CAMAR: Continuous Actions Multi-Agent Routing

Artem Pshenitsyn, Aleksandr Panov, Alexey Skrynnik

Abstract

Multi-agent reinforcement learning (MARL) is a powerful paradigm for solving cooperative and competitive decision-making problems. While many MARL benchmarks have been proposed, few combine continuous state and action spaces with challenging coordination and planning tasks. We introduce CAMAR, a new MARL benchmark designed explicitly for multi-agent pathfinding in environments with continuous actions. CAMAR supports cooperative and competitive interactions between agents and runs efficiently at up to 100,000 environment steps per second. We also propose a three-tier evaluation protocol to better track algorithmic progress and enable deeper analysis of performance. In addition, CAMAR allows the integration of classical planning methods such as RRT and RRT* into MARL pipelines. We use them as standalone baselines and combine RRT* with popular MARL algorithms to create hybrid approaches. We provide a suite of test scenarios and benchmarking tools to ensure reproducibility and fair comparison. Experiments show that CAMAR presents a challenging and realistic testbed for the MARL community.

Code — https://github.com/AIRI-Institute/CAMAR.git

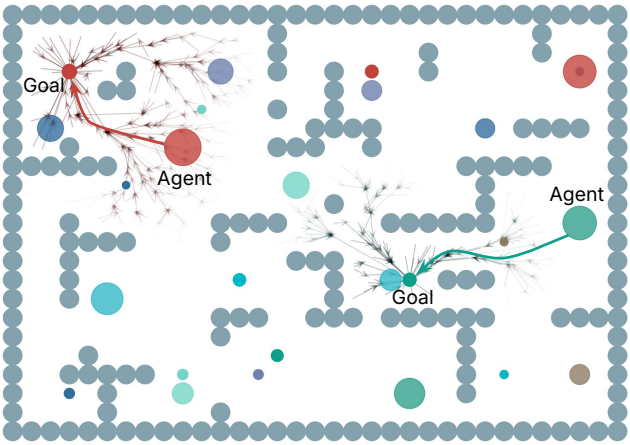


Figure 1: An example scenario from the proposed CAMAR benchmark. Agents are represented as filled circles. Each agent aims to reach its goal while avoiding collisions. The small arrows for the red and green agents indicate segments of paths generated by RRT*, providing guidance for the RL algorithms.

Introduction

Multi-agent reinforcement learning (MARL) has shown strong results in cooperative and competitive settings. Many recent studies explore how MARL can solve tasks that require coordination and decision-making in complex environments [1, 2, 3, 4, 5, 6].

One important group of such tasks is multi-agent path finding (MAPF), where several agents must move to their goals without collisions. Traditional MAPF is often studied on discrete grids, but real robots move in continuous space, follow realistic dynamics, and must plan smooth trajectories. This continuous version of MAPF is critical for a wide range of real-world domains where multiple robots or agents must coordinate in shared environments. Straightforward applications include warehouse logistics, drone swarm coordination, etc., where agents must move efficiently, avoid collisions, and cooperate to reach common goals.

Learnable approaches have recently become effective for solving MAPF and cooperative navigation tasks [7, 5, 6, 4], demonstrating that reinforcement learning can handle many agents simultaneously. However, most existing environments still simplify the problem using grid worlds or discrete actions that cannot model the smooth movements required in realistic systems.

Generally, robots must plan and coordinate while avoiding walls and other robots simultaneously. The task of multi-robot navigation and avoiding collisions with dynamic obstacles is fundamental in robotics and has been studied for a long time [8, 9, 7, 10]. Many current MARL environments do not fully capture this problem. Most simplify navigation into discrete actions or low-dimensional maps that cannot represent real movement.

At the same time, high-fidelity simulators like Gazebo[1], Isaac Sim[2], AirSim [11], or Flightmare [12] support realistic physics and 3D rendering, but they are designed for robotic control and perception research, not for large-scale MARL. These simulators often run slower and cannot efficiently simulate hundreds of agents at once.

To study large-scale continuous navigation, we need envi-

[1] https://gazebosim.org/home

[2] https://developer.nvidia.com/isaac/sim

ronments that combine the best of both worlds: fast simulation for MARL research and continuous dynamics similar to high-fidelity robotics simulators. Such environments should run efficiently on GPU, support realistic interactions, and allow testing both learned and classical path planning methods in a unified framework.

We have identified three main limitations in existing MARL environments. First, many use discrete environments, which cannot represent the smooth movements and physical dynamics important in real applications [13, 14, 15]. Second, while some benchmarks do support continuous environments and complex tasks, they do not scale well when the number of agents and obstacles becomes large [16]. Third, some environments can scale and use continuous states and actions, but their tasks are too simple and do not challenge agents to develop advanced policies [17, 18, 19].

To bridge the gap between multi-robot systems and MARL research, we introduce the CAMAR (Continuous Actions Multi-Agent Routing) Benchmark. Specifically, we make the following contributions:

- We introduce CAMAR, an extremely fast environment with GPU acceleration support (using JAX), achieving speeds exceeding 100,000 steps per second. It is designed for multi-agent navigation and collision avoidance tasks in continuous state and action spaces.
- We propose an evaluation protocol that includes both training and holdout task instances, as well as a suite of metrics and performance indicators to assess agents' generalization capabilities.
- We provide strong baselines for benchmarking, including state-of-the-art MARL algorithms and classical path planning methods commonly used in robotics, and conduct an extensive experimental study to evaluate their performance across diverse scenarios.

Related Work

Many multi-agent reinforcement learning (MARL) benchmarks have been developed in recent years, each focusing on different challenges such as coordination, navigation, or multi-agent planning. A key ability for agents in robotics is to move and adapt in complex environments. Some recent environments explore these abilities, but they differ widely in important properties.

These differences include whether the environment supports continuous spaces, GPU acceleration, and high simulation throughput. Some platforms support only discrete actions, while others offer more realistic continuous control. Environments also vary in their ability to handle large groups of agents or support partial observability.

Other practical features also matter for researchers. Some environments are fully implemented in Python and support flexible task generation, which makes them easier to extend and use in large experiments. Others may lack documentation or require heavy customization.

Finally, only a few MARL environments offer standard evaluation protocols, automated tests, or package installation via PyPi. These features are useful for ensuring fair comparisons between methods, improving reproducibility, and making environments easier to maintain. In our work, we aim to address these points by extending existing protocols and providing a high-performance, flexible platform.

To sum up, our benchmark builds on the strengths of existing environments and aims to complement ongoing efforts in MARL research. Over the years, a range of benchmarks has helped researchers tackle challenges in coordination, planning, and navigation. Well-known examples include SMAC [20, 17, 19], Jumanji [21], POGEMA [14, 15], MPE [22, 18], and VMAS, each of which introduces useful features for different kinds of tasks.

SMAC [20, 17, 19] is popular for testing strategic decision-making, but it uses discrete actions and does not scale well for large environments. Jumanji [21] supports GPU acceleration and procedural generation, but its focus is not on navigation or planning. POGEMA [14, 15] is strong in grid-based navigation and procedural tasks with many agents, but it does not use continuous states or actions, which are important in robotics.

MPE [22, 18] has played an important role in early MARL research, but it cannot scale to hundreds of agents efficiently. VMAS [16] builds on MPE [22, 18] by adding its physics and continuous dynamics, making it better for robotics. However, it can still be slow and difficult to scale to larger agent populations or complex maps.

Many environments also lack evaluation protocols. This makes it difficult to compare algorithms fairly and limits the reproducibility of results. Performance issues also appear in environments that depend heavily on CPU-GPU communication, which slows down training and makes large-scale experiments harder.

Finally, popular robotics simulators such as Gazebo [23], Webots [24], and ARGoS [25] are powerful tools for realistic physics-based simulation and robotic deployment. Each of these frameworks serves specific use cases focused on accurate physical modeling and hardware-oriented evaluation. In contrast, our environment CAMAR is designed to complement these tools by focusing on large-scale MARL and benchmarking, particularly for continuous MAPF tasks. Rather than replacing existing simulators, CAMAR extends the set of available research tools, bridging the gap between large-scale MARL studies and robotics-oriented simulation. A more detailed analysis of environments is provided in Appendix H.

Continuous Observations and Actions Robots usually operate in continuous state and action spaces, so it's important for benchmarks to reflect these conditions. Environments like VMAS and MPE [22, 18] use continuous actions and observations, while others, like Trash Pickup [27] or POGEMA [15], use discrete representations, which limit realism in robotics simulations.

GPU Support In multi-agent environments, each agent often has its observations and rewards. This can lead to heavy data transfers between the CPU and the GPU, especially when training deep RL models. Benchmarks like Jumanji [21] and SMAX [19] are initially built with GPU acceleration, helping to reduce training time. In contrast, many

Environment / Simulator	Repository	Cont. Observations	Cont. Actions	GPU Support	Scalability >500 Agents	Partially observable	Heterogeneous agents	Performance >10K SPS	Python based	Procedural generation	Requires generalization	Evaluation protocols	Tests & CI	PyPI Listed
Waterworld (SISL) [26]	link	✓	✓	✗	✗	✓	✗	✓	✓	✗	✗	✗	✓	✓
RWare [13]	link	✗	✗	✗	✗	✓	✗	✗	✓	✗	✓	✗	✓	✓
RWare (Jumanji) [21]	link	✗	✗	✓	✗	✓	✗	✗	✓	✗	✓	✗	✓	✓
RWare (Pufferlib) [27]	link	✗	✗	✗	✓	✓	✗	✓	✗	✗	✓	✗	✓	✓
Trash Pickup (Pufferlib) [27]	link	✗	✗	✗	✓	✓	✗	✓	✗	✗	✓	✗	✓	✓
SMAC [20]	link	✓	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗
SMACv2 [17]	link	✓	✗	✗	✗	✓	✓	✗	✗	✗	✓	✗	✗	✗
SMAX (JaxMARL) [19]	link	✓	✓	✓	✗	✓	✓	✓	✓	✗	✓	✗	✓	✓
MPE [22, 18]	link	✓	✓	✗	✓	✓	✓	✗ / ✓ [3]	✓	✗	✓	✗	✓	✓
MPE (JaxMARL) [19]	link	✓	✓	✓	✓	✓	✓	✗ / ✓ [3]	✓	✗	✓	✗	✓	✓
JaxNav (JaxMARL) [28]	link	✓	✓	✓	✗	✓	✗	✗	✓	✗	✗	✗	✓	✓
Nocturne [29]	link	✓	✓	✗	✓	✓	✗	✗	✗	✗	✗	✓	✓	✗
POGEMA [14]	link	✗	✗	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓	✓
VMAS [4] [16]	link	✓	✓	✓	✗	✓	✓	✗ / ✓ [4]	✓	✗	✗ / ✓ [4]	✗	✓	✓
SMART [30]	link	✓	✓	✗	✓	✓	✗	✗	✗	✗	✓	✗	✗	✗
Gazebo [23]	link	✓	✓	✓	✗	✓	✓	✗	✗	✗	✗	✗	✓	✗
Webots [24]	link	✓	✓	✓	✗	✓	✓	✗	✗	✗	✗	✗	✓	✗
ARGoS [25]	link	✓	✓	✗	✓	✓	✓	✗	✗	✗	✗	✗	✓	✗
CAMAR (Ours)	link	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 1: Comparison of multi-agent reinforcement learning (MARL) environments and simulators. Each row corresponds to a specific environment or a particular implementation of it. The columns indicate key properties, including support for continuous observations and actions, GPU acceleration, scalability beyond 500 agents, partial observability, heterogeneous agents, and whether the simulator can exceed 10K simulation steps per second (SPS). Additional columns specify if the environment is implemented fully in Python, supports procedural generation, requires generalization across different maps or tasks, includes evaluation protocols, and provides built-in tests or continuous integration (CI). The "Repository" column contains links to the official source code for each environment. CAMAR, listed at the bottom, is our proposed environment.

older environments like SMAC [20, 17] and MPE [22, 18] do not support GPU-based simulation.

Scalability >500 Agents When the number of agents grows, decision-making becomes harder. A good benchmark should scale well to hundreds or thousands of agents. Environments like POGEMA [14, 15] and Trash Pickup [27] can run with thousands and even millions of agents. Others, like VMAS, MPE [22, 18], and SMAC [20, 17], are limited to much smaller groups.

Partially Observable In most real-world scenarios, agents see only part of the environment. This is known as partial observability and is a common feature in almost every RL environment. SMAC [20, 17], RWare [13], and POGEMA [14, 15] support this feature by limiting each agent's view.

Heterogeneous Agents In real life, agents and robots can have different sensors, shapes, or goals. Benchmarks like VMAS and SMAC [20, 17] allow agents to behave differently, making cooperation more complex. Others, like POGEMA [14, 15] or RWare [13], usually involve homogeneous agents.

Performance >10K Steps/s High simulation speed is essential when agents need millions of interactions to learn. For example, the Pufferlib [27] environment can run with a speed of up to 1M steps per second. This allows researchers to train and develop models more quickly. Environments like SMAC [20, 17] or VMAS can be slower, especially when running with many agents.

Python Based Using Python makes it easier for researchers to understand and modify the environment. SMAX [19], VMAS, and POGEMA [14, 15] are implemented entirely in Python. This helps with faster development and integration into learning frameworks.

Procedural Generation Procedural generation helps create diverse tasks that reduce the chance of overfitting. POGEMA [14, 15] and Jumanji [21] use this method to generate complex tasks automatically. Other environments, like RWare [13], use fixed layouts that limit diversity.

Requires Generalization Only a few environments offer separate training and testing tasks, making it difficult to test whether agents generalize well to new situations. POGEMA [14, 15] and SMACv2 [17] include test scenarios that allow researchers to check generalization. Many others

use the same tasks during both training and testing.

Evaluation Protocols To compare algorithms fairly, we need well-defined test cases and metrics. Benchmarks like SMACv2 [17] and Jumanji [21] include evaluation protocols to make results reproducible and meaningful. Many other environments do not have standard evaluation tools or test setups.

Tests & CI CI pipeline and testing suite are vital for collaborative development and maintenance of open-source projects. These practices help ensure code reliability and facilitate contributions from the whole research community.

PyPi Listed When an environment is available on PyPi, it becomes easier for others to install and use. Benchmarks like Jumanji [21], and Trash Pickup [27] are available on PyPi. This lowers the barrier to entry and helps increase adoption in the research community.

CAMAR Environment

CAMAR is designed for continuous-space planning tasks in multi-agent environments. In this environment, multiple agents move toward their goals while avoiding both static obstacles and other moving agents. The simulation happens in a fully continuous two-dimensional space, without any predefined grids. Agents interact by applying forces, which control their movement through a simple and computationally efficient dynamic model. This approach makes the environment more realistic and easier to scale for many agents.

Dynamic Model & Action Space A key part of CAMAR is its collision model. Similar to MPE [22, 18] and VMAS [16], CAMAR uses a force-based system. Agents receive repulsive forces from nearby agents and obstacles. The collision force applied to agent i from object j is calculated using a smooth contact model, as shown in the equation below.

$$\begin{cases} \vec{f}_{ij}^{\text{collision}}(t) = f_0 \frac{\Delta\vec{x}_{ij}(t)}{\|\Delta\vec{x}_{ij}(t)\|} k \log\left(1 + e^{\frac{-(\|\Delta\vec{x}_{ij}(t)\| - d_{\min})}{k}}\right), \\ \text{if } \|\Delta\vec{x}_{ij}(t)\| < d_{\min}; \\ \\ \vec{f}_{ij}^{\text{collision}}(t) = 0, \\ \text{otherwise.} \end{cases}$$

Here, contact force f_0 regulates the magnitude of the repulsive force, penetration softness k controls the smoothness of the contact dynamics, $\Delta\vec{x}_{ij}(t)$ is a displacement vector between agent i and an object j at time t, $d_{\min}$ defines the minimum allowable distance before collision is activated.

When two objects overlap, the force grows smoothly without sudden changes. When there is no overlap, the collision force is zero. This smooth behavior helps keep agent movement more realistic and stable. The total collision force acting on agent i is calculated by summing forces from all nearby objects: $\vec{f}i^c(t) = \sum j \vec{f}_{ij}^{\text{collision}}(t)$.

The full environment state is updated using the collision force and agents' actions. CAMAR supports multiple types of dynamic models. Currently, we provide two built-in models: `HolonomicDynamic` and `DiffDriveDynamic`.

HolonomicDynamic This model is simple and similar to the one used in MPE [22, 18]. Each agent has a position and velocity. The agent moves by applying a 2D force. The next state is calculated using the semi-implicit Euler method, as described in the equation below.

$$\begin{cases} \vec{v}_i(t+dt) = (1 - \texttt{damping})\vec{v}_i(t) + \frac{\vec{f}_i^a(t) + \vec{f}_i^c(t)}{\texttt{m}} dt \\ \vec{v}_i(t+dt) := \begin{cases} \vec{v}_i(t+dt), \text{if } \|\vec{v}_i(t+dt)\| < \texttt{max_speed} \\ \frac{\vec{v}_i(t+dt)}{\|\vec{v}_i(t+dt)\|} \cdot \texttt{max_speed}, \text{otherwise.} \end{cases} \\ \vec{\text{pos}}_i(t+dt) = \vec{\text{pos}}_i(t) + \vec{v}_i(t+dt)dt \end{cases}$$

Here, $\vec{f}_i^a(t)$ is the 2D action force of agent i, `damping` - scalar in the range $[0, 1)$ that controls velocity decay over time, `m` is the agent mass for applying forces, `max_speed` regulates maximum speed of an agent preserving direction but limiting speed, dt is the time step duration between updates.

DiffDriveDynamic Differential-drive robot model is another built-in dynamic in CAMAR. Each agent has a 2D position $\vec{\text{pos}}_i(t)$ and a heading angle $\theta_i(t)$. The agent chooses a 2D action: one value for linear speed $u_i^a(t)$ and one for angular speed $\omega_i^a(t)$. The motion is updated based on this action using equation below.

$$\begin{cases} u_i(t) = \text{clip}(u_i^a(t), -\texttt{max_u}, \texttt{max_u}) \\ \omega_i(t) = \text{clip}(\omega_i^a(t), -\texttt{max_w}, \texttt{max_w}) \\ \vec{v}_i(t) = [u_i(t)\cos(\theta_i(t)); u_i(t)\sin(\theta_i(t)] + \frac{\vec{f}_i^c(t)}{m} dt \\ \vec{\text{pos}}_i(t+dt) = \vec{\text{pos}}_i(t) + \vec{v}_i(t)dt \\ \theta_i(t+dt) = \theta_i(t) + \omega_i(t)dt \end{cases}$$

Here, `max_u` and `max_w` are constraints on agent velocities.

Observations Each agent in CAMAR receives a local observation centered around itself. The size of the observation window can be set by the user. This observation system is inspired by LIDAR sensors but avoids using ray tracing. Instead, CAMAR provides a simple and efficient vector-based observation.

Each agent observes nearby objects using a penetration-based vector representation, which ensures smooth and continuous observations. For every object in the environment (either an agent or a static landmark), the observation is computed as a normalized vector pointing from the agent to the object. If the object is far away, outside the agent's sensing window, the observation becomes a zero vector. This method avoids discontinuities and helps agents better generalize across different object sizes.

[3] SPS decreases gracefully with many agents and obstacles.

[4] VMAS is a framework consisting of many different scenarios, while some scenarios run efficiently with a speed exceeding 10K SPS, other, complex ones don't; the same applies for generalization.

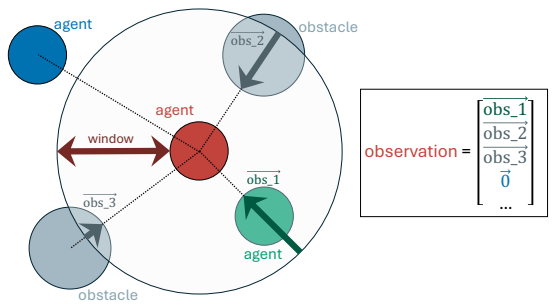


Figure 3: LIDAR-inspired vector observations in CAMAR. Each agent detects nearby objects using penetration vectors, and receives a normalized goal direction.

$$\begin{cases} \Delta\vec{o}_j = \vec{o}_j - \vec{\text{pos}} \\ \vec{\text{obs_j}} = \begin{cases} \Delta\vec{o}_j \cdot \left(1 - \frac{\texttt{window}+R_j}{\|\vec{o}_j\|}\right), \\ \text{if } \|\Delta\vec{o}_j\| - R - R_j < \texttt{window} \\ \vec{0}, \\ \text{otherwise.} \end{cases} \\ \vec{\text{obs_j}} := \frac{\vec{\text{obs_j}}}{\texttt{window}} \end{cases}$$

This observation is computed for each agent in a fully vectorized manner. Afterward, only the top `max_obs` closest objects are kept to form the final observation vector. Here, $\vec{o}_j$ is the 2D position of object j, R is the agent radius, R_j the radius of object j, `window` is a parameter that sets how far the agent can sense objects nearby.

In addition to obstacle information, each agent also gets an ego-centric vector pointing to its goal. This vector is clipped, normalized and concatenated to the final observation. This structure helps agents understand both their surroundings and the direction they need to move.

Circle-Based Discretization One simple way to simulate a world is to use geometry rules to check if objects overlap. Ray tracing is a common method for detecting collisions based on the shapes of objects. However, ray tracing is hard to implement in a way that is fast on a GPU. Simple versions of ray tracing are slow and better suited for CPU simulations.

Another method is to discretize the world and only check collisions with nearby objects. Based on the dynamic model described above, it is enough to calculate the distance between objects.

In CAMAR, every object is represented as a circle. This choice has several advantages. Checking the distance between two circles is simple and fast. It does not require special cases like ray tracing does. It also avoids the complexity of handling different shapes like rectangles or polygons. Because of this, the simulation can easily run on GPUs with many agents at the same time. This design allows CAMAR to simulate large-scale multi-agent tasks efficiently and with high performance.

Map Generators Although every object in CAMAR is represented as a circle, it is still possible to create complex and detailed maps. A complex structure can be made by combining many smaller circles close together. The more circles that are used, the more accurate the map becomes. This method allows users to simulate walls, tunnels, mazes, and other complicated shapes even though the basic element is always a circle.

CAMAR includes several types of built-in maps. It also gives users the ability to add custom map generators, even if they are not compatible with JAX just-in-time compilation [32]. A custom map generator can be connected easily by using the `string_grid` or batched_string_grid formats. This flexibility allows users to design many kinds of environments, from simple random setups to complex and realistic maps.

The current set of built-in maps and generators includes (see Fig. 4):

4a `random_grid`: A map where obstacles, agents, and goals are placed randomly on a grid with a predefined size.

4b `labmaze_grid`: Maps generated using LabMaze[5] [33] - maze generator with connected rooms.

4c `movingai`: Integrated two-dimensional maps from the MovingAI benchmark [31], adapted for continuous planning tasks. They can also be used in a `batched_string_grid` manner.

4d `caves_cont`: A continuous type of map where caves are generated using Perlin noise, a common method in video games for creating realistic and varied landscapes.

4e `string_grid`: A grid map based on a text layout. Obstacles are placed according to characters in a string, similar to the MovingAI benchmark [31]. Agent and goal positions can be fixed or random, depending on the free cells in the string.

4f `batched_string_grid`: Similar to `string_grid`, but supports different obstacle layouts across parallel environments. This allows training on multiple map variations at once.

Reward Function CAMAR uses a scalar reward for each agent at every time step. This reward is the sum of four terms: a goal reward, a collision penalty, a movement-based reward, and a collective success reward:

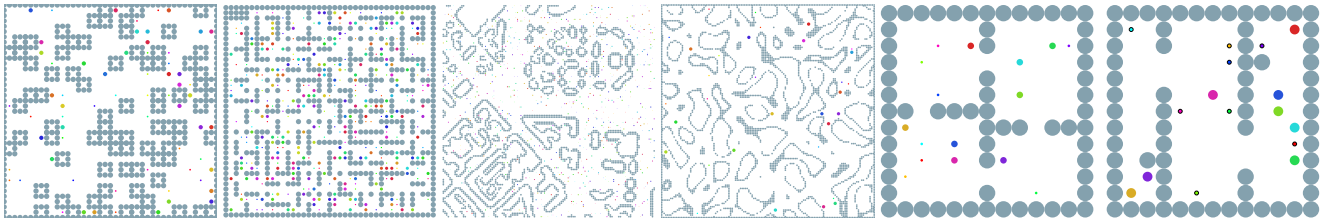
$$r_i(t) = r_{\text{all_g}}(t) + r_{\text{on_g}_i}(t) + r_{\text{collision}_i}(t) + r_{\text{g_dist}_i}(t)$$

The terms are defined as follows:

$$\begin{cases} r_{\text{all_g}}(t) = +0.5, \text{ if } \forall i : \vec{x}_i(t) - \vec{x}_{\text{g}_i}\| \leq R_{\text{g}}; \\ r_{\text{on_g}_i}(t) = +0.5, \text{ if } \|\vec{x}_i(t) - \vec{x}_{\text{g}_i}\| \leq R_{\text{g}}; \\ r_{\text{collision}_i}(t) = -1, \text{ if } \exists j : \|\Delta\vec{x}_{ij}(t)\| < d_{\min}; \\ r_{\text{g_dist}_i}(t) = +\texttt{shaping}\cdot \\ \quad \cdot \left(\|\vec{x}_i(t-dt) - \vec{x}_{\text{g}_i}\| - \|\vec{x}_i(t) - \vec{x}_{\text{g}_i}\|\right) \end{cases}$$

Here, $\vec{x}_i(t)$ is the position of agent i at time t, $\vec{x}_{\text{g}_i}$ is the goal position for agent i, R_{g} is the distance threshold to count as reaching the goal (goal radius), $\Delta\vec{x}_{ij}(t)$ is the vector between agent i and another object j, $d_{\min}$ is the minimal

[5] https://github.com/google-deepmind/labmaze



(a) random_grid (b) labmaze_grid (c) movingai (d) caves_cont (e) string_grid (f) batched_string_grid

Figure 4: A rich collection of maps for multi-agent planning in continuous spaces in CAMAR: support for both continuous and grid landscapes together with MovingAI collection [31].

distance between agent i and object j (for circle objects $d_{\min} = R_i + R_j$ where R_i and R_j are radii), `shaping` is a user-defined coefficient that controls the strength of the movement-based term.

To support cooperation, the environment gives an extra reward when all agents reach their goals. In this case, each agent receives an additional reward of +0.5.

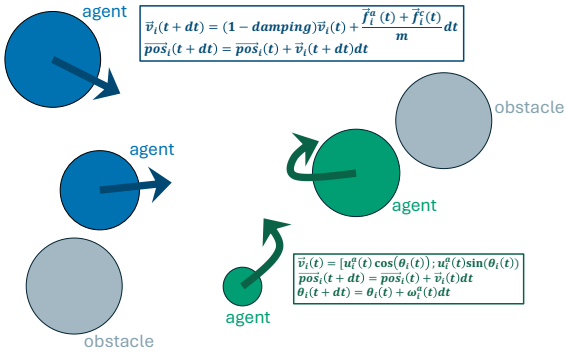


Figure 5: Illustration of heterogeneous agents with different sizes and dynamics supported by CAMAR. Blue agents are governed by `HolonomicDynamic`, while green agents follow `DiffDriveDynamic`. All agents navigate a shared environment while avoiding gray obstacles.

Heterogeneous agents Additionally, CAMAR supports heterogeneous agents in both size and dynamics. All map generators can produce agents with different properties, making it possible to study diverse multi-agent systems inspired by real-world scenarios.

For example, some agents can use `HolonomicDynamic`, while others follow `DiffDriveDynamic`. These agents operate together in a shared space, interact with the same obstacles, and must coordinate their movements despite having different control rules (Fig. 5). Each agent follows its own dynamics model, but all agents contribute to a single global simulation.

CAMAR also supports agents with different sizes. Each agent can have its own radius, which affects how it moves and avoids collisions. This adds complexity to coordination and planning.

Metrics In multi-agent pathfinding research, it is common to use metrics such as success rate (SR), flowtime (FT), makespan (MS), and coordination (CO) to compare different methods. We adopt these metrics in our work and define them more formally as follows in the equation below:

$$\begin{cases} \text{SR} = \frac{1}{N}\sum\limits_{i=1}^{N} \mathbf{1}\{\|\vec{x}_i(T) - \vec{x}_{\text{goal}}\| \leq R_{\text{g}}\}; \\ \text{FT} = \sum\limits_{i=1}^{N} t_i; \\ \text{MS} = \max\limits_{i=1,\dots,N} t_i; \\ \text{CO} = 1 - \frac{C}{N \times T} \end{cases}$$

Here, N is the number of agents, T is the maximum episode length, t_i is the time step when agent i first reaches its goal, $t_i = T$ for unfinished agents, C is the total per-agent-per-timestep number of collisions over the episode.

Evaluation Protocols To support rigorous and reproducible benchmarking, CAMAR includes a standardized suite of evaluation protocols inspired by and extending prior work on cooperative MARL evaluation [34]. We adapt that framework for continuous multi-agent pathfinding, focusing on generalization across both agent count and map structure.

We propose three evaluation tiers: **Easy**, **Medium**, and **Hard**, each targeting a different level of generalization. **Easy** evaluates performance on unseen start and goal positions using the same map type and number of agents as during training. **Medium** tests generalization to maps with similar structure but a different number of agents and obstacle parameters. **Hard** measures generalization to fully unseen map types from the MovingAI street collection, often with a different number of agents than during training.

Each tier follows a defined training and evaluation setup using introduced metrics, aggregated by the Interquartile Mean (IQM) with 95% confidence intervals (CI_{95}) for fair comparison across methods and difficulty levels.

All experiments use fixed JAX [32] random seeds for reproducibility. In total, each protocol involves training multiple models and running thousands of evaluation episodes. We include detailed training and evaluation scripts and provide all protocol maps in the public CAMAR repository.

These protocols help the community better track progress on continuous multi-agent pathfinding and identify which algorithms generalize well to more realistic conditions. Sample

Algorithm	random_grid				labmaze_grid			
	SR ↑	FT ↓	MS ↓	CO ↑	SR ↑	FT ↓	MS ↓	CO ↑
IPPO	0.410±0.001	1695±10	160.0±0.0	1.000±0.000	0.213±0.013	2104±14	160.0±0.0	1.000±0.000
MAPPO	0.830±0.001	984±5	151.4±0.3	1.000±0.000	0.568±0.004	1484±8	160.0±0.0	1.000±0.000
IDDPG	0.335±0.001	1851±10	160.0±0.0	1.000±0.000	0.167±0.000	2772±14	160.0±0.0	0.996±0.000
MADDPG	0.041±0.000	2508±12	160.0±0.0	0.913±0.001	0.027±0.000	2745±12	160.0±0.0	0.854±0.001
ISAC	0.115±0.001	2523±14	160.0±0.0	1.000±0.000	0.047±0.000	2808±12	160.0±0.0	1.000±0.000
MASAC	0.281±0.001	1843±11	160.0±0.0	0.856±0.001	0.105±0.001	2098±12	160.0±0.0	0.781±0.001
RRT*+IPPO	0.420±0.001	1426±9	160.0±0.0	1.000±0.000	0.511±0.001	1316±6	160.0±0.0	0.999±0.000
RRT*+MAPPO	0.828±0.001	971±5	150.4±0.3	1.000±0.000	0.556±0.001	1326±7	160.0±0.0	0.999±0.000
RRT*+IDDPG	0.280±0.001	2181±12	160.0±0.0	1.000±0.000	0.189±0.000	2635±14	160.0±0.0	0.997±0.000
RRT*+MADDPG	0.037±0.000	2953±15	160.1±0.0	0.984±0.000	0.037±0.000	2918±14	160.1±0.0	0.969±0.000
RRT*+ISAC	0.143±0.000	2618±13	160.0±0.0	1.000±0.000	0.058±0.000	2749±13	160.0±0.0	1.000±0.000
RRT*+MASAC	0.054±0.000	2511±14	160.0±0.0	1.000±0.000	0.034±0.000	2854±15	160.0±0.0	0.994±0.000
RRT*+PD	0.678±0.002	2010±59	160.0±0.0	0.997±0.000	0.692±0.004	1807±49	160.0±0.0	0.971±0.002
RRT+PD	0.413±0.014	2440±264	160.0±0.0	0.788±0.021	0.528±0.021	2049±251	160.0±0.0	0.558±0.025

Table 2: Extended performance comparison across different algorithms in the `random_grid` and `labmaze_grid` environments including integrated RRT* with off-policy algorithms additionally. Reported metrics are SR (Success Rate), FT (Flowtime), MS (Makespan), and CO (Coordination), each shown as IQM±CI_{95}. Confidence intervals are symmetric for clarity and computed using 1K bootstrapped samples. Arrows indicate the direction of improvement: ↑ denotes higher is better, ↓ indicates lower is better. Tan boxes highlight the best-performing approach. The CO metric is not colored here for visibility.

efficiency curves, metric-vs-agent-count plots, and performance profiles are supported and recommended for deeper analysis.

Experimental Evaluation

In this section, we evaluate both the scalability and benchmarking capabilities of our environment. We begin by training and testing a set of popular MARL algorithms, as well as classical non-learnable and hybrid methods. These experiments show that the environment supports a wide range of navigation and coordination strategies. We also present results from a simple heterogeneous-agent scenario to demonstrate support for heterogeneous MARL research. Finally, we measure the performance of the simulator in terms of simulation speed, and compare it with VMAS using a shared experimental setup.

Experimental Setup

We evaluate six MARL algorithms: IPPO, MAPPO, IDDPG, MADDPG, ISAC [35], and MASAC. In addition, we include two non-learnable baselines, RRT+PD and RRT*+PD, and two hybrid methods, RRT*+IPPO and RRT*+MAPPO.

All methods are evaluated on two procedurally generated map types: `random_grid` and `labmaze_grid`, each with 6 versions that vary in obstacle density and agent count (8 or 32)[6]. For `labmaze_grid`, an additional connection probability ranging from 0.4 to 1.0 is used to test different maze complexities. Generation details for all training and evaluation maps are provided in Appendix D.

[6]Our current Medium-tier protocol includes tasks with 8 and 32 agents, but we plan to extend it in future versions to support larger agent populations and more complex settings as methods advance.

The "independent" variants (IPPO [1], IDDPG, ISAC) train each agent using its own policy and value function. These methods do not use centralized critics or information sharing across agents. In contrast, the multi-agent versions (MAPPO, MADDPG, MASAC) use centralized critics during training to improve coordination. All approaches use parameter sharing, meaning that agents use the same neural network weights.

Each algorithm is trained for 20M (IPPO, MAPPO) and 2M (IDDPG, MADDPG, ISAC, MASAC) steps per scenario. The training is done independently for each of the 12 map variations. After training, we evaluate the models on both seen and unseen tasks to test their generalization. In total, we train 532 models and evaluate them across 5184 tasks, with 1000 episodes per task. The experiments were run on a single NVIDIA H100 GPU and took around 1000 hours in total.

For the non-learning baselines, we use RRT+PD and RRT*+PD. These methods use classical planning algorithms to generate a path to the goal for each agent. Each path is generated using either RRT (with 50,000 iterations) [36] or RRT* (with 3000 iterations). The agent then follows the path using a simple PD controller.

We also evaluate hybrid methods where agents receive additional RRT* information during training and evaluation. At the start of each episode, RRT* generates sample paths from the goal to the agent's position. These paths and their estimated costs are included in the agent's observation, enabling the policy to learn from approximate cost-to-go values without invoking RRT* at every step.

To evaluate simulator performance, we measure simulation speed in steps per second (SPS) on a 20×20 `random_grid` map with 0.3 obstacle density (120 obstacles). We vary the number of agents and parallel environments to assess CAMAR's scalability. For fair comparison, we benchmark CA-

MAR against VMAS [16] using identical map size, agent count, and a single NVIDIA H100 GPU.

Benchmark

The main results are presented in Table 2, which reports success rate (SR), flowtime (FT), makespan (MS), and coordination (CO) for each algorithm on two types of maps.

On the `random_grid` map, MAPPO achieves the highest SR, with good FT and full CO. Its centralized critic helps agents plan jointly and share useful information. RRT*+MAPPO shows similar SR but performs better in FT and MS, which shows that adding planning support improves movement efficiency.

RRT*+IPPO also improves over IPPO alone, although the gains are smaller. RRT*+PD achieves high SR without learning, but it performs worse in CO. This is expected because it plans for each agent independently. The planner uses full map knowledge but does not consider other agents during execution. As a result, agents sometimes collide in narrow areas.

RRT* with Off-Policy Algorithms We also evaluate hybrid methods that combine RRT* with off-policy MARL algorithms. As shown in Table 2, the results are mixed. Some methods benefit from additional planning information. For example, RRT*+ISAC slightly improves over ISAC alone in both success rate (SR) and coordination (CO), especially in the `random_grid` setting. RRT*+IDDPG also shows moderate improvement, achieving better SR and lower flowtime (FT) than IDDPG.

However, in other cases, the integration leads to weaker results. RRT*+MASAC and RRT*+MADDPG perform poorly, with SR staying below 0.05 in both maps. One possible reason is the increased input size to the models. Off-policy algorithms with centralized critics, such as MASAC and MADDPG, must process long input vectors that include observations and RRT* features from all agents. This can make training unstable and reduce generalization.

Overall, while combining RRT* with off-policy methods can help in some settings, the results suggest that better integration strategies or more expressive policy architectures may be needed to fully benefit from hybrid approaches.

Among MARL methods, IPPO and IDDPG perform well in SR, but their FT is worse than MAPPO. ISAC shows high CO but low SR, possibly due to its focus on smooth motion and risk reduction. MASAC and MADDPG perform poorly. In particular, MADDPG shows very low SR and fails to explore even in the simpler tasks. We investigate these issues in more detail in the Appendix D.

The `labmaze_grid` map is more challenging. Narrow passages make reward signals sparse. In this setting, SR drops across all MARL methods. RRT*+PD performs the best in SR, showing the value of full-path planning when learning signals are limited. However, its CO score remains lower than MARL baselines.

The simpler RRT+PD baseline performs worse than RRT*+PD across all metrics. It produces longer and less efficient paths because RRT does not optimize for path cost and often generates jagged routes. On the `labmaze_grid`, RRT+PD still performs better in SR than many learning-based methods, but with higher FT and lower CO. This result shows that even basic planning methods can outperform MARL baselines in sparse reward environments, but their inability to consider other agents during execution limits coordination and overall efficiency.

These findings highlight a trade-off. Planning methods like RRT*+PD solve complex tasks more reliably, while learning-based policies handle coordination better. To explore this further, we evaluated hybrid methods. These approaches combine RRT* samples with MARL observations to guide exploration. However, results show only small gains or similar performance. One possible reason is the size of the observation vector, which may be too large for the simple policy network to process effectively. Future work may improve integration with better architectures. See Appendix D for more detailed analysis of performance of algorithms.

Heterogeneous Agents

To show CAMAR's support for heterogeneous agents, we modify a simple coordination task called `give_way`. In this new version, two agents must pass through a narrow corridor, but only the smaller red agent can fit into the central chamber. The larger blue agent cannot enter this area and must wait for the other to pass.

We evaluate both standard and heterogeneous versions of IPPO and MAPPO. In the heterogeneous versions (HetIPPO and HetMAPPO), agents use separate policy models. Results in Fig. 6b show that HetIPPO performs better than its shared-policy version. However, HetMAPPO fails, likely because the centralized critic struggles with large and diverse input spaces.

(a) `hetero_give_way` scenario

Algorithm	SR
IPPO	0.5
HetIPPO	0.7
MAPPO	0.5
HetMAPPO	0.0

(b) SR results.

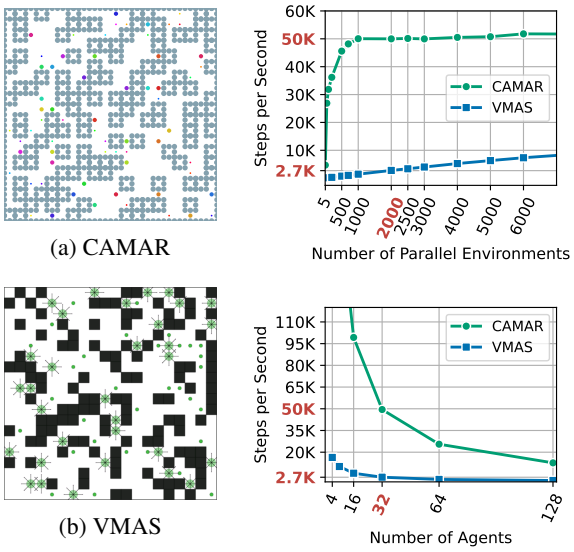
Figure 6: Example of heterogeneous agent coordination (a). Success rates of algorithms are shown in (b).

This experiment shows that CAMAR supports agents with different sizes and models. It can be used to study coordination and policy learning in heterogeneous teams. More details and discussion are provided in the Appendix D.

Scalability Analysis

We evaluate the scalability of our environment by measuring how the simulation speed changes when we increase the number of parallel environments, agents, and obstacles. Fig. 7 and 8 show the results of these experiments.

Fig. 7a shows a snapshot of the CAMAR environment with 32 agents on the 20×20 `random_grid` map. Agents move smoothly while avoiding the 120 fixed obstacles. Fig. 7b shows the same scenario in VMAS [16].



(a) CAMAR

(b) VMAS

Figure 7: Scalability comparison between CAMAR and VMAS across different settings, evaluating the impact of increasing parallel environments, agent count, and obstacles on simulation speed.

First, we measure the SPS as we increase the number of parallel environments from 5 to 7000, keeping 32 agents fixed. VMAS [16] scales almost linearly but reaches less than 10 000 SPS at 7000 environments. In contrast, CAMAR scales quickly up to 1000 environments and then maintains around 50 000 SPS even as more are added. This shows that CAMAR can simulate many parallel tasks without losing speed, which is important for fast training.

Next, we fix the number of environments at 2000 and vary the agent count from 4 to 128. CAMAR keeps over 100 000 SPS with fewer than 16 agents and remains above 10 000 SPS even with 128 agents. VMAS [16] starts near 20 000 SPS with 4 agents but drops sharply to 500 SPS at 128 agents. In this comparison, CAMAR is up to 20x faster than VMAS when many agents act at once.

Overall, CAMAR delivers much higher throughput and keeps strong performance as the number of agents increases. VMAS can boost its speed by adding more parallel environments, but it would need about 50 000 parallel environments to match the throughput that CAMAR provides at 2000 environments. We also compare CAMAR scalability with other modern MARL environments, which can be found in the Appendix E.

We also study CAMAR under more extreme conditions. In Fig. 8, we increase the number of agents up to 800 while keeping the number of circle obstacles fixed at 4160. As the number of agents grows, the simulation speed drops, but our environment still maintains about 1400 SPS with 800 agents. This result shows that CAMAR can simulate very large multi-agent teams and still produce enough experience for learning. It also means that the total number of observations remains high, because each agent generates one observation per step.

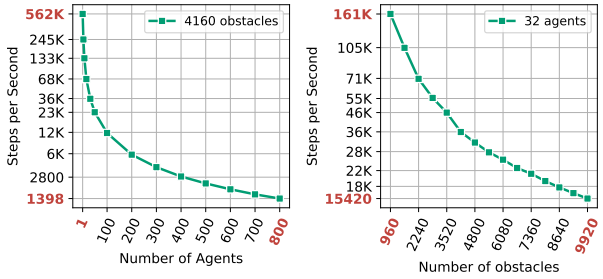


Figure 8: Scalability of CAMAR with increasing numbers of agents and obstacles under more extreme conditions. Despite a decrease in Steps per Second (SPS) as the number of agents grows to 800, CAMAR maintains approximately 1400 SPS, demonstrating its capability to simulate large multi-agent teams. Additionally, although SPS decreases, the total number of gathered observations remains high, as it scales proportionally with the number of agents.

Finally, we test how the number of obstacles affects scalability. We fix the number of agents at 32 and increase the number of obstacles up to 9920. The results, shown in Fig. 8, demonstrate that although SPS decreases with more obstacles, CAMAR still achieves about 15 400 SPS at the highest tested complexity. This indicates that CAMAR remains robust even in very cluttered environments.

In conclusion, CAMAR achieves around 50 000 SPS in complex settings with 32 agents and many obstacles (Fig. 7a), while maintaining stable simulation performance. Overall, the speed in CAMAR depends mainly on the number of simulated objects. Across different tasks, it reaches up to 161 000 SPS with 32 agents and 960 obstacles, and as high as 562 000 SPS when simulating a single agent with 4160 obstacles (Fig. 8), confirming CAMAR's capability to operate at over 100 000 SPS.

Conclusion

This paper introduces CAMAR, a high-performance benchmark for continuous-space multi-agent reinforcement learning. CAMAR combines realistic dynamics with efficient simulation, supporting over 100 000 steps per second using JAX [32]. It includes a diverse set of navigation tasks, a standardized evaluation protocol with built-in metrics, and a range of strong baselines from both classical, learning-based, and hybrid methods. These components enable reliable, scalable, and reproducible evaluation of MARL algorithms.

References

[1] Christian Schroeder De Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip HS Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

[2] Matteo Bettini, Ajay Shankar, and Amanda Prorok. Heterogeneous multi-robot reinforcement learning. In *AAMAS*, 2023.

[3] 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. *Advances in neural information processing systems*, 35:24611–24624, 2022.

[4] Mehul Damani, Zhiyao Luo, Emerson Wenzel, and Guillaume Sartoretti. Primal _2: Pathfinding via reinforcement and imitation multi-agent learning-lifelong. *IEEE Robotics and Automation Letters*, 6(2):2666–2673, 2021.

[5] Alexey Skrynnik, Anton Andreychuk, Maria Nesterova, Konstantin Yakovlev, and Aleksandr Panov. Learn to follow: Decentralized lifelong multi-agent pathfinding via planning and learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17541–17549, 2024.

[6] Anton Andreychuk, Konstantin Yakovlev, Aleksandr Panov, and Alexey Skrynnik. Mapf-gpt: Imitation learning for multi-agent pathfinding at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 23126–23134, 2025.

[7] Brian Angulo, Aleksandr Panov, and Konstantin Yakovlev. Policy optimization to learn adaptive motion primitives in path planning with dynamic obstacles. *IEEE Robotics and Automation Letters*, 8(2):824–831, 2022.

[8] Weinan Chen, Wenzheng Chi, Sehua Ji, Hanjing Ye, Jie Liu, Yunjie Jia, Jiajie Yu, and Jiyu Cheng. A survey of autonomous robots and multi-robot navigation: Perception, planning and collaboration. *Biomimetic Intelligence and Robotics*, page 100203, 2024.

[9] Binyu Wang, Zhe Liu, Qingbiao Li, and Amanda Prorok. Mobile robot path planning in dynamic environments through globally guided reinforcement learning. *IEEE Robotics and Automation Letters*, 5(4):6932–6939, 2020.

[10] Vassilissa Lehoux-Lebacque, Tomi Silander, Christelle Loiodice, Seungjoon Lee, Albert Wang, and Sofia Michel. Multi-agent path finding with real robot dynamics and interdependent tasks for automated warehouses. In *ECAI 2024*, pages 4393–4401. IOS Press, 2024.

[11] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics: Results of the 11th international conference*, pages 621–635. Springer, 2017.

[12] Yunlong Song, Selim Naji, Elia Kaufmann, Antonio Loquercio, and Davide Scaramuzza. Flightmare: A flexible quadrotor simulator. In *Conference on Robot Learning*, pages 1147–1157. PMLR, 2021.

[13] Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. *arXiv preprint arXiv:2006.07869*, 2020.

[14] Alexey Skrynnik, Anton Andreychuk, Konstantin Yakovlev, and Aleksandr I Panov. Pogema: partially observable grid environment for multiple agents. *arXiv preprint arXiv:2206.10944*, 2022.

[15] Alexey Skrynnik, Anton Andreychuk, Anatolii Borzilov, Alexander Chernyavskiy, Konstantin Yakovlev, and Aleksandr Panov. Pogema: A benchmark platform for cooperative multi-agent pathfinding. In *The Thirteenth International Conference on Learning Representations*, 2025.

[16] Matteo Bettini, Ryan Kortvelesy, Jan Blumenkamp, and Amanda Prorok. Vmas: A vectorized multi-agent simulator for collective robot learning. In *International Symposium on Distributed Autonomous Robotic Systems*, pages 42–56. Springer, 2022.

[17] Benjamin Ellis, Jonathan Cook, Skander Moalla, Mikayel Samvelyan, Mingfei Sun, Anuj Mahajan, Jakob Foerster, and Shimon Whiteson. Smacv2: An improved benchmark for cooperative multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 36:37567–37593, 2023.

[18] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

[19] Alexander Rutherford, Benjamin Ellis, Matteo Gallici, Jonathan Cook, Andrei Lupu, Gardar Ingvarsson, Timon Willi, Akbir Khan, Christian Schroeder de Witt, Alexandra Souly, et al. Jaxmarl: Multi-agent rl environments in jax. *arXiv preprint arXiv:2311.10090*, 2023.

[20] Mikayel Samvelyan, Tabish Rashid, Christian Schroeder De Witt, Gregory Farquhar, Nantas Nardelli, Tim GJ Rudner, Chia-Man Hung, Philip HS Torr, Jakob Foerster, and Shimon Whiteson. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*, 2019.

[21] Clément Bonnet, Daniel Luo, Donal Byrne, Shikha Surana, Sasha Abramowitz, Paul Duckworth, Vincent Coyette, Laurence I Midgley, Elshadai Tegegn, Tristan Kalloniatis, et al. Jumanji: a diverse suite of scalable reinforcement learning environments in jax. *arXiv preprint arXiv:2306.09884*, 2023.

[22] Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. *arXiv preprint arXiv:1703.04908*, 2017.

[23] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ international conference on intelligent robots and systems (IROS)(IEEE Cat. No. 04CH37566)*, volume 3, pages 2149–2154. Ieee, 2004.

[24] Olivier Michel. Cyberbotics ltd. webots™: professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, 1(1):5, 2004.

[25] Carlo Pinciroli, Vito Trianni, Rehan O'Grady, Giovanni Pini, Arne Brutschy, Manuele Brambilla, Nithin Mathews, Eliseo Ferrante, Gianni Di Caro, Frederick

Ducatelle, et al. Argos: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm intelligence*, 6:271–295, 2012.

[26] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *Autonomous Agents and Multi-agent Systems: AAMAS 2017 Workshops, Best Papers, São Paulo, Brazil, May 8-12, 2017, Revised Selected Papers 16*, pages 66–83. Springer, 2017.

[27] Joseph Suarez. Pufferlib: Making reinforcement learning libraries and environments play nice. *arXiv preprint arXiv:2406.12905*, 2024.

[28] Alexander Rutherford, Michael Beukman, Timon Willi, Bruno Lacerda, Nick Hawes, and Jakob Nicolaus Foerster. No regrets: Investigating and improving regret approximations for curriculum discovery. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024.

[29] Eugene Vinitsky, Nathan Lichtlé, Xiaomeng Yang, Brandon Amos, and Jakob Foerster. Nocturne: a scalable driving benchmark for bringing multi-agent learning one step closer to the real world. *Advances in Neural Information Processing Systems*, 35:3962–3974, 2022.

[30] Jingtian Yan, Zhifei Li, William Kang, Yulun Zhang, Stephen Smith, and Jiaoyang Li. Advancing mapf towards the real world: A scalable multi-agent realistic testbed (smart). *arXiv preprint arXiv:2503.04798*, 2025.

[31] Nathan R Sturtevant. Benchmarks for grid-based pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(2):144–148, 2012.

[32] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, et al. Jax: composable transformations of python+ numpy programs. *GitHub repository*, 2018.

[33] Charles Beattie, Joel Z Leibo, Denis Teplyashin, Tom Ward, Marcus Wainwright, Heinrich Küttler, Andrew Lefrancq, Simon Green, Víctor Valdés, Amir Sadik, et al. Deepmind lab. *arXiv preprint arXiv:1612.03801*, 2016.

[34] Rihab Gorsane, Omayma Mahjoub, Ruan John de Kock, Roland Dubb, Siddarth Singh, and Arnu Pretorius. Towards a standardised performance evaluation protocol for cooperative marl. *Advances in Neural Information Processing Systems*, 35:5510–5521, 2022.

[35] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. Pmlr, 2018.

[36] Steven LaValle. Rapidly-exploring random trees: A new tool for path planning. *Research Report 9811*, 1998.

[37] Albert Bou, Matteo Bettini, Sebastian Dittert, Vikash Kumar, Shagun Sodhani, Xiaomeng Yang, Gianni De Fabritiis, and Vincent Moens. Torchrl: A data-driven decision-making library for pytorch, 2023.

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

Appendix Contents

Sections	Contents
Appendix A	Evaluation Protocol
Appendix B	Code Examples
Appendix C	Discussion of Limitations
Appendix D	Extended Benchmark Results
Appendix E	Scalability Analysis Details
Appendix F	Future Work and Directions
Appendix G	Implementation Details
Appendix H	Extended Related Work

Appendix A — Evaluation Protocol

Evaluation Suite Overview. To make results easy to compare and fully reproducible, CAMAR adopts the "Standardised Performance Evaluation Protocol for Cooperative MARL" by [34] and *extends* it to continuous multi-agent path-finding. We keep the core ideas, fixed training budgets, multiple random seeds, and strict uncertainty estimates, but add path-finding-specific stress-tests:

- **Axis 1: Agent count.** We vary the number of agents to see whether a method still works when the team grows.
- **Axis 2: Map difficulty.** We change obstacle density and map geometry to check if the learned policy still solves harder layouts.

Three difficulty tiers.

- **Easy** uses the same map, agent count, and obstacle parameters for training and testing. Only the random seeds that place starts and goals change. This tells us whether an algorithm can *solve* the problem at all.
- **Medium** trains on one map setting but evaluates on 12 variants that share the same map types while changing agent count and obstacle density. This probes generalization within the same domain and includes testing generalization across agents. The current protocol is designed for reproducibility and fair comparison across methods. However, since the environment scales efficiently, we plan to extend the Medium tier in future releases to include a wider range of agent counts and task complexities as MARL methods progress.
- **Hard** trains on any user-chosen maps—except the MovingAI street set—and then tests on those unseen street maps

with new agent counts. This is a near-real-world stress test.

Standardized Evaluation Suite for CAMAR

Input. A set of maps $\mathcal{M}$, task sets $\mathcal{T}_m$, and a pool of algorithms $\mathcal{A}$.

1. Default settings

- Training steps T: **2M** (off-policy) or **20M** (on-policy).
- Independent runs R: **3** seeds.
- Evaluation episodes E: **1 000** per interval.
- Evaluation intervals $\mathcal{I}$ (based on available resources): every **10K-100K** steps (off-policy) or **100K-1000K** steps (on-policy).

2. Metrics

- *Returns* G for sample-efficiency plots.
- *Success Rate* (SR), *Flowtime* (FT), *Makespan* (MS), and *Coordination* (CO).
- Per-task: mean G_t^a over E episodes with 95% CIs.
- Per-protocol: build an $(R \times |\mathcal{T}|)$ matrix of normalised returns, then report IQM, optimality gap, probability of improvement, and performance profiles (all with 95% bootstrap CIs).

3. Three difficulty tiers

1. **Easy**: train and test on the same map/agent count. 12 models $\times$ 3 seeds.
2. **Medium**: train on one map, test on all 12 maps and aggregate. 12 models $\times$ 3 seeds.
3. **Hard**: train on any maps (except MovingAI), test on MovingAI [31] street maps with new agent counts. Single model (preferably 3 seeds).

4. Reporting checklist

- Hyper-parameters, network sizes, and compute budget.
- Map generation settings for each tier.
- Final IQM $\pm$95% CI for SR, FT, MS, CO (mandatory). Sample-efficiency curves are recommended for Easy and Medium, optional for Hard. In Hard also plot each metric against the number of agents to test the ability to scale.

Each tier follows the default budget: 2M steps for off-policy and 20M steps for on-policy algorithms, three random seeds, and evaluations every 10K-100K or 100K-1000K steps. We keep the JAX seed fixed at `5` and split it to generate all evaluation keys, which makes every run exactly repeatable.

Reporting and analysis. We require final IQM $\pm$ 95% CI of Success Rate, Flowtime, Makespan, and Coordination. Sample-efficiency curves are strongly recommended for Easy and Medium and optional for Hard if resources allow. For the Hard tier, we also ask for plots of each metric versus the number of agents because scalability is a key question.

Appendix B — Code Examples

The CAMAR library provides a simple and flexible interface for building custom multi-agent pathfinding environments. Below we show two examples that demonstrate how to create environments using different map generators and agent dynamics.

Example 1 (Fig. 9) shows how to create an environment with the `random_grid` map. It uses the string-based API to pass all configuration options directly. This includes the number of agents, the size range for agents and goals, and the dynamic model to use. We set the dynamic to `HolonomicDynamic`, and also customize the observation window and the shaping factor. This example shows how to use heterogeneous sizes for agents and goals, which can help simulate more realistic scenarios.

```python
from camar import camar_v0

env1 = camar_v0(
    map_generator="random_grid",
    dynamic="HolonomicDynamic",
    pos_shaping_factor=2.0,
    window=0.25, # obs window
    max_obs=10, # max num of obj in obs
    map_kwargs={
        "num_agents": 16,
        "agent_rad_range": (0.01,  0.05 ),
        "goal_rad_range":  (0.001, 0.005),
    },
    dynamic_kwargs={
        "mass": 2.0,
    }
)
```

Figure 9: Example 1. Creating a CAMAR environment using random grid maps and holonomic agents. The environment is loaded using the string-based API, which supports configuration via YAML or inline dictionary.

Example 2 (Fig. 10) shows how to use maps from the MovingAI benchmark [31]. These maps are loaded by name using the map generator function movingai(). This example also shows how to use heterogeneous agent dynamics by combining different models (e.g. `DiffDriveDynamic` and `HolonomicDynamic`). This is done using the class-based API with MixedDynamic. The number of agents of each type is specified, and the total is passed to the environment. This allows testing how different types of agents can cooperate in complex environments.

These examples can be used as templates for building new scenarios in CAMAR. Users can adjust map parameters, agent settings, or dynamic models to suit their research needs. The CAMAR design supports quick changes to both environment structure and agent behavior using only a few lines of code.

Appendix C — Limitations

CAMAR offers fast and flexible simulation, but there are still important gaps. First, all built-in scenarios use static obstacles. We do not yet provide maps with moving obstacles, so agents cannot train on fully dynamic scenes. While the engine supports different sizes for agents, goals, and landmarks, the

	Easy	Medium	Hard
Purpose	Test that the method can solve the problem without testing generalization.	Test that the method can solve the problem and generalise across similar map types including varying number of agents.	Test that the method can generalise to near-real-world settings.
How to train and evaluate	**Train on** `map_X` (see Appendix). **Evaluate on** *the same* `map_X`. **Repeat** for all 12 maps	**Train on** `map_X` (see Appendix). **Evaluate on** *all 12* maps. **Repeat** for all 12 maps	**Train on** any map collection excluding MovingAI **Evaluate on** MovingAI street collection with varying agent counts
Number of models	12 trained models × 3 seeds	12 trained models × 3 seeds	1 trained model
Number of evals	12 evaluations × 1K episodes	144 evaluations × 1K episodes	30 evaluations × 1K episodes × varying number of agents
Report	**Metrics** - if needed **sample-efficiency curves** - strongly recommended	**Metrics** - mandatory **sample-efficiency curves** - if needed	**Metrics** - mandatory **sample-efficiency curves** - if resources allow **metrics vs num_agents** - mandatory

Table 3: Overview on 3-tier evaluation protocols presenting the purpose, number of trained models and evaluations for each protocol

```python
from camar import camar_v0
from camar.maps import movingai
from camar.dynamics import MixedDynamic,
     DiffDriveDynamic, HolonomicDynamic

dynamics_batch = [
    DiffDriveDynamic(mass=1.0),
    HolonomicDynamic(mass=10.0),
]
# 8 diffdrive + 24 holonomic = 32 total
num_agents_batch = [8, 24]
mixed_dynamic = MixedDynamic(
    dynamics_batch=dynamics_batch,
    num_agents_batch=num_agents_batch,
)

map_generator = movingai(
    map_names=["street/Denver_0_1024",
               "bg_maps/AR0072SR"],
    num_agents=sum(num_agents_batch),
)

env2 = camar_v0(
    map_generator=map_generator,
    dynamic=mixed_dynamic,
    pos_shaping_factor=2.0,
    window=0.25,
    max_obs=10,
)
```

Figure 10: Example 2. Creating a CAMAR environment with MovingAI maps [31] and mixed agent dynamics. This example demonstrates the class-based API for explicitly specifying heterogeneous agent behavior.

built-in map generators do not include variations in landmark sizes. We believe that adding new maps with obstacles of different sizes and dynamic behavior is an important direction for future work. Fortunately, CAMAR's modular design makes it easy to extend in this way.

Second, although we support two built-in dynamics models (`HolonomicDynamic` and `DiffDriveDynamic`), and allow mixing them to build heterogeneous teams, both use basic integration schemes: either explicit or non-explicit Euler. To achieve stable simulation with larger time steps (for example, dt = 0.1), a frameskip must be applied. Without frameskip, simulation becomes unstable with large dt values, and a smaller integration step like dt = 0.005 must be used together with frameskip = 20. Adding more stable integration methods, such as Runge-Kutta, could improve stability and allow efficient simulation with fewer steps. For example, using Runge-Kutta with dt = 0.05 and frameskip = 2 (for a total of 8 integration steps per forward pass) might offer a better trade-off between speed and accuracy.

Third, there is no built-in communication mechanism between agents. Adding message passing would help study how algorithms use shared information.

Appendix D — Extended Benchmark

Performance Analysis

Aggregate Analysis and Performance Profiles We also report aggregate return metrics across all evaluated methods in Fig.13. These charts, produced using the MARL-EVAL toolkit[34], show median, interquartile mean (IQM), mean return, and optimality gap, with normalized scores. Among MARL baselines, MAPPO performs best across all metrics, followed by IPPO. ISAC and IDDPG achieve lower scores and are less consistent, while MADDPG performs the worst and shows the largest optimality gap. Among hybrid methods, RRT*+MAPPO and RRT*+IPPO achieve strong results that match or exceed their baselines. However, RRT*+MASAC and RRT*+MADDPG perform poorly. This may be caused

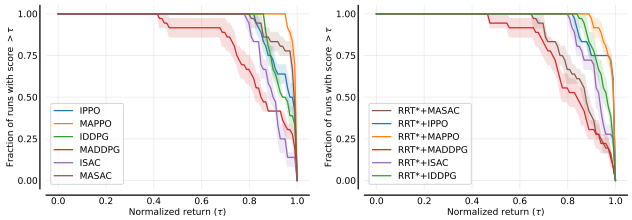


Figure 11: Performance profiles based on normalized return. Each line shows the fraction of evaluation runs that achieved a score above a given threshold τ.

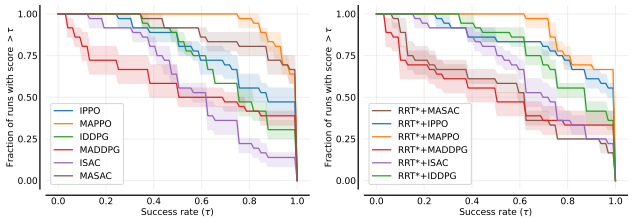


Figure 12: Performance profiles based on success rate. Each line shows the fraction of evaluation runs that achieved SR above a threshold τ.

by the larger input space, especially for centralized critics that struggle with high-dimensional observations.

To further evaluate consistency and reliability, we present performance profiles in Fig.11 and Fig.12. These plots show the fraction of evaluation runs where each algorithm achieves a score above a given threshold. The results confirm our earlier findings: MAPPO and IPPO maintain good performance across tasks, while MADDPG and MASAC drop quickly. RRT*+MAPPO and RRT*+IPPO remain strong among hybrid approaches, while RRT*+MADDPG and RRT*+MASAC again show poor results.

These charts together highlight the differences in generalization ability, robustness, and effectiveness across MARL and hybrid methods. They also support our earlier analysis of SR, FT, MS, CO (Table 2). Overall, these findings show that RRT* integration can improve performance for some methods, but not all. The effect depends on the algorithm and how it handles the added input complexity.

Sample Efficiency. Figure 14 shows the mean success rate evolving during training, computed with the MARL-EVAL toolkit [34]. MAPPO learns the fastest, rising above 0.60 after roughly $3M$ steps and then remaining stable. MASAC improves quickly at first but plateaus below 0.45. IPPO starts slowly yet keeps improving and nearly reaches the MASAC curve near the end of training. IDDPG attains a modest plateau early and then changes little. ISAC climbs slightly in the first million steps and then flattens out, while MADDPG stays under 0.15 for the entire run.

DDPG and SAC analysis We further investigated the weak performance of multi-agent DDPG and SAC algorithms in our benchmark. Specifically, we observed that both MADDPG and MASAC achieved lower success rates and higher

Algorithm	SR ↑	FT ↓	MS ↓	CO ↑	Ep. Return ↑
IDDPG (SC)	0.77	406	134	0.996	266
ISAC (SC)	0.21	1060	160	0.999	92
MADDPG (SC)	0.00	1241	160	0.997	-1.1
MADDPG (NSC)	0.00	1259	160	0.998	-0.9
MASAC (SC)	0.26	813	159	0.996	67
MASAC (NSC)	0.44	780	160	1.000	193

Table 4: Performance comparison of multi-agent DDPG and SAC variants on `random_grid` with 8 agents. SC: shared critic, NSC: not shared critic.

episode lengths compared to their independent counterparts (IDDPG and ISAC). To understand these results, we considered two possible factors that could negatively affect learning with centralized critics.

First, the credit assignment problem becomes harder as the number of agents increases. In both MADDPG and MASAC, a single centralized critic must estimate value based on the combined actions and states of all agents. The critic's target is a simple mean across agents in this version, which can blur important agent-specific differences. This makes learning slow and unstable.

Second, the input size of the centralized critic grows with the number of agents. For example, in our setting with 8 agents, the critic receives a long concatenated vector of all observations and actions. This high-dimensional input makes the learning problem more difficult, especially for relatively small networks like MLPs. Previously discussed performance of RRT*+MASAC and RRT*+MADDPG also supports this hypothesis. These hybrid methods show much lower returns and success rates compared to RRT*+IPPO or RRT*+MAPPO. This drop in performance is likely caused by the added RRT* information increasing the input size even further, making the learning task more difficult for centralized critics.

To test these ideas, we trained new versions of MADDPG and MASAC where the critic network is no longer shared between agents. Instead, each agent has its own critic (without parameter sharing), while still using centralized training. The results are shown in Table 4. For MADDPG, there was no improvement: both the shared and non-shared versions failed to learn the task, with success rate remaining at 0.0. This further supports the hypothesis that MADDPG struggles with large input vectors and credit assignment, even when the critics are separated. In contrast, MASAC improved with non-shared critics: the success rate rose from 0.26 to 0.44 and return nearly tripled. This suggests that SAC can better tolerate large input sizes, likely due to its entropy regularization and smoother policy updates. However, even this improved MASAC variant still performs worse than ISAC, which does not face centralized credit assignment at all. This shows that credit assignment remains a challenge in MASAC, despite its robustness to large inputs. Notably, the poor performance of RRT*+MADDPG and RRT*+MASAC in earlier experiments aligns with these findings.

Together, these findings confirm that both factors—the size of the critic input and the difficulty of shared credit

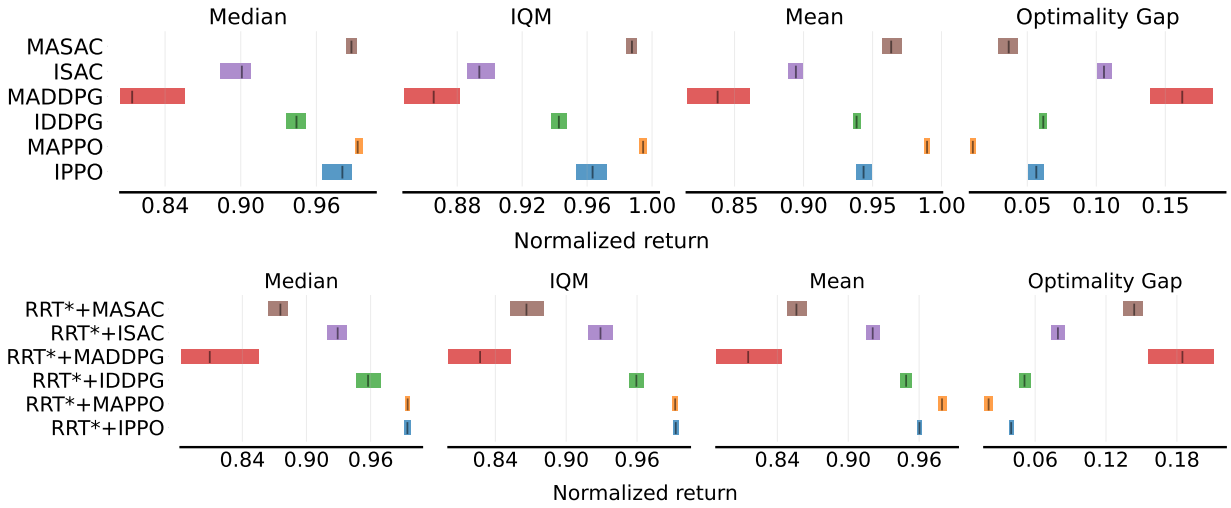


Figure 13: Normalized return scores. Top: MARL baselines. Bottom: RRT* hybrid methods. Metrics include median, interquartile mean (IQM), mean, and optimality gap. Higher values are better except for optimality gap. Each bar shows 95% confidence intervals.

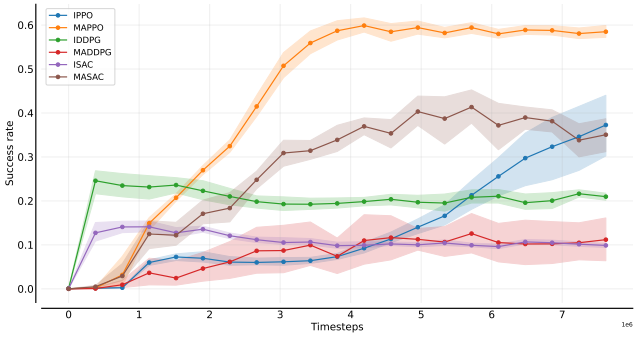


Figure 14: Sample-efficiency curves: mean success rate versus environment steps. Computed using MARL-EVAL.

assignment—can reduce the performance of centralized off-policy MARL methods. We include these results in Table 4.

Benchmark Maps

CAMAR includes three benchmark map types designed to evaluate navigation and coordination in multi-agent scenarios. These maps differ in layout, complexity, and agent configurations.

random_grid map is a 20 × 20 grid with randomly placed rectangular obstacles (with circle-based discretization). Agents are initialized in free cells, and goals are placed at randomly chosen locations. The scenario uses holonomic dynamics and includes parameters for obstacle density, agent radius, goal radius, and others. During benchmarking, we generate six different `random_grid` instances: three maps with 8 agents and three with 32 agents. Each of these variations is evaluated under three different obstacle densities (0.0, 0.05, 0.15), which define how cluttered the map becomes. These tasks are relatively simple but still useful to test multi-agent pathfinding (MAPF) algorithms.

labmaze_grid is based on procedural maze generation. It creates more structured maps using multiple rectangular rooms and corridors. The parameter `extra_connection_probability` controls how many connections exist between rooms: 1.0 means the map is fully connected, while lower values introduce more isolated areas and bottlenecks. As with the random grid, we evaluate 6 instances: three maps with 8 agents and three with 32 agents, each under three different connection probabilities (0.4, 0.65, 1.0). The dynamics are the same as in the random grid, allowing consistent comparison across layouts. These tasks provide harder challenges for coordination and navigation.

These benchmark maps are scalable, fast to simulate, and allow flexible control over structure and difficulty. Together, they form a comprehensive testbed for MAPF.

Heterogeneous Support

The `hetero_give_way` map tests heterogeneous multi-agent coordination. It is based on a simple corridor scenario where one larger agent cannot enter the central chamber, while the smaller agent can pass through. This setup forces agents to learn implicit turn-taking or yielding behavior. Although the dynamics remain the same (`HolonomicDynamic`), this configuration highlights differences in agent capabilities. Basic algorithms like MAPPO and IPPO struggle in this task. Nonetheless, CAMAR supports such heterogeneous settings out of the box and can be used to advance research in coordination strategies for agents with diverse sizes, dynamics, and behaviors.

Appendix E — Extended Scalability Analysis

We also study CAMAR scalability in comparison to other popular MARL environments to better understand its limits.

Comparing with Other Environments. In Table 5, we compare CAMAR to other environments with GPU support like JaxMARL and Jumanji. CAMAR is faster than VMAS and the MPE baselines, and it reaches speeds close to the best GPU simulators. For example, CAMAR runs at 338K SPS with 4 agents and 50K SPS with 32 agents. Some JaxMARL environments reach higher throughput—for example, SMAX achieves up to 1.2M SPS—but their performance drops more quickly as the number of agents grows. JaxNav, for instance, cannot run with more than 32 agents. While SMAX and RWARE show strong throughput, they use simplified settings like open space or discrete grids and do not support procedurally generated obstacle layouts. CAMAR, in contrast, supports continuous control and dense obstacles, which brings more realism. It trades a small drop in speed for a richer environment that better matches real-world MAPF challenges.

Agents	4	8	16	32	64	128
CAMAR	338K	189K	99K	50K	25K	13K
VMAS	16K	10K	5.5K	2.7K	1.2K	447
JaxNav	159K	61K	13.6K	353	-	-
SMAX	0.8M	1.0M	1.2M	0.8M	0.3M	0.1M
RWARE	1.4M	1.1M	0.8M	0.5M	0.3M	0.1M
MPE-2	0.6M	0.8M	1.2M	0.7M	0.5M	0.2M
MPE-1120	11.8K	11.2K	11.9K	11.1K	9.7K	7.7K

Table 5: SPS vs number of agents. Comparison across environments with GPU support. MPE-2 uses the default simple-tag configuration with 2 circle obstacles, while MPE-1120 uses 1120 circles matching the number of circle obstacles in CAMAR. '-' indicates that the environment was too slow to even finish episode.

Appendix F — Future Work

We highlight several research directions that can be explored using CAMAR:

1. CAMAR enables testing with large agent populations, supporting the development of scalable multi-agent reinforcement learning (MARL) methods. Future work can focus on improving how algorithms handle hundreds of agents operating in dense, dynamic environments.
2. CAMAR provides a natural platform for studying inter-agent communication. Large-scale coordination often requires agents to exchange information with nearby teammates, which remains an open challenge. The environment offers the necessary tools to explore such localized communication mechanisms.
3. CAMAR includes realistic continuous navigation tasks, making it suitable for integrating classical planning approaches (RRT, RRT*) with learning-based agents. Combining long-term planning with reinforcement learning could improve both sample efficiency and robustness.
4. CAMAR supports agents with different dynamics and properties, enabling research on heterogeneous teams. We plan to extend the current list of dynamics. For the moment, we include differential-drive and holonomic robots, but we are going to support more options, such as car-like and 2D quadrotor dynamics.
5. CAMAR's fully modular design allows users to easily redefine observation and reward functions through wrapping. We plan to introduce additional observation types, including simulated 2D LiDAR, to further diversify sensing modalities for navigation tasks.
6. Another direction of future extensions is to extend classical baselines beyond the current RRT+PD and RRT* setup by adding widely used multi-robot navigation methods such as PRM+PD (Probabilistic roadmap for global planning with PD controller), ORCA, and MPPI. Similarly, we will expand learning-based baselines to include RNN-based algorithms and architectures using graph attention or transformers, going beyond the standard MLP-based backbones.
7. We aim to incorporate post-generation connectivity checks to ensure that all agents can reach their goals. While computationally expensive, this process will improve the reliability and fairness of evaluation.
8. CAMAR can be extended beyond multi-agent pathfinding. Future releases will include new task types such as pick-up-and-delivery, cooperative transportation, and dynamic obstacle scenarios, making CAMAR more general testbed for large-scale MARL research.

We hope CAMAR will support the community in studying these and other open problems in multi-agent learning and planning.

Appendix G — Implementation Details

Integrations

CAMAR is designed to work easily with modern reinforcement learning tools. It follows the Gymnax interface, which is already familiar to many researchers working with RL environments on top of JAX [32].

We also provide a wrapper for TorchRL [37], which allows users to integrate CAMAR into PyTorch-based pipelines. In addition, CAMAR supports integration with BenchMARL [38], a framework for evaluating MARL algorithms. These integrations make it easy to use CAMAR with popular RL frameworks.

Vectorized setup

To maintain high simulation speed, CAMAR uses efficient vectorized operations based on JAX [32]. Agents with the same dynamic model are grouped together and updated in parallel. For agent sizes, two cases are supported: if all agents have the same radius, it is treated as a constant during JAX compilation [32], giving faster simulation. If agents have different sizes, their radii are passed as vectors being part of the environment state, which still allows efficient processing and supports randomized sizes during training.

Appendix H — Extended Related Work

Waterworld (SISL) — The Waterworld environment, part of the SISL (Stanford Intelligent Systems Laboratory) suite

[26], is a continuous control benchmark where multiple agents move in a bounded two-dimensional space to collect moving targets ("food") while avoiding harmful objects ("poison"). Agents have continuous observations and actions, and their motion dynamics are simple and lightweight, which allows high simulation speeds for training.

The environment places a small number of obstacles randomly in the scene, with the default setup containing only one obstacle. This limited variation does not require agents to generalize across different maps in a meaningful way. Waterworld does not support heterogeneous agents, evaluation protocols, or complex multi-stage tasks, and the scenario remains structurally the same across runs.

Multi-Robot Warehouse (RWare) — The Multi-Robot Warehouse environment [13] simulates robots moving in a warehouse to collect and deliver requested goods. The layout is grid-based, and agents must coordinate to navigate around shelves and other robots. The default version is implemented in Python using a discrete state and action space. It supports partial observability but does not provide procedural generation: the warehouse layout is fixed. As a result, agents do not need to generalize across different maps.

Several re-implementations of RWare exist. The Jumanji version [21] rewrites the environment in JAX, which improves simulation speed and allows hardware acceleration on GPUs, but keeps the original fixed-layout design and task structure. The PufferLib version [27] modifies the environment to support large-scale parallel simulation.

None of the RWare versions support heterogeneous agents, continuous control, or evaluation protocols. Despite this, RWare remains a widely used benchmark for discrete-space multi-agent pathfinding and cooperative delivery tasks.

Trash Pickup (PufferLib) — The Trash Pickup environment [27] is a grid-based multi-agent task where agents move around a map to collect pieces of trash and deliver them to designated drop-off locations. It is implemented in C with a Python API for controlling agents, which allows efficient simulation. The environment uses discrete state and action spaces and supports partial observability.

The layout is fixed, and trash positions follow a predefined spawn pattern. While trash locations may vary between episodes, the underlying map structure remains the same. This means the environment does not require generalization across different maps. The design supports scalability to more than 500 agents.

The Trash Pickup benchmark does not provide continuous control, heterogeneous agents, or formal evaluation protocols. However, it offers a simple and repeatable cooperative task that can be scaled to large numbers of agents, making it a useful test case for studying coordination efficiency in discrete environments.

StarCraft Multi-Agent Challenge (SMAC) — The StarCraft Multi-Agent Challenge [20] is one of the most widely used benchmarks in the MARL community. In SMAC, a team of StarCraft II units controlled by independent agents must cooperate to defeat an opposing team. The environment is partially observable, and by default uses discrete action spaces. The underlying maps are fixed, which allows agents to solve the tasks without relying on observation inputs by simply memorizing the optimal action sequence for each map. This significantly reduces the need for generalization across scenarios.

SMACv2 — SMACv2 [17] addresses one of the key limitations of SMAC by introducing randomly generated maps with random positions of units, which prevents memorization of fixed action sequences and forces agents to generalize their policies across start positions. The rest of the environment remains the same as SMAC, with partially observable states and discrete actions by default. Like SMAC, SMACv2 does not include an evaluation protocol, but it can be evaluated using the protocol proposed in [34].

SMAX (JaxMARL) — SMAX [19] is a JAX-based re-implementation of SMACv2 that leverages hardware acceleration for faster simulation. In addition to performance improvements, SMAX introduces the option to switch from discrete to continuous action spaces, making it more flexible for testing different types of MARL algorithms.

Multi-Agent Particle Environment (MPE) — The Multi-Agent Particle Environment [22, 18] is a lightweight 2D simulator for testing cooperative, competitive, and communication multi-agent tasks. Agents and landmarks are represented as simple geometric shapes, and their interactions follow basic physical dynamics. MPE supports both continuous and discrete action spaces, and scenarios can be either fully or partially observable. Procedural generation is not used in the default scenarios, although obstacle or landmark positions can be randomized in some tasks. This means generalization across different maps is not strictly required. The environment does not include a built-in evaluation protocol.

MPE (JaxMARL) — The JaxMARL re-implementation of MPE [19] offers hardware acceleration through JAX, which enables much faster simulation compared to the original Python implementation. Functionally, it mirrors MPE in terms of available scenarios, physics, and agent capabilities. It supports both continuous and discrete action spaces. Like the original MPE, it does not include procedural map generation by default and does not provide an evaluation protocol.

JaxNav (JaxMARL) — JaxNav [28] is a navigation-focused benchmark implemented within the JaxMARL framework. It places agents in a continuous 2D space where they must navigate to goal locations while avoiding static obstacles. The environment supports both continuous and discrete action spaces. Maps are fixed in the default setup, so agents do not need to generalize to unseen layouts. JaxNav is implemented in JAX for hardware acceleration but does not provide an evaluation protocol.

Nocturne — Nocturne [29] is a 2D partially observed driving simulator implemented in C++ for performance, with a Python API for training and evaluation. It focuses on realistic autonomous driving scenarios sourced from the Waymo Open Dataset. The environment provides a set of benchmark tasks in which agents control autonomous vehicles interacting with

traffic participants whose trajectories are taken from recorded data. This setup captures complex multi-agent interactions requiring coordination, prediction of other agents' intentions, and handling of partial observability.

Nocturne does not use procedurally generated maps, as all scenes are drawn from fixed datasets. While the simulator benefits from its C++ implementation, it does not provide GPU acceleration, which limits its throughput for large-scale MARL experiments.

POGEMA — The Partially Observable Grid Environment for Multiple Agents [14] is a grid-based benchmark for partially observable multi-agent pathfinding. Agents operate with ego-centric local observations and must reach individual goals while avoiding collisions. A key feature of POGEMA is its procedural map generation, producing diverse layouts—random maps, mazes, and warehouse-style scenarios—that require agents to generalize to unseen environments. It also offers integrated evaluation protocols with standardized metrics, enabling consistent comparison across reinforcement learning, planning, and hybrid methods. Implemented in Python, POGEMA supports scalable multi-agent experiments with both classical and learning-based policies.

VMAS — The Vectorized Multi-Agent Simulator for Collective Robot Learning [16] is a PyTorch-based, vectorized 2D physics framework designed for efficient multi-robot benchmarking. It includes a modular interface for defining custom scenarios, alongside a set of built-in multi-robot tasks, that each involve a relatively small number of agents. VMAS stands out for its GPU-accelerated, batch-simulated environments. It supports inter-agent communication and sensors (e.g., LIDAR).

SMART — Scalable Multi-Agent Realistic Testbed [30] is a physics-based simulator tailored for bridging Multi-Agent Path Finding (MAPF) algorithms and real-world performance. Designed for scalability, SMART supports simulation of thousands of agents, enabling evaluation of large-scale deployments. However, its simulation speed is relatively low (around 335 SPS for 100 agents), which can limit large-scale reinforcement learning experiments. The platform targets both academic researchers and industrial users who lack access to extensive physical robot fleets, offering a realistic environment to test MAPF performance under near-real conditions.

Gazebo — Gazebo is a widely used open-source 3D robotics simulator that integrates tightly with ROS. It supports detailed physics simulation, realistic sensors (e.g., LIDAR, cameras), and high-quality rendering. While Gazebo allows multi-robot experimentation and moderate scalability, it does not support GPU acceleration for simulation logic - GPU use is limited to rendering and sensor visuals only. This limitation can make simulations of large multi-agent systems slow or resource-heavy.

Webots — Webots is a commercial-grade 3D robotics simulator offering a wide array of built-in robot models, sensors, and actuator libraries. It provides realistic physics simulation, high-quality graphics, and ROS support. Webots is well-suited for prototyping and academic or industrial robotic research. However, from a multi-agent perspective, it's not optimized for running large numbers of agents at scale—simulation speed tends to drop significantly as agent count grows, making massive multi-robot evaluations challenging.

ARGoS — ARGoS [25] is an open-source, modular simulator designed specifically for swarm robotics and large-scale multi-agent systems. Its architecture supports running thousands of simple agents efficiently by combining multiple physics engines and leveraging parallel computation. ARGoS is customizable, allowing researchers to define their own robots, sensors, actuators, etc.