# TerraShift: A Multi-Agent Reinforcement Learning Environment for Cooperative Terrain Control and Object Sorting

Zach Oines

May 2025

**Abstract**

TerraShift is a multi-agent reinforcement-learning testbed for cooperative terrain control on dense shape display (pin/column) arrays. Implemented in Unreal Engine 5 within the UnrealRLLabs framework—which provides synchronized multi-environment rollouts, shared-memory interprocess communication between Unreal and Python, and config-driven experiments—the environment models a height-field surface that exerts in-plane forces on objects, enabling routing and sorting to color-matched goals. A practical multi-agent PPO variant with a counterfactual baseline (MA-POCA) is employed together with a compact perception stack: a shared embedding trunk converts the heightmap into patch tokens (CNN $\rightarrow$ Swin), fuses object and agent tokens via cross-attention, and provides temporal memory with a GRU, followed by separate policy, value, and baseline heads.

On a $15 \times 15$ grid with 8 agents and up to 4 objects, the system learns stable, cooperative behaviors such as basin-forming, cluster-break, and ball-to-ball avoidance maneuvers that move objects to assigned goals. Initial pretraining on this simplified setting ($\approx$96M environment steps) yields steadily improving returns but remains resource-intensive and sensitive to configuration. Finetuning from that checkpoint to a larger grid with more agents reacquires competent behaviors in $\approx$2.4M additional steps. Analysis covers failure modes (e.g., contact discreteness and coupling on pin arrays), highlights design choices that improved stability, and outlines practical steps to scale task complexity. TerraShift thus provides a reproducible UE5 benchmark and reference implementation for studying learning-based control on shape display platforms.

## 1 Introduction

Densely actuated shape displays promise general-purpose manipulation for tasks such as routing, staging, and sorting. By deforming a height-field, these platforms generate in-plane forces that move objects toward goals. This work investigates whether distributed, feedback-based policies can

accomplish these tasks without an explicit analytic model, using *TerraShift*, a new Unreal Engine testbed, and a multi-agent reinforcement-learning (MARL) approach. The setting is challenging due to dense actuator coupling, shifting contact sets, non-smooth, contact-rich dynamics, and a scale-related learning burden.

## 1.1 General Problem and Approach

The problem considered is to shape a high-dimensional, deformable surface to sort objects into target regions. In TerraShift, cooperative agents each modulate their own localized Gaussian wavelets to influence object motion on a grid of vertically actuated columns. Agents receive multi-stream observations (heightmap, object, and agent-context) and act in continuous space. The study evaluates attention-based policies trained with PPO+MA-POCA to determine whether learned strategies can manage deformable-terrain manipulation at scale. The discussion first situates TerraShift within prior work, then details the specific difficulties of this domain.

## 1.2 Context and Prior Work

Actuated surfaces span a diverse family of mechanisms for in-plane object handling, including MEMS arrays, ciliary and vibratory surfaces, variable-morphology platforms, rotor tiles, and mobile-unit swarms, with shared themes of programmability, distributed actuation, and optional sensor integration [3]. Within this space, pin-based shape displays form a distinct line of work. *Relief* demonstrated interactive terrain generation with a 120-pin tabletop display [18]. *inFORM* scaled this idea to 900 pins with depth sensing and projection mapping to render shapes and manipulate objects, with responsive movement to user interaction via hand tracking [9]. *TRANSFORM* presented an expressive, large-scale public art installation of interconnected shape displays, using similar techniques [14]. *Materiable* conveyed material properties such as elasticity and viscosity through dynamic deformation in response to physical interaction [23]. These systems typically employ analytic or heuristic control. At higher actuator densities and complex contact conditions, these analytic controllers become increasingly brittle, thereby motivating data-driven approaches

like reinforcement learning.

## 1.3 Why the Problem Is Hard

Controlling densely actuated shape displays is challenging for three interconnected reasons. *(i) Coupling and discreteness.* Many actuators may simultaneously support a single object. As contacts shift, each command alters load distribution and friction, so the mapping from actuation to the object's overall push and twist (wrench) is highly state dependent and often discontinuous. Classical array controllers can tame coupling by exposing a small set of actuation primitives that approximately decouple translation and rotation and by closing independent feedback loops on those primitives. For example, Luntz et al.'s wheel tiles run separate pose-feedback loops for planar translation and rotation before mapping the resulting velocities to module speeds, while noting that slip keeps the modes imperfectly decoupled [22]. Shape displays do not naturally admit such linear, global primitives. Forces arise indirectly from local slopes, friction, and a moving set of supports. Small shape changes can destabilize contacts, which undermines modular control. *(ii) Contact-rich, non-smooth dynamics.* Unlike rigid grasping, motion arises through rolling, slipping, impacts, and launching (non-prehensile). Even one-DoF arrays require sequential throw-and-catch maneuvers to reposition objects [1], which underscores the difficulty when scaling up. *(iii) Scale and learning burden.* High actuator counts expand the joint action space, increasing the temporal and spatial dependencies for coordination, which complicates exploration, non-stationarity, and credit assignment in MARL. Centralized critics (e.g., MADDPG) and attention-augmented critics (e.g., MA-POCA) temper these instabilities, yet surveys highlight that auxiliary mechanisms remain necessary to manage residual non-stationarity and exploration burden [21, 25, 5].

## 1.4 Why MARL

A centralized controller managing hundreds of pins results in an unwieldy joint action vector and fragile exploration. Hierarchical RL can reduce effective dimensionality via temporally abstract skills [26]. Action elimination and pruning can also focus the search [32]. However, both approaches

add design and tuning overhead in non-smooth, contact-rich regimes. A multi-agent factorization is better aligned with shape display hardware, allowing for $K$ localized controllers, each with a modest number of continuous parameters (e.g., Gaussian wavelet). Centralized-critic / decentralized-actor methods keep those local policies low-dimensional while conditioning on the joint context needed for coordination during training [21, 25]. Counterfactual-advantage variants such as COMA and its MA-POCA extension then attribute shared rewards back to individual agents, improving cooperative credit assignment [8, 5].

## 1.5 Contributions

*This paper contributes:*

- **UnrealRLLabs: A modular MARL framework in Unreal Engine.** Supports synchronized multi-environment rollouts, shared-memory interprocess communication between Unreal and Python, and high-throughput data capture; experiments are configured via seedable files to aid reproducibility.
- **TerraShift: A terrain-manipulation benchmark.** Cooperative agents deform an $N \times N$ shape display to sort colored objects into goal regions.
- **Multi-agent & Multi-datastream Modeling.** A CTDE PPO+MA-POCA agent that fuses heightmap, object, and agent-context streams in a shared trunk with memory and separate policy/baseline/value heads.

## 1.6 Paper Organization

Section 2 reviews background and related work. Section 3 presents the TerraShift environment, observation streams, action parameterization, network architecture, and training setup. Section 4 reports empirical results. Section 5 discusses limitations and future directions.

# 2 Background & Related Work

## 2.1 MARL Simulation Platforms

Selecting an appropriate simulation platform is crucial for MARL research. The project began by reviewing several options based on criteria including MARL support, physics/rendering fidelity, and usability. Relevant platforms are briefly reviewed below:

### 2.1.1 Unity ML-Agents

Leveraging the Unity engine, ML-Agents offers accessible 3D environments with moderate physics fidelity and integrated multi-agent support [15]. Its C# API and Python interface facilitate rapid prototyping, though performance may be limited by the underlying physics engine and scalability constraints in complex scenarios [16].

### 2.1.2 NVIDIA Isaac Sim and Isaac Lab

NVIDIA Isaac Sim delivers high-fidelity robotics simulation inside Omniverse, combining advanced physics, photorealistic rendering, and sophisticated sensor models that make it attractive for sim-to-real studies. A companion Isaac Lab framework further streamlines MARL experiments [13]. The trade-off is that Isaac Sim's steep computational requirements and overall system complexity can pose significant adoption barriers [16].

### 2.1.3 Google Brax

Brax is a JAX-based differentiable physics engine optimized for speed and massive parallelization on accelerators [10]. It enables rapid training iterations and supports the creation of diverse simulation scenarios, including potential MARL setups, via extensions like the Braxlines Composer API [11]. However, its suitability for complex MARL tasks can be limited by poor scaling and high computational cost as agent count increases. Additionally, challenges related to documentation and available resources have been noted [16].

## 2.2 Rationale for Using Unreal Engine

Unreal Engine (UE) was selected for TerraShift due to its balance of high-fidelity simulation, developer accessibility, and established use in robotics research.

### 2.2.1 Physics and Rendering Fidelity

UE's physics engines (including Chaos) handle complex multi-body dynamics effectively, crucial for TerraShift's physical manipulation task. Its advanced rendering produces photorealistic visuals and sensor data, vital for perception-based RL and sim-to-real transfer, as demonstrated by simulators like CARLA [7].

### 2.2.2 Extensibility and Developer Tools

UE provides a flexible development environment with intuitive tools (Unreal Editor, Blueprints visual scripting) and powerful C++ access for performance-critical code. Features like live coding and a rich Marketplace (offering assets and plugins like AGX Dynamics for advanced physics [2]) accelerate research and development.

### 2.2.3 Community and Precedents

UE's adoption by major simulation projects like CARLA [7] and Microsoft AirSim [29] validates its capacity for demanding robotics simulation tasks requiring accurate physics and sensing, aligning with TerraShift's requirements. While Epic Games has introduced a 'Learning Agents' plugin aimed at game AI bots, its focus is not on general-purpose MARL research [16]. It lacks the specific features required for the MA-POCA-based approach employed here (hence the UnrealRLLabs framework).

In essence, UE's combination of fidelity, tools, and community support made it a suitable choice for developing the TerraShift environment.

## 2.3 Influential MARL Algorithms

TerraShift's agent design draws upon several key MARL algorithms:

### 2.3.1 Proximal Policy Optimization

PPO provides a stable foundation for single-agent policy gradient updates via its clipped surrogate objective [28]. PPO is adapted for the multi-agent policy updates in this work (Section 3.3.4).

### 2.3.2 Multi-Agent Deep Deterministic Policy Gradient

MADDPG pioneered the Centralized Training with Decentralized Execution (CTDE) paradigm for continuous MARL using centralized critics [21]. The design adopts the CTDE structure, utilizing centralized networks during training (Sections 3.3.2 and 3.3.3) with all agents operating synchronously (lock-step), a valuable design simplification.

### 2.3.3 Counterfactual Multi-Agent Policy Gradients

COMA addresses multi-agent credit assignment using a centralized critic and a counterfactual baseline to assess individual agent contributions [8]. This principle informs the baseline network design (Section 3.3.3).

### 2.3.4 Multi-Agent Posthumous Credit Assignment

MA-POCA builds on COMA and uses attention mechanisms to handle variable agent numbers and assign credit effectively, even with early agent termination [5]. MA-POCA directly influences the agent's attention-based critic/baseline architecture and advantage calculation (Eqs. (5) and (7)), crucial for coordinating the terrain-deforming agents.

## 2.4 Cross-Attention Mechanisms

Attention mechanisms, particularly cross-attention as popularized by the Transformer [31], enable selective information processing, beneficial for MARL coordination. Prior work demonstrated attention-based critics improving MARL scalability [12]. Vision Transformers (ViT) [6] further showed attention's effectiveness in processing image patches for visual understanding. The concept of cross-attention extends the original Transformer's encoder-decoder attention, allowing one data modality (e.g., agent states) to attend to another (e.g., environment features). Recent MARL applications, like MACAC for cooperative driving [19], leverage cross-attention mechanisms to fuse locally observed environmental data with information about interacting agents for enhanced coordination. This motivation underlies the use of cross-attention in the shared embedding network (Section 3.3.1, Fig. 5), detailed in Algorithm 2 in the Appendix, enabling agents to integrate multi-scale visual cues from the central state with their own observations for improved coordination.

# 3 Materials and Methods

## 3.1 Unreal Engine Architecture: UnrealRLLabs

TerraShift is implemented within Unreal Engine 5 using **UnrealRLLabs**, a custom framework designed for RL experiments.

### 3.1.1 Config-Driven Design

UnrealRLLabs employs a config-driven approach where environment parameters (agent count, terrain size, rewards, etc.) are defined in external JSON files. This allows rapid iteration and ensures reproducibility by loading configurations at runtime.

### 3.1.2 Parallel Environment Instancing

The framework supports concurrent execution of multiple independent TerraShift instances within a single UE process, spawned via UE's actor system. This CPU-bound parallelization enhances data collection throughput. Tests showed feasibility for 4-16 instances on typical multi-core CPUs.

### 3.1.3 Data Flow and Communication

The standard RL interaction loop [30] maps onto UE's tick cycle, managed by the UnrealRLLabs framework as outlined in Algorithm 1. In each simulation step $i$, the framework first gathers the outcomes $(r_e, d_e, t_e)$ from the previous step for each parallel environment $e$. These outcomes, along with the previous state $S^e_{i-1}$, action $A^e_{i-1}$, and current state $S^e_i$, form a complete transition tuple which is stored in a shared experience buffer. If an environment signals termination ($d_e$) or truncation ($t_e$), it is reset, providing a new initial state. Next, the current states $\mathbf{S}_i$ from all environments are collected and sent to the external Python agent process via the shared memory bridge (Section 3.1.4). The agent process computes the actions $\mathbf{A}_i$ based on $\mathbf{S}_i$ and sends them back via the shared memory bridge. Finally, each environment $e$ applies its corresponding action $A^e_i$ using $\text{Env}[e].\textsc{Act}(A^e_i)$. Periodically (e.g., when the buffer reaches a specified size), the framework signals the Python agent via the shared memory bridge to perform a training update using the collected transitions. The overall system architecture, illustrating the interaction between the UE and Python processes, the flow of data (states, actions, transitions), and the role of the shared memory bridge, is depicted in Fig. 1.

### 3.1.4 Shared Memory Communication Implementation

To minimize latency between C++ UE environments and the Python agent process, the implementation uses a shared memory bridge (Fig. 1). Dedicated memory regions for observations, actions, and control signals are mapped into both processes using native OS APIs (Windows memory mapping and synchronization primitives). This direct memory access avoids network serialization overhead, enabling low-latency communication crucial for high-throughput RL data collection,

**Algorithm 1** UnrealRLLabs Abstracted Data Flow

**Require:** $N$ parallel environments Env$[e]$, remote Agent process

Initialize all Env$[e]$ and shared memory bridge
**for** simulation step $i \leftarrow 1 \ldots T$ **do**
  **if** $i > 1$ **then** ▷ Collect outcomes from previous step
    **for** $e \leftarrow 1 \ldots N$ **do**
      $(r_e, d_e, t_e) \leftarrow$ Env$[e]$.Outcome()
      Append $(S^e_{i-1}, A^e_{i-1}, r_e, S^e_i, d_e, t_e)$ to Transitions buffer
      **if** $d_e \vee t_e$ **then**
        Env$[e]$.Reset() $\Rightarrow$ $S^e_i$ re-initialized
      **end if**
    **end for**
  **end if**
  $\mathbf{S}_i \leftarrow$ [ Env$[1]$.State(), ..., Env$[N]$.State() ] ▷ Gather current states
  Send $\mathbf{S}_i$ to Agent via shared memory
  Receive $\mathbf{A}_i$ from Agent via shared memory ▷ Agent computes actions
  **for** $e \leftarrow 1 \ldots N$ **do** ▷ Apply actions
    Env$[e]$.Act$(A^e_i)$
  **end for**
  **if** Update condition met **then** ▷ e.g., buffer size reached
    Signal Agent to Update(Transitions) via shared memory bridge
  **end if**
**end for**

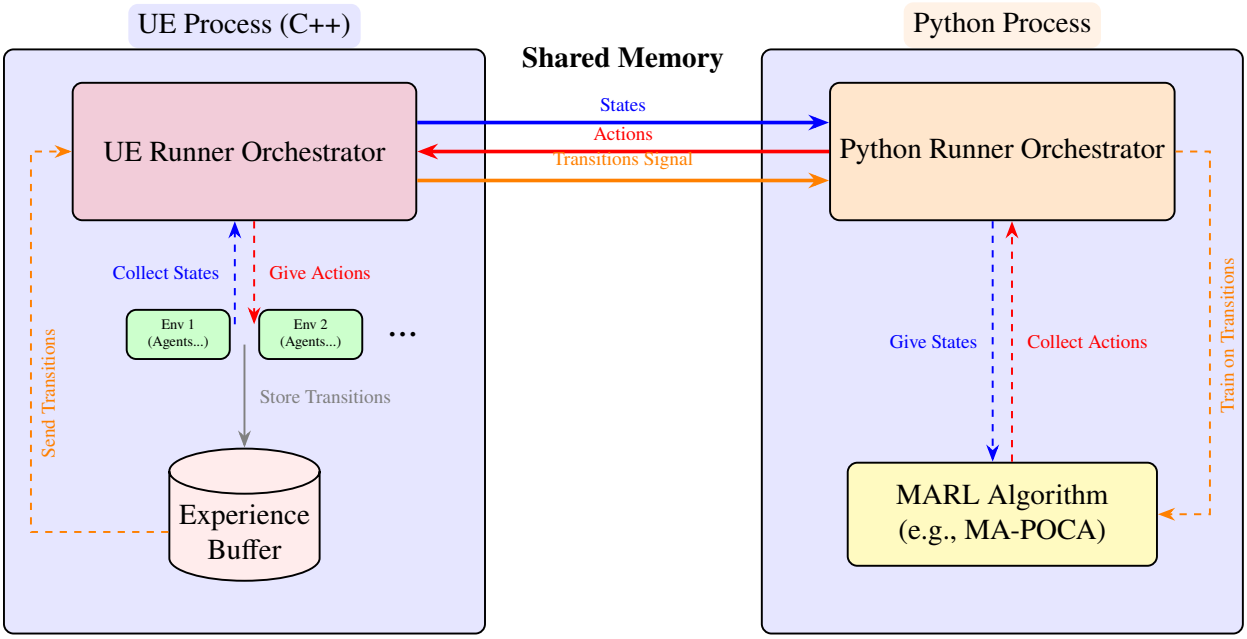


Figure 1: High-level architecture of the UnrealRLLabs framework, showing communication between the Unreal Engine (C++) and Python processes via a shared memory bridge.

though it increases implementation complexity for memory management and synchronization.

## 3.2 TerraShift Environment Design

### 3.2.1 Environment Layout

The TerraShift environment consists of a static base platform supporting a deformable terrain grid ($N \times N$ ellipsoid columns), as shown in Fig. 2. Column heights are updated based on the agent-generated wave simulation (Section 3.2.3). Physics-enabled spheres are spawned on the grid and moved by the deforming columns. Randomly placed colored zones serve as goals; an object reaching a goal's radius is considered successful. Objects falling off the platform trigger failure.

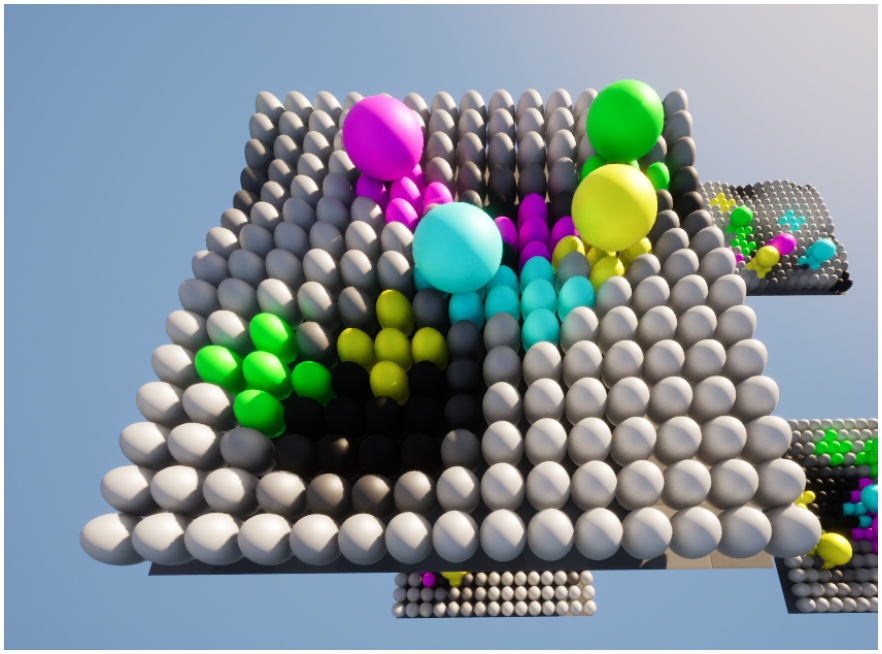

Figure 2: TerraShift environment: deformable column grid, physics object (sphere), and goal regions on a base platform.

### 3.2.2 Modular Environment Features

To facilitate systematic experimentation, TerraShift uses JSON configuration files for modularity. These allow modification of key parameters—such as environment setup (grid size, agent count,

object count), object properties, terrain dynamics, and reward structure—without recompiling. Key parameters for the representative experiment are detailed in Table 1 in the Appendix.

### 3.2.3 Multi-Agent Procedural Terrain Generation

Inspired by procedural noise functions [17], TerraShift's terrain is deformed by the superposition of $K$ agent-controlled Gaussian wavelets. Each agent $i$ contributes a localized height change $h_i(x, y)$ based on its parameters (amplitude $A_i$, standard deviations $\sigma_{x,i}, \sigma_{y,i}$, position $\mu_i$, orientation $\theta_i$):

$$h_i(x, y) = A_i \exp\left(-\frac{1}{2}\left[\left(\frac{x_i'}{\sigma_{x,i}}\right)^2 + \left(\frac{y_i'}{\sigma_{y,i}}\right)^2\right]\right),$$

where $(x_i', y_i')$ are coordinates rotated relative to the agent's frame. The total height is $H(x, y) = \sum_{i=1}^{K} h_i(x, y)$. Agents modify their parameters via actions interpreted as parameter deltas. Figure 3 illustrates the concept.

To avoid edge discontinuities, agent positions wrap around the grid and the Gaussian kernel treats opposite edges as adjacent. This produces a seamless surface and allows patterns to flow across boundaries. However wrapping can be disabled to use clamped edges if needed.

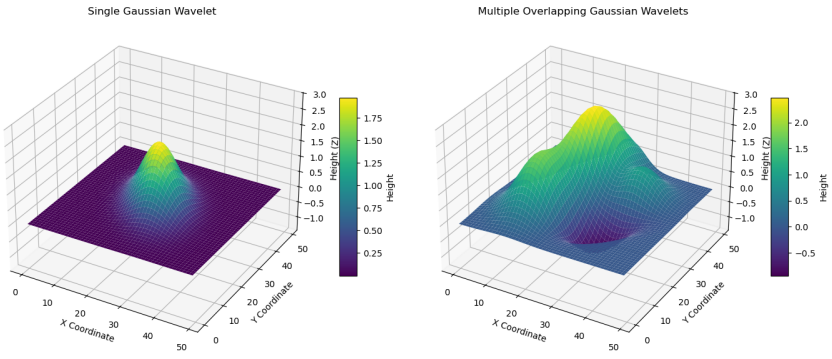


Figure 3: Left: Single Gaussian wavelet. Right: Superposition of multiple wavelets creating complex terrain.

### 3.2.4 State (Observations)

TerraShift is formulated as a fully observable MARL problem structured as a Markov Decision Process (MDP) [30]. The observation includes global and agent-specific components:

- **Central State ($s_t^{\text{central}}$):** In the experiments reported here, the setup uses a single-channel height map ($1 \times H \times W$) as the central image state (Fig. 4). The framework also supports an optional overhead greyscale image channel. In addition, the configuration includes a fixed-length *grid-object sequence*: up to $M$ entries, each a 9-D feature vector comprising normalized object position $[x, y, z]$, assigned goal position $[x, y, z]$, and object velocity $[v_x, v_y, v_z]$. Unused slots are zero-padded and accompanied by a padding mask.

- **Agent State ($o_t^i$):** A 9D vector per agent $i$: 2D Position, Orientation, Amplitude, Sigmas (X/Y), Linear Velocity (X/Y), Angular Velocity of its Gaussian wavelet. Normalized to $[-1, 1]$.

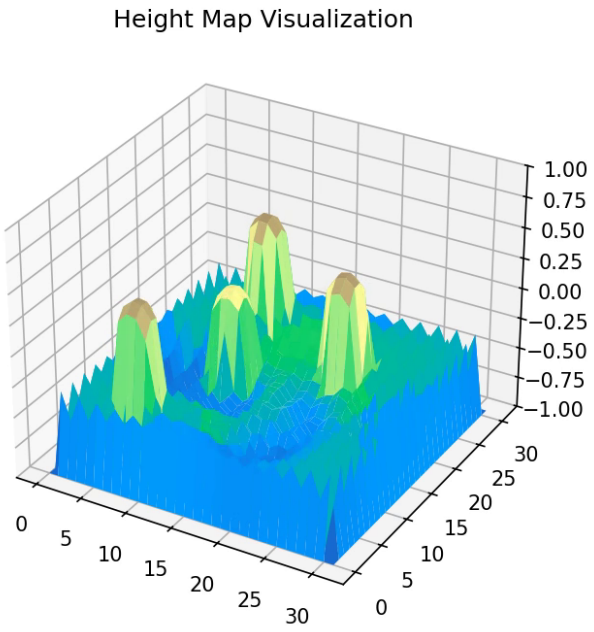


Figure 4: Central state height map used in the reported experiments (e.g., $32 \times 32 \times 1$).

### 3.2.5 Action Space

Each agent $i$ selects a continuous action $a_t^i \in \mathbb{R}^6$, corresponding to changes in its wavelet's X/Y Velocity, Angular Velocity, Amplitude, Sigma X, and Sigma Y when using action deltas. All agents act simultaneously.

### 3.2.6 Reward Function

The total reward at timestep $t$ is accumulated across all $M$ physics objects currently active in the environment: $R_t = \sum_{j=1}^{M} R_t^j$. The per-object reward $R_t^j$ combines sparse terminal rewards with optional dense shaping terms, configurable via JSON. If object $j$ reaches its goal, it receives $R_{\text{goal}}$; if it falls off, it receives a penalty $P_{\text{fall}}$. Active objects incur a small per-step penalty $P_{\text{step}}$. Optional dense shaping terms include:

- Potential-based reward shaping [24], where $\Phi(s^j)$ is a potential function (e.g., negative distance of object $j$ to its goal), $\gamma$ is the discount factor, and $S_{\text{potential}}$ is a scaling factor:

$$r^j_{\text{potential}} = S_{\text{potential}} \cdot (\gamma \Phi(s^j_{t+1}) - \Phi(s^j_t)) \tag{1}$$

- Distance improvement, rewarding reduction in distance to the goal, scaled by $S_{\text{dist}}$:

$$r^j_{\text{dist}} = S_{\text{dist}} \cdot (\text{dist}(s^j_t) - \text{dist}(s^j_{t+1})) \tag{2}$$

- Velocity alignment, encouraging object velocity toward the goal, scaled by $S_{\text{align}}$. It is calculated via cosine similarity as:

$$r^j_{\text{align}} = S_{\text{align}} \cdot \left( \frac{\mathbf{v}^j_{\text{obj}}}{\|\mathbf{v}^j_{\text{obj}}\|} \cdot \frac{\mathbf{p}^j_{\text{goal}} - \mathbf{p}^j_{\text{obj}}}{\|\mathbf{p}^j_{\text{goal}} - \mathbf{p}^j_{\text{obj}}\|} \right) \tag{3}$$

  This yields a reward between -1 (moving directly away) and +1 (moving directly toward).

### 3.2.7 Episode Termination and Reset

Episodes end upon task completion/failure (all objects reach goals or fall off) or exceeding a configured time limit. Resetting involves clearing object states, randomizing agent parameters, and placing objects/goals in new non-overlapping positions.

## 3.3 MARL Agent Architecture

The MARL agent follows the CTDE paradigm, comprising: (1) the shared embedding network processing observations (Section 3.3.1), (2) a shared policy network (actor) $\pi_\theta$ outputting actions (Section 3.3.4), (3) a centralized value network (critic) $V_\phi$ (Section 3.3.2), and (4) a centralized counterfactual baseline network $B_\psi$ for credit assignment (Section 3.3.3), inspired by COMA [8] and MA-POCA [5].

### 3.3.1 Shared Embedding Network

The shared embedding network maps the central state $S_C$—comprising the height map, grid-object sequence, and any optional camera views—and the per-agent observations $O_A$ into per-agent feature vectors used by the actor, the centralized critic, and the counterfactual baseline. The design combines three ideas: (i) a compact, multi-scale spatial backbone with adaptive tokenization, (ii) cross-attention from agents to spatial/object tokens for context fusion, and (iii) recurrent memory after the backbone for short-horizon dynamics.

*Central spatial backbone.* A CNN stem produces low-level features which are downsampled to a fixed spatial grid via adaptive average pooling before projecting to patch embeddings $E_{patch}$. 2D sinusoidal positional encodings are added and the sequence passes through Swin-inspired windowed self-attention layers with optional patch merging (without the original shifted-window alternation) to capture multi-scale context at near-linear cost in image size [20]. Finally, an adaptive tokenization module selects a bounded number of informative spatial tokens (base/min/max budget), balancing the sequence length contributed by the central state against other components (agent vectors, grid-

object sequences). This keeps computation stable as $H \times W$ grows while preserving salient terrain cues (ridges, basins, goal neighborhoods).

*Agent encoding and fusion.* Agent observations are encoded by a small MLP to form $E_{agent\_init}$. Cross-attention [31] uses agent embeddings as queries over central-state tokens (keys/values) to inject global context relevant to each agent (e.g., nearby slopes, objects, goals). In parallel, the grid-object sequence is linearly embedded into object tokens and participates in the same cross-attention. In the reported experiments, agent ID embeddings and positional encodings are enabled and a *parallel fusion* scheme is used in which agents attend to spatial and object tokens concurrently before a lightweight fusion step. A subsequent self-attention block lets agents attend to one another to support coordination. The result is a set of per-agent embeddings $\mathbf{e}_t$ that reflect local state, global surface context, and peers.

*Temporal memory.* To capture short-term dynamics (e.g., object movement characteristics), a gated recurrent unit (GRU) [4] follows the shared embedding in the agent trunk. At inference time a single GRU step is applied to the flattened per-agent embeddings and the hidden state is carried across steps (reset on episode boundaries). During training, sequences are processed with the same GRU.

The resulting per-agent embeddings feed the shared policy network and the centralized value/baseline paths. An overview appears in Fig. 5, and the forward pass is detailed in Algorithm 2.

### 3.3.2 Centralized Critic (Value Network)

The centralized state-value function $V_\phi(s_t)$ takes the per-agent embeddings $\mathbf{e}_t$ from Section 3.3.1 as input. The embeddings first pass through a Residual Self-Attention (RSA) block—following MA-POCA's use of attention to aggregate variable agent sets [5, Sec. 4.2]—and then the attended features are mean-pooled across agents to obtain a single global descriptor $\bar{\mathbf{e}}_t$. This pooled vector is fed to a lightweight MLP head to produce the scalar value prediction $V_\phi(s_t) = V_{\text{head}}(\bar{\mathbf{e}}_t)$. The critic is trained to minimize the MSE between $V_\phi(s_t)$ and the target return $R_t$, calculated using the

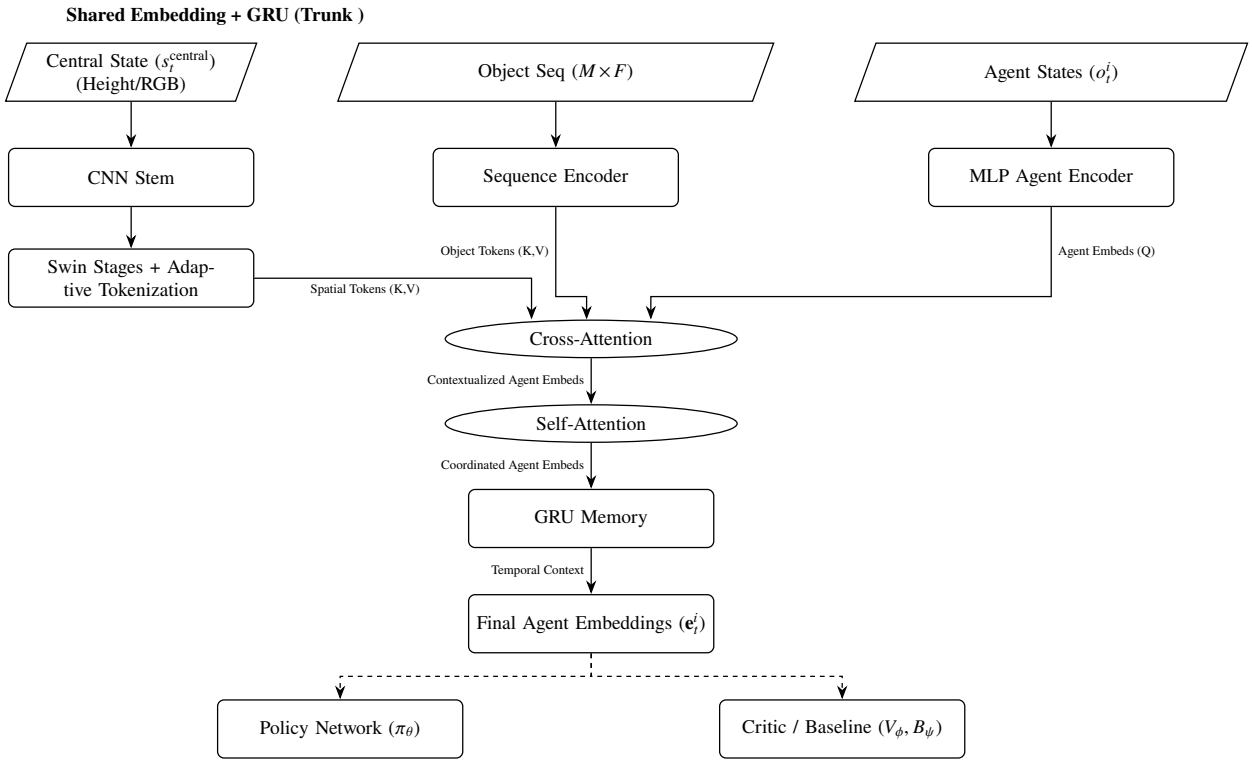


Figure 5: Architecture of the shared embedding trunk network used in the reported experiments. The central height map inputs pass through a CNN stem and Swin-style stages with adaptive tokenization to produce a bounded set of spatial tokens. In parallel, both the agent and grid-object state data are linearly embedded into separate sequences (with positional/ID encodings). Against the agent sequence embeddings, cross-attention uses parallel fusion to combine spatial/object context, followed by agent self-attention. A lightweight GRU provides short-horizon temporal memory before producing the final per-agent embeddings ($\mathbf{e}_t^i$) used by the policy and critic/baseline networks. See Algorithm 2 for the forward pass.

$TD(\lambda)$ return (related to GAE [27]):

$$\mathcal{L}^{VF}(\phi) = \hat{\mathbb{E}}_t \left[ (V_\phi(s_t) - R_t)^2 \right] \tag{4}$$

where $R_t = \hat{A}_t^{\text{GAE}(\gamma,\lambda)} + V_\phi(s_t)$ and $\hat{\mathbb{E}}_t$ denotes the empirical average over a batch of transitions.

### 3.3.3 Counterfactual Baseline Network (MA-POCA)

To improve credit assignment [8], the implementation uses a per-agent baseline $B_\psi^i$ following MA-POCA [5]. This network estimates the expected return excluding agent $i$'s contribution. It uses an attention mechanism (RSA), analogous to the critic, over embeddings derived from agent $i$'s state and other agents' state-action pairs. This allows the baseline computation to adapt flexibly to the available agents and their actions [5, Sec. 4.2, Eq. 7]:

$$B_\psi^i(\mathbf{e}_t, \mathbf{a}_t) \approx Q_{\text{baseline\_head}}(\text{RSA}(g(e_t^i), \{f(e_t^j, a_t^j)\}_{j \neq i})) \tag{5}$$

It is trained via MSE against the same target return $R_t$ used for the critic:

$$\mathcal{L}^{Baseline}(\psi) = \hat{\mathbb{E}}_t \left[ \sum_{i=1}^{K} (B_\psi^i(\mathbf{e}_t, \mathbf{a}_t) - R_t)^2 \right] \tag{6}$$

where the sum is over the $K$ agents in the system, and $\hat{\mathbb{E}}_t$ denotes the empirical average over a batch.

### 3.3.4 Policy Network (Actor)

A single policy network $\pi_\theta$, shared across agents, maps agent embeddings $e_t^i$ from Section 3.3.1 to action distributions. For continuous control, the implementation uses a Beta policy (bounded in $[-1, 1]$). The policy is updated using the PPO clipped surrogate objective [28]. The advantage estimate $\hat{A}_t^i$ uses the counterfactual baseline [5]:

$$\hat{A}_t^i = R_t - B_\psi^i(\mathbf{e}_t, \mathbf{a}_t) \tag{7}$$

With probability ratio $r_t^i(\theta) = \frac{\pi_\theta(a_t^i|e_t^i)}{\pi_{\theta_{old}}(a_t^i|e_t^i)}$, the objective is:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[ \sum_{i=1}^{K} \min \left( r_t^i(\theta)\hat{A}_t^i, \text{clip}(r_t^i(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t^i \right) \right] \tag{8}$$

where $\hat{\mathbb{E}}_t$ denotes the empirical average over a batch. An entropy bonus $S[\pi_\theta](s_t) = \sum_{i=1}^{K} H(\pi_\theta(\cdot|e_t^i))$ is added to encourage exploration. The final policy loss combines the clipped objective and the entropy bonus:

$$\mathcal{L}^{Policy}(\theta) = -\hat{\mathbb{E}}_t \left[ L^{CLIP}(\theta) + c_2 S[\pi_\theta](s_t) \right] \tag{9}$$

where $c_2$ is the entropy coefficient.

#### 3.3.5 Training Procedure

The training loop adapts PPO [28] for the MARL/CTDE setting using MA-POCA. The detailed procedure is outlined in Algorithm 3 in the Appendix. In summary, it involves collecting trajectories, computing GAE-based returns and per-agent counterfactual advantages, and then performing mini-batch updates over multiple epochs to optimize policy, critic and baseline networks using their respective loss functions (Eqs. (4), (6) and (9)).

# 4 Results

While the simplified benchmark yields consistent improvement, training remained resource-intensive and sensitive to configuration. Below is a summary of empirical observations; Section 5 interprets these findings in context and outlines implications.

## 4.1 Experimental Setup

Two regimes are reported: a **pretrain** run on a single, simplified benchmark and a **finetuning** run that continues from the pretrain checkpoint on a larger grid with more agents.

### 4.1.1 Pretrain Regime

Training operates on a $15 \times 15$ grid with 8 agents and up to 4 physics objects with random goals. Episodes are time-limited (respawn on goal/out-of-bounds) with a maximum step cap. Observations comprise a central height map ($32 \times 32 \times 1$) and a grid-object sequence (max length 4, feature size 9). Rewards combine dense distance shaping with sparse terminal events. For the learning algorithm and networks, the MA-POCA CTDE agent described in Section 3.3 is used; run settings are summarized as follows: 8 parallel environments, sequence length 32, buffer size 1024, mini-batches of 32, 4 epochs per update, totaling $\approx$ 96M environment steps. The complete values are shown in Table 1 (Supplemental).

### 4.1.2 Finetuning Regime

Starting from the pretrain checkpoint, training continues on a larger task (e.g., $30 \times 30$ grid, 16 agents, 4 objects, 2 environments) without changing the overall reward philosophy and other hyperparameters. The finetuning phase required about 2.4M additional environment steps to reacquire competent behavior.

## 4.2 Learning Curves

Figure 6 shows that mean per-step reward—the primary metric tracked—increases steadily on the simplified benchmark.

## 4.3 Qualitative Behaviors

Several recurring coordination patterns emerge: *basin-forming*, where agents shape a concave pocket that captures an object and biases its motion toward a goal; *cluster-breaking*, where transient ridges and local gradients separate nearby objects to reduce interference; and *ball-to-ball avoidance*, where surface modulations steer trajectories to minimize inter-object collisions. Figure 7 illustrates cooperative basin-forming in four frames.

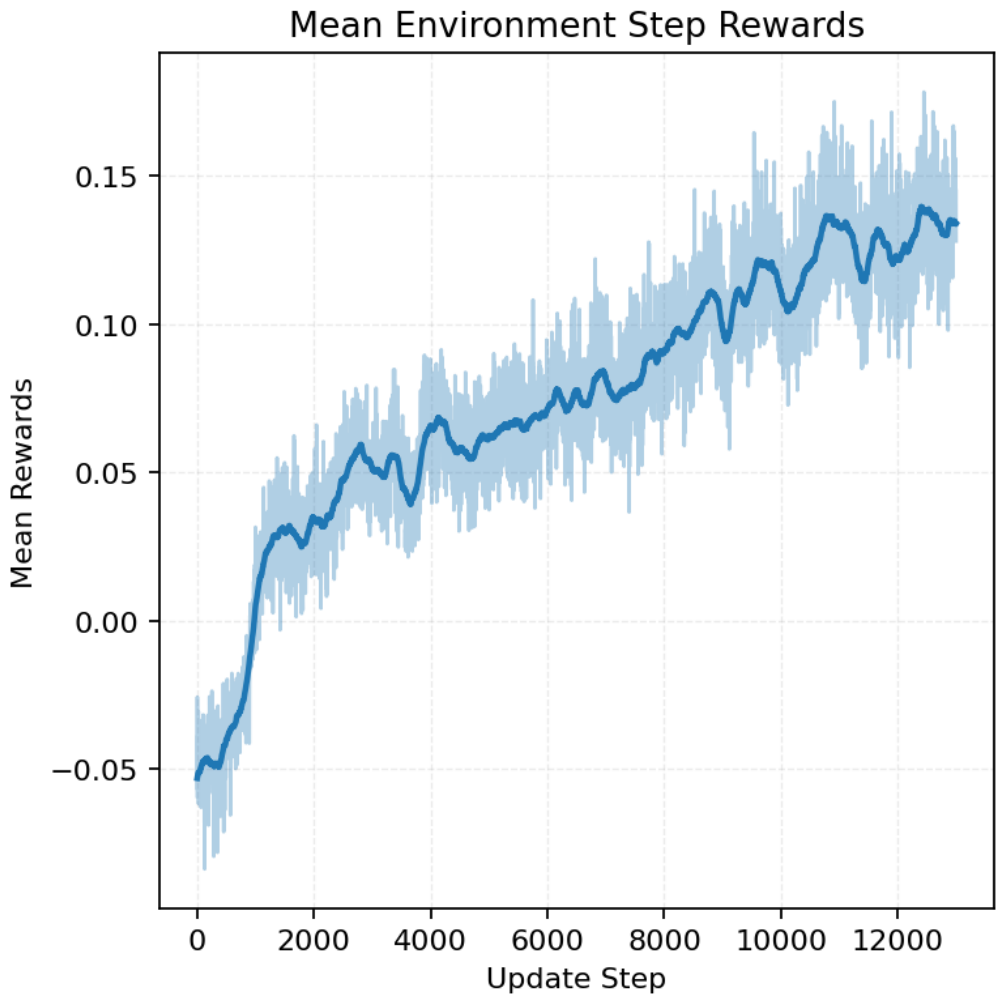


Figure 6: **Pretrain curve (15×15, 8 agents, 4 objects, 8 Environments).** Mean per-step rewards increase over training.

## 4.4 Finetuning

Initializing from the pretrain checkpoint, finetuning on a larger grid with more agents yields rapid reacquisition of successful strategies with a small additional data budget. Figure 8 shows mean per-step rewards for the transfer run.

## 4.5 Failure Modes

Typical failure cases include (i) *edge escapes* when a ridge carries an object off the platform, (ii) *border parking* where agents stall near boundaries, (iii) *oscillatory chasing* from over-reactive

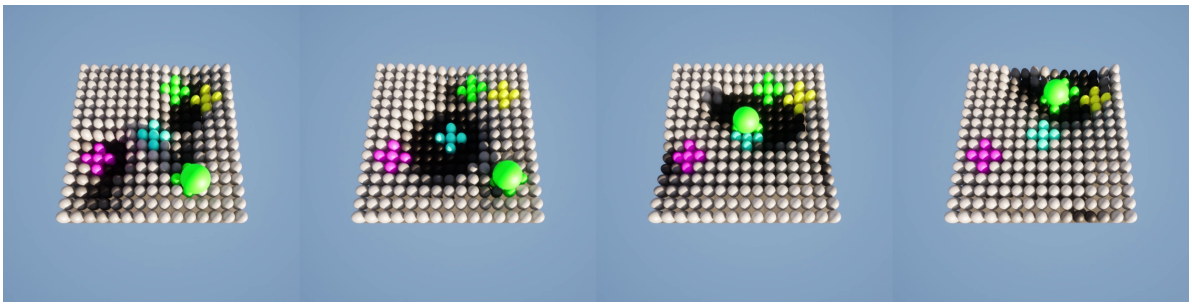

Figure 7: **Basin-forming (annotated sequence).** Multiple agents converge to form a large, spinning circular basin that dilates to draw in an object, then shepherds it along a graded exit toward its goal.

amplitude/sigma updates, and (iv) *over-smoothing* that yields surfaces too flat to impart velocity.

## 4.6 Training Challenges

While the simplified benchmark yields consistent improvement, training remained resource-intensive and sensitive to configuration.

### 4.6.1 Resource Demand

The base model required roughly two weeks on a workstation with an RTX 4090 (24 GB VRAM), AMD Ryzen 9 8945HS, and 64 GB RAM. Memory use scaled approximately linearly with the number of agents, forcing trade-offs among batch size, number of parallel environments, and network capacity.

### 4.6.2 Sensitivity to Settings

Performance was sensitive to both algorithmic hyperparameters and environment settings. Beyond reward magnitudes, it was observed that *object scale and mass* as well as physics-material properties (e.g., *friction* and *restitution/bounce*) materially affected whether motion was smooth and directed versus ballistic and unstable. Small changes in wavelet delta ranges (e.g., amplitude, sigma, velocity) could tip the behavior from rolling to bouncing, stalling learning. Considerable time was spent finetuning the overall system through trial and error. For the representative run, fairly conservative settings were chosen. The complete values are shown in Table 1 (Supplemental).

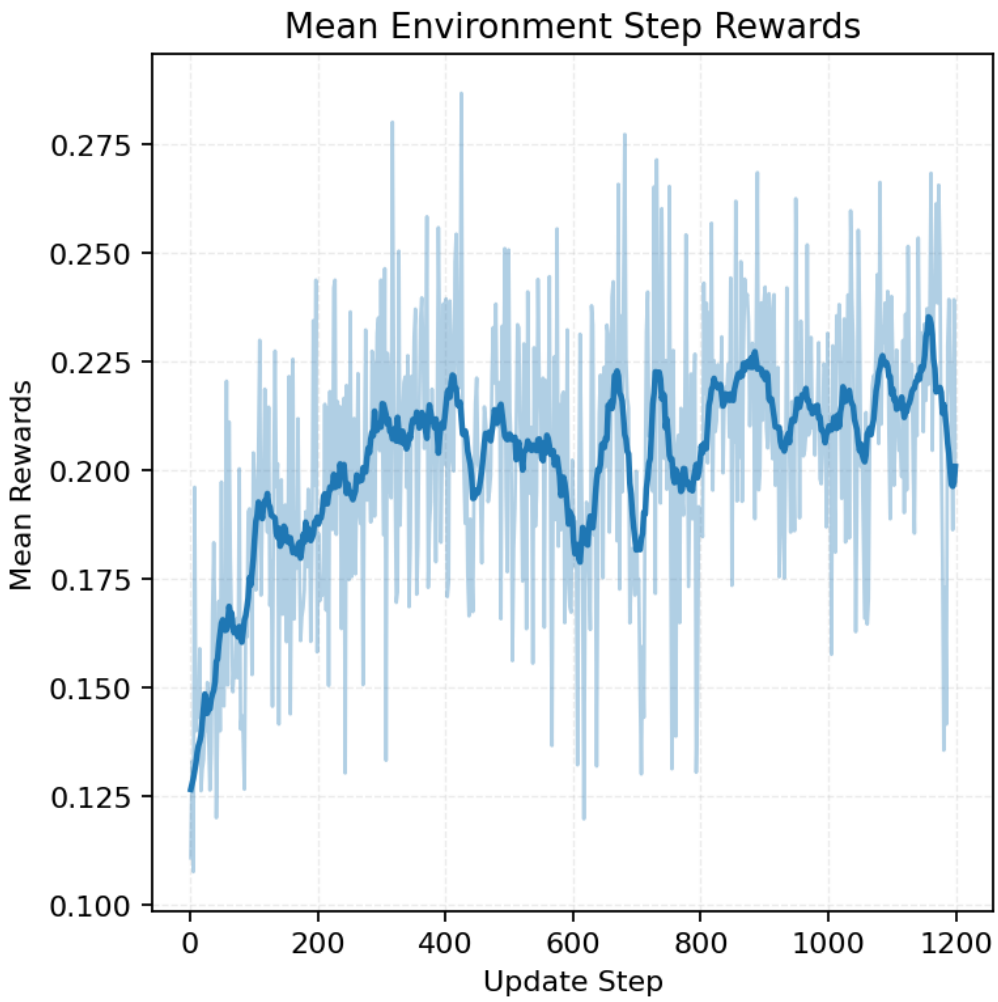


Figure 8: **Finetune curve (30×30, 16 agents, 4 objects).** Mean per-step rewards rise quickly from the pretrained initialization.

#### 4.6.3 Task Scale and Non-Stationarity

Scaling TerraShift along any of its task axes increases optimization difficulty and compute. Larger grids increase effects of coupling and often make early learning brittle; more agents amplify non-stationarity (each learner's target shifts as teammates change), stressing credit assignment; and more objects increase interaction complexity and reward variance (collisions/interference), yielding noisier advantages. Centralized training with a counterfactual baseline (MA-POCA) and attention-based critics mitigated-but did not eliminate-these effects. In practice, the workflow standardized on a modest $15 \times 15$ grid for stable pretraining, then finetuned to larger settings.

#### 4.6.4 Actuator Geometry

When reducing grid size, using cubic columns (as in classic shape displays) proved too coarse for nuanced control. Ball-to-pin contact tended to be intermittent and catchy in these setups. Swapping to *ellipsoidal* column geometry improved contact continuity with rolling objects, resulting in smoother and more controllable surface-object interactions.

#### 4.6.5 Representation Balance

Because the central height map (32×32) dominates the token budget relative to the smaller agent and object sequences, the approach relied on CNN downsampling with Swin-style stages and adaptive tokenization to cap the number of central tokens. These choices improved stability without materially increasing runtime.

# 5 Discussion

This section revisits the results in Section 4 and relates them to goals from Section 1. The aim is to (i) synthesize what the experiments demonstrate under the constraints of long training time and fragile convergence, (ii) explain how the observed failure modes arise from the fundamental difficulty of dense terrain manipulation, and (iii) outline directions for scaling up the study.

## 5.1 Summary of Findings

#### 5.1.1 Consistent Learning Dynamics on the Benchmark

In the simplified benchmark (15×15 grid, 8 agents, 4 objects), mean per-step rewards improve steadily during pretraining (see Fig. 6), indicating ongoing policy advancement despite complex non-prehensile dynamics. The overall trend is a salient signal, considering the environment's variance and sensitivity.

### 5.1.2 Emergent Multi-Agent Coordination

Qualitative rollouts consistently reveal behaviors like basin-forming that translate into object-to-goal transport (cf. Fig. 7). These patterns are not scripted and appear to arise from the policy optimizing under PPO/MAPOCA with shared embeddings and attention. This suggests agents acquired non-trivial cooperation primitives suited to the task.

### 5.1.3 Transfer from Small to Larger Tasks

A pretrained policy adapts quickly under finetuning on the more demanding setting (larger grid and agent pool), with rapid reacquisition of reward compared to training from scratch (see Fig. 8), in settings where training the larger task from scratch did not reliably converge. This supports the claim that the representations learned during pretraining capture reusable structures of object motion over deformable terrain.

### 5.1.4 Robustness in the Face of Training Cost and Fragility

Achieving the above required multi-week runs and extensive tuning (reward shaping, learning-rate schedules, and stability safeguards). That meaningful behaviors emerged consistently despite this fragility validates that the overall system design is workable for TerraShift.

### 5.1.5 Reproducibility at the Behavioral Level

Similar transport strategies and reward trajectories were observed across independent seeds. In the absence of full ablations, this cross-run behavioral consistency and the pretrain→finetune transfer act as convergent evidence that the components cohere into an effective solution.

## 5.2 Limitations and Underlying Difficulty

### 5.2.1 Sparse and Delayed Feedback Induces Passivity Without Shaping

Early runs often resulted in risk-averse policies (low-energy wavelets, object avoidance). This is consistent with long-horizon, sparse rewards: infrequent successes were too rare to offset penalties and contact instability. Introducing a blend of dense shaping (e.g., per-step object progress) and increased event frequency (e.g., respawning objects after OOB/GOAL) enabled a usable learning signal and unlocked progress.

### 5.2.2 Scale Amplifies Non-Stationarity and Variance

Increasing the grid size, agent count, or object count independently destabilized learning: more agents worsened non-stationarity and credit assignment; more objects increased reward variance; larger grids made object motion less predictable and controllable. These effects mirror Section 1's argument that scale expands the joint action/state space and compounds coupling, making end-to-end learning brittle. A curriculum (small→large) was necessary for stability.

### 5.2.3 Actuation Noise and Contact Discontinuity

Small changes to wavelet ranges (amplitude/width) or material properties (friction/restitution) led to significant, often chaotic alterations in object trajectories, particularly near contact transitions. This directly reflects dense coupling on pin arrays: many columns share support, so seemingly local commands alter global object behaviors, making the action→motion mapping highly non-smooth.

### 5.2.4 Temporal Returns Compression Improved Stability

Reducing the discount factor $\gamma$ and GAE parameter $\lambda$ made the policy more myopic and reactive, which proved beneficial under contact discontinuities and reward variance. Shortening effective horizons improved the signal-to-noise ratio in returns/advantages and reduced brittle, over-optimistic long-horizon plans. Although macro behaviors may show meandering object-to-goal

paths, particularly in larger environmental configurations (fine-tuning).

## 5.3 Future Work

### 5.3.1 Curriculum Learning

The pretrain→finetune result suggests a systematic curriculum along axes such as grid size, number of agents, and number of objects, augmented with intermediate objectives. Each stage should provide denser feedback and simpler credit assignment, while transfer between stages preserves coordination and reduces retraining time.

### 5.3.2 Extending TerraShift to Other Actuators and Control Modes

Beyond height-field wavelets, TerraShift can support programmable surfaces with alternative primitives (e.g., ciliary arrays, vibrational fields, rotor tiles) and complementary parameterizations (e.g., continuous surface vector fields). This would enable comparative studies of expressivity, energy, latency, and learning stability across actuator families under the same MARL task template.

### 5.3.3 Performance Enhancements

Integrating GPU-accelerated sensing and tracking advances in GPU physics within UE to increase simulation throughput and enable larger-scale experiments. The current pipeline relies on CPU-based scene capture components and line traces, so sensor fidelity and the number of parallel environments are bounded by the CPU. Likewise, Chaos—the primary UE physics engine—remains CPU-bound today, reinforcing the need for accelerator-aware upgrades. Improved throughput would render broader ablations feasible.

# Conclusion

This work contributes (i) **UnrealRLLabs**, a lightweight, modular, config-driven framework for parallel multi-agent RL in UE5; (ii) **TerraShift**, a MARL benchmark for cooperative, continuous

control of deformable terrain to manipulate multiple objects; and (iii) a **multi-datastream, multi-agent architecture** trained with PPO+MA-POCA.

Experimentally, the results demonstrate reproducible multi-agent coordination on a 15×15 benchmark and effective transfer through pretraining → finetuning to larger configurations, where training from scratch proved brittle (see Section 5). These results demonstrate feasibility with extended training times and unstable convergence. They also highlight core challenges, such as reward modeling, non-smooth contact, and scale related non-stationarity, which complicate end-to-end learning.

While some questions remain, the framework and environment provide a practical testbed for studying programmable surfaces using MARL. As outlined in Section 5.3, the three near-term paths are: curriculum design across task scales and intermediate objectives; extending TerraShift to alternative actuation and control parameterizations; and performance improvements (GPU sensing/physics) to enable broader and larger studies. These tools aim to promote further research in actuated surfaces and MARL.

Source code, TerraShift assets, and experiment configurations are available at `https://github.com/zachoines/UnrealRLLabs`.

# Acknowledgments

## Disclosure of Generative AI

Generative AI tools assisted in auxiliary tasks for this work, including brainstorming, prose writing, grammar checking, code generation, and error troubleshooting, in accordance with the University of Minnesota's academic integrity principles. All substantive contributions, including core concepts, analysis, and conclusions, are the author's original work. AI-generated content was critically reviewed, and the author assumes full responsibility for the manuscript.

# References


[1] Adel Akbarimajd, Majid Nili Ahmadabadi, and Bahram Beigzadeh. Dynamic object manipulation by an array of 1-dof manipulators: Kinematic modeling and planning. *Robotics and Autonomous Systems*, 55(6):444–459, 2007.

[2] Algoryx Simulation AB. AGX Dynamics for Unreal. `https://www.algoryx.se/agx-dynamics-for-unreal/`, 2024. Accessed: 2025-04-19.

[3] Edoardo Bianchi, Francisco Javier Brosed Dueso, and José A. Yagüe-Fabra. In-plane material handling: A systematic review. *Applied Sciences*, 14(16):7302, 2024.

[4] Kyunghyun Cho, Bart van Merri"enboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[5] Andrew Cohen, Ervin Teng, Vincent-Pierre Berges, Ruo-Ping Dong, Hunter Henry, Marwan Mattar, Alexander Zook, and Sujoy Ganguly. On the use and misuse of absorbing states in multi-agent reinforcement learning. *arXiv preprint arXiv:2111.05992*, 2022.

[6] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

[7] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In Sergey Levine, Vincent Vanhoucke, and Ken Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning (CoRL 2017)*, volume 78 of *Proceedings of Machine Learning Research*, pages 1–16. PMLR, 13–15 Nov 2017.

[8] Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. *arXiv preprint arXiv:1705.08926*, 2018.

[9] Sean Follmer, Daniel Leithinger, Alex Olwal, Akimitsu Hogge, and Hiroshi Ishii. inFORM: Dynamic physical affordances and constraints through shape display. *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology (UIST '13)*, pages 417–426, 2013.

[10] C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax – a differentiable physics engine for large scale rigid body simulation. *arXiv preprint arXiv:2106.13281*, 2021.

[11] Shixiang Shane Gu, Manfred Diaz, C. Daniel Freeman, Hiroki Furuta, Seyed Kamyar Seyed Ghasemipour, Anton Raichuk, Byron David, Erik Frey, Erwin Coumans, and Olivier Bachem. Braxlines: Fast and interactive toolkit for rl-driven behavior engineering beyond reward maximization. *arXiv preprint arXiv:2110.04686*, 2021.

[12] Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning (ICML 2019)*, volume 97 of *Proceedings of Machine Learning Research*, pages 2961–2970. PMLR, 09–15 Jun 2019.

[13] Isaac Lab Project Developers. Isaac Lab: A unified framework for robot learning on isaac sim. `https://github.com/isaac-sim/IsaacLab`, 2024. Accessed: 2025-04-19.

[14] Hiroshi Ishii, Daniel Leithinger, Sean Follmer, Amit Zoran, Philipp Schoessler, and Jared Counts. TRANSFORM: Embodiment of "radical atoms" at milano design week. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '15)*, pages 687–694, New York, NY, USA, 2015. Association for Computing Machinery.

[15] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2020.

[16] Michael Kaup, Cornelius Wolff, Hyerim Hwang, Julius Mayer, and Elia Bruni. A review of nine physics engines for reinforcement learning research. *arXiv preprint arXiv:2407.08590*, 2024.

[17] Ares Lagae, Sylvain Lefebvre, Robert L. Cook, Tony DeRose, George Drettakis, David S. Ebert, J. P. Lewis, Ken Perlin, and Matthias Zwicker. A survey of procedural noise functions. *Computer Graphics Forum*, 29(8):2579–2600, 2010.

[18] Daniel Leithinger and Hiroshi Ishii. Relief: A scalable actuated shape display. In *Proceedings of the 4th International Conference on Tangible, Embedded, and Embodied Interaction (TEI '10)*, pages 221–222, New York, NY, USA, 2010. Association for Computing Machinery.

[19] Kun Liang, Xin Zheng, Zhenghao Chen, Chunjing Wu, and Jian Zhang. Efficient collaborative multi-agent driving via cross-attention and concise communication. `https://www.researchgate.net/publication/382286104_Efficient_Collaborative_Multi-Agent_Driving_via_Cross-Attention_and_Concise_Communication`, 2024. Accessed: 2025-04-19.

[20] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Xiaogang Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[21] 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 30 (NIPS 2017)*, pages 6379–6390, 2017.

[22] Jonathan E. Luntz, William C. Messner, and Howie Choset. Distributed manipulation using discrete actuator arrays. *The International Journal of Robotics Research*, 20(7):553–583, 2001.

[23] Ken Nakagaki, Luke Vink, Jared Counts, Daniel Windham, Daniel Leithinger, Sean Follmer, and Hiroshi Ishii. Materiable: Rendering dynamic material properties in response to direct physical touch with shape changing interfaces. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*, pages 2764–2772, New York, NY, USA, 2016. Association for Computing Machinery.

[24] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, page 278–287, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[25] Zepeng Ning and Lihua Xie. A survey on multi-agent reinforcement learning and its application. *Journal of Automation and Intelligence*, 3(2):73–91, June 2024.

[26] Shubham Pateria, Budhitama Subagdja, Ah-Hwee Tan, and Chai Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys (CSUR)*, 54(5):1–35, 2021.

[27] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2018.

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

[29] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. AirSim: High-fidelity visual and physical simulation for autonomous vehicles. *arXiv preprint arXiv:1705.05065*, 2018. Appears in Field and Service Robotics 2017.

[30] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2nd edition, 2018. Second edition. Available at `http://incompleteideas.net/book/the-book.html`.

[31] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30 (NIPS 2017)*, pages 5998–6008, 2017.

[32] Tom Zahavy, Matan Haroush, Nadav Merlis, Daniel J. Mankowitz, and Shie Mannor. Learn what not to learn: Action elimination with deep reinforcement learning. In *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*, pages 3582–3593, 2018.

# A Supplemental Details

This appendix provides supplementary details referenced in the main text, including the pseudocode for the feature extractor and the key experimental parameters.

**Algorithm 2** Shared Embedding Network Forward Pass (Swin-inspired + Adaptive Tokenization)

**Require:** Central image $S_C$ ($B\times C\times H\times W$), grid-object sequence $G$ ($B\times M\times F$), optional grid-object mask $G_{\text{mask}}$ ($B\times M$), agent states $O_A$ ($B\times A\times Obs_{dim}$), optional prior GRU state $H_{\text{prev}}$

**Ensure:** Final per-agent embeddings $E_{final}$ ($B\times A\times D$), updated GRU state $H_{\text{next}}$

**function** SharedEmbeddingNetworkForward($S_C, G, O_A, G_{\text{mask}}, H_{\text{prev}}$)

▷ Notation: $B$ batch size; $A$ agents; $D$ embedding dim; $M$ max objects; $F$ object feature dim; $H\times W$ height map resolution; $Obs_{dim}$ agent obs dim.

$S'_C \leftarrow \text{AddCoords}(S_C)$ ▷ Append 2D coords

$F_{cnn} \leftarrow \text{CNNStem}(S'_C)$

$F_{pooled} \leftarrow \text{AdaptivePool}(F_{cnn})$ ▷ Target spatial scale

$E_{proj} \leftarrow \text{ProjectionConv}(F_{pooled})$ ▷ To embed dim

$X \leftarrow \text{FlattenAndNorm}(E_{proj}) + \text{SinCosPosEmb2D}()$ ▷ $(B, H_pW_p, D)$

$H_p \leftarrow \text{target\_pool\_scale}$, $W_p \leftarrow \text{target\_pool\_scale}$

**for** each Swin stage $(w, \text{merge})$ **do**

$X \leftarrow \text{WindowedSelfAttention}(X; w)$ ▷ Non-shifted windows

**if** merge = true **then**

$(X, H_p, W_p) \leftarrow \text{PatchMerge}(X, H_p, W_p)$

**end if**

**end for**

**if** Adaptive tokenizer enabled **then** ▷ Select informative tokens

$E_{patch}, \_ \leftarrow \text{AdaptiveSpatialTokenizer}(X; \text{base/min/max})$

**else**

$E_{patch} \leftarrow X$

**end if**

▷ Grid-object sequence ⇒ object tokens

$E_{obj} \leftarrow \text{LinearSequenceEncoder}(G) + \text{SinCosPosEmb1D}(M)$ ▷ $(B, M, D)$

▷ Agent encoding

$E_{agent\_init} \leftarrow \text{AgentEncoder}(O_A)$ ▷ $(B, A, D)$

$ID_A \leftarrow [0, 1, \ldots, A-1]$ ▷ Implicit agent indices

$E_{id} \leftarrow \text{IDEmbedding}(ID_A) + \text{IDPosEnc}(ID_A)$

$E_{agent\_init} \leftarrow \text{LayerNorm}(E_{agent\_init} + E_{id})$

$E \leftarrow E_{agent\_init}$

**for** transformer layer $l = 1 \ldots L$ **do**

$\mathcal{M} \leftarrow \text{BuildComponentMasks}(G_{\text{mask}}, \ldots)$

$E \leftarrow \text{ParallelCrossAttention}(E, \{E_{patch}, E_{obj}, \ldots\}, \mathcal{M})$ ▷ Per-component attention with softmax gating

$E \leftarrow \text{AgentSelfAttention}(E)$ ▷ Residual self-attention across agents

$E \leftarrow \text{FeedForwardBlock}(E)$

**end for**

$(E, H_{\text{next}}) \leftarrow \text{GRUMemory}(E, H_{\text{prev}})$ ▷ Optional recurrent memory

$E \leftarrow \text{FinalLayerNorm}(E)$

$E_{final} \leftarrow \text{OutputMLP}(E)$

**return** $E_{final}, H_{\text{next}}$

**end function**

**Algorithm 3** MA-POCA Training Update (Simplified)

**Require:** Networks: Shared Embedding $E_\omega$, Policy $\pi_\theta$, Critic $V_\phi$, Baseline $B_\psi$
**Require:** Collected batch of transitions $\mathcal{D} = \{(s_t, \mathbf{a}_t, r_t, s_{t+1}, d_t, t_t)\}_{t=1..T}$
**Require:** Loss weights $c_{policy}, c_{value}, c_{baseline}$
Compute GAE returns $R_t$ for $\mathcal{D}$ using $V_\phi$
**for** update epoch $e \leftarrow 1 \ldots E_{update}$ **do**
  **for** mini-batch $m \subset \mathcal{D}$ **do**
    $\mathbf{e}_t \leftarrow E_\omega(s_t)$ ▷ Calculate shared embeddings
    $v_t \leftarrow V_\phi(\mathbf{e}_t)$ ▷ Calculate value estimates
    $b_t^i \leftarrow B_\psi(\mathbf{e}_t, \mathbf{a}_t)$ ▷ Calculate baseline estimates
    $\hat{A}_t^i \leftarrow R_t - b_t^i$ ▷ Calculate MA-POCA advantages (Eq. (7))
    Optionally normalize advantages $\hat{A}_t^i$ across mini-batch $m$
    $\mathcal{L}_{policy} \leftarrow$ CalculatePolicyLoss$(\pi_\theta, \hat{A}_t^i, \mathbf{e}_t)$ ▷ Using Eqs. (8) and (9)
    $\mathcal{L}_{value} \leftarrow$ CalculateValueLoss$(v_t, R_t)$ ▷ Using Eq. (4)
    $\mathcal{L}_{baseline} \leftarrow$ CalculateBaselineLoss$(b_t^i, R_t)$ ▷ Using Eq. (6)
    $\mathcal{L}_{total} \leftarrow c_{policy}\mathcal{L}_{policy} + c_{value}\mathcal{L}_{value} + c_{baseline}\mathcal{L}_{baseline}$ ▷ Combine losses
    Optimize parameters $\omega, \theta, \phi, \psi$ using $\nabla\mathcal{L}_{total}$ ▷ e.g., AdamW step
  **end for**
**end for**
*// For implementation details, see MAPOCAAgent.py*

Table 1: Key parameters for the representative TerraShift experiment (pretrain).

| | |
|---|---|
| **(a) Task & environment** | |
| **Terrain grid** | $15 \times 15$ actuated columns; height range $[-6, 6]$; |
| **Observations** | Height map $32 \times 32 \times 1$; grid-object sequence (max 4, 9-D entries); agent sequence (8, 9-D entries) |
| **Actions** | 6-D deltas $(\Delta v_x, \Delta v_y, \Delta\dot{\theta}, \Delta\text{amp}, \Delta\sigma_x, \Delta\sigma_y)$ with ranges $[-1, 1]$, $[-0.785, 0.785]$, $[-6, 6]$, $[0.2, 2.0]$ |
| **Rewards** | Distance-improvement shaping (scale 100); goal +10; out-of-bounds $-2.5$; time-step penalty $-10^{-4}$ |
| **Episode termination** | Max 1024 steps with respawn on goal/out-of-bounds |
| **(b) Embedding & memory** | |
| **Embedding trunk** | Dimension 128; three transformer layers with 8-head parallel cross-attention over spatial/object tokens |
| **Central CNN stem** | Channels $[16, 32, 32]$, kernels $[5, 3, 3]$, strides $[2, 2, 1]$, group norms $[4, 8, 8]$; pooled to $4 \times 4$ with coordinate channels |
| **Swin refinement** | Window sizes 2 and 4 with final patch merge; adaptive tokenizer (base/min/max tokens $12/8/16$) |
| **Sequence branch** | Linear stem $[64, 128]$ with positional encodings for grid-object features |
| **Temporal memory** | Two-layer GRU (hidden size 128, dropout 0.05) applied after transformer trunk |
| **(c) Policy, value, and optimization** | |
| **Policy head** | Shared Beta policy (hidden 192, min concentration 1.01, dropout 0.05) |
| **Critic / baseline** | Centralized attention pooling (8 heads) with per-head MLPs (hidden 128) for value and counterfactual baseline |
| **Discount / GAE** | $\gamma = 0.95$, $\lambda = 0.92$ |
| **PPO clip range** | $0.2 \rightarrow 0.1$ (linear decay over 10k iterations) |
| **Entropy coefficient** | $0.005 \rightarrow 0.001$ over 10k iterations |
| **Learning rates** | Trunk/policy $1 \times 10^{-4}$; value/baseline $2 \times 10^{-4}$ (each decays by 10× over 10k iterations); Adam optimizer |
| **Regularization** | Dropout 0.05 on trunk/policy/critic; max grad norm $5.0 \rightarrow 0.5$ over 10k iterations |
| **(d) Data collection & training loop** | |
| **Parallel environments** | 8 synchronous UE instances; action repeat 2 |
| **Batch & epochs** | Batch size 1024 steps (x8 envs); split into padded sequences of 32 steps each for GRU; shuffled mini-batch of 32 sequences; 4 epochs per update |