POPGEN: GENERATING SEMANTICALLY DIVERSE POPULATIONS WITH LLMS FOR ZERO-SHOT COORDINATION

Anonymous authors
Paper under double-blind review

ABSTRACT

In multi-agent reinforcement learning (MARL), exposing a cooperator to partners with *diverse* strategies during training is key to better collaboration at test time. However, existing methods typically rely on trajectory-level proxies to measure diversity, which may label two distinct trajectories of the same strategy as diverse. Our key insight is instead to leverage LLMs' ability to describe *behaviorally* distinct agents in environments, similar to how humans can recognize high-level strategies. To this end, we introduce POPGEN, a framework that synthesizes *semantically* diverse agents from scratch using LLMs. POPGEN first generates a diverse set of textual descriptions of agent behaviors, then translates each behavior into a code policy that can be executed within the environment. We show that training a cooperator against a POPGEN population achieves robust zero-shot cooperation performance across four collaborative environments, and that the resulting policies adapt to diverse partners. Additionally, we measure how diversity scales with larger POPGEN populations and provide insights on how population diversity impacts cooperation and how it can conflict with the quality of the agents. Our code is available at https://github.com/anon/repo.[1]

1 INTRODUCTION

Training agents that collaborate effectively with unseen partners is a key challenge in multi-agent systems like robotics or cooperative video games. The task, known as zero-shot coordination (ZSC), requires a cooperator to coordinate with partners it has never interacted with, whose conventions, skill level, and strategy are unknown at test time. This is challenging because cooperative tasks typically admit many equally effective but mutually incompatible conventions, so a cooperator trained with one arbitrary convention may fail when paired with a partner that follows a different one (Hu et al., 2020; Carroll et al., 2019).

A common approach to ZSC is population play, which trains a cooperator against a population of partner agents. The effectiveness of population play depends on the *diversity* of the population against which the cooperator is trained, since a high coverage of the strategy space during training increases the likelihood that the test-time partner is in-distribution for the cooperator, allowing them to collaborate effectively (Strouse et al., 2021; Lupu et al., 2021; Sarkar et al., 2023). Existing methods build such populations by optimizing diversity proxies computed from trajectory statistics (Lupu et al., 2021; Zhao et al., 2023). However, these measures may fail to capture what humans perceive as meaningfully distinct strategies (Bettini et al., 2024c; Wang et al., 2025). For example, two chefs may take different trajectories in a kitchen while following the same strategy (e.g., only carrying onions). The issue is not that these trajectories are wrongly classified as different (they are indeed different), but that optimizing for such diversity can lead to undesired training populations that are diverse in their motion but do not cover the distinct strategies that matter for coordination (Sarkar et al., 2023).

In contrast to these low-level measures, humans naturally describe trajectories at a higher level, recognizing groups of strategies. Our key idea is to use LLMs as a higher-level interface for specifying differences through natural language, as humans do. This allows us to represent behaviors in

[1] Code is provided in the supplementary material for double-blind review.

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.

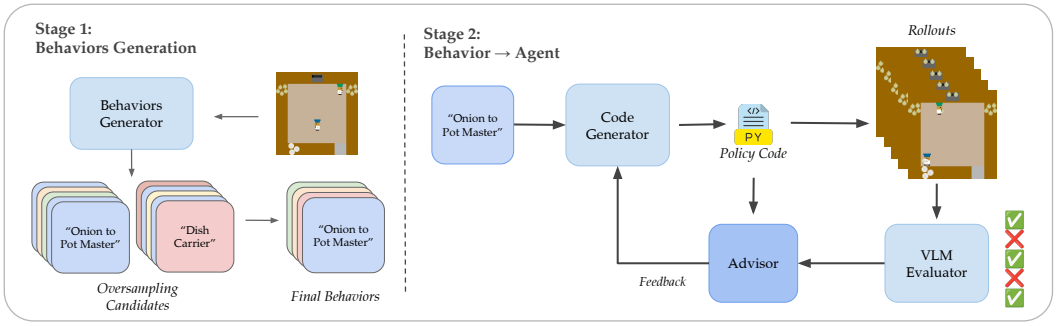


Figure 1: **Overview of POPGEN.** *Left (Stage 1):* An LLM produces a diverse set of behaviors at the text level using oversampling. *Right (Stage 2):* Each behavior description enters an iterative refinement loop: a generator translates the behavior into an executable policy (Policy Code), which is evaluated with a VLM through environment rollouts. An advisor then analyzes rollout results and provides feedback to guide subsequent refinement attempts.

terms of their high-level *semantics* rather than their low-level actions, providing a natural space for expressing diverse behaviors. We introduce POPGEN, a framework that leverages LLMs to define semantically diverse behaviors in language space and translate them into executable policies that can interact with the environment. POPGEN operates in two stages. First, it generates a diverse set of behavior descriptions in natural language, where high-level semantic differences can be expressed explicitly. We use tailored prompting methods to balance the quality of behavior descriptions with their semantic diversity, encouraging descriptions that are both meaningful and genuinely distinct. Second, POPGEN grounds these high-level semantic descriptions into executable agent behaviors by synthesizing policy execution code. We generate the policy code in an iterative loop, where structured feedback from an advisor module guides progressive refinement.

We use the diverse population generated by POPGEN for population-play training and evaluate its impact on cooperation with novel partners. We find that training with semantically diverse partners improves robust cooperation, demonstrating the value of diverse behaviors. We further show that semantic diversity grows with larger POPGEN populations, and that POPGEN increases diversity while controlling for the quality of the synthetic agents (Pugh et al., 2016). Finally, we show that turning textual behavior descriptions into executable policy code, rather than LLM-generated reward functions, yields partners that more faithfully exhibit their specified behaviors and leads to more robust cooperators.

In summary, our contributions are:

- We propose POPGEN, a framework that generates diverse agent populations from scratch by leveraging LLMs' understanding of semantic diversity.
- We show that training a cooperator against POPGEN achieves the best performance across four collaborative environments when matched against other population-play baselines.
- We provide extensive ablations that aim to understand the link between semantic diversity and downstream cooperation, as well as insights on methods to turn textual descriptions into executable agents using LLMs.

2 RELATED WORK

Generating diverse populations for ZSC. Zero-shot coordination requires a cooperator to work with teammates it never saw during training (Hu et al., 2020). Population play achieves this by training an agent against a set of partners, and how well population play works depends on the behavioral diversity of the population against which an agent is trained (Bettini et al., 2024a; McKee et al., 2022). Existing methods for generating agent populations vary from pooling checkpoints across random seeds (Strouse et al., 2021) to optimizing explicit diversity objectives over trajectory or action distributions (Lupu et al., 2021; Zhao et al., 2023), and more recently to sampling partners from a generative model of human play (Liang et al., 2024). These diversity measures all operate in trajectory, action, or latent space, where distances may not reflect the strategic distinctions a human

would recognize (Bettini et al., 2024c; Wang et al., 2025). We instead leverage the semantic diversity captured at the text level to generate diverse populations of agents.

LLM-generated agents. LLMs can translate natural-language specifications into agents in several forms. LLM-as-Agent systems directly interact with the environment but require costly LLM calls during rollout (Huang et al., 2022; Yao et al., 2023), while reward-generation methods translate textual objectives into reward functions and train RL policies (Ma et al., 2024; Xie et al., 2024). However, designing rewards that faithfully capture high-level behaviors is challenging, as RL policies can optimize the reward without exhibiting the intended behavior (Choi et al., 2025). This challenge is pronounced in multi-agent settings, where their interactions depends on other agents. Policy code instead provides a direct representation of behavior by explicitly encoding its decision logic (Liang et al., 2023; Wang et al., 2023; Fu et al., 2026; Ahn et al., 2026). Recent advances in LLM code generation further enable refining such policies through execution feedback (Fu et al., 2026; Ahn et al., 2026). In our experiments, we show that Policy Code more reliably realizes the specified behaviors than RL policies, resulting in higher behavioral adherence and stronger cooperator performance.

3 Problem Formulation

Multi-agent interaction. To formalize multi-agent coordination, we adopt the Dec-POMDP framework (Bernstein et al., 2002; Oliehoek & Amato, 2016). A Dec-POMDP is a tuple $\mathcal{M} = \langle \mathcal{I}, \mathcal{S}, \{\mathcal{A}^i\}_{i\in\mathcal{I}}, \{\mathcal{O}^i\}_{i\in\mathcal{I}}, \{Z^i\}_{i\in\mathcal{I}}, T, R, \gamma \rangle$, where $\mathcal{I} = \{1, \dots, n\}$ is the set of agent indices and $\mathcal{S}$ is the state space. Each agent i has an action space $\mathcal{A}^i$ and an observation space $\mathcal{O}^i$. From state s_t at time step t, the observation function Z^i defines the probability of agent i receiving an observation $o_t^i \sim Z^i(\cdot \mid s_t)$. An action a_t^i of agent i is sampled from its policy $\pi^i(\cdot \mid h_t^i)$, where $h_t^i = (o_0^i, \dots, o_t^i)$ is agent i's observation history. The transition function $T(\cdot \mid s_t, a_t)$ defines the probability of the next state s_{t+1} given the joint action $a_t = (a_t^1, \dots, a_t^n)$. R is the shared reward function $r_t = R(s_t, a_t)$, and $\gamma \in [0, 1)$ is the discount factor.

Training against populations of partners. In this work, we consider a symmetric two-player setting in which the two agents are exchangeable. For notational simplicity, we designate agent 1 as the cooperator and agent 2 as the partner. Population play proceeds in two phases. It first constructs a population $\Pi = \{\pi_1^2, \dots, \pi_K^2\}$ of K partner policies that exhibit diverse coordination behaviors. In the second phase, we train a cooperator policy π_ϕ^1 with parameters ϕ to maximize the expected return averaged over partners sampled from Π:

$$\phi^* = \arg\max_\phi \mathbb{E}_{\pi_k^2 \sim \text{Uniform}(\Pi)} \left[J(\pi_\phi^1, \pi_k^2) \right] = \arg\max_\phi \frac{1}{K} \sum_{k=1}^{K} J(\pi_\phi^1, \pi_k^2), \tag{1}$$

where $J(\pi_\phi^1, \pi_k^2) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t \right]$ is the expected discounted shared return when the cooperator π_ϕ^1 is paired with partner π_k^2. In PopGen, each partner policy π_k^2 is associated with a natural-language behavior description b_k. We denote the corresponding behavior set by $\mathcal{B} = \{b_1, \dots, b_K\}$.

4 PopGen: Generating Diverse Agent Populations

We introduce PopGen, a framework for synthesizing semantically diverse populations in multi-agent environments. The resulting population can then be used to train a cooperator for zero-shot cooperation. As illustrated in Figure 1, PopGen leverages LLMs in two stages.

4.1 Stage 1: Generating Semantically Diverse Behaviors

The first stage feeds an environment description as input to an LLM, and queries it to generate various behaviors that agents could elicit in it (e.g., an "onion-to-pot master" in `overcooked`). The objective here is to generate a pool of *diverse* behaviors, in order to train a cooperator against it (Lupu et al., 2021; Bettini et al., 2024a). Such diversity is hard to measure directly from policies, since two policies can take different actions while following the same strategy (Bettini et al., 2024c; Wang et al., 2025). We instead measure *semantic* diversity from the behaviors' textual descriptions.

Specifically, we encode each description b_k into a unit-normalized text embedding e_k and quantify the diversity of a behavior set $\mathcal{B} = \{b_1, \ldots, b_K\}$ as the average pairwise distance between its embeddings:

$$D(\mathcal{B}) = \frac{2}{K(K-1)} \sum_{k<\ell} \sqrt{2\big(1 - e_k^\top e_\ell\big)}. \tag{2}$$

The tension between quality and diversity of agent populations. An important challenge when targeting textual diversity in the population is to avoid achieving it at the cost of degrading the quality of the generations. For instance, one can simply generate diverse but gibberish text as the behaviors for an environment to optimize diversity. This problem, known as the quality-diversity (Zhang et al., 2020) trade-off in NLP, is a result of the two metrics directly conflicting when optimizing for diversity[2]. In our setup, we define behavior *quality* along the following axes: *realism* (would a real player plausibly play this way?); *relevance* (is the behavior relevant to the environment?); and *specificity* (is the behavior specific enough so that it encodes a recognizable style of play?). Concretely, we compute this by grading each axis from 0 to 5 with an LLM[3], and sum these into a score out of 15.

POPGEN is carefully designed to address this tension, decoupling the search for diverse behaviors from the selection of relevant ones to preserve quality of generations. The LLM is first queried to generate a set of candidates by oversampling (i.e., generating many more candidates than expected), then re-queried to select the highest-quality candidates among the set as shown in Figure 1. Additionally, POPGEN employs a simple trick from Verbalized Sampling (VS), which queries the LLM to output each candidate's probability alongside the generation and has shown strong empirical results (Zhang et al., 2025). We empirically show that POPGEN can generate behavior populations that are both high-quality and semantically diverse in Section 5.3.

4.2 STAGE 2: INSTANTIATING BEHAVIORS AS AGENTS

The second stage translates each textual behavior into a policy that interacts with the environment and other agents using an LLM. An LLM can act as the policy itself, outputting an action at every step (Yin et al., 2025; Wang et al., 2024), generate a reward function for an RL policy to optimize (Ma et al., 2024; Xie et al., 2024; Li et al., 2025), or directly implement the decision logic as executable code (Liang et al., 2023; Wang et al., 2023). POPGEN adopts the latter, modeling each agent's policy as a Python function, which is cheap to execute and avoids training an RL agent per behavior. Section 5.4 compares this choice with SemDiv's reward-function approach (Li et al., 2025).

Measuring adherence with an evaluator. A key requirement in Stage 2 is to evaluate whether a generated policy actually follows its target behavior. For this, we define *adherence*. Adherence is measured by rolling out a candidate policy π with a partner, and asking a *VLM* to classify whether the trajectory τ aligns with the intended behavior b based on screenshots and state sequences of τ. The classifier outputs 1 if a trajectory aligns with b and 0 otherwise. We repeat this on M different initial environment states to prevent a candidate from achieving high adherence only under a favorable configuration (Zheng et al., 2023; Choi et al., 2025), and report the final score as the fraction of accepted rollouts $\tau_1, \ldots, \tau_M$:

$$\mathrm{Adh}(\pi, b) = \frac{1}{M} \sum_{m=1}^{M} \mathrm{VLM}(\tau_m, b) \in [0, 1]. \tag{3}$$

Iterative refinement loop. Generating a policy that reliably exhibits a specified behavior can be challenging in complex multi-agent environments (Choi et al., 2025). We address this challenge with an iterative refinement procedure (Madaan et al., 2023; Khattab et al., 2024) based on the above adherence measure. The LLM iteratively generates a candidate policy from the behavior and

[2]The intuition is the following: optimizing diversity means expanding towards uncovered axes *unconditionally*, while optimizing quality *restricts* the search space to fewer points.

[3]The judge prompt is detailed in Appendix E.3.

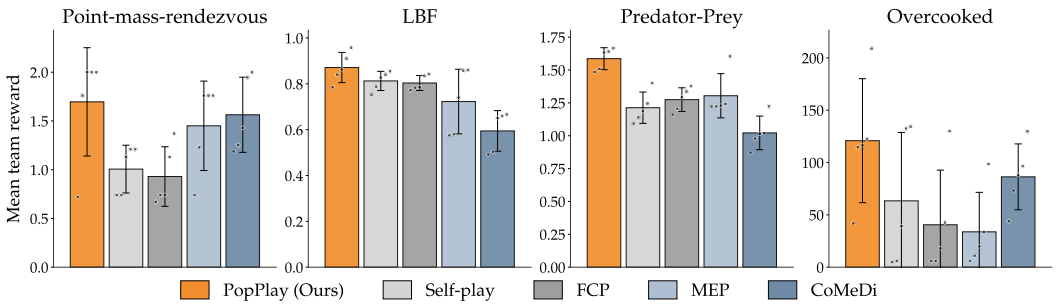


Figure 2: **Zero-shot coordination with held-out partners.** Each bar reports the average team reward (higher is better) when each cooperator is paired with the held-out partners, which do not appear in any training population. Error bars give the standard deviation across five runs, and dots mark individual runs. POPPLAY attains the highest reward in all four environments. Table 5 reports the reward against each individual partner.

environment description, and gets rich feedback from a separate LLM advisor based on candidate policy rollouts. This loop is repeated until the candidate policy reaches a target level of adherence or all attempts are exhausted. The refinement loop is detailed in Algorithm 1

4.3 POPULATION PLAY WITH POPGEN PARTNERS

A POPGEN population can serve directly as the partner population for population play. We call this POPPLAY, which trains a cooperator π^1_ϕ against a POPGEN partner population. Given partner policies $\Pi_{\text{POPGEN}} = \{\pi^2_1, \ldots, \pi^2_K\}$, we optimize the cooperator's expected return over sampled partners with PPO (Schulman et al., 2017). The partner policies are frozen, so only π^1_ϕ is updated. Because POPGEN partners are executable Policy Code, they add negligible overhead to the training loop compared to sampling from a learned generative model or maintaining a live population of RL agents. Rather than sampling partners uniformly during training, we draw them using the rank-based prioritized scheme of MEP (Zhao et al., 2023), which samples partners that the cooperator performs poorly with at higher probability, preventing the cooperator from overfitting to easy-to-coordinate partners while neglecting hard ones (Appendix B.1).

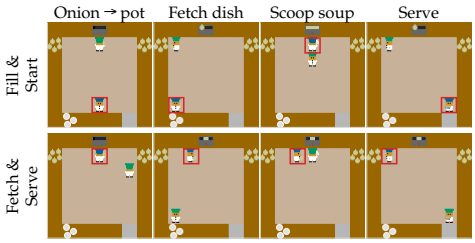
5 EXPERIMENTS

We evaluate POPGEN as a method for generating synthetic populations to train cooperators. After describing the setup (Section 5.1), we test whether cooperators trained against POPGEN populations generalize to novel partners (Section 5.2), analyze how POPGEN produces semantically diverse agents and why this helps cooperation (Section 5.3), and compare policy code against LLM-generated reward functions for implementing behaviors (Section 5.4).

5.1 EXPERIMENTAL SETUP

Environments. We consider four multi-agent environments with discrete and continuous action spaces. In Point-Mass Rendezvous (`PMR`) (Lowe et al., 2017), two agents must agree on which of four equally valid landmarks to meet at. In Level-Based Foraging (`LBF`) (Albrecht & Ramamoorthy, 2015), agents must time their load actions to jointly collect food whose level exceeds any single agent's. In Predator-Prey (`PP`) (Lowe et al., 2017), agents choose between capturing low-reward rabbits alone and coordinating to capture high-reward stags. In `Overcooked` (Carroll et al., 2019), two chefs must divide subtasks (filling pots with onions, cooking, serving) to deliver soups. Details are in Appendix A.1.

LLM Models. We employ Gemini-3-Flash as the generator model and adherence classifier, and Gemini-3-Pro as the feedback advisor (see Figure 1) in our experiments (Google DeepMind, 2025). We use Gemini-Embedding-001 to obtain behavior embeddings, which we use to measure semantic diversity.



(a) Rendered rollouts of POPPLAY cooperator (red box) paired with a partner. POPPLAY switches roles based on the partner's behavior.

Method	Onion → pot	Start pot	Scoop soup	Serve soup
POPPLAY	26% (2.7)	26% (2.7)	27% (2.8)	21% (2.2)
Self-Play	0% (0.0)	0% (0.0)	56% (1.8)	44% (1.4)
FCP	0% (0.0)	0% (0.0)	62% (1.2)	38% (0.7)
MEP	9% (0.2)	13% (0.3)	52% (1.1)	26% (0.5)
CoMeDi	68% (6.1)	21% (1.9)	6% (0.5)	5% (0.5)

(b) Share of each subtask completed by the cooperator, averaged over the four held-out partners, with the subtask count per episode in parentheses.

Method	PMR	LBF	PP	Overcooked
POPPLAY (ours)	$\mathbf{1.41 \pm 0.89}$	$\mathbf{0.76 \pm 0.16}$	$\mathbf{1.33 \pm 0.13}$	$\mathbf{31.2 \pm 44.6}$
Self-Play (SP)	0.00 ± 0.00	0.71 ± 0.11	1.04 ± 0.14	0.0 ± 0.0
FCP	0.00 ± 0.00	0.73 ± 0.08	1.07 ± 0.14	0.0 ± 0.0
MEP	0.62 ± 0.57	0.68 ± 0.14	0.98 ± 0.09	2.3 ± 3.7
CoMeDi	1.40 ± 0.52	0.49 ± 0.08	0.64 ± 0.21	11.2 ± 0.5

(c) Worst-case team reward against held-out partners. For each run, we report the reward with the held-out partner on which the cooperator scores lowest (mean ± std over five runs).

Figure 3: **Cooperator role adaptation in `overcooked` and robustness to held-out partners.** (a) Rollouts show POPPLAY taking semantically diverse behaviors interpretable to humans. (b) POPPLAY spreads its actions across subtasks while the baselines tend to focus on a subset of actions. Per-partner counts are in Table 8. (c) POPPLAY achieves the highest worst-case reward in all environments.

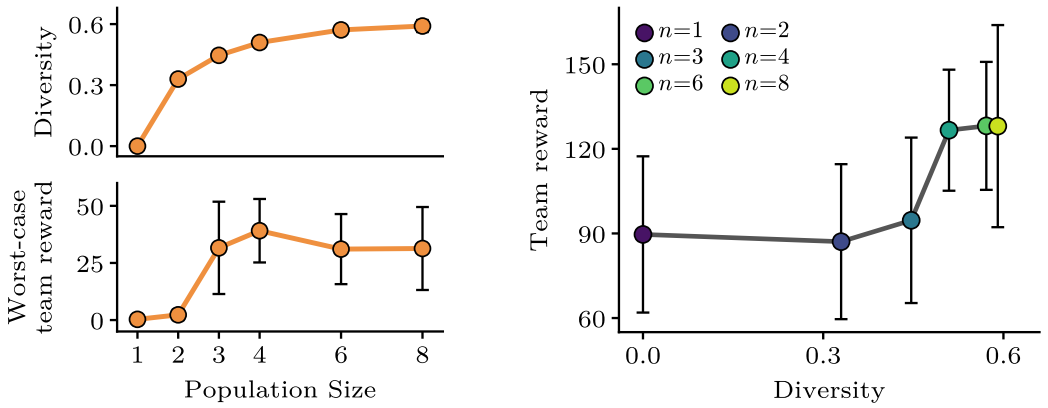
Cooperator training. We compare POPPLAY, which trains a cooperator via population play against POPGEN partners, with four population-play baselines. *Self-Play (SP)* (Strouse et al., 2021) trains MAPPO self-play agents from independent seeds. *Fictitious Co-Play (FCP)* (Strouse et al., 2021) adds their intermediate checkpoints to cover a range of skill levels. *Maximum Entropy Population-based Training (MEP)* (Zhao et al., 2023) diversifies the population with a population-entropy objective and trains the cooperator with prioritized partner sampling. *CoMeDi* (Sarkar et al., 2023) adds partners that minimize cross-play reward with the existing population, encouraging incompatible conventions, and trains the cooperator with an auxiliary behavior-cloning loss on each partner's self-play rollouts. All cooperators except CoMeDi share the same PPO (Schulman et al., 2017) backbone and number of training frames. Appendices B.1 and B.3 detail the baselines and their hyperparameters in our setup.

ZSC evaluation. We evaluate each cooperator's zero-shot coordination by pairing it with *held-out partners* that were not encountered during training. These are fixed, scripted agents with semantically distinct behaviors, specified in advance without LLMs. The held-out behaviors are never provided to the LLMs during our experiments, preventing POPGEN from being biased toward the evaluation behaviors. Appendix B.2 lists the held-out partners for each environment.

5.2 TRAINING COOPERATORS WITH POPGEN POPULATIONS

We train a cooperator against a POPGEN population (referred to as POPPLAY), and compare how well it coordinates with held-out partners against the baselines. We then analyze the resulting trajectories in `overcooked` to see how each cooperator divides the work with its partner.

Generalization to novel partners. Figure 2 reports the average team reward across different held-out partners of each method (mean and standard deviation over five runs). POPPLAY achieves the highest mean return in every environment, clearly outperforming the baselines in `LBF` and `PP`, while attaining a higher mean return than the strongest baseline, CoMeDi, in `PMR` and `overcooked`, with overlapping error bars. This consistently high mean performance indicates robust generalization across held-out partners. To further assess robustness to individual partners, Figure 3c reports the team reward when each cooperator is paired with its worst-performing held-out partner. POPPLAY



(a) Diversity of the training pool and worst-case team reward as a function of population size.

(b) Team reward with held-out partners as a function of training pool diversity.

Figure 4: **Scaling the POPGEN population in `overcooked`.** POPPLAY cooperators trained against pools of n distinct POPGEN agents at a fixed budget. (a) Larger populations are more diverse and raise both worst-case and average (b) team reward with held-out partners, with gains plateauing beyond $n = 4$. Error bars show the standard error over runs.

maintains the highest worst-case rewards, whereas Self-Play and FCP fall to zero reward with their worst partners in `PMR` and `overcooked`. Table 5 reports the breakdown of average team reward for each partner.

POPPLAY adapts its role to the partner. We examine whether POPPLAY can show semantically different behaviors when paired with different partners. `overcooked` provides a natural setting for this analysis, as it requires a cooperator to handle different subtasks according to the partner chef's preference. Figure 3a illustrates this behavior by showing the *same* POPPLAY cooperator paired with different partners. When the partner fetches onions, POPPLAY takes the dish and serves the soup. When the partner takes the dish, POPPLAY fetches onions and cooks instead. Figure 3b shows the distribution of subtasks completed by the cooperator. The POPPLAY cooperator performs all subtasks with substantial count, while other methods almost never load the onion (Self-Play, FCP, and MEP) or are biased toward single actions (CoMeDi on pot loading). In particular, Self-Play and FCP exhibit a severe mode collapse in population generation where their generated populations predominantly focus on onion fetching. As a result, the corresponding cooperators rarely need to fetch onions themselves, leading to the absence of onion-fetching behavior with held-out partners. We study these failures in detail in Appendix B.5, highlighting the importance of behavioral diversity in the training population for learning robust partner-adaptive behavior.

5.3 GENERATING DIVERSE POPULATIONS FOR COOPERATOR TRAINING

Diversity in training agents is the core element driving cooperator performance as it allows the cooperator to adapt to a wide range of partners at test time (Lupu et al., 2021; Bettini et al., 2024a). In this section, we evaluate the diversity of POPGEN populations. First, we investigate how diversity scales with POPGEN population size, and measure its downstream impact on the cooperator's ability to adapt to new partners. We then evaluate how our generation method targets high semantic diversity while maintaining the quality of the behaviors (i.e., the quality-diversity trade-off introduced in Section 4.1).

Larger populations for more diversity, and more robustness. We train `overcooked` cooperators against POPGEN populations with $n \in \{1, 2, 3, 4, 6, 8\}$ distinct behaviors. The training budget is fixed for fair comparison, so every pool contains 8 partners obtained by replicating the n distinct agents (i.e., $n = 1$ corresponds to 8 copies of a single partner). For each size, we run the experiment 5 times (i.e., we generate 5 independent populations and train one cooperator on each), so the variation we report covers the population generation.

As shown in Figure 4a, semantic diversity increases with larger POPGEN population sizes. It grows from 0 at $n = 1$ to 0.51 at $n = 4$, and saturates afterwards as the first behaviors already cover most of the environment. These gains in diversity naturally translate into better cooperator performance with new partners (Bettini et al., 2024a), with an inflection point suggesting that the population captures the diversity needed to represent most aspects of `overcooked` (Figure 4b, $+34\%$ performance when adding the 4th agent). More importantly, larger populations make the cooperator more robust. We measure it as the reward with the held-out partner each cooperator coordinates with least (Figure 4a, bottom). With $n \leq 2$, every cooperator fails completely with at least one held-out partner (worst-case reward around 0 in all runs), whereas from $n = 3$ onwards, worst-case reward significantly jumps (up to 39 on average). This last result means that a larger population is more likely to include a partner resembling any given unseen one, so the cooperator is less likely to face a partner it has never learned to complement.

Preserving training agents' quality while increasing diversity. Naively pushing for diversity tends to erode quality, as introduced in Section 4.1. Preserving quality matters for cooperator training, as incoherent or degenerate behaviors provide a noisy signal and can lead the cooperator to learn compensatory strategies that are useless, or even harmful, when paired with realistic partners (Liang et al., 2024). We evaluate POPGEN's ability to preserve a correct level of quality in agent behaviors while aiming for semantic diversity. We compare POPGEN against three baselines. *DIR* generates each behavior independently, with no knowledge of previous generations, and thus cannot reach high diversity but serves as a quality oracle. *Verbalized Sampling (VS)* (Zhang et al., 2025) is a method designed to preserve quality when generating diverse outputs, and was evaluated on standard NLP tasks (e.g., QA, story generation). Finally, *SemDiv* (Li et al., 2025) is the pipeline closest to ours: it generates textual behaviors sequentially, conditioning each new behavior on all previous ones. For each method, we generate a set of 30 behaviors for each environment listed in Section 5.1, and measure *diversity* and *quality*[4] as defined in Section 4.1.

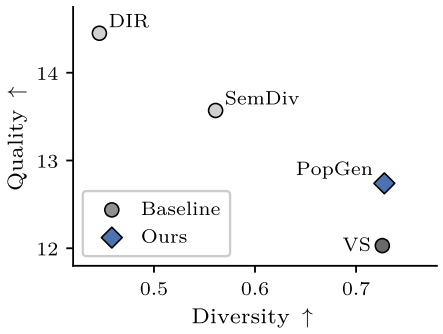


Figure 5: **Diversity and quality of POPGEN population agents.** Quality and diversity of a generated set of 30 behaviors, averaged across the environments. POPGEN matches the diversity of VS while improving quality, and is substantially more diverse than DIR and SemDiv.

Figure 5 presents the results. POPGEN achieves the best trade-off overall. It reaches the highest diversity (0.728), on par with VS (0.726), while producing clearly better behaviors at that diversity level (12.74 vs. 12.03, a 6% improvement in quality). Compared to SemDiv, POPGEN gives up 6% of quality for a 30% gain in diversity, which is considerable for the task. As expected, DIR obtains the highest quality (14.45) but by far the lowest diversity (0.446), with POPGEN being 63% more diverse.

5.4 MODELING AGENT BEHAVIORS AS POLICY CODE FOR POPULATION PLAY

In its second stage, POPGEN uses an LLM to turn each textual behavior description into an executable *policy* that can act in the environment (see Figure 1). In this section we ablate our approach, comparing POPGEN's approach that uses an LLM to model behaviors using executable code that scripts agent action (which we refer to as Policy Code), to SemDiv (Li et al., 2025), which queries the LLM to craft a reward function to train a neural network agent following expected behavior (which we refer to as Reward Function). For both methods, we start from the same set of textual behaviors, so that we avoid a potential confounder from stage 1. We measure the performance as the team reward when the cooperator is paired with new partners. Additionally, we compute the training population average adherence to their behaviors, as introduced in Section 4.2, to measure how well each method performs at the task and link this with downstream cooperation.

[4]We employ Gemini-3-Pro as the judge for quality.

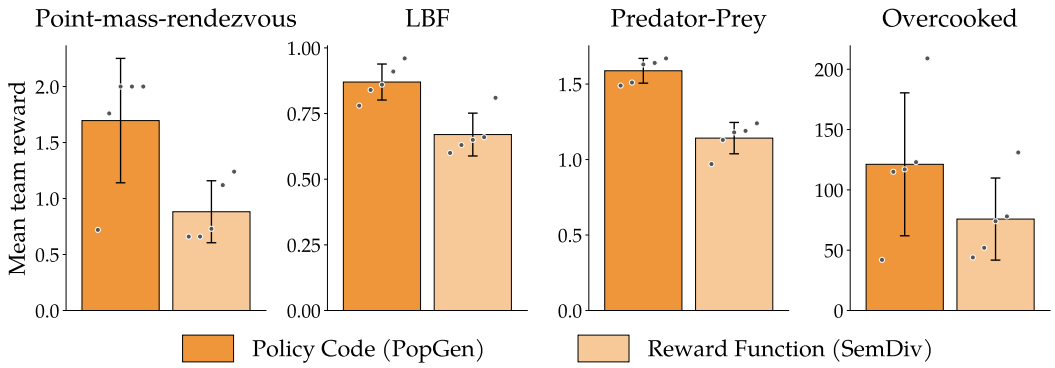


Figure 6: **Population play with policy code and reward-function partners.** Mean team reward against held-out partners of cooperators whose training partners are modeled as executable code (like POPGEN) or as neural network policies trained with RL on LLM-generated reward functions (like SemDiv). The set of behaviors is kept identical, so that only agent modeling method changes. Error bars give the standard deviation across five runs, and dots mark individual runs.

Figure 6 shows that cooperators trained with Policy Code partners (i.e., as in POPGEN) achieve higher reward against held-out partners in all four environments. Regarding adherence, Policy Code partners achieve an average adherence score of 0.71 ± 0.11, compared with 0.55 ± 0.30 for Reward Function partners across all environments. Consistent with the pattern observed in Table 11, the adherence gap is larger in the discrete environments: Policy Code and Reward Function achieve scores of 0.66 and 0.40 respectively in `LBF`, and 0.70 and 0.23 in `Overcooked`. In summary, Policy Code partners exhibit the intended behaviors more reliably in the discrete environments, and lead to more robust cooperators in all four.

6 CONCLUSION

In this work, we explore *semantic* diversity, rather than trajectory-level statistics, as a basis for training agents that cooperate robustly with unseen partners. We introduce POPGEN, which constructs diverse populations by first generating diverse behaviors in text space and then translating them into executable agents through advisor-guided refinement. We show that cooperators trained against POPGEN populations generalize to new partners, outperforming population-play baselines in all four environments studied. We further study the link between semantic diversity and zero-shot cooperation, and show that training against semantically diverse partners improves the cooperator's worst-case performance by exposing it to a wide range of strategies. We also provide insights on how to preserve the quality of the generated agents (e.g., keeping behaviors realistic) while optimizing for diversity. Finally, we show that implementing behaviors as executable policy code, rather than as LLM-generated reward functions, yields partners that follow their intended behaviors more faithfully, particularly in discrete environments, and leads to more robust cooperators.

Limitations. Our work focuses on increasing *behavior coverage*, but we evaluate only against synthetic partners. Prior work reports that training against human-like populations transfers to coordination with humans (Carroll et al., 2019; Liang et al., 2024), but a human study remains necessary to confirm that the semantic diversity of POPGEN also improves human-AI coordination. In addition, because POPGEN relies on LLMs for both behavior and policy generation, the generated populations, and hence cooperator performance, may vary across runs.

Future work. Two directions follow naturally. First, *online adaptive behavior generation* would move behavior generation inside the cooperator training loop, so that POPGEN can target the cooperator's current weaknesses (Zhao et al., 2023). Second, the quality of the refinement loop is bounded by the context provided to the LLM. Advances in context engineering (Mei et al., 2025; Khattab et al., 2024; Agrawal et al., 2026) and in code LLMs and coding agents (Chen et al., 2021; Rozière et al., 2024; Kimi, 2026) offer a direct path to improving both the advisor's feedback and the generator's code.

AI USE STATEMENT

In this work, we used generative AI tools to propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, support qualitative and thematic data analysis, and interpret results. We have not used generative AI tools to help develop theoretical models or conceptual frameworks, formulate mathematical claims, or provide critical ingredients for proving mathematical claims, and generating synthetic data sets, assisting in the writing of proofs, assisting with translation, and cleaning and reformatting datasets are not applicable to this work. Additionally, we used generative AI tools to create or modify scientific figures or images, suggest experimental parameters, create or edit software code, brainstorm, edit a research paper to improve readability, and format references. We have reviewed all AI-assisted work. LLM-generated code and figures were verified by the authors, and the authors manually checked all AI-assisted paper writing. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

The behavior generation and refinement pipeline is specified in Section 4, with prompts detailed in Appendix E. Environments are described in Appendix A.1, and held-out evaluation partners, cooperator baselines, and hyperparameters in Appendix B. The per-environment numbers behind Figure 2 are tabulated in Appendix B.4. Anonymized source code is provided as supplementary material.

REFERENCES

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Reincarnating reinforcement learning: Reusing prior computation to accelerate progress. In *Advances in Neural Information Processing Systems*, 2022.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. Gepa: Reflective prompt evolution can outperform reinforcement learning, 2026. URL https://arxiv.org/abs/2507.19457.

Sanghyun Ahn, Wonje Choi, Junyong Lee, Jinwoo Park, and Honguk Woo. Towards reliable code-as-policies: A neuro-symbolic framework for embodied task planning. *Advances in Neural Information Processing Systems*, 38:75428–75459, 2026.

Stefano V. Albrecht and Subramanian Ramamoorthy. A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems, 2015. URL https://arxiv.org/abs/1506.01170.

Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4): 819–840, 2002.

Matteo Bettini, Ryan Kortvelesy, and Amanda Prorok. The impact of behavioral diversity in multi-agent reinforcement learning. *arXiv preprint arXiv:2412.16244*, 2024a.

Matteo Bettini, Amanda Prorok, and Vincent Moens. Benchmarl: Benchmarking multi-agent reinforcement learning. *Journal of Machine Learning Research*, 25(217):1–10, 2024b.

Matteo Bettini, Ajay Shankar, and Amanda Prorok. System neural diversity: Measuring behavioral heterogeneity in multi-agent learning, 2024c. URL https://arxiv.org/abs/2305.02128.

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*, 2019.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Seoyeon Choi, Kanghyun Ryu, Jonghoon Ock, and Negar Mehr. Craft: Coaching reinforcement learning autonomously using foundation models for multi-robot coordination tasks. *arXiv preprint arXiv:2509.14380*, 2025.

Benjamin Ellis, Jonathan Cook, Skander Moalla, Mikayel Samvelyan, Mingfei Sun, Anuj Mahajan, Jakob N. Foerster, and Shimon Whiteson. Smacv2: An improved benchmark for cooperative multi-agent reinforcement learning, 2023. URL https://arxiv.org/abs/2212.07489.

Letian Fu, Justin Yu, Karim El-Refai, Ethan Kou, Haoru Xue, Huang Huang, Wenli Xiao, Li Fei-Fei, Guanya Shi, Jiajun Wu, S. Shankar Sastry, Yuke Zhu, Ken Goldberg, and Linxi Fan. Cap-x: A framework for benchmarking and improving coding agents for robot manipulation. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=4JRO9plGAI.

Google DeepMind. Gemini 3 pro model card. Technical report, Google, November 2025. URL https://storage.googleapis.com/deepmind-media/Model-Cards/Gemini-3-Pro-Model-Card.pdf.

Hengyuan Hu, Adam Lerer, Alex Peysakhovich, and Jakob Foerster. "other-play" for zero-shot coordination. In *International Conference on Machine Learning (ICML)*, pp. 4399–4410. PMLR, 2020.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning*, 2022.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into self-improving pipelines. In *International Conference on Learning Representations*, 2024.

Kimi. Kimi k3: Open frontier intelligence, 2026. URL https://arxiv.org/abs/2607.24653.

Lihe Li, Lei Yuan, Pengsen Liu, Tao Jiang, and Yang Yu. LLM-assisted semantically diverse teammate generation for efficient multi-agent coordination. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 36743–36764. PMLR, 13–19 Jul 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 *IEEE International Conference on Robotics and Automation (ICRA)*, 2023.

Yancheng Liang, Daphne Chen, Abhishek Gupta, Simon S. Du, and Natasha Jaques. Learning to cooperate with humans using generative agents. *arXiv preprint arXiv:2411.13934*, 2024.

Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, 2017.

Andrei Lupu, Brandon Cui, Hengyuan Hu, and Jakob Foerster. Trajectory diversity for zero-shot coordination. In *International Conference on Machine Learning (ICML)*, 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 *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, et al. Self-refine: Iterative refinement with self-feedback. *Advances in Neural Information Processing Systems*, 2023.

Kevin R McKee, Joel Z Leibo, Charlie Beattie, and Richard Everett. Quantifying the effects of environment and population diversity in multi-agent reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 36(1):21, 2022.

Lingrui Mei et al. A survey of context engineering for large language models. *arXiv preprint arXiv:2507.13334*, 2025.

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

Justin K Pugh, Lisa B Soros, and Kenneth O Stanley. Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40, 2016.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024. URL https://arxiv.org/abs/2308.12950.

Bidipta Sarkar, Andy Shih, and Dorsa Sadigh. Diverse conventions for human-AI collaboration. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

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

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research*, 2023.

Ren-Jian Wang, Ke Xue, Yutong Wang, Peng Yang, Haobo Fu, Qiang Fu, and Chao Qian. Diversity from human feedback, 2025. URL https://arxiv.org/abs/2310.06648.

Yen-Jen Wang, Bike Zhang, Jianyu Chen, and Koushil Sreenath. Prompt a robot to walk with large language models. In *2024 IEEE 63rd conference on decision and control (CDC)*, pp. 1531–1538. IEEE, 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*, 2022.

Justin Wong, Yury Orlovskiy, Michael Luo, Sanjit A. Seshia, and Joseph E. Gonzalez. SimpleStrat: Diversifying language model generation with stratification. *arXiv preprint arXiv:2410.09038*, 2024.

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 *International Conference on Learning Representations*, 2024.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023.

Yida Yin, Zekai Wang, Yuvan Sharma, Dantong Niu, Trevor Darrell, and Roei Herzig. In-context learning enables robot action prediction in llms. In *2025 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 8972–8979. IEEE, 2025.

Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of PPO in cooperative multi-agent games. In *Advances in Neural Information Processing Systems*, 2022.

Hugh Zhang, Daniel Duckworth, Daphne Ippolito, and Arvind Neelakantan. Trading off diversity and quality in natural language generation, 2020. URL https://arxiv.org/abs/2004.10450.

Jiayi Zhang, Simon Yu, Derek Chong, Anthony Sicilia, Michael R. Tomz, Christopher D. Manning, and Weiyan Shi. Verbalized sampling: How to mitigate mode collapse and unlock LLM diversity. *arXiv preprint arXiv:2510.01171*, 2025.

Rui Zhao, Jinming Song, Hu Haifeng, Yang Gao, Yi Wu, Zhongqian Sun, and Yang Wei. Maximum entropy population-based training for zero-shot human-ai coordination. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2023.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

APPENDIX

A POPGEN DETAILS

A.1 ENVIRONMENTS

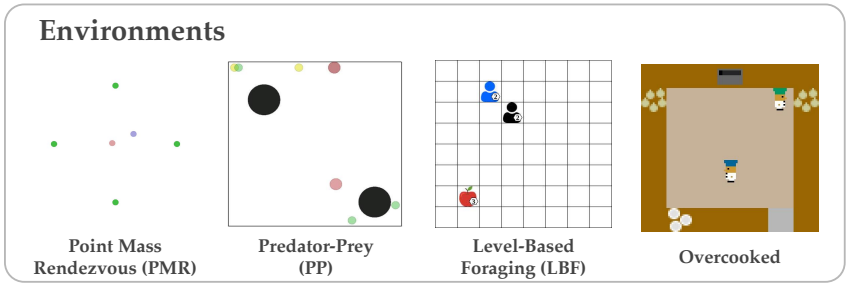


Figure 7: **Environments used for cooperator training.** From left to right, `PMR` requires agents to navigate toward and converge on the same landmark. `PP` requires agents to coordinate to capture prey, with different rewards between individually capturing a rabbit and jointly capturing a stag. `LBF` requires agents to combine their carrying capacities to collect food items whose levels may exceed the level of a single agent. Finally, `overcooked` requires two chefs to coordinate the sequential subtasks of preparing, cooking, and delivering soups.

Table 1: Multi-agent environments in POPGEN cover cooperative settings across discrete and continuous action spaces. We note that adversarial agents (prey) in PP follow fixed policies defined by the environment, so only the ego team is generated and trained.

Env	Num. Agents	Action Space	Observation	Objective
LBF	2 (ego: 2)	Discrete (6)	Partial	Collect all food items
Overcooked	2 (ego: 2)	Discrete (6)	Global	Serve as many soups as possible
PMR	2 (ego: 2)	Continuous (2D)	Global	Meet at the same landmark
PP	7 (ego: 2, adv.: 5)	Continuous (2D)	Partial	Capture prey

Figure 7 and Table 1 show the environments used for cooperator training, covering different action spaces, agent composition, observability, and reward structure.

Level-Based Foraging (LBF) (Albrecht & Ramamoorthy, 2015). `LBF` is a cooperative grid-world in which agents navigate a discrete map and collect food items of varying levels. A team receives {collected food's level /total food level in the field} as a reward when they collect the food.

Overcooked (Carroll et al., 2019). `overcooked` places two chefs in a shared kitchen where they must fetch onions, load a pot, cook soup, plate it, and deliver it to the serving counter. They receive a reward for completing each subtask. A single delivery therefore chains several dependent subtasks, and either chef can perform any of them. This makes the division of labor, rather than individual skill, the main driver of team reward. We use the *large_room* layout in our experiments.

Point-Mass Rendezvous (PMR) (Lowe et al., 2017). `PMR` asks two point-mass agents to meet at the same landmark, with four landmarks placed evenly on a circle. The team receives a reward of 2 only when both agents converge on the same landmark, and every landmark is equally valid.

Predator-Prey (PP) (Lowe et al., 2017). `PP` places agents in a continuous environment alongside multiple prey types with different capture requirements. Rabbits can be caught individually and provide a reward of $+0.5$, while stags require coordinated capture by multiple predators but yield a reward of $+2$. We train a team of 2 predators while there are 3 rabbits and 2 stags in the field.

Algorithm 1 Iterative Behavior-to-Agent Refinement

Require: target behavior b, environment description s, adherence threshold α, max iteration I_{max}, rollouts per attempt M
Ensure: best candidate policy $\pi^\star$ for behavior b

$f_1 \leftarrow \emptyset$
for $i = 1, \ldots, I_{max}$ **do**
  $\pi_i \leftarrow \textsc{Generator}(b, s, f_i)$ ▷ regenerate given prior feedback
  $\{\tau_m\}_{m=1}^{M} \leftarrow \textsc{Rollout}(\pi_i, M)$
  $\mathrm{Adh}_i, \{\mathrm{rationale}_m\}_{m=1}^{M} \leftarrow \mathrm{VLM}(b, \{\tau_m\}_{m=1}^{M})$ ▷ fraction of rollouts like b
  if $\mathrm{Adh}_i \geq \alpha$ **then**
    return π_i
  end if
  $f_{i+1} \leftarrow \textsc{Advisor}(b, s, \pi_i, \{\mathrm{rationale}_m\}_{m=1}^{M})$ ▷ feedback for next attempt
end for
return $\pi_{i^\star}$ where $i^\star = \arg\max_{i \in \{1,\ldots,I_{max}\}} \mathrm{Adh}_i$ ▷ return best attempt

A.2 Iterative Refinement for Adherence

Evaluating adherence to behavior. We estimate the adherence score in Equation 3 with a VLM-as-a-judge that observes the trajectory and sampled frames from the rendered rollout and assigns a binary adherence label. The prompts passed to the VLM are available in Appendix E. When we roll out a candidate policy to measure adherence, we pair the generated agent with a fixed scripted partner. In `LBF`, the partner moves to the closest food item and loads it. In `PP`, it follows the scripted predator policy of Lowe et al. (2017), which avoids solo attempts and moves in to coordinate a capture when a stag is present. In `overcooked`, we use the oracle planning policy provided in the `overcooked_ai` repository (Carroll et al., 2019). In `PMR`, it selects a landmark uniformly at random and moves toward it.

Feedback loop. Algorithm 1 details the refinement loop employed in PopGen. Given a target behavior b and environment description s, the Generator proposes a candidate policy π_i, which is executed for M rollouts. A VLM judge inspects each rollout, deciding whether it exhibits b and giving a rationale; the adherence score Adh_i is the fraction of rollouts judged consistent with b. If $\mathrm{Adh}_i \geq \alpha$, the policy is accepted immediately. Otherwise, an Advisor turns the judge's rationales, together with b, s, and the current policy, into targeted feedback f_{i+1}, and the generator regenerates the policy conditioned on that feedback. The loop runs for at most I_{max} iterations; if no attempt reaches the threshold, the highest-adherence policy is returned.

Advisor. The advisor analyzes rollout evaluation results and produces structured feedback that guides the next refinement attempt. The prompts are shown in Appendix E.

B Cooperator Training

B.1 Baselines

Every baseline follows the two-stage population play recipe. The first stage builds a partner population $\Pi = \{\pi_1, \ldots, \pi_K\}$, and the second stage trains a cooperator π_ϕ against it. The baselines differ mainly in how the first stage produces the partners, which is the stage PopGen replaces with language-guided policy code generation. Throughout this section we write $J(\pi, \pi')$ for the expected discounted team return of the pair (π, π'), so $J(\pi, \pi)$ is a *self-play* return.

Self-Play (SP). Self-Play trains one policy by pairing it with copies of itself, so run k starting from seed ξ_k solves $\pi_k = \arg\max_\pi J(\pi, \pi)$. We launch K independent MAPPO runs and keep their final checkpoints,

$$\Pi_{\mathrm{SP}} = \{\pi_1, \ldots, \pi_K\}, \tag{4}$$

so the only source of diversity is the random initialization and the sampling noise of each run. Runs that converge to the same convention leave the population nearly degenerate, and the cooperator then inherits the brittleness of a single self-play agent.

Fictitious Co-Play (FCP). FCP reuses the same self-play runs and adds their intermediate checkpoints, on the grounds that a cooperator should also handle partners of lower skill. Let $\pi_{k,t}$ denote run k after t gradient steps, with T the final step. FCP retains C checkpoints per run at steps $t_1 < \cdots < t_C = T$, which yields

$$\Pi_{\text{FCP}} = \{\pi_{k,t_c}\}_{k=1,\ldots,K}^{c=1,\ldots,C}, \qquad |\Pi_{\text{FCP}}| = KC. \tag{5}$$

We use $K = 4$ runs and $C = 3$ checkpoints per run, sampled evenly from the early, middle, and final stages of training.

Maximum Entropy Population-based Training (MEP). MEP trains the K partners jointly and rewards them for acting differently from one another. Let $\bar{\pi}(a \mid o) = \frac{1}{K}\sum_{k=1}^{K} \pi_k(a \mid o)$ be the mean population policy. Each partner maximizes the team return augmented with the entropy of this mean policy,

$$\max_{\pi_1,\ldots,\pi_K} \sum_{k=1}^{K} \mathbb{E}_{\tau\sim(\pi_k,\pi_k)} \left[\sum_t \gamma^t \big(r_t + \eta\, \mathcal{H}\big(\bar{\pi}(\cdot \mid o_t)\big)\big) \right], \tag{6}$$

where η weights the population entropy bonus. We use $\eta = 0.001$ for LBF and PMR and $\eta = 0.005$ for PP and Overcooked. This bonus decomposes into an individual and a collective term,

$$\mathcal{H}(\bar{\pi}) = \underbrace{\frac{1}{K}\sum_{k=1}^{K} \mathcal{H}(\pi_k)}_{\text{stochasticity of each partner}} + \underbrace{\text{JSD}(\pi_1, \ldots, \pi_K)}_{\text{disagreement between partners}}, \tag{7}$$

so maximizing it rewards both randomness within a partner and divergence between partners.

The second stage prioritizes partners that the cooperator performs poorly with by replacing $\text{Uniform}(\Pi)$ in Equation 1 with the rank-based prioritized sampling scheme. Specifically, each partner is sampled with probability

$$p(\pi_k) = \frac{\text{rank}(1/J(\pi_\phi, \pi_k))^\beta}{\sum_{j=1}^{K} \text{rank}(1/J(\pi_\phi, \pi_j))^\beta}, \tag{8}$$

where rank assigns 1 to the partner with the highest return and K to the partner with the lowest, and β controls the strength of the prioritization. During training, we estimate $J(\pi_\phi, \pi_k)$ as the mean episodic return from the most recent rollouts with π_k, and sample a partner from p at the start of each episode. POPPLAY uses the same sampling scheme, but with a population of POPGEN-generated policy code.

CoMeDi. CoMeDi builds the population one partner at a time and encourages each new partner to adopt a convention that is distinct from those already represented. Suppose $\pi_1, \ldots, \pi_{k-1}$ have been trained. For a candidate π, we identify the existing partner with which it achieves the highest return,

$$j^\star(\pi) = \arg\max_{j<k} \; J(\pi, \pi_j), \tag{9}$$

and treat this partner as the candidate's nearest existing convention. The new partner then maximizes its own self-play return while minimizing its return when paired with this nearest neighbor,

$$\pi_k = \arg\max_{\pi} \; J(\pi, \pi) \; - \; \lambda_{\text{XP}}\, J\big(\pi, \pi_{j^\star(\pi)}\big) \; + \; \lambda_{\text{MP}}\, J_{\text{MP}}\big(\pi, \pi_{j^\star(\pi)}\big). \tag{10}$$

The mixed-play return J_{MP} prevents the candidate from achieving a low paired return simply by failing deliberately. Its rollout initially pairs π with $\pi_{j^\star}$, then switches at a random step $t \sim \text{Uniform}(\{0, \ldots, T\})$ to using π for both agents, and scores only the reward collected after the switch. Maximizing this return encourages π to remain competent in states reached when paired with the existing partner. Thus, a low $J(\pi, \pi_{j^\star})$ reflects a genuinely different convention rather than deliberate failure when paired with an unfamiliar partner. We use $\lambda_{\text{XP}} = 0.25$ and $\lambda_{\text{MP}} = 1.0$ from the original implementation.

The CoMeDi cooperator. CoMeDi also replaces the best response of Equation 1 with a multi-task objective in the style of QDagger (Agarwal et al., 2022). Its convention-aware cooperator $\hat{\pi}$ minimizes

$$\mathcal{L}(\hat{\pi}, \Pi) = -J(\hat{\pi}, \hat{\pi}) \; - \; \frac{\lambda}{|\Pi|} \sum_{\pi \in \Pi} \mathbb{E}_{(o,a)\sim\pi}\big[\log \hat{\pi}(a \mid o)\big], \tag{11}$$

where $(o, a) \sim \pi$ denotes observation-action pairs drawn from self-play rollouts of π, and λ weights the behavior-cloning term. The demonstrations come from a fixed population, so this cooperator only covers the conventions that Π already contains.

B.2 Held-out Evaluation Partners

The held-out partners are agents that are never seen during training and are paired with cooperators only at evaluation time. We design four partners per environment. Table 2 lists the partners for each environment.

Env.	Name	Description
LBF	Closest-Food Greedy	Moves to the nearest food and loads it when possible.
	Team-Center Food	Targets food near the center of the two agents.
	Teammate-Biased Food	Targets the food cheapest for the teammate to reach.
	Synchronized Loading	Waits at its food until the teammate is adjacent.
PP	Closest-Prey Greedy	Chases the nearest prey, whether it is a rabbit or a stag.
	Teammate-Biased Prey	Targets the prey the teammate is also pressuring.
	Rabbit Only	Always pursues the nearest rabbit, never a stag.
	Stag Only	Pursues stags only, so it scores only if the partner joins.
Overcooked	Fill only	Only fetches onions and loads the pot.
	Fetch & Serve	Only fetches a dish and delivers soup.
	Fill & Start	Loads onions and starts the pot, but never serves.
	Wait & Serve	Waits at the pot until the soup is cooked, then serves.
PMR	Closest-Landmark Greedy	Moves toward the currently nearest landmark.
	Farthest-Landmark Commitment	Always moves toward the currently farthest landmark.
	Teammate-Biased Landmark	Selects the landmark closest to the teammate.
	Regroup then Diverge	Approaches the teammate, then heads to a far landmark.

Table 2: Held-out evaluation partners for each environment: Level-Based Foraging (LBF), Predator-Prey (PP), Overcooked, and Point-Mass Rendezvous (PMR).

B.3 Hyperparameters

We train every cooperator with PPO (Schulman et al., 2017) on a recurrent actor and critic. We keep the partner policies frozen so that only the cooperator is updated. Table 3 reports the hyperparameters. Training populations contain 4 partners for PopPlay, Self-Play, MEP, and CoMeDi and 12 for FCP, which pools three checkpoints from each of four self-play runs. In the population generation stage, baseline populations are trained with MAPPO (Yu et al., 2022), since their agents learn jointly.

Table 3: Cooperator training hyperparameters. The upper block lists the settings shared by every environment and every method. The lower block lists the per-environment settings.

Shared across all environments	
Discount factor γ	0.99
Learning rate	10^{-4}
PPO clip ϵ	0.2
GAE λ	0.9
Policy network	GRU
Prioritized sampling β	1.0

Parameter	**LBF**	**Overcooked**	**PMR**	**PP**	**Football**
Environment frames	4M	4M	30M	80M	100M

B.4 Results Table

Table 4: **Average team reward of each cooperator against evaluation and training partners.** We report reward against held-out *eval* behaviors and against the *train* partners the cooperator saw during training. We report the reward averaged across partners (mean $\pm$ standard deviation over five runs). Best value per environment is in **bold**.

Method	**Partners**	PMR	LBF	PP	Overcooked
Self-Play (SP)	Eval	1.01 ± 0.25	0.81 ± 0.04	1.21 ± 0.12	63.38 ± 65.37
	Train	1.60 ± 0.22	0.78 ± 0.02	1.36 ± 0.14	133.79 ± 151.41
FCP	Eval	0.93 ± 0.31	0.80 ± 0.03	1.27 ± 0.09	40.53 ± 52.23
	Train	$\mathbf{1.68 \pm 0.30}$	0.80 ± 0.02	1.31 ± 0.19	157.29 ± 134.16
MEP	Eval	1.45 ± 0.46	0.72 ± 0.14	1.30 ± 0.17	33.79 ± 37.73
	Train	1.32 ± 0.17	0.65 ± 0.30	1.31 ± 0.32	126.05 ± 104.18
CoMeDi	Eval	1.56 ± 0.39	0.59 ± 0.09	1.02 ± 0.13	86.33 ± 31.48
	Train	1.51 ± 0.15	0.48 ± 0.14	0.88 ± 0.13	$\mathbf{312.96 \pm 75.00}$
PopPlay (ours)	Eval	$\mathbf{1.70 \pm 0.56}$	$\mathbf{0.87 \pm 0.07}$	$\mathbf{1.59 \pm 0.08}$	$\mathbf{120.88 \pm 59.25}$
	Train	1.64 ± 0.45	$\mathbf{0.94 \pm 0.03}$	$\mathbf{1.71 \pm 0.29}$	211.66 ± 122.13

Table 4 reports the per-environment numbers behind Figure 2. We report two different setups: one against the held-out eval partners and the other against the train partners. PopPlay attains the highest eval reward in every environment. The reward difference between eval and train partners measures how much of its reward is lost due to out-of-distribution partners.

Table 5 breaks the eval row of Table 4 down to the individual partner. We read each row as the partner picking the cooperator it scores highest with. Fourteen of the 16 partners picked PopPlay as the most cooperative agent, and the remaining two, both in `overcooked`, picked CoMeDi.

Table 5: **Team reward against each held-out partner.** We report mean ± standard error of the team reward across 5 runs. For every partner, **bold** marks the best cooperator and every method whose error bar overlaps it. The final row counts the partners for which a method attains the highest average.

Held-out partner	POPPLAY (ours)	SP	FCP	MEP	CoMeDi
Point-Mass Rendezvous (PMR)					
Closest-Landmark Greedy	**1.58**$_{\pm 0.26}$	0.82$_{\pm 0.08}$	0.92$_{\pm 0.16}$	1.01$_{\pm 0.02}$	**1.46**$_{\pm 0.21}$
Farthest-Landmark Commitment	**1.60**$_{\pm 0.40}$	**0.80**$_{\pm 0.49}$	0.40$_{\pm 0.40}$	**1.20**$_{\pm 0.49}$	**1.45**$_{\pm 0.22}$
Teammate-Biased Landmark	**2.00**$_{\pm 0.00}$	**2.00**$_{\pm 0.00}$	**2.00**$_{\pm 0.00}$	**1.99**$_{\pm 0.01}$	1.93$_{\pm 0.04}$
Regroup then Diverge	**1.60**$_{\pm 0.40}$	0.40$_{\pm 0.40}$	0.40$_{\pm 0.40}$	**1.60**$_{\pm 0.40}$	**1.42**$_{\pm 0.23}$
Level-Based Foraging (LBF)					
Closest-Food Greedy	**0.76**$_{\pm 0.07}$	**0.72**$_{\pm 0.05}$	**0.73**$_{\pm 0.04}$	**0.71**$_{\pm 0.07}$	0.49$_{\pm 0.03}$
Team-Center Food	**0.90**$_{\pm 0.02}$	0.82$_{\pm 0.02}$	0.81$_{\pm 0.01}$	0.71$_{\pm 0.06}$	0.62$_{\pm 0.04}$
Teammate-Biased Food	**0.92**$_{\pm 0.01}$	0.87$_{\pm 0.01}$	0.86$_{\pm 0.02}$	0.77$_{\pm 0.05}$	0.66$_{\pm 0.04}$
Synchronized Loading	**0.90**$_{\pm 0.02}$	0.84$_{\pm 0.02}$	0.82$_{\pm 0.01}$	0.70$_{\pm 0.07}$	0.60$_{\pm 0.04}$
Predator-Prey (PP)					
Closest-Prey Greedy	**1.35**$_{\pm 0.06}$	1.06$_{\pm 0.06}$	**1.28**$_{\pm 0.05}$	1.16$_{\pm 0.10}$	1.01$_{\pm 0.03}$
Teammate-Biased Prey	**1.81**$_{\pm 0.12}$	1.43$_{\pm 0.07}$	1.39$_{\pm 0.05}$	1.48$_{\pm 0.16}$	1.20$_{\pm 0.11}$
Rabbit Only	**1.38**$_{\pm 0.06}$	1.21$_{\pm 0.04}$	1.21$_{\pm 0.02}$	1.05$_{\pm 0.02}$	1.24$_{\pm 0.03}$
Stag Only	**1.81**$_{\pm 0.11}$	1.15$_{\pm 0.08}$	1.22$_{\pm 0.13}$	1.53$_{\pm 0.14}$	0.64$_{\pm 0.09}$
Overcooked					
Fill only	**36**$_{\pm 18}$	12$_{\pm 0}$	12$_{\pm 0}$	**17**$_{\pm 4}$	11$_{\pm 0}$
Fetch & Serve	53$_{\pm 29}$	0$_{\pm 0}$	0$_{\pm 0}$	2$_{\pm 2}$	**141**$_{\pm 18}$
Fill & Start	**341**$_{\pm 66}$	**242**$_{\pm 117}$	150$_{\pm 93}$	113$_{\pm 70}$	81$_{\pm 44}$
Wait & Serve	53$_{\pm 29}$	0$_{\pm 0}$	0$_{\pm 0}$	2$_{\pm 2}$	**112**$_{\pm 20}$
Best cooperator	**14**/16	0/16	0/16	0/16	2/16

B.5 MODE COLLAPSE IN SELF-PLAY POPULATIONS

Self-Play populations built from independent seeds often collapse to a few conventions. To show this, we pair each self-play policy with itself and record which strategy the pair follows. In `PMR`, different seeds can converge to different landmarks, but each policy commits to a single landmark, and a population of four covers only 2.8 of the four landmarks on average (Table 6). Since Self-Play has no mechanism that promotes diversity across seeds, several seeds can settle on the same landmark, and all four policies in Run 5 settle on the same one. In `overcooked`, self-play pairs mostly put onions into the pot and rarely fetch a dish or scoop soup (Table 7). Cooperators trained against these populations therefore rarely see partners that take the other roles. FCP suffers from the same collapse, since its checkpoints follow the same semantic behaviors as the self-play runs and differ only in skill level.

Table 6: **Self-Play in Point-Mass Rendezvous.** Each column is one self-play policy paired with itself. We run 5 self-play experiments, each producing 4 policies from different seeds, and evaluate each policy for 20 episodes. Rows give the percentage of episodes in which the pair meets at each landmark, and the largest in each column is in bold. Counting each policy's most frequent landmark, only Run 1 covers all four landmarks, and all four policies in Run 5 settle on West, since Self-Play has no principled way of promoting diversity across seeds.

Landmark (%)	Run 1				Run 2				Run 3				Run 4				Run 5			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
East	0	0	0	**100**	35	**100**	0	0	0	0	0	0	**85**	0	0	**100**	0	0	0	0
North	**100**	15	0	0	0	0	**75**	0	0	0	**100**	0	15	0	30	0	0	0	0	0
West	0	0	**75**	0	0	0	0	0	**100**	0	0	0	0	0	**70**	0	**100**	**100**	**70**	**100**
South	0	**85**	25	0	**65**	0	25	**100**	0	**100**	0	**100**	0	**100**	0	0	0	0	30	0

Table 7: **Self-Play in Overcooked.** Each column is one self-play policy paired with itself. We run 5 self-play experiments, each producing 4 policies from different seeds, and evaluate each policy for 20 episodes. Rows give each pipeline stage's share of all stage actions performed by both agents (%), and the largest in each column is in bold.

Stage (%)	Run 1				Run 2				Run 3				Run 4				Run 5				
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	Mean
Fetch onion	**37**	**40**	**51**	**38**	**58**	**51**	**40**	**54**	**41**	**49**	**35**	**46**	**43**	**42**	**57**	**34**	**51**	**47**	**35**	**39**	44
Onion→pot	31	36	38	35	38	36	35	39	33	28	33	45	33	35	34	34	38	42	35	36	36
Start pot	9	12	0	12	0	0	12	0	8	8	11	0	11	10	0	10	0	0	12	12	6
Fetch dish	11	12	11	8	4	13	13	7	11	8	11	10	11	13	9	10	10	11	10	12	10
Scoop soup	11	0	0	8	0	0	0	0	8	8	11	0	2	0	0	10	0	0	10	1	3
Serve	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

B.6 Task Event Counts in `overcooked`

Table 8 breaks down the subtask completion shares in `overcooked` from Figure 3b by held-out partner, reporting the partner's counts alongside those of the cooperator. PopPlay shifts toward subtasks left open by the partner, whereas Self-Play, FCP, and MEP remain focused on plating and show little activity when the partner already occupies those subtasks.

Table 8: **Cooperator task events per held-out partner in `overcooked`.** Per-partner breakdown of Figure 3b. Fill & Start and Fill only load the pot but never serve. Fetch & Serve and Wait & Serve plate and serve but never fetch an onion. Counts are averaged per episode over 50 episodes.

		Cooperator				
Method	**Partner**	Onion → pot	Start pot	Fetch dish	Scoop soup	Serve soup
PopPlay	Fill & Start	0.23	0.22	10.62	10.35	8.14
	Fill only	0.23	0.78	2.34	0.74	0.72
	Fetch & Serve	5.18	4.85	0.36	0.02	0.00
	Wait & Serve	5.20	4.85	0.41	0.03	0.00
Self-Play	Fill & Start	0.00	0.00	7.96	7.16	5.55
	Fill only	0.00	0.00	22.47	0.00	0.00
	Fetch & Serve	0.00	0.00	0.05	0.00	0.00
	Wait & Serve	0.00	0.00	0.05	0.00	0.00
FCP	Fill & Start	0.00	0.00	5.51	4.84	2.96
	Fill only	0.00	0.00	1.28	0.00	0.00
	Fetch & Serve	0.00	0.00	0.27	0.00	0.00
	Wait & Serve	0.00	0.00	0.27	0.00	0.00
MEP	Fill & Start	0.03	0.32	4.63	3.76	2.05
	Fill only	0.03	0.37	1.24	0.37	0.08
	Fetch & Serve	0.32	0.18	0.23	0.04	0.00
	Wait & Serve	0.31	0.18	0.23	0.05	0.00
CoMeDi	Fill & Start	0.59	0.06	2.90	1.93	1.91
	Fill only	0.54	0.07	1.89	0.03	0.03
	Fetch & Serve	12.83	4.16	0.63	0.00	0.00
	Wait & Serve	10.31	3.30	0.71	0.02	0.01

C DIVERSE BEHAVIOR GENERATION

In this section, we extensively ablate methods to generate diverse text while retaining maximum quality (Pugh et al., 2016). We include two additional environments for completeness, as the following experiments were cheaper to run on it[5].

Football (Bettini et al., 2024b). The Football environment introduces a team setting in which agents must maintain positions relative to the ball, goals, and teammates while dynamically switching between offensive and defensive roles.

SMACv2 (Ellis et al., 2023). SMACv2 consists of small-scale StarCraft multi-agent combat scenarios in which allied units coordinate against a stronger enemy force under partial observability. In our setting, two allied Marines face four enemy Marines, so success depends on tactical movement, focus fire, spacing, retreat timing, and target selection.

Table 9: **Comparison of prompting methods for generating N behaviors.** k is the number of candidates per LLM call (in our experiments $k = 5$). y_i denotes the i-th generated candidate, $\hat{p}_i$ its verbalized probability, and $\pi(\cdot|x)$ the LLM's output distribution conditioned on prompt x. For Multi-Turn and VS-Multi, h_{i-1} denotes the conversation history up to turn $i-1$, and t the turn index.

Method	LLM Calls	Candidates	Turns	Prompt Example	Definition
1. Instance-level Prompt					
Direct (DIR)	N	1	1	"Think step-by-step, generate a football player behavior"	$y_i \sim \pi(y\|x_{\text{CoT}})$
2. List-level Prompt					
One-Pass (OP)	1	N	1	"Generate 30 football player behaviors"	$(y_1,\ldots,y_N) \sim \pi(y_1,\ldots,y_N\|x_{\text{OP}})$
Sequence (SEQ)	$\lceil N/k \rceil$	k	1	"Generate 5 football player behaviors"	$(y_1,\ldots,y_k) \sim \pi(y_1,\ldots,y_k\|x_{\text{seq}})$
Multi-Turn (MT)	N	1	N	Turn 1: "Generate a football player behavior" Turn 2+: "Generate another football player behavior"	$y_i \sim \pi(y\|x_{\text{multi}}, h_{i-1})$
SimpleStrat (SS) (Wong et al., 2024)	$\lceil N/k \rceil$	k	1	"Generate a football player behavior following these characteristics: [strata]"	$y_i \sim \pi(y \mid x_{\text{strat}_i}),\ \text{strat}_i \sim \hat{P}_{\text{strata}}$
3. Distribution-level Prompt					
Verbalized-Sampling (VS) (Zhang et al., 2025)	$\lceil N/k \rceil$	k	1	"Think step-by-step, generate 5 football player behaviors with probabilities"	$(y_1,\hat{p}_1),\ldots,(y_k,\hat{p}_k) \sim \pi(\cdot\|x_{\text{VS-CoT}})$
VS-MT (Zhang et al., 2025)	$\lceil N/k \rceil$	k	$\lceil N/k \rceil$	Turn 1: "Generate 5 player behaviors with probabilities" Turn 2+: "Generate 5 more with probabilities"	$\left((y_1^{(t)},\hat{p}_1^{(t)}),\ldots,(y_k^{(t)},\hat{p}_k^{(t)})\right) \sim \pi(\cdot\|x_{\text{VS}}, h_{t-1})$
VS-OS (ours)	$\lceil m/k \rceil + 1$	k	1	(VS) "Generate $m{>}N$ player behaviors with probs" (Merge) "Combine into N best behaviors"	$\{(y_i,\hat{p}_i)\}_{i=1}^{m} \sim \pi(\cdot\|x_{\text{VS}}),\ m > N$ $\tilde{y}_{1:N} = f_\pi(\{(y_i,\hat{p}_i)\}_{i=1}^{m})$

Diverse text generation methods. Table 9 presents the generation methods we evaluate, organized by the level at which diversity is encouraged. *Instance-level* methods (Direct) generate one behavior per independent LLM call with chain-of-thought reasoning (Wei et al., 2022), relying solely on sampling stochasticity for diversity. *List-level* methods ask the model to produce multiple behaviors in a single output: One-Pass generates all N behaviors at once, Sequence generates them in batches of k, and Multi-Turn uses conversation history so that each new behavior is conditioned on previously generated ones. SimpleStrat (Wong et al., 2024) first asks the LLM to identify meaningful diversity axes (strata) for the environment, then samples behaviors conditioned on randomly selected strata combinations. *Distribution-level* methods build on Verbalized Sampling (VS) (Zhang et al., 2025), which prompts the model to output behaviors together with verbalized probabilities, encouraging it to sample from the tails of its distribution rather than collapsing to the mode. VS-MT extends this across conversation turns, accumulating history. Finally, we propose *VS-Oversample* (VS-OS), which generates $m > N$ candidate behaviors with verbalized probabilities, then uses a second LLM call to merge and deduplicate them into the N best behaviors. This two-step approach decouples diversity exploration (oversampling) from quality control (merging), allowing broader coverage without sacrificing coherence. Figure 8 compares all methods, of which Section 5.3 reports a subset: POPGEN corresponds to VS-OS, SemDiv to MT, and DIR and VS to themselves.

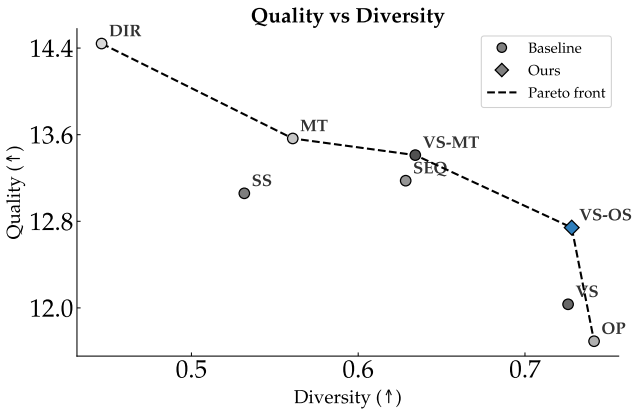


Figure 8: **Quality and diversity of all behavior generation methods.** Quality and diversity of a generated set of 30 behaviors, averaged across the four environments, for every method in Table 9. VS-OS is the method used in POPGEN.

Table 10: **Quality and diversity of oversampling variants across environments.** Diversity is measured as average pairwise embedding distance; quality as total score. Best value per column in **bold**. The Sequential $\times 2$ variant — referred to as `VS-OS` throughout the paper — achieves the best diversity–quality trade-off overall.

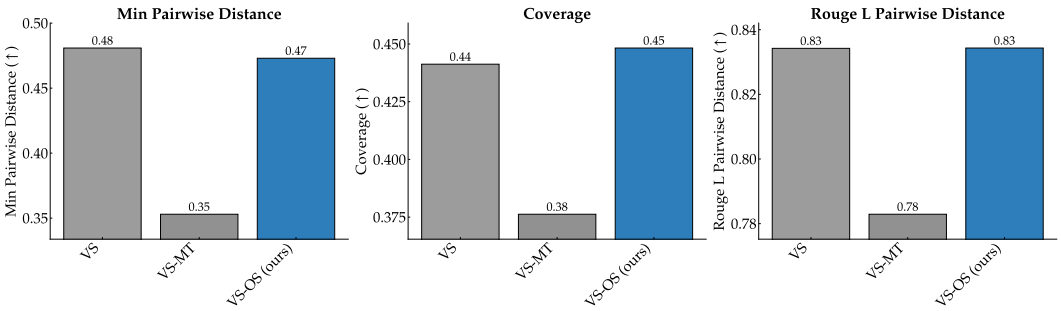
		Diversity (↑)							Quality (↑)						
Type	**Ratio**	football	lbf	pmr	pp	smacv2	overcooked	avg	football	lbf	pmr	pp	smacv2	overcooked	avg
Baseline (VS)	×1	0.730	**0.726**	0.636	0.691	**0.756**	0.612	0.692	12.77	11.16	13.52	12.20	12.00	**13.32**	12.50
One Pass	×2	0.737	0.710	0.645	0.725	0.752	0.634	**0.701**	12.47	12.24	13.78	11.90	12.44	13.30	12.69
	×4	0.729	0.684	0.644	0.718	0.747	**0.684**	**0.701**	12.50	12.18	13.84	13.04	12.57	13.20	12.89
Independent	×2	0.737	0.654	0.637	0.735	0.738	0.616	0.686	12.44	11.97	13.84	12.93	12.20	13.26	12.77
	×4	0.726	0.672	0.643	0.721	0.749	0.614	0.688	**13.01**	**12.57**	**13.88**	12.77	12.11	13.10	12.91
Sequential	×2	0.733	0.703	**0.649**	**0.737**	0.740	0.613	0.696	**13.01**	11.97	13.78	**13.14**	**12.87**	13.20	**13.00**
	×4	**0.738**	0.685	0.640	0.719	0.694	0.594	0.678	12.84	12.36	13.74	12.66	12.80	13.22	12.94

C.1 OVERSAMPLING TO PRESERVE QUALITY IN BEHAVIOR GENERATION

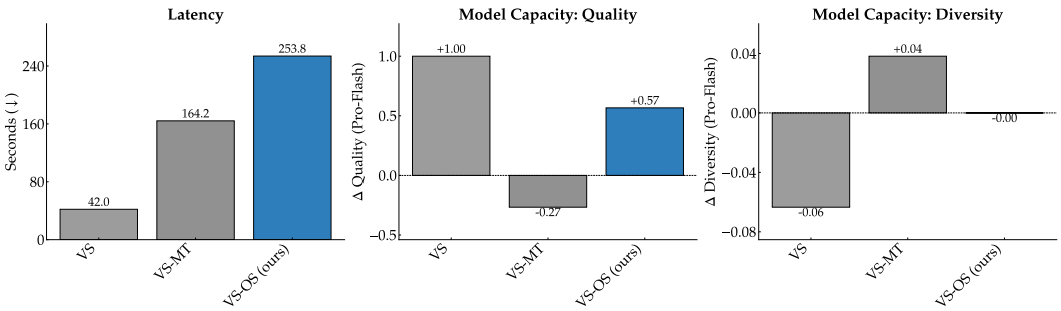
Oversampling ablation. Table 10 ablates how we oversample, comparing One-Pass, Independent, and Sequential strategies at ratios $\times 2$ and $\times 4$ against vanilla VS ($\times 1$). All variants improve quality over the VS baseline while maintaining comparable diversity, confirming that the merging step filters low-quality candidates without collapsing diversity. Sequential $\times 2$ achieves the best overall balance, reaching the best average quality (13.00) while staying within 0.005 of the highest average diversity (0.696 against 0.701), and we adopt it as the default VS-OS configuration throughout the paper. Increasing the ratio to $\times 4$ leaves diversity essentially unchanged for One-Pass and Independent while lowering it for Sequential, suggesting that an overly large candidate pool gives the merging step too much freedom to prune unusual behaviors in favor of safe ones.

Diversity analysis and scaling of VS-OS. Figure 9 refines the comparison between VS-OS and its single-pass counterpart. Both achieve comparable lexical dissimilarity, but they diverge on embedding-level measures: VS-OS recovers the worst-case (minimum pairwise) distance and coverage that multi-turn VS sacrifices (Figure 9a), indicating that oversample-and-merge fills gaps a single pass leaves unexplored rather than duplicating its distribution. Figure 9b shows why this matters at scale: when upgrading the generator from Flash to Pro, multi-turn VS gains quality ($+1.00$) but *loses* diversity (-0.06), whereas VS-OS translates the same upgrade into gains on *both* axes ($+0.57$ quality, $+0.04$ diversity). Because exploration and selection are separate stages, a more capable model improves each independently.

[5]For both, the main issue is rollout completion time, which make them hard to include in the rest of the paper as training is therefore quite long. Football rollouts can be quite long before anything concrete happens, and SMACv2 runs a slow StarCraft backend, which was hard to optimize for fast training.



(a) Detailed diversity metrics: minimum pairwise embedding distance, embedding space coverage, and lexical dissimilarity (1 − Rouge-L).



(b) Scaling VS-OS: generation latency in seconds (left); quality improvement Δ when scaling the generator from Flash to Pro (center); diversity gain Δ from the same scaling (right).

Figure 9: **Diversity breakdown and scaling properties of VS variants.** (a) VS-MT sacrifices worst-case diversity (minimum pairwise distance drops to 0.35) and coverage (0.38) relative to single-pass VS (0.48, 0.44): iterative refinement tends to homogenize outputs. VS-OS recovers both metrics (0.47, 0.45) by decoupling exploration from selection—oversampling casts a wide net, then a dedicated merging step curates without collapsing variety. Lexical dissimilarity is comparable across all three. (b) The decoupled design of VS-OS trades additional latency for better scaling behavior. When moving from Flash to Pro, VS-MT gains quality ($+1.00$) but *loses* diversity (-0.06), confirming that multi-turn refinement with a stronger model amplifies the homogenization effect. VS-OS, by contrast, translates the stronger model into gains on *both* axes ($+0.57$ quality, $+0.04$ diversity): because exploration and selection are separate stages, a more capable model improves the quality of each stage independently without one undermining the other.

Transfer of findings from other domains. Figure 8 includes generation methods that were tested in other domains. SimpleStrat (13.07 quality, 0.532 diversity) falls below the Pareto front, indicating that stratified partitioning, while effective in general-knowledge QA (Wong et al., 2024), does not transfer well to generating valid agent strategies where identifying meaningful diversity axes is itself hard. In contrast, the core findings of Verbalized Sampling for creative writing carry over: prompting the model to verbalize a distribution of open-ended responses consistently improves diversity across all VS variants. A consistent secondary pattern is that multi-turn interaction improves quality at the cost of diversity, as conversation history anchors subsequent generations around themes established in earlier turns.

D Modeling Behaviors in Agents with LLMs

In its second stage, PopGen turns each textual behavior description into an executable *policy* that can act in the environment (see Figure 1). These policies must adhere to their behaviors (i.e., act according to their textual descriptions), otherwise the cooperator would be taught to cooperate with noisy partners, leading to arbitrary conventions that could harm cooperation with real partners. This section ablates this part. We report agent adherence, as introduced in Section 4.2, on two splits: a *dev* set on which the LLM optimizes during refinement, and a *test* set of 10 environment initializations

Table 11: **Adherence scores of POPGEN agents across environments and splits.** We use `gemini-3-flash-preview` as the agent generator or player in LLM-A. For each method, we report first-attempt and best-attempt adherence (out of 5 attempts) along with the improvement Δ. Avg. columns show the mean across environments. Best values per method and split are in **bold**.

		Behavior Adherence (Dev)							Behavior Adherence (Test)						
Method	**Attempt**	Football	LBF	PMR	PP	SMACv2	Overcooked	Avg.	Football	LBF	PMR	PP	SMACv2	Overcooked	Avg.
LLM-A	First	38.3 ± 10.4	13.3 ± 10.4	33.3 ± 7.7	58.3 ± 10.4	43.3 ± 7.6	30.7 ± 8.0	36.2 ± 9.1	10.4 ± 1.9	25.4 ± 1.9	26.7 ± 5.8	93.3 ± 5.2	38.3 ± 7.3	30.7 ± 6.5	37.5 ± 4.8
	Best	$\mathbf{38.3 \pm 10.4}$	$\mathbf{13.3 \pm 10.4}$	$\mathbf{50.7 \pm 7.8}$	$\mathbf{76.7 \pm 10.4}$	$\mathbf{66.7 \pm 2.9}$	$\mathbf{62.7 \pm 9.3}$	$\mathbf{51.4 \pm 8.5}$	$\mathbf{10.4 \pm 1.9}$	$\mathbf{25.4 \pm 1.9}$	$\mathbf{32.7 \pm 6.6}$	$\mathbf{93.3 \pm 5.2}$	$\mathbf{45.8 \pm 4.7}$	$\mathbf{45.3 \pm 9.1}$	$\mathbf{42.2 \pm 4.9}$
	Δ	0.0	0.0	+17.4	+18.3	+23.3	+32.0	+15.2	0.0	0.0	+6.0	0.0	+7.5	+14.6	+4.7
PC	First	83.0 ± 7.6	43.0 ± 8.4	56.0 ± 8.3	95.0 ± 3.5	28.0 ± 5.7	34.7 ± 10.8	56.6 ± 7.4	63.5 ± 16.1	53.5 ± 3.9	58.0 ± 7.5	74.0 ± 8.4	26.0 ± 8.8	41.3 ± 10.5	52.7 ± 9.2
	Best	$\mathbf{99.0 \pm 2.2}$	$\mathbf{71.0 \pm 6.5}$	$\mathbf{84.0 \pm 5.9}$	$\mathbf{98.0 \pm 2.7}$	$\mathbf{68.0 \pm 6.7}$	$\mathbf{68.0 \pm 10.3}$	$\mathbf{81.3 \pm 5.7}$	$\mathbf{66.0 \pm 12.6}$	$\mathbf{61.0 \pm 5.0}$	$\mathbf{66.7 \pm 6.1}$	$\mathbf{74.0 \pm 8.4}$	$\mathbf{51.5 \pm 8.8}$	$\mathbf{63.3 \pm 9.3}$	$\mathbf{63.8 \pm 8.4}$
	Δ	+16.0	+28.0	+28.0	+3.0	+40.0	+33.3	+24.7	+2.5	+7.5	+8.7	0.0	+25.5	+22.0	+11.1
RF	First	68.0 ± 21.4	1.0 ± 2.2	50.7 ± 8.5	68.0 ± 2.7	18.0 ± 10.4	28.0 ± 11.2	39.0 ± 9.4	46.5 ± 12.1	7.5 ± 3.5	48.0 ± 7.9	64.5 ± 5.1	20.0 ± 10.6	28.7 ± 11.1	35.9 ± 8.4
	Best	$\mathbf{97.0 \pm 2.7}$	$\mathbf{6.0 \pm 5.5}$	$\mathbf{70.7 \pm 7.3}$	$\mathbf{86.0 \pm 8.9}$	$\mathbf{44.0 \pm 18.2}$	$\mathbf{42.7 \pm 11.0}$	$\mathbf{57.7 \pm 8.9}$	$\mathbf{66.5 \pm 7.4}$	$\mathbf{12.5 \pm 3.5}$	$\mathbf{49.3 \pm 7.5}$	$\mathbf{71.0 \pm 5.2}$	$\mathbf{33.0 \pm 6.2}$	$\mathbf{40.7 \pm 10.3}$	$\mathbf{45.5 \pm 6.7}$
	Δ	+29.0	+5.0	+20.0	+18.0	+26.0	+14.7	+18.7	+20.0	+5.0	+1.3	+6.5	+13.0	+12.0	+9.6

Table 12: **Behavior modeling methods.** Each method specifies a distinct mechanism for instantiating a target behavior b in an agent policy π. o is the observation of the environment, and a is the sampled action.

Name	**Abbr.**	**Description**
LLM-Embodied Agent	LLM-A	Uses an LLM directly as the decision-making agent, conditioned on textualized observations and a behavior specification: $\pi_{\text{LLM-A}} \sim \text{LLM}(o, b)$
Reward Function	RF	An LLM generates a reward function $r_b : (o, a) \to \mathbb{R}$, whose optimization via reinforcement learning yields a neural network policy π_{RF} exhibiting the target behavior.
Policy Code	PC	An LLM synthesizes an executable policy $f_b : o \to a$ that directly maps observations to actions, serving as the policy.

never observed by the refinement loop. The former evaluates how well the LLM refines policies across attempts, while the latter controls for overfitting.

D.1 BEHAVIOR MODELING METHODS

We compare different methods to emulate a behavior in a policy. Table 12 presents them.

Performance. Across all environments, Policy Code (PC) achieves the highest average adherence on both the dev set (81.3%) and the test set (63.8%). The advantage is most pronounced in discrete-action environments: on `LBF`, PC reaches 71.0% dev adherence versus 6.0% for RF, and on `smacv2`, 68.0% versus 44.0%. Because RF operates at the parameter level, small changes in the reward signal often fail to discriminate between a limited set of discrete actions, whereas PC directly specifies decision logic. LLM-A achieves high adherence on `PP` (93.3% test) but fails on longer-horizon environments such as `football` (10.4% test) and `LBF` (25.4% test).

Table 13: **Average generation and rollout latency per method.** Generation latency covers code synthesis (PC) or RL training (RF); rollout latency is for a single episode. LLM-A has no generation step (—) but incurs high rollout cost from per-timestep LLM inference. Best per column in **bold**. `smacv2` is excluded due to engine overhead dominating latencies.

Method	**Generation latency (s)** $\downarrow$	**Rollout latency (s)** $\downarrow$
LLM-A	—	110.1 ± 1.5
RF	2878.6 ± 239.5	11.7 ± 0.6
PC	$\mathbf{8.1 \pm 0.2}$	$\mathbf{9.8 \pm 1.9}$

Latency. Latency becomes crucial when generating diverse agent populations at scale. We compare the latency of the different methods, including both generation of the modeling artifact and

Table 14: **Policy iterative refinement.** Improvement in adherence (first- vs. best-attempt, out of 5 attempts) on the dev and test splits. Avg. columns show the mean across environments. Best values per split are in **bold**.

Split	Attempt	LBF	PMR	PP	Overcooked	Avg.
Dev	First	43.0 ± 8.4	56.0 ± 8.3	95.0 ± 3.5	34.7 ± 10.8	57.2 ± 7.8
	Best	$\mathbf{71.0 \pm 6.5}$	$\mathbf{84.0 \pm 5.9}$	$\mathbf{98.0 \pm 2.7}$	$\mathbf{68.0 \pm 10.3}$	$\mathbf{80.3 \pm 6.4}$
	Δ	$+28.0$	$+28.0$	$+3.0$	$+33.3$	$+23.1$
Test	First	53.5 ± 3.9	58.0 ± 7.5	74.0 ± 8.4	41.3 ± 10.5	56.7 ± 7.6
	Best	$\mathbf{61.0 \pm 5.0}$	$\mathbf{66.7 \pm 6.1}$	$\mathbf{74.0 \pm 8.4}$	$\mathbf{63.3 \pm 9.3}$	$\mathbf{66.3 \pm 7.2}$
	Δ	$+7.5$	$+8.7$	0.0	$+22.0$	$+9.6$

rollout time. Table 13 presents the results. PC is the most practical method. It requires only 8.1s to generate the script and takes 9.8s per rollout on average across the environments, orders of magnitude faster than RF which runs a full policy training (2878.6s generation), and cheaper at inference than LLM-A (110.1s rollout).

D.2 Impact of refinement loop on PopGen agents adherence

Refinement loop to improve adherence. We compare the adherence scores of PopGen's final policies (best of 5 attempts with refinement) against those of the first-attempt policies. The difference between the two represents improvements that can be attributed to PopGen's refinement loop. Table 14 reports the results. PopGen's iterative refinement of policies substantially improves the adherence ($+23.1$ points on average across environments on the *dev* set). The results transfer to the unseen initializations of the *test* set, with the average adherence rising from 56.7% to 66.3% ($+9.6$ points). `PP` is the exception: the adherence on *dev* is already near ceiling on the first attempt (95.0%), leaving almost no room for refinement to improve on.

Feedback examples. Box D.2 contrasts the first refinement step on `LBF` with and without the advisor. The two edits are similar in size ($+98/-72$ vs. $+67/-78$ lines), but they differ in kind. With the advisor, the generator makes structural fixes aimed at coordination. It corrects an inverted North/South mapping, scores food using the teammate's motion vector, and moves to a free cell adjacent to the target so both agents can load. It also commits to a target for three steps to avoid oscillation. Without the advisor, the generator patches symptoms. It adds a Load action that never checks whether the teammate is adjacent, and its fallback moves the agent toward the teammate rather than the food. It also computes the teammate's heading but never uses it. The raw rollouts thus signal that teammate motion matters, but without the advisor's diagnosis the generator cannot act on it.

Advisor Diff (trunc.) — `attempt_0` → `attempt_1` (+98 / -72 lines)

```
@@ movement direction mapping
-    if abs(dr) >= abs(dc) and abs(dr) > 0:
-        return torch.tensor(1) if dr > 0 else torch.tensor(2)
+    action = 2 if dr > 0 else 1  # SOUTH if dr>0, NORTH if dr<0
+    return torch.tensor(action)

@@ target food scoring via teammate motion vector
-    closest_dist = float('inf')
-    for f in foods:
-        dist = abs(f['r'] - team_pos[0]) + abs(f['c'] - team_pos[1])
+    def food_score(food):
+        f_dr = food[0] - team_row
+        f_dc = food[1] - team_col
+        return (abs(f_dr - team_dr) + abs(f_dc - team_dc),
↪ abs(f_dr), abs(f_dc))
+    best_food = sorted(foods, key=food_score)[0] if foods else None

@@ support-cell selection (closest cardinal cell, not
↪ teammate-occupied)
+    for d_row, d_col in [(-1,0),(1,0),(0,-1),(0,1)]:
+        cand_r, cand_c = target_row + d_row, target_col + d_col
+        if not (cand_r == team_row and cand_c == team_col):
+            valid_positions.append((cand_r, cand_c))
+    best_pos = min(valid_positions,
+                   key=lambda p:
↪ abs(p[0]-self_row)+abs(p[1]-self_col))

@@ target commitment window
+    if not hasattr(policy, 'target_countdown'):
+        policy.target_countdown = 3
+    if policy.target_countdown == 0:
+        policy.target_food = best_food
+        policy.target_countdown = 3
```

No-Advisor Diff (trunc.) — `attempt_0` → `attempt_1` (+67 / -78 lines)

```
@@ opportunistic LOAD without teammate adjacency check
+    if abs(target_r - self_r) <= 1 and abs(target_c - self_c) <= 1:
+        if food0_l > 0 and (self_l + team_l >= food0_l):
+            return torch.tensor(5)

@@ fallback: move toward teammate rather than support cell
+    dr_to_t = team_r - self_r
+    dc_to_t = team_c - self_c
+    if dr_to_t != 0:
+        return torch.tensor(2 if dr_to_t > 0 else 1) \
+               if abs(dr_to_t) > abs(dc_to_t) \
+               else torch.tensor(3 if dc_to_t > 0 else 4)

@@ heading estimation computed but never used in food selection
+    dr = policy.team_pos[-1][0] - policy.team_pos[-2][0]
+    dc = policy.team_pos[-1][1] - policy.team_pos[-2][1]
+    new_dir = 1 if dr > 0 else -1  # stored but unused
```

E PROMPTS

E.1 ADVISOR

Advisor — System Prompt

```
You are an advisor for a code generation pipeline.
Your task is to diagnose why a generated code fails to represent its
target behavior in rollouts and provide actionable feedback to
↪  improve
the next code attempt.

You will be given:
- The intended behavior
- The environment description
- The code attempt (type: {generation_type})
- Evaluation results from multiple rollout executions.

Focus on the most important failure patterns and give specific,
actionable suggestions to fix the code.
```

Advisor — User Prompt

```
Code generation type: {generation_type}

Inputs
--------------
## Behavior
{behavior}

## Environment Description
{environment_description}

## Code Attempt
```
{code_attempt}
```

## Rollout Evaluation Results
{metrics_str}
--------------
Analyze the results and provide feedback for the next code attempt.
```

E.2 ADHERENCE EVALUATOR

Adherence Evaluator — System Prompt

```
You are an expert evaluator of whether an agent's actions match a given
behavior description.
You will be given:
- a behavior description
- agent rollouts (trajectory states and possibly image sequences)

Your task is to determine whether the ego agent's actions adhere to the
behavior description.

# Guidelines:
- Evaluate alignment with the described behavior only.
- Do not consider any other possible goals.
- Focus only on the ego agent.
- Consider interactions with other agents only when relevant to behavior
  following or when they affect the ego agent's ability to demonstrate
  the behavior.
- Use states as the primary source of truth and images as supporting
evidence.
- When serialized states are present, interpret them literally and use them
  as the main basis for factual claims.

In rare cases, the agent may be unable to fully demonstrate the expected
behavior due to environment limitations or teammate actions. If the ego
agent still makes a clear, reasonable, good-faith effort to follow the
behavior, you may mark it as adherent. You must explicitly explain the
constraint and justify the decision in the reasoning field.

# Environment Description
{environment_description}

# Output Format
Return a JSON object with exactly these fields:
- "evaluation": a boolean indicating whether the rollout demonstrates
  the behavior
- "reasoning": a detailed explanation of why the rollout does or does
  not demonstrate the behavior, including the single most decisive reason
  for the verdict and specific actions or states when relevant
```

Adherence Evaluator — User Prompt

```
Behavior:
{behavior}

Rollout (showing {N} serialized sampled states and {M} sampled images
in chronological order; step numbers are meaningful):
------------------------------------------------------------
[image_1] ... [image_M]          # interleaved chronologically
[state_1]
[state_2]
...
[state_N]
------------------------------------------------------------
```

E.3 BEHAVIOR QUALITY JUDGE

System Prompt

```
You are an expert in evaluating the quality of agent behavior
descriptions.  You will be provided with a behavior description and
an environment description.  Your task is to evaluate the quality
of the behavior across three rubrics:

1.  **Relevance** (0-5):  Is the behavior relevant to the
environment?
- 0:  Completely irrelevant, behavior has nothing to do with the
environment or its objectives
- 1:  Mostly irrelevant, only tangentially related to the
environment
- 2:  Somewhat relevant, addresses the environment but misses key
aspects
- 3:  Moderately relevant, relates to the environment but could
apply to many other contexts
- 4:  Highly relevant, clearly designed for this environment with
appropriate context
- 5:  Perfectly relevant, deeply tied to the environment's unique
characteristics and objectives

2.  **Realism** (0-5):  Could a real player plausibly play this
way, given the environment's constraints?
- 0:  Violates hard environment constraints (e.g., impossible
actions, broken physics)
- 1:  Technically allowed but contradicts the environment so
fundamentally that no human would ever play this way
- 2:  Extremely unlikely -- requires bizarre intent or
misunderstanding, but not strictly impossible
- 3:  Unusual but conceivable -- a small fraction of players (even
0.01%) might genuinely adopt this style
- 4:  Plausible -- an uncommon but recognizable play style that
some players would intentionally choose
- 5:  Natural -- a common or intuitive way to play that many
players would independently arrive at

3.  **Specificity** (0-5):  Is the behavior specific enough to be
actionable?
- 0:  Extremely vague, no actionable guidance (e.g., "play well")
- 1:  Very generic, could mean almost anything (e.g., "be
strategic")
- 2:  Somewhat vague, general direction but lacking concrete
details
- 3:  Moderately specific, provides guidance but leaves room for
interpretation
- 4:  Specific, clear behavior and conditions with minor
ambiguities
- 5:  Highly specific, precise behavior, triggers, and success
criteria that leave no ambiguity

Provide detailed reasoning for each score.
```

User Prompt

```
Environment Description:
{environment_description}

Behavior:
{behavior}

Please evaluate this behavior's quality across the three rubrics
(relevance, realism, specificity).
```