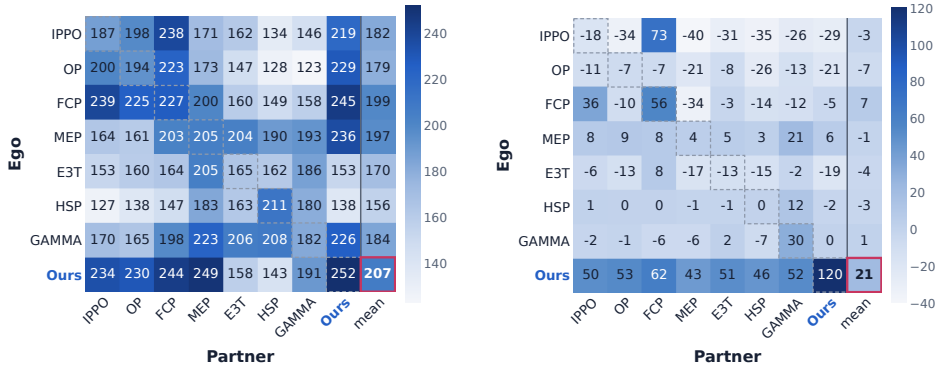


Figure 17: Learning curves in two- to five-player Hanabi.

F.2 Coordination with Held-Out Partners

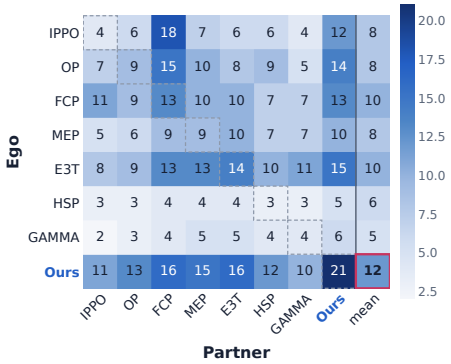
Beyond cross-play between different seeds of the same method, we further evaluate held-out XP in two stronger settings: partners trained by a different method, and partners that no evaluated method trained against.

Partners trained by other methods. Inter-algorithm cross-play pairs each ego policy with those of the other methods; we report one condition per environment (Figure 18). PPC attains the highest mean return in all three, with the largest margin in Overcooked-v2, where every baseline averages near zero or below while PPC, as the ego, stays positive with every partner. Since partners are fixed, this gain comes from the Promisee's trust-based adaptation.



(a) Overcooked-v1, Cramped Room

(b) Overcooked-v2, Grounded Coord. Ring



(c) Hanabi, 4 players

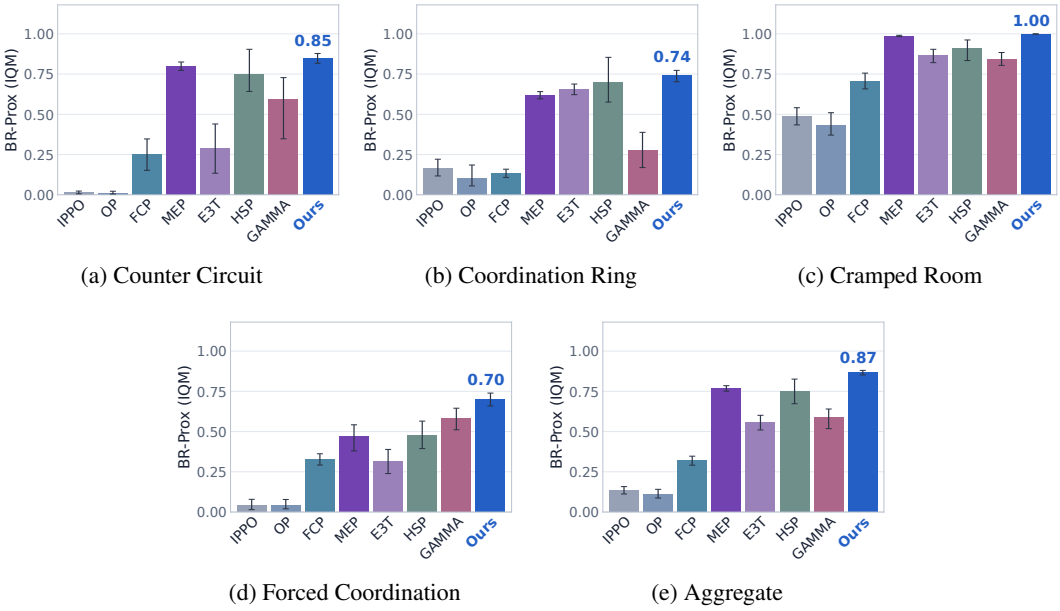
Figure 18: Inter-algorithm cross-play. Rows: ego method; columns: partner method; dashed cells: same-method pairs. The last column averages over all cross-method pairs involving that method as ego or partner. In Hanabi, the remaining seats are filled with copies of the partner method.

Best-response proximity. We additionally evaluate all methods against partners unseen by any method during training, following ZSC-Eval (Wang et al., 2024). For each Overcooked-v1 layout, we train 36 partners with random hidden utilities (Yu et al., 2023), keep the 10 that require the most distinct best responses (BR-Div), and use their mid and final checkpoints, giving 20 reference partners. With BR_p an approximate best response to partner p, the score for ego policy π in seat k is

$$\mathrm{BRProx}(\pi, p, k) = \min\left(1,\ \frac{R(\pi, p, k)}{R(\mathrm{BR}_p, p, k)}\right). \tag{15}$$

Pairs whose best response itself fails to deliver are excluded, since the ratio is then undefined; at least 72.5% of pairs remain in every layout (Figure 20). We report the IQM (Agarwal et al., 2021) over the remaining (partner, seat, seed) triples with 95% bootstrap confidence intervals over seeds.

PPC achieves the highest BR-Prox on every layout, with an aggregate of 0.866 compared with 0.769 for the strongest baseline, MEP (Figure 19). The diversity-based baselines follow: HSP, trained on the same family of biased partners from which this pool is constructed, reaches 0.70–0.91 where both seats can perform every subtask, but falls to 0.477 on Forced Coordination, while PPC retains 0.700. Covering a range of utility biases therefore does not supply the handoff convention an asymmetric layout requires, whereas a promise states which seat yields.



(a) Counter Circuit (b) Coordination Ring (c) Cramped Room

(d) Forced Coordination (e) Aggregate

Figure 19: BR-Prox on Overcooked-v1. Bars show the IQM and error bars the 95% confidence interval over seeds; (e) pools the triples of all four layouts.

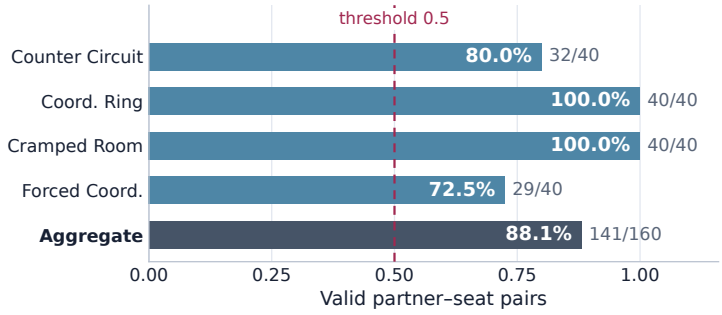
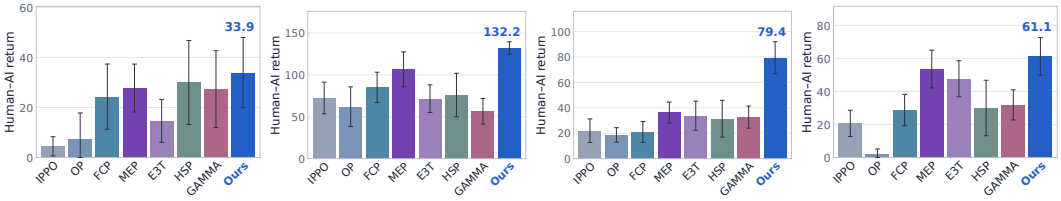


Figure 20: Fraction of partner–seat pairs whose best response obtains a positive return.

F.3 Per-Layout Human Evaluation

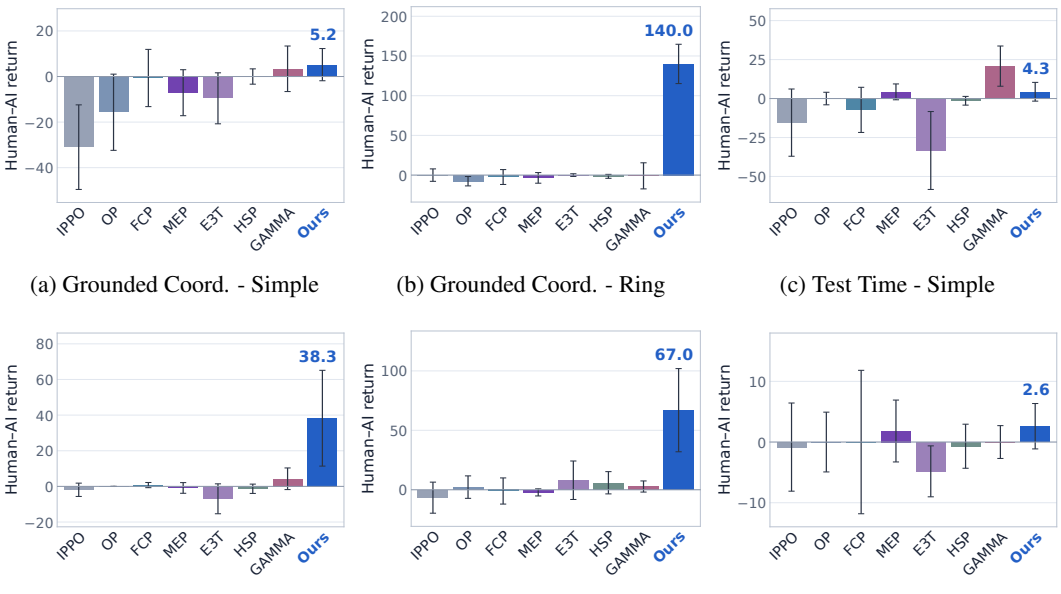
The human evaluation of Figure 4 pools layouts within each environment, which can hide where promise-based coordination helps. Here we report every Overcooked layout separately, under the protocol of Appendix E.2 and with the interface of Appendix E.3. Hanabi is evaluated with two players only, so Figure 4 already reports it per condition. The vertical axis is scaled per layout, since the attainable return differs across layouts.

PPC obtains the highest human–AI return in all four Overcooked-v1 layouts (Figure 21), including the ones that force handoffs across a divider. In Overcooked-v2 (Figure 22) the gains concentrate in the layouts where human–AI teams complete deliveries at all, reaching 140.0 on Grounded Coord. Ring and 67.0 on Demo Cook Simple, while the layouts in which even the best team rarely succeeds leave little room for any method to separate. Baselines fall below the return of an idle policy in several Overcooked-v2 layouts, because an incorrect delivery is penalized.



(a) Forced Coordination (b) Cramped Room (c) Coordination Ring (d) Counter Circuit

Figure 21: Human evaluation on Overcooked-v1, per layout. Mean episode return with 95% confidence intervals; the vertical axis is scaled per layout.



(a) Grounded Coord. - Simple (b) Grounded Coord. - Ring (c) Test Time - Simple

(d) Test Time - Wide (e) Demo Cook - Simple (f) Demo Cook - Wide

Figure 22: Human evaluation on Overcooked-v2, per layout. Mean episode return with 95% confidence intervals; the vertical axis is scaled per layout.

G Additional Analyses

This appendix first repeats the analyses of Section 5.3 on Overcooked-v2, where partial observability requires agents to reveal or infer the recipe. Appendix G.1 covers promise composition and refinement gains, Appendix G.2 human behavior coverage, and Appendix G.3 component ablations and the LLM comparison. Unless stated otherwise, these results are averaged over the six layouts and follow the protocols of Section 5.3. Appendix G.4 then examines the sensitivity of PPC to the number of categories and promises, and Appendix G.5 reports the computational cost of all methods.

G.1 Promise Composition

p_1: Agents add recipe ingredients only to the fuller empty or same-ingredient pot.

p_2: Cook takes a plate only while soup cooks and scoops ready soup at once.

p_3: Recipe-side agent stands below the pot matching the recipe.

Set	SP	XP	$\mathrm{Div}_{\mathrm{fix}}$	Div_{η}
Null (IPPO)	178	−51	0.050	
p_1	183	189	0.181	0.241
p_2	193	193	0.370	0.493
p_3	208	161	0.430	0.554
$p_1 + p_2$	186	184	0.203	0.353
$p_1 + p_3$	187	182	0.450	0.637
$p_2 + p_3$	213	189	0.581	0.766
$p_1 + p_2 + p_3$	**224**	**207**	**0.599**	**0.779**

Table 13: Analysis of promise combinations on Overcooked-v2.

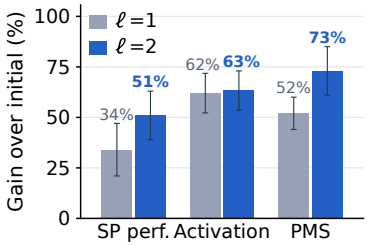


Figure 23: Gains from refinement on Overcooked-v2.

Promise combinations. We focus on `Test Time Wide`, where only the agent on the recipe side can observe the recipe, so successful task completion critically depends on conveying it to the partner through behavior. Table 13 evaluates all subsets of three representative promises in this layout, with $\mathrm{Div}_{\mathrm{fix}}$ and Div_{η} defined as in Table 2. IPPO attains a high SP of 178 but a negative XP of -51, as its self-play conventions fail with unseen partners. As in Overcooked-v1, individual promises induce distinct, task-useful behaviors and substantially improve coordination with unseen partners; for example, p_1 barely changes SP but raises XP from -51 to 189, while p_3, which conveys the recipe through the agent's position, attains the highest SP among single promises (208) together with high diversity. Combining promises generally further improves both coordination and diversity, with the full set achieving the highest SP, XP, and diversity under both enforcement settings (224, 207, 0.599, and 0.779).

Refinement gains. Figure 23 further shows gains over the initial promises in SP performance, activation, and PMS across refinement rounds. Activation improves by 62% at $\ell = 1$ and changes little at $\ell = 2$ (63%), indicating that a single round suffices to make promises reliably activated. In contrast, SP performance and PMS continue to improve at $\ell = 2$, from 34% to 51% and from 52% to 73%, respectively, a larger second-round gain than in Overcooked-v1 (Figure 5). Once promises are reliably activated, the second round thus mainly improves their separation and performance, which is consistent with the greater behavioral diversity required to convey recipe information. Overall, refinement improves both coordination performance and promise separation in Overcooked-v2, where the second round is even more beneficial than in Overcooked-v1.

G.2 Human Behavior Coverage

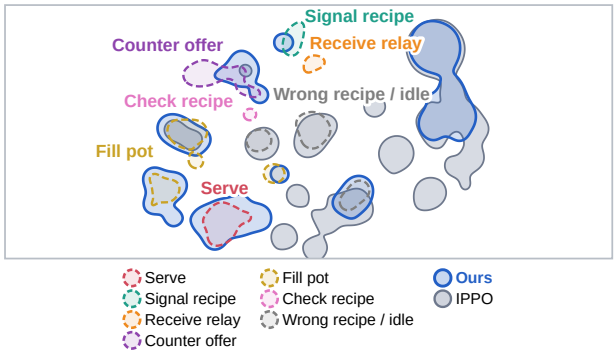


Figure 24: Joint UMAP of human and agent behavior on Overcooked-v2.

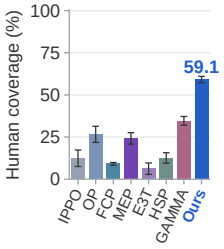


Figure 25: Human behavior coverage.

Behavior	Ours	IPPO
Serve	**99.4** ±1.0	3.7 ±5.2
Fill pot	**88.8** ±2.3	29.3 ±10.0
Counter offer	**31.4** ±5.8	2.3 ±1.9
Check recipe	9.5 ±2.1	9.5 ±3.9
Receive relay	**3.6** ±2.3	0.0 ±0.0
Signal recipe	**21.0** ±3.1	0.0 ±0.0
Wrong / idle†	**31.2** ±3.2	92.6 ±2.3
All	**59.1** ±1.7	12.6 ±2.8

Table 14: Per-behavior human coverage (%). †Lower is better.

We examine how well the behaviors induced by each method cover coordination patterns observed in human play on Overcooked-v2, using human–human trajectories from this environment. To place human and agent behaviors in a common space, we pool the six Overcooked-v2 layouts, remove stay steps, and cut each agent's trajectory into 16-step windows, sampling equally many windows per method and layout. Each window is described by recipe-aware interaction events (e.g., adding a correct or incorrect ingredient, delivering a correct or incorrect soup, placing items on a counter, pressing the recipe button), the objects the agent faces, and its proximity to the partner, and is assigned a rule-based behavior type. All windows are embedded jointly with UMAP (McInnes et al., 2018), using the behavior types as a categorical target, and a method's coverage is the fraction of human windows lying inside its 90%-mass kernel-density region. The Overcooked-v1 analysis in Figure 7 follows the same windowing and coverage definition.

Figure 25 measures overall human behavior coverage, while Figure 24 visualizes human–human and agent cross-play trajectories in the shared UMAP space, with behavior types identified in the space. Beyond the behaviors in Overcooked-v1, human play in Overcooked-v2 includes recipe-related behaviors: checking the recipe indicator, signaling the recipe to the partner, and offering ingredients on a shared counter, together with a failure type that groups incorrect deliveries and idling. PPC covers 59.1% of the human behavior space, more than 1.6 times that of the strongest baseline, GAMMA.

Table 14 further quantifies coverage for each behavior type. PPC almost fully covers the core task behaviors, serving and filling pots, at 99.4% and 88.8% compared with 3.7% and 29.3% for IPPO. PPC also conveys the recipe as humans do, covering 21.0% of signal-recipe and 31.4% of counter-offer behaviors, which IPPO rarely exhibits (0.0% and 2.3%). In contrast, IPPO's coverage concentrates on the failure type, reaching 92.6% compared with 31.2% for PPC, which is consistent with its near-zero or negative human–AI returns in Figure 4. Checking the recipe, which each agent performs on its own, is covered equally by both (9.5%), and receive relay remains rare for both. As in Overcooked-v1, PPC achieves this coverage without using human behavior data for promise design or agent training. These results show that promises capture the recipe-conveying behaviors that partial observability requires, helping PPC coordinate with unseen human partners.

G.3 Component Ablations

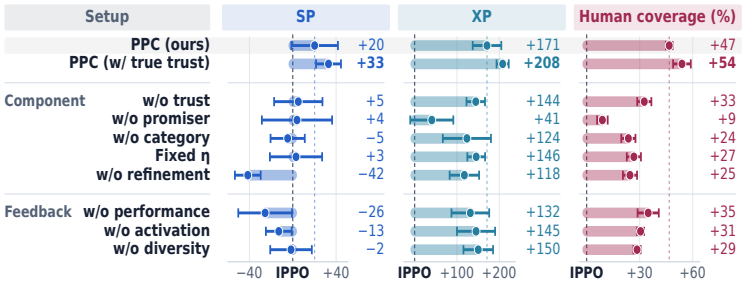


Figure 26: Component ablations of PPC on Overcooked-v2.

LLM	SP	XP
Qwen3.5-9B	110±65	91±57
Qwen3.5-27B	154±33	127±41
Claude Opus 4.8	161±23	138±36
GPT-4o	155±48	129±43
GPT-5.6 Sol	**174±21**	**141±19**

Table 15: LLM comparison on Overcooked-v2.

Component ablations. To assess the contribution of PPC's components on Overcooked-v2, Figure 26 evaluates the same variants as Figure 8. Unlike in Overcooked-v1, PPC with true trust clearly outperforms PPC, with larger gains in SP, XP, and human coverage (+33, +208, and +54 compared with +20, +171, and +47). Since the recipe is visible to only one agent, the Promisee must infer its partner's promises from partially observed behavior, which makes trust harder to estimate but more valuable when accurate. Accordingly, removing trust reduces all three gains (+5, +144, and +33). Among the components, removing Promisers causes the largest drops in XP and human coverage (+41 and +9), and removing refinement causes the largest drop in SP, falling below IPPO (-42), consistent with the larger refinement gains in Figure 23. Excluding promise categories or fixing η also reduces both performance and human coverage. For refinement, every feedback metric improves SP, XP, and human coverage, with the performance feedback contributing most to SP. Overall, these results confirm the trends of Overcooked-v1, while suggesting that more accurate trust estimation could further improve PPC under partial observability.

LLM comparison. Table 15 further shows consistent performance across LLMs, as in Table 4. Every model yields XP between 91 and 141, well above the best baseline average of 50.5 (FCP in Table 1). Smaller models degrade more than in Overcooked-v1 and show higher variance across seeds, since promises in Overcooked-v2 must also condition on recipe information that only some agents observe, which places more demand on the state summary and the generated code. Overall, PPC remains relatively robust to the choice of LLM on Overcooked-v2, while stronger models become more beneficial under partial observability.

G.4 HYPERPARAMETER SENSITIVITY

	Overcooked-v1		Overcooked-v2	
Setting	SP	XP	SP	XP
Categories M ($K/M = 5$)				
$M = 1$	226±10	211±14	144±44	90±43
$M = 5$	**291±11**	**277±18**	**174±21**	**141±19**
$M = 10$	259±38	254±22	124±36	110±33
Promises K ($M = 5$)				
$K = 5$	252±13	208±25	155±32	10±50
$K = 25$	**291±11**	**277±18**	**174±21**	**141±19**
$K = 50$	253±24	265±31	112±28	93±34

Table 16: Sensitivity of PPC to the number of promise categories M and the number of promises K. Each entry reports the mean return ± standard deviation over seeds, averaged over the layouts of each environment. Varying M keeps $K/M = 5$ promises per category, and varying K keeps $M = 5$. Shaded rows correspond to the default setting ($M = 5$, $K = 25$).

We examine the sensitivity of PPC to the number of promise categories M and the number of promises K. Table 16 varies $M \in \{1, 5, 10\}$ with $K/M = 5$ promises per category, and $K \in \{5, 25, 50\}$ with $M = 5$, in which case K also determines the population size K/M. We report SP and XP averaged over the layouts of Overcooked-v1 and Overcooked-v2. A single category ($M = 1$) covers only one coordination decision and lowers returns, while ten categories ($M = 10$) over-constrain each Promiser and also degrade performance (124 SP and 110 XP in Overcooked-v2). With $M = 5$, a single Promiser ($K = 5$) sharply reduces XP (208 in Overcooked-v1 and 10 in Overcooked-v2), whereas ten Promisers ($K = 50$) make Promisee training less stable, lowering returns and increasing variance. Overall, PPC is robust to the choice of M and K: apart from the degenerate single-Promiser setting, every configuration achieves at least 211 XP in Overcooked-v1 and 90 XP in Overcooked-v2, and the same configuration performs best in both environments. We therefore use $M = 5$ and $K = 25$ as the default for all environments.

G.5 Computational Cost

All methods were trained on NVIDIA GeForce RTX 3090 (24 GB) GPUs (Appendix C.5). Training time is reported in GPU hours and minutes, i.e. the number of GPUs multiplied by the wall-clock time needed to train one ego policy, and excludes evaluation.

Training cost. Table 17 compares training cost with the baselines for every condition. For multi-stage methods the totals include every stage: population construction and ego training for FCP, MEP, and HSP. For PPC, the cost of the shared Promiser population, including the trust estimator, is amortized over the Promisee seeds trained against it, and the cost of one Promisee is added.

Layout	IPPO	OP	FCP	MEP	E3T	HSP	GAMMA	Ours
Overcooked-v1								
Counter Circuit	25m	25m	53m	2h06m	25m	4h17m	2h56m	1h18m
Coordination Ring	23m	23m	50m	1h58m	25m	4h01m	2h44m	1h22m
Cramped Room	23m	23m	48m	1h54m	23m	3h53m	2h39m	1h32m
Forced Coordination	24m	23m	51m	1h59m	24m	4h04m	2h46m	1h15m
Overcooked-v2								
Grounded Coord. Simple	23m	25m	49m	1h55m	25m	4h27m	2h47m	1h34m
Grounded Coord. Ring	24m	23m	51m	1h59m	23m	4h36m	2h53m	1h42m
Test Time Simple	23m	23m	50m	1h57m	25m	4h30m	2h49m	1h23m
Test Time Wide	24m	24m	50m	1h58m	24m	4h33m	2h51m	1h12m
Demo Cook Simple	24m	23m	50m	1h58m	24m	4h33m	2h51m	1h18m
Demo Cook Wide	25m	24m	51m	2h01m	25m	4h39m	2h55m	1h29m
Hanabi								
2 Players	2h04m	2h07m	4h45m	11h04m	2h27m	30h40m	19h03m	17h18m
3 Players	2h50m	2h54m	6h33m	15h15m	3h23m	42h15m	26h14m	20h46m
4 Players	3h37m	3h42m	8h22m	19h29m	4h26m	54h01m	33h33m	22h50m
5 Players	4h47m	4h53m	11h04m	25h47m	6h00m	71h25m	44h22m	26h16m

Table 17: Training time per condition in GPU hours and minutes, covering every training stage of each method. Our method is shaded.

Language-model cost. The language model is called four times offline per condition: once for the state summary, once for the initial promise set, and once per refinement round (Appendix C.1). No language-model calls are made during training, evaluation, or deployment. Table 18 reports usage estimated from the archived prompts and responses. Cost is computed at the GPT-5.6 Sol API rates of $4 and $20 per million input and output tokens, counting visible text. Every environment requires at most 20 minutes of generation and costs less than $0.60 per condition.

Item (per condition)	Overcooked-v1	Overcooked-v2	Hanabi
Language-model calls	4	4	4
Generation time	13m	20m	9m
Input tokens	48,897	75,078	69,523
Output tokens	15,456	12,747	15,026
Cost (USD)	0.50	0.56	0.58

Table 18: Mean language-model usage per condition.