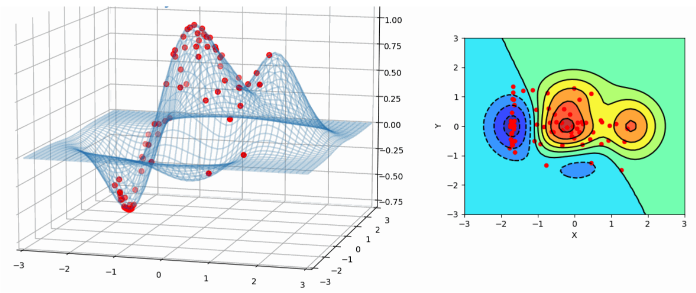
# Particle Swarm Optimization

## Learning Swarm Intelligence with RL

Collective Intelligence - Spring 2026

Teams of 2-3 students | 6-8 weeks



## Overview

This project explores **learned particle swarm optimization (PSO)** using multi-agent reinforcement learning. Instead of hand-tuning hyperparameters like inertia weight $\omega$, cognitive coefficient $c_1$, and social coefficient $c_2$, agents learn to dynamically adjust these parameters based on the current optimization landscape.

Your task is to extend the base implementation with **one of three research directions** below. Each task involves implementing new functionality, running rigorous experiments, and documenting results with visualizations and analysis.

**What is already provided**

- `src/envs/env.py` — TorchRL-based multi-agent PSO environment
- `src/envs/dynamic_functions.py` — Static and dynamic benchmark functions (Sphere, Rastrigin, Eggholder, etc.)
- `src/main.py` — Hydra-based training loop with PPO
- `src/eval.py` — Evaluation with 2D/3D visualization and GIF generation
- `src/visualizer.py` — SwarmVisualizer class for particle animation
- Docker containerization for reproducibility

**Teams & Timeline**

- **Teams**: 2-3 students per team
- **Duration**: 6-8 weeks (start early!)
- **Checkpoint 1** (Week 3): Implementation outline + preliminary results
- **Checkpoint 2** (Week 6): Working code + initial experiments
- **Final** (Week 8): Full submission + presentation

# Task Options

Each team selects **one task**. All tasks have comparable scope and grading weight.

## Task 1: Communication Topologies & Information Flow

**Research Question**: How do different swarm communication topologies (global best, local best, dynamic neighborhoods) affect convergence and diversity in learned PSO?

### Hypothesis

Restricting information flow through local topologies (ring, von Neumann, dynamic k-nearest) will maintain diversity longer, improving performance on multimodal landscapes, while global best topology converges faster on unimodal functions.

### Implementation Requirements

1. **Topology System** (`src/envs/topology.py` – new file):
    - Implement: Global Best (gBest), Local Best (lBest) with ring topology
    - Implement: Von Neumann (grid) neighborhood, k-Nearest dynamic neighbors
    - Topology abstraction: Topology.get_neighbors(particle_id) interface
    - Allow topology changes during optimization (learned or scheduled)
2. **Environment Extension** (`src/envs/env.py` modification):
    - Add topology-aware observation: include neighbor personal bests
    - Modify reward to use neighborhood best instead of global best
    - Config-driven topology selection via Hydra
3. **Diversity Metrics** (`src/eval/diversity.py` – new file):
    - Position diversity: mean pairwise distance, convex hull volume
    - Velocity diversity: velocity alignment coefficient
    - Information spread: how fast does best solution propagate?

### Technical Guidance

- Use adjacency matrices for efficient neighbor queries
- Dynamic k-nearest: recompute neighbors every N steps (configurable)
- Configs: `src/configs/topology/` with global.yaml, ring.yaml, knearest.yaml
- Consider graph neural network observations for topology-aware policies

### Expected Deliverables

- Working topology system with 4+ topology types
- Diversity metrics integrated into evaluation pipeline
- Comparison: 4 topologies × 3 landscapes (unimodal, multimodal, dynamic) × 5 seeds
- 4-6 plots: convergence curves, diversity over time, final fitness distributions, topology comparison heatmaps
- 2-3 GIFs: side-by-side topology comparison on same function
- Analysis: when does local topology help? Trade-offs between exploration and exploitation?

---

## Task 2: Curriculum Learning & Adaptive Difficulty

**Research Question**: Can progressively increasing optimization difficulty (dimensionality, function complexity, landscape dynamics) improve sample efficiency and generalization?

### Hypothesis

Starting training on simpler problems (low dimensions, unimodal functions) and gradually introducing complexity will produce policies that generalize better to hard problems compared to training directly on difficult tasks.

### Implementation Requirements

1. **Curriculum Manager** ( `src/training/curriculum.py` – new file):
    - Dimension curriculum: train 2D → 5D → 10D → 30D
    - Function curriculum: Sphere → Rosenbrock → Rastrigin → Eggholder
    - Dynamics curriculum: static → slow dynamics → fast dynamics
    - Progression criteria: performance threshold, training steps, or learned
2. **Generalization Testing** ( `src/eval/generalization.py` – new file):
    - Zero-shot transfer: evaluate on unseen dimensions/functions
    - Performance gap: curriculum-trained vs direct-trained on hard problems
    - Learning curves: sample efficiency comparison
3. **Adaptive Difficulty** (bonus extension):
    - Automatic difficulty adjustment based on agent performance
    - Multi-armed bandit for curriculum stage selection
    - Self-paced learning: agents request harder problems

### Technical Guidance

- Dimension-agnostic policy: use attention or recurrent networks for variable input sizes
- Function encoding: one-hot, learned embeddings, or meta-learning
- Configs: `src/configs/curriculum/` with dimension.yaml, function.yaml, combined.yaml
- Consider domain randomization as baseline comparison

### Expected Deliverables

- Working curriculum system with 3+ curriculum types
- Generalization evaluation across unseen problems
- Comparison: curriculum vs direct training on final tasks (5 seeds each)
- 4-6 plots: learning curves, generalization matrices, curriculum progression, sample efficiency
- 2-3 GIFs: policy behavior on easy vs hard problems
- Analysis: optimal curriculum ordering? When does curriculum help/hurt? Transfer limits?

---

## Task 3: Diversity-Preserving Reward Shaping

**Research Question**: Can auxiliary rewards for swarm diversity prevent premature convergence and improve global optimization on deceptive landscapes?

### Hypothesis

Adding intrinsic motivation rewards for maintaining swarm diversity (position spread, velocity diversity, exploration bonuses) will improve performance on multimodal and deceptive functions where standard PSO suffers from premature convergence.

### Implementation Requirements

1. **Diversity Rewards** (`src/envs/diversity_reward.py` – new file):
    - Position entropy: reward for spread in search space
    - Novelty bonus: reward for visiting unexplored regions (archive-based)
    - Velocity alignment penalty: discourage all particles moving same direction
    - Scalarization: $r = \alpha \cdot r_{\texttt{fitness}} + (1 - \alpha) \cdot r_{\texttt{diversity}}$
2. **Adaptive Weighting**:
    - Schedule: high diversity weight early, decay over optimization
    - Performance-based: increase diversity when stuck, decrease when improving
    - Learned: let the policy learn to balance via multi-objective heads
3. **Premature Convergence Detection** (`src/eval/convergence_analysis.py`):
    - Detect: when does swarm collapse? Distance to centroid metrics
    - Compare: baseline vs diversity-rewarded policies
    - Measure: escape rate from local optima

### Technical Guidance

- Position entropy: $H = -\sum_i p_i \log p_i$ over discretized search space bins
- Novelty archive: store visited positions, reward distance to k-nearest archive points
- Configs: `src/configs/reward/` with baseline.yaml, diversity.yaml, adaptive.yaml
- Test on deceptive functions: Rastrigin, Schwefel, multimodal custom landscapes

### Expected Deliverables

- 3+ diversity reward mechanisms with configurable weights
- Adaptive weighting system (at least 2 strategies)
- Comparison: baseline vs diversity rewards on 4+ functions (5 seeds each)
- 4-6 plots: diversity over time, convergence detection, $\alpha$ sensitivity, local optima escape rates
- 2-3 GIFs: baseline (premature convergence) vs diversity-rewarded (sustained exploration)
- Analysis: optimal diversity-fitness trade-off? Function-dependent tuning? Failure modes?

---

## Shared Requirements (All Tasks)

All teams must follow these common guidelines to ensure quality and reproducibility.

**Code Quality**
- Follow existing code structure (TorchRL patterns, Hydra configs)
- Update relevant folder READMEs documenting new modules
- Add 2-3 unit tests for core functionality
- Hydra configs for all experiments

**Experiments**
- Run on both **low-dimensional** (2D-5D) and **high-dimensional** (10D-30D) problems
- Test on multiple function types: unimodal (Sphere), multimodal (Rastrigin), dynamic
- Use fixed seeds for reproducibility (report all seeds in README)
- Minimum 5 seeds per experimental condition

**Documentation**
- Update root README.md with new "Semester Contribution" section:
  - Research question & hypothesis
  - Implementation summary (what was added/modified)
  - Key results (embed 2-3 key plots/GIFs)
  - Conclusions & limitations
  - Future work
- Keep it concise (500-800 words max)

## Grading Rubric

**Total: 100 points**

Each task has the same grading structure. All requirements must use Hydra configs and be reproducible.

### Implementation (40 pts)

- Core functionality working (new modules/classes follow codebase conventions) **20 pts**
- Integration with existing system (configs, training loops, evaluation) **10 pts**
- Code quality (documentation, tests, readable) ........................... **10 pts**

### Experiments (30 pts)

- Rigorous experimental design (controls, baselines, multiple seeds) ... **10 pts**
- Sufficient scale (low-D + high-D, 5+ conditions) ........................ **10 pts**
- Reproducibility (fixed seeds, configs, clear instructions) ........... **10 pts**

### Results & Analysis (20 pts)

- Clear visualizations (4-6 plots + 2-3 GIFs with captions) ............. **10 pts**
- Insightful analysis (hypothesis testing, failure cases, limitations) . **10 pts**

### Documentation (10 pts)

- Semester Contribution in root README (clear, well-structured) .......... **6 pts**
- Updated folder READMEs for modified modules .............................. **4 pts**

### Bonus (up to +10 pts)

- Novel extension beyond task requirements (e.g., new functions, theoretical analysis, hybrid approaches) ........................................... **+5 pts**
- Exceptional results (paper-quality plots, surprising insights, strong baselines) **+5 pts**

---

## Presentation

Brief 10-minute presentation with slides covering: (1) Research question, (2) Implementation highlights, (3) Key results (3-4 plots/GIFs), (4) Failure analysis, (5) Future work. Live demo or pre-recorded GIF required.

## Submission

- **GitHub**: Push all code, configs, and updated READMEs to repository
- **Canvas**: Submit PDF slides and link to GitHub branch/PR
- **Artifacts**: Include plots/GIFs in `images/semester_contribution/`
- **Team**: Document individual contributions in README

---

**Prepared by:** Tamás Takács
**Semester:** 2025/26/2