CAMAR: Continuous Actions Multi-Agent Routing

Artem Pshenitsyn[1,2], **Aleksandr Panov**[1,2], **Alexey Skrynnik**[1,2]

[1]CogAI Lab, Moscow, Russia
[2]MIRAI, Moscow, Russia
pshenitsyn@cogailab.com, skrynnikalexey@gmail.com

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.

Appendix with extended discussions can be found at arxiv version of the paper — https://arxiv.org/abs/2508.12845

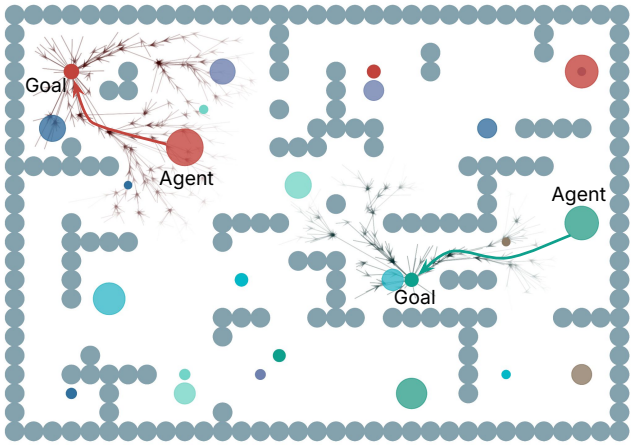


Figure 1: An example scenario from the proposed CAMAR benchmark. Agents are represented as filled circles. Each agent aims to reach its goal while avoiding collisions. The small arrows for the red and green agents indicate segments of paths generated by RRT*, providing guidance for the RL algorithms.

Introduction

Multi-agent reinforcement learning (MARL) has shown strong results in cooperative and competitive settings, and many recent studies explore how MARL can solve tasks that need coordination in complex environments (De Witt et al. 2020; Bettini, Shankar, and Prorok 2023; Yu et al. 2022; Damani et al. 2021; Skrynnik et al. 2024; Andreychuk et al. 2025). One important group of such tasks is multi-agent pathfinding (MAPF), where several agents must reach their goals without collisions. Classic MAPF is usually studied on discrete grids, but real robots move in continuous space, follow dynamics, and must plan smooth paths. This continuous version of MAPF is important for many domains such as warehouse logistics, drone swarm coordination, and other systems where many robots must move safely and efficiently.

Learnable methods have recently become effective for MAPF and cooperative navigation (Skrynnik et al. 2024; Andreychuk et al. 2025; Damani et al. 2021). However, many existing environments still use grid worlds or discrete actions that do not match real robot behavior. In practice, robots must plan and coordinate while avoiding both static and dynamic obstacles, a long-standing problem in robotics (Chen et al. 2024; Wang et al. 2020; Angulo, Panov, and Yakovlev 2022; Lehoux-Lebacque et al. 2024). Many MARL environments simplify this problem too much by using low-dimensional maps or unrealistic movement models.

High-fidelity simulators like Gazebo[1], Isaac Sim[2], AirSim (Shah et al. 2017), and Flightmare (Song et al. 2021) offer realistic physics, but they focus on robot control and perception, not on large-scale MARL. These tools often run slower and cannot simulate hundreds of agents at once. To study large-scale continuous navigation, we need environments that combine high speed with continuous dynamics, run efficiently on GPU, and allow testing both learned and

[1]https://gazebosim.org/home
[2]https://developer.nvidia.com/isaac/sim

planning-based methods in a single benchmark.

We identify three main gaps in current MARL environments. First, many use discrete action spaces that cannot represent smooth motion (Papoudakis et al. 2020; Skrynnik et al. 2022, 2025). Second, while some environments do support continuous states and actions, they do not scale to large numbers of agents and obstacles (Bettini et al. 2022). Third, other environments can scale but offer simple tasks that do not require strong coordination or realistic navigation skills (Ellis et al. 2023; Lowe et al. 2017; Rutherford et al. 2023).

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

To compare existing MARL environments, we evaluate them in Table 1 across key features such as continuous control, GPU support, scalability, and usability. An extended version of the Table 1 with descriptions of comparison features and detailed analysis of each environment are provided in Appendix H.

Prominent benchmarks include SMAC (Samvelyan et al. 2019; Ellis et al. 2023; Rutherford et al. 2023), Jumanji (Bonnet et al. 2023), POGEMA (Skrynnik et al. 2022, 2025), MPE (Mordatch and Abbeel 2017; Lowe et al. 2017), and VMAS (Bettini et al. 2022). SMAC enables testing of strategic behavior but uses discrete actions and scales poorly. Jumanji supports GPUs and procedural generation but is not focused on navigation. POGEMA handles large-agent navigation with procedural maps but lacks continuous control. MPE, while foundational, does not scale to many agents; VMAS adds physical realism but still struggles with performance and scalability.

Many environments lack evaluation protocols and suffer from slow training due to CPU-GPU bottlenecks. Simulators like Gazebo (Koenig and Howard 2004), Webots (Michel 2004), and ARGoS (Pinciroli et al. 2012) offer realistic continuous dynamics but are not optimized for efficient MARL training. These gaps highlight the need for a new benchmark supporting scalable, high-performance multi-agent learning.

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 (Mordatch and Abbeel 2017; Lowe et al. 2017) and VMAS (Bettini et al. 2022), 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.

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

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 (Mordatch and Abbeel 2017; Lowe et al. 2017). 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.

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

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

Environment / Simulator	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
RWare (Jumanji) (Bonnet et al. 2023)	✗	✗	✓	✗	✓	✗	✗	✓	✗	✓	✗	✓
SMAC (Samvelyan et al. 2019)	✓	✗	✗	✗	✓	✓	✗	✗	✗	✗	✗	✗
SMACv2 (Ellis et al. 2023)	✓	✗	✗	✗	✓	✓	✗	✗	✗	✓	✗	✗
SMAX (JaxMARL) (Rutherford et al. 2023)	✓	✓	✓	✗	✓	✓	✓	✓	✗	✓	✗	✓
MPE (Mordatch and Abbeel 2017; Lowe et al. 2017)	✓	✓	✗	✓	✓	✓	✗/✓ [3]	✓	✗	✓	✗	✓
MPE (JaxMARL) (Rutherford et al. 2023)	✓	✓	✓	✓	✓	✓	✗/✓ [3]	✓	✗	✓	✗	✓
POGEMA (Skrynnik et al. 2022)	✗	✗	✗	✓	✓	✗	✓	✓	✓	✓	✓	✓
VMAS [4] (Bettini et al. 2022)	✓	✓	✓	✗	✓	✓	✗/✓ [4]	✓	✗	✗/✓ [4]	✗	✓
Gazebo (Koenig and Howard 2004)	✓	✓	✓	✗	✓	✓	✗	✗	✗	✗	✗	✓
Webots (Michel 2004)	✓	✓	✓	✗	✓	✓	✗	✗	✗	✗	✗	✓
ARGoS (Pinciroli et al. 2012)	✓	✓	✗	✓	✓	✓	✗	✗	✗	✗	✗	✓
CAMAR (Ours)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table 1: Comparison of different MARL environments and simulators.

time, `m` is the agent mass for applying forces, `max_v` 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

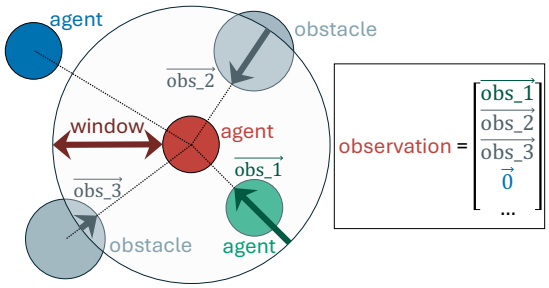


Figure 3: LIDAR-inspired vector observations in CAMAR. Each agent detects nearby objects using penetration vectors, and receives a normalized goal direction.

(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.

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

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

[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.

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 (Bradbury et al. 2018). 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] (Beattie et al. 2016) - maze generator with connected rooms.

4c `movingai`: Integrated two-dimensional maps from the MovingAI benchmark (Sturtevant 2012), adapted for continuous planning tasks. They can also be used in a `batched_string_grid` manner.

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

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 (Sturtevant 2012). 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:

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

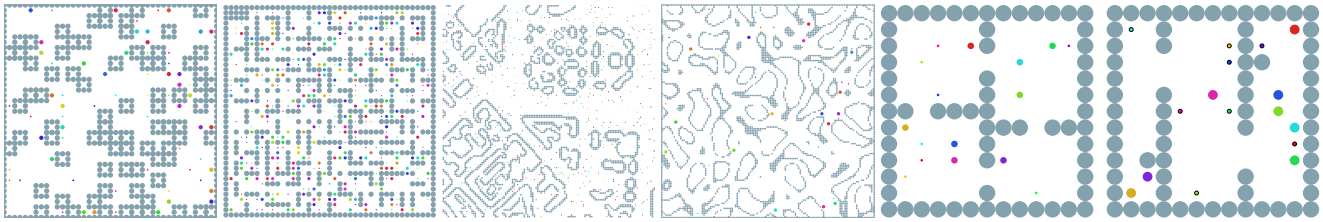
Here, $\vec{x}_i(t)$ is the position of agent i at time t, $\vec{x}_{\text{g}_i}$ is the goal position for agent i, R_{g} is the distance threshold to count as reaching the goal (goal radius), $\Delta\vec{x}_{ij}(t)$ is the vector between agent i and another object j, $d_{\min}$ is the minimal distance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

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.



(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 (Sturtevant 2012).

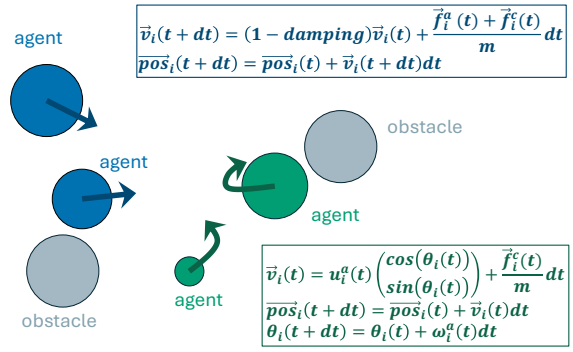


Figure 5: Illustration of heterogeneous agents with different sizes and dynamics supported by CAMAR. Blue agents are governed by `HolonomicDynamic`, while green agents follow `DiffDriveDynamic`. All agents navigate a shared environment while avoiding gray obstacles.

Metrics The evaluation metrics are defined as follows: the success rate is $\text{SR} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}\{\|\vec{x}_i(T) - \vec{x}_{\text{goal}}\| \leq R_{\text{g}}\}$; the flowtime is $\text{FT} = \frac{1}{N}\sum_{i=1}^{N} t_i$; the makespan is $\text{MS} = \max_{i=1,\ldots,N} t_i$; and the coordination is $\text{CO} = 1 - \frac{C}{N \times T}$. 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 (Gorsane et al. 2022). 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 (Bradbury et al. 2018) 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 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 6 MARL algorithms: IPPO, MAPPO, IDDPG, MADDPG, ISAC (Haarnoja et al. 2018), and MASAC. In addition, we include 2 non-learnable baselines, RRT+PD and RRT*+PD, and 6 hybrid methods: RRT*+IPPO, RRT*+MAPPO, RRT*+IDDPG, RRT+MADDPG, RRT*+ISAC, and RRT*+MASAC.

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

[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.

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: Performance comparison across different algorithms in the random_grid and labmaze_grid environments. 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. **Bold values** highlight the best-performing approach.

complexities. Generation details for all training and evaluation maps are provided in Appendix D.

The "independent" variants (IPPO (De Witt et al. 2020), 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) (LaValle 1998) 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 CAMAR against VMAS (Bettini et al. 2022) using identical map size, agent count, and a single NVIDIA H100 GPU.

Benchmark

The main results are shown in Table 2. On the random_grid map, MAPPO reaches the highest success rate with strong coordination. Adding planning improves efficiency. RRT*+MAPPO gives faster routes than MAPPO, and RRT*+IPPO improves over IPPO. The classic RRT*+PD baseline reaches high success without learning, but shows low coordination because it plans for each agent alone.

For off-policy algorithms, results are mixed. RRT* improves ISAC and IDDPG slightly, but weakens MASAC and MADDPG. These methods use a centralized critic and must process long input vectors that include RRT* features. This makes training unstable and harms generalization. Among MARL baselines, IPPO and IDDPG reach good success but slower flowtime than MAPPO. MASAC and MADDPG fail in both maps.

The labmaze_grid setting is harder due to narrow corridors and sparse rewards. All MARL baselines drop in success. RRT*+PD performs best, which shows the value of full-path planning when learning signals are weak. The simpler RRT+PD planner gives worse paths but still beats many MARL baselines in success.

More detailed analysis of these results and hybrid methods is given in Appendix D.

Heterogeneous Agents

To demonstrate support for heterogeneous teams, we extend the give_way task so that two agents must pass through a

(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).

narrow corridor where only the smaller red agent can enter the central chamber. The larger blue agent must wait.

We compare IPPO and MAPPO with shared policies to their heterogeneous versions, where each agent has its own model. As shown in Fig. 6b, HetIPPO performs better, but HetMAPPO fails, likely because the centralized critic cannot handle the larger and more diverse input space.

This experiment shows that CAMAR can model agents with different sizes and abilities and is suitable for studying coordination in heterogeneous teams. More details are in Appendix D.

Scalability Analysis

We study how CAMAR scales when we increase the number of parallel environments, agents, and obstacles. The full results are shown in Fig. 7. Visual examples of CAMAR and VMAS running the same scenario are provided in Appendix E.

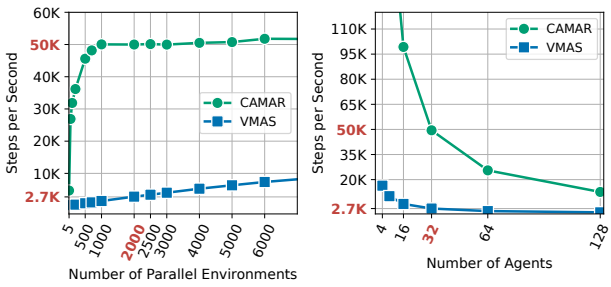
We first fix the number of agents at 32 and measure the simulation speed when we increase the number of parallel environments from 5 to 6000+. VMAS scales roughly linearly but stays below 10 000 steps per second (SPS) at 6000 environments. In contrast, CAMAR rises quickly to about 1000 environments and then stays close to 50 000 SPS even as we add more parallel environments. This shows that CAMAR can support fast and stable training with many vectorized environments.

Next, we fix the number of environments at 2000 and increase the number of agents from 4 to 128. CAMAR keeps more than 100 000 SPS when the agent count is below 16 and remains above 10 000 SPS even at 128 agents. VMAS begins near 20 000 SPS with 4 agents but drops to about 500 SPS at 128 agents. In this setting, CAMAR is up to 20 times faster when many agents act together.

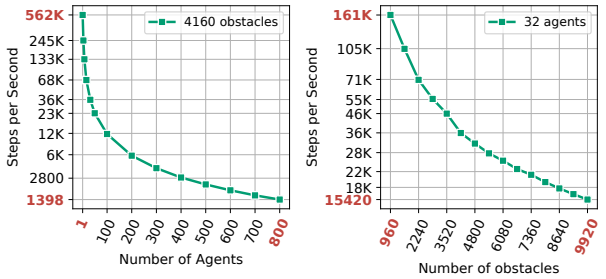
We also test extreme conditions by keeping 4160 fixed and increasing the number of agents up to 800. The speed goes down as more agents are added, but CAMAR still maintains about 1400 SPS at 800 agents. Since every agent produces one observation per step, the total amount of data remains high.

Finally, we test how obstacle count alone affects speed. We fix the number of agents at 32 and increase the number of obstacles up to 9920. CAMAR still reaches about 15 000 SPS in this most cluttered setting. This shows that the system remains robust even in very dense maps.

In summary, CAMAR achieves around 50 000 SPS on complex scenarios with many obstacles and 32 agents. The



(a) CAMAR vs VMAS with increasing number of environments, agents, and obstacles.



(b) CAMAR handles up to 800 agents, maintaining 1400 SPS and high observation throughput.

Figure 7: CAMAR achieves robust scalability in multi-agent simulation, surpassing VMAS in performance across varied settings and supporting efficient simulation of large agent populations with high observation rates.

speed depends mainly on the number of objects in the scene. In other tasks it reaches up to 161 000 SPS with 32 agents and 960 obstacles, and up to 562 000 SPS when simulating a single agent with 4160 obstacles. These results confirm that CAMAR can operate above 100 000 SPS. Visualisations of the test scenes are shown in Appendix E. Further comparisons with other MARL environments are also given in Appendix E.

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 (Bradbury et al. 2018). 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.

Acknowledgments

The study was supported by the Ministry of Economic Development of the Russian Federation (agreement No. 139-15-2025-013, dated June 20, 2025, IGK 000000C313925P4B0002).

References

Andreychuk, A.; Yakovlev, K.; Panov, A.; and Skrynnik, A. 2025. Mapf-gpt: Imitation learning for multi-agent pathfinding at scale. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 23126–23134.

Angulo, B.; Panov, A.; and Yakovlev, K. 2022. Policy optimization to learn adaptive motion primitives in path planning with dynamic obstacles. *IEEE Robotics and Automation Letters*, 8(2): 824–831.

Beattie, C.; Leibo, J. Z.; Teplyashin, D.; Ward, T.; Wainwright, M.; Küttler, H.; Lefrancq, A.; Green, S.; Valdés, V.; Sadik, A.; et al. 2016. Deepmind lab. *arXiv preprint arXiv:1612.03801*.

Bettini, M.; Kortvelesy, R.; Blumenkamp, J.; and Prorok, A. 2022. Vmas: A vectorized multi-agent simulator for collective robot learning. In *International Symposium on Distributed Autonomous Robotic Systems*, 42–56. Springer.

Bettini, M.; Shankar, A.; and Prorok, A. 2023. Heterogeneous Multi-Robot Reinforcement Learning. In *AAMAS*.

Bonnet, C.; Luo, D.; Byrne, D.; Surana, S.; Abramowitz, S.; Duckworth, P.; Coyette, V.; Midgley, L. I.; Tegegn, E.; Kalloniatis, T.; et al. 2023. Jumanji: a diverse suite of scalable reinforcement learning environments in jax. *arXiv preprint arXiv:2306.09884*.

Bradbury, J.; Frostig, R.; Hawkins, P.; Johnson, M. J.; Leary, C.; Maclaurin, D.; Necula, G.; Paszke, A.; VanderPlas, J.; Wanderman-Milne, S.; et al. 2018. JAX: composable transformations of Python+ NumPy programs. *GitHub repository*.

Chen, W.; Chi, W.; Ji, S.; Ye, H.; Liu, J.; Jia, Y.; Yu, J.; and Cheng, J. 2024. A survey of autonomous robots and multi-robot navigation: Perception, planning and collaboration. *Biomimetic Intelligence and Robotics*, 100203.

Damani, M.; Luo, Z.; Wenzel, E.; and Sartoretti, G. 2021. PRIMAL _2: Pathfinding via reinforcement and imitation multi-agent learning-lifelong. *IEEE Robotics and Automation Letters*, 6(2): 2666–2673.

De Witt, C. S.; Gupta, T.; Makoviichuk, D.; Makoviychuk, V.; Torr, P. H.; Sun, M.; and Whiteson, S. 2020. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*.

Ellis, B.; Cook, J.; Moalla, S.; Samvelyan, M.; Sun, M.; Mahajan, A.; Foerster, J.; and Whiteson, S. 2023. Smacv2: An improved benchmark for cooperative multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 36: 37567–37593.

Gorsane, R.; Mahjoub, O.; de Kock, R. J.; Dubb, R.; Singh, S.; and Pretorius, A. 2022. Towards a standardised performance evaluation protocol for cooperative marl. *Advances in Neural Information Processing Systems*, 35: 5510–5521.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, 1861–1870. Pmlr.

Koenig, N.; and Howard, A. 2004. 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, 2149–2154. Ieee.

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

Lehoux-Lebacque, V.; Silander, T.; Loiodice, C.; Lee, S.; Wang, A.; and Michel, S. 2024. Multi-Agent Path Finding with Real Robot Dynamics and Interdependent Tasks for Automated Warehouses. In *ECAI 2024*, 4393–4401. IOS Press.

Lowe, R.; Wu, Y. I.; Tamar, A.; Harb, J.; Pieter Abbeel, O.; and Mordatch, I. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30.

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

Mordatch, I.; and Abbeel, P. 2017. Emergence of Grounded Compositional Language in Multi-Agent Populations. *arXiv preprint arXiv:1703.04908*.

Papoudakis, G.; Christianos, F.; Schäfer, L.; and Albrecht, S. V. 2020. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. *arXiv preprint arXiv:2006.07869*.

Pinciroli, C.; Trianni, V.; O'Grady, R.; Pini, G.; Brutschy, A.; Brambilla, M.; Mathews, N.; Ferrante, E.; Di Caro, G.; Ducatelle, F.; et al. 2012. ARGoS: a modular, parallel, multi-engine simulator for multi-robot systems. *Swarm intelligence*, 6: 271–295.

Rutherford, A.; Ellis, B.; Gallici, M.; Cook, J.; Lupu, A.; Ingvarsson, G.; Willi, T.; Khan, A.; de Witt, C. S.; Souly, A.; et al. 2023. Jaxmarl: Multi-agent rl environments in jax. *arXiv preprint arXiv:2311.10090*.

Samvelyan, M.; Rashid, T.; De Witt, C. S.; Farquhar, G.; Nardelli, N.; Rudner, T. G.; Hung, C.-M.; Torr, P. H.; Foerster, J.; and Whiteson, S. 2019. The starcraft multi-agent challenge. *arXiv preprint arXiv:1902.04043*.

Shah, S.; Dey, D.; Lovett, C.; and Kapoor, A. 2017. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics: Results of the 11th international conference*, 621–635. Springer.

Skrynnik, A.; Andreychuk, A.; Borzilov, A.; Chernyavskiy, A.; Yakovlev, K.; and Panov, A. 2025. POGEMA: A Benchmark Platform for Cooperative Multi-Agent Pathfinding. In *The Thirteenth International Conference on Learning Representations*.

Skrynnik, A.; Andreychuk, A.; Nesterova, M.; Yakovlev, K.; and Panov, A. 2024. Learn to follow: Decentralized lifelong multi-agent pathfinding via planning and learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 17541–17549.

Skrynnik, A.; Andreychuk, A.; Yakovlev, K.; and Panov, A. I. 2022. POGEMA: partially observable grid environment for multiple agents. *arXiv preprint arXiv:2206.10944*.

Song, Y.; Naji, S.; Kaufmann, E.; Loquercio, A.; and Scaramuzza, D. 2021. Flightmare: A flexible quadrotor simulator. In *Conference on Robot Learning*, 1147–1157. PMLR.

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

Wang, B.; Liu, Z.; Li, Q.; and Prorok, A. 2020. Mobile robot path planning in dynamic environments through globally guided reinforcement learning. *IEEE Robotics and Automation Letters*, 5(4): 6932–6939.

Yu, C.; Velu, A.; Vinitsky, E.; Gao, J.; Wang, Y.; Bayen, A.; and Wu, Y. 2022. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in neural information processing systems*, 35: 24611–24624.