Computers & Graphics 133 (2025) 104467

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Special Section on STAG 2024

Render, Encode, Plan: A simple pipeline for hybrid RL-DL learning inside Unreal Engine☆

Daniele Della Pietra [a], Nicola Garau [a,b,*]

[a] University of Trento, Via Sommarive 14, Trento, 38123, Italy
[b] CNIT - Consorzio Nazionale Interuniversitario per le Telecomunicazioni, Italy

ARTICLE INFO

Dataset link: https://mmlab-cv.github.io/REP/

Keywords:
Reinforcement learning
Computer graphics
Game engines
Intelligent agents

ABSTRACT

Learning is an iterative process that requires multiple forms of interaction with the environment. During learning, we experience the world through the repetition of observations and actions, gaining an insight into which combination of these leads to the best results, according to our goals. The same paradigm has been applied to traditional reinforcement learning (RL) over the years, with impressive results in 3D navigation and planning. On the other hand, the computer vision community has been focusing mostly on vision-related tasks (e.g. classification, segmentation, depth estimation) using deep learning (DL). We present **REP: Render, Encode, Plan**, a unified framework to train embodied agents of different kinds (humanoids, vehicles, and drones) inside Unreal Engine, showing how a combination of RL and DL can help to shape intelligent agents that can better sense the surrounding environment. The main advantage of our method is the combination of different sensory modalities, including game state observations and vision features, that allow the agents to share a similar structure in their observations and rewards, while defining separate rewards based on their goals. We demonstrate impressive generalization capabilities on large-scale realistic 3D environments and on multiple dynamically changing scenarios, with different goals and rewards. All code, complete experiments, and environments will be available at https://mmlab-cv.github.io/REP/.

1. Introduction

Intelligent agents are autonomous entities that can perceive the surrounding environment, make decisions based on their perceptions and internal goals, and take actions to achieve those goals, with some degree of autonomy, adaptability, reasoning and goal-oriented behavior. Intelligent agents can be *embodied*, meaning that they interact with the world through a physical or simulated body. Moreover, the definition of *digital twin* can be applied to intelligent embodied agents whenever a specific physical system is represented by a synchronized digital replica, allowing real-time monitoring, prediction, simulation and optimization.

Intelligent agents can interact with the environment through *actions*, with the objective of maximizing *cumulative rewards*, usually through reinforcement learning (RL). Unlike supervised, self-supervised and unsupervised learning, reinforcement learning does not rely on a fixed data set for training, but it dynamically receives *observations*, which are continuously gathered from the environment. For these reasons, RL is particularly well suited for highly dynamic and complex scenarios, such as video games, simulations, autonomous driving, and robotics.

Before RL gained popularity, nonplayer characters (NPCs) and intelligent agents were typically controlled using *finite state machines* (FSMs) and *behavior trees* (BTs) [1,2]. These modular control structures are easy to understand, interpret, and debug, making them particularly popular in commercial game development and simulation environments. FSMs model agent behavior as a collection of discrete states with hand-defined transitions, while BTs extend this concept by introducing a hierarchical and more flexible decision structure. However, despite their robustness and simplicity, both FSMs and BTs are inherently limited by their scripted nature. Designing such structures is time-consuming and requires domain expertise to anticipate every possible scenario the agent may encounter. As a result, they struggle to generalize to unseen or dynamically changing environments and often exhibit repetitive or predictable behaviors once players discover their logic [3]. In contrast, learning-based approaches – especially those employing reinforcement learning – enable agents to adapt their policies based on interaction experience rather than static design, allowing for more flexible and emergent behaviors. Nonetheless, this shift from handcrafted logic to learned behavior introduces new challenges, such as sample

☆ This article is part of a Special issue entitled: 'Special Section STAG2024' published in Computers & Graphics.

* Corresponding author at: University of Trento, Via Sommarive 14, Trento, 38123, Italy.
E-mail addresses: daniele.dellapietra@unitn.it (D.D. Pietra), nicola.garau@unitn.it (N. Garau).

https://doi.org/10.1016/j.cag.2025.104467
Received 4 April 2025; Received in revised form 31 October 2025; Accepted 3 November 2025
Available online 13 November 2025

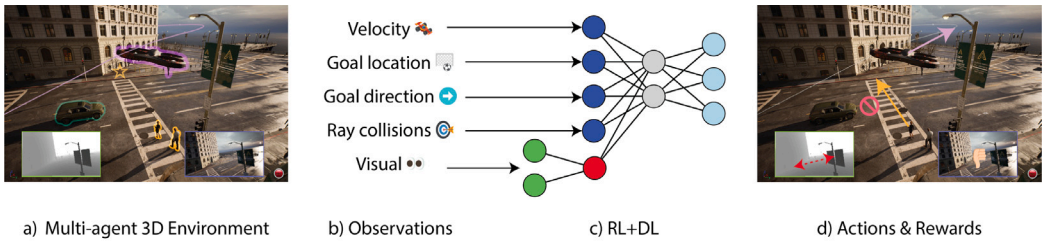

Fig. 1. We present *REP - Render, Encode, Plan*, a novel framework to train different kinds of intelligent agents inside Unreal Engine 5. **(a)** Different kinds of agents can be spawned into arbitrarily large 3D environments, while **(b)** capturing diverse kinds of observations (including visual), that **(c)** are then encoded an hybrid RL+DL procedure. The encoded observations give rise to **(d)** a set of actions for each agent, that receives rewards to improve over time.

inefficiency, stability issues, and reduced interpretability, highlighting the ongoing trade-off between controllability and adaptability in NPC design.

On the other hand, NPCs and intelligent agents usually require real-time decision-making, which is possible through decision trees but hard if modeled using deep learning (DL) alone, since that would require creating enormous datasets with a multitude of different observations, as well as fine-tuning them when new scenarios take place.

Current research in RL has focused on learning from non-visual observations [4], such as ray collisions, distances, lidar and sonar, achieving promising results and adequate generalization to new environments. However, these kinds of observations are inherently sparse and do not capture semantic details at all. Several works in the literature have proposed to train RL agents directly from pixels, demonstrating that visual cues can be used as a substitute for classic observations in RL, despite the increased complexity due to the high input resolution ($W \times H \times C$ pixels, W = width, H = height, C = channels). For an overview of deep reinforcement learning methods in character animation, virtual crowds and navigation, see the survey by Kwiatkowski et al. [5], which discusses how visual and multi-modal observations are used in modern frameworks. Some attempts have been made to cast the problem of reinforcement learning as conditional sequence modeling using transformers [6], demonstrating that RL can benefit from the recent technical advancements of DL.

One important example of combining RL with other learning paradigms is the mixing of supervised and reinforcement learning from human feedback in ChatGPT [7] and other generative pre-trained transformers. This hybrid DL-RL approach combines the strengths of deep learning in *static* tasks like text and image generation and understanding, with the creation of reward models to aid generalization to previously unseen scenarios.

Partially inspired by this double-sided approach, our goal is to enable the design, training, and deployment of RL-DL hybrid intelligent embodied agents, combining the best of the two words in a simple, streamlined way.

2. Contributions

In this paper, we present **Render, Encode, Plan (REP)**, a simple framework for training intelligent embodied agents inside Unreal Engine 5, using multiple kinds of observations and actions. REP allows the developers to easily combine multiple kinds of sensory modalities for different agents, while focusing on shaping the reward function. Moreover, we fully integrate our work with Eclipse Ditto, making it possible to train intelligent embodied agents that behave like digital twins of their physical counterpart, unlocking the possibility of deploying the synthetically trained agents in the real world.

We build on previous works [8,9], that focused on pedestrians mostly. Dynamic Crowd Routing [8] employed only the game state observations to train humanoid agents to behave like a crowd of pedestrians, while Peek-a-bot [9] agents were allowed to use pre-trained visual observations only. REP generalizes these two ideas by **enabling a single architecture that can handle both kinds of observations simultaneously**. Unlike previous works, we introduce support for **multiple agent classes** (humanoids, vehicles and drones), describe our shared-memory communication scheme for training custom visual encoders, and provide a publicly available implementation that can be integrated with new tasks (see Table 1). Furthermore, we allow the user to train a DNN to parse visual stimuli and encode them into visual observations, by using a dedicated fast shared memory portion that we implemented inside the Learning Agents [10] RL framework. At inference time, the trained neural network can be loaded into NNE [11] for faster inference, effectively combining the pros of DL and RL into a unified architecture, as can be seen in Fig. 1. We provide results on multiple scenarios that use game state observations, visual observations, and a combination of both, demonstrating the effectiveness of our approach.

To clarify the contribution of this paper with respect to other existing methods, we summarize our key novelties here:

- **In-engine RL+DL training.** To our knowledge, REP is the only existing toolkit that allows the in-engine training of intelligent agents using a combination of deep and reinforcement learning tasks that go beyond game state observations, such as visual or sensory data coming from cameras and other sensors. This opens up new possibilities for the real-world generalization of in-engine trained digital twins.
- **Unified multi-modal framework.** We design a single unified observation vector that integrates physical observations (e.g. velocities, ray sensors) with learned visual embeddings. This unification allows heterogeneous agents (vehicles, humanoids and drones) to use a common pipeline while retaining task-specific reward functions, simultaneously sharing the same training loop for enhanced training speed.
- **Shared-memory inference and NNE integration.** We implement a fast shared-memory interface within the Learning Agents library to transfer images to a neural encoder on the host machine and stream the resulting features back into the RL loop. At inference time the trained encoder can be deployed using the Neural Network Engine (NNE) for real-time execution.
- **Digital-twin support.** Through integration with Eclipse Ditto, we synchronize Unreal Engine agents with their physical counterparts, providing a basis for digital-twin training and real-world deployment. Industry reports emphasize that digital twins provide the foundation for reinforcement learning–driven decision support and automation [12].
- **Expanded evaluation.** We extend previous benchmarks with new agent types (cars and drones) and more complex scenarios that combine dynamic goals, dynamic obstacles and visual navigation. Ablation studies and comparisons with game-state-only baselines are included to demonstrate the benefit of multi-modal observations.

Additional videos, supplementary results, code, data, and qualitative comparisons can be found on the following page: https://mmlab-cv.github.io/REP/.

Table 1
Overview of the state-of-the-art frameworks that allow the creation of intelligent agents inside game engines. Our **REP** framework can generalize to multiple kinds of tasks and applications (games, simulations, navigation and robotics), by using a combination of physical and visual observations. The majority of the works have been implemented inside custom or highly specialized game engines, or in general with a reduced feature set. Recent works [8,9,13] have been focusing on implementing intelligent agents inside Unreal Engine 5, which allows to simulate much more photorealistic and complex environments, compared to the widely adopted OpenAI Gym.

Area	Application field	Method	Observation type	Multi-task agents	Engine	Environment complexity
Robotics	Robotic arm	[14]	Visual only	✗	MuJoCO, PyGame	Bounded
	Robotic arm	[15]	Visual only	✗	OpenAI Gym	Bounded
	Robot soccer	[16]	Visual only	✗	MuJoCo	Bounded
Navigation	Indoor navigation	[17]	Visual only	✗	OpenAI Gym	Bounded
	Indoor navigation	[18]	Visual + keypoints	✓	Gibson Dataset	Bounded
	Indoor navigation	[19]	Visual + memory module	✗	Gibson Dataset	Bounded
	Indoor navigation	[20]	Visual + language	✗	Gibson Dataset	Bounded
	Indoor navigation	[21]	Multi-sensory	✗	Matterport3D	Bounded
	Indoor navigation	[8]	Game state	✓	Unreal Engine 5	Bounded
	Drone navigation	[22]	Visual + game state	✗	Gazebo	Bounded
	Drone navigation	[23]	Game state	✗	C++ based engine	Bounded
Gaming	Atari games	[24]	Visual only	✓	OpenAI Gym	Bounded
	Doom	[25]	Visual + game state	✗	ViZDoom	Bounded
	2D+3D games	[26]	Visual only	✓	OpenAI Gym, ViZDoom	Open-world
	Quake III	[27]	Visual + game state	✗	DeepMind Lab	Open-world
	Fortnite	[13]	Game state	✓	Unreal Engine 5	Open-world
	Dota 2	[28]	Game state	✓	Source 2	Open-world
	StarCraft II	[29]	Game state	✓	StarCraft II Engine	Open-world
Multiple	3D games, Navigation, Robotics	[9]	Visual only	✓	Unreal Engine 5	Open-world
		REP	Visual + game state	✓	Unreal Engine 5	Open-world

3. Related work

3.1. Game state-based RL

Game state-based reinforcement learning (RL) uses structured environmental states rather than raw visual data. Classic benchmarks such as MuJoCo [30] and OpenAI Gym [31] offer symbolic state information (e.g., velocities, positions), simplifying the learning process. Notable successes include AlphaStar, achieving mastery in StarCraft II [29], and OpenAI's Dota agent [28], demonstrating professional-level performance. Structured state inputs remain vital for robotics and autonomous vehicles, notably drones, allowing rapid learning of complex flight dynamics [23,32]. Moreover, behavioral cloning from structured states has proven efficient for training robust agents [33]. Recent studies have also applied state-based RL in commercial gaming environments such as Fortnite [13] and complex navigation tasks [8].

3.2. Vision-based RL

Vision-based RL directly utilizes visual input, compelling agents to extract relevant features for decision-making. Mnih et al. [24] demonstrated pixel-based RL achieving human-level Atari game performance. Navigation tasks extensively use visual RL, such as egocentric indoor navigation [17–20]. Intrinsic curiosity-driven exploration methods [34] further promote exploration through prediction errors from visual data. Significant progress has also been achieved in vision-based RL for robotic applications such as robot soccer [16] and robotic arm [14,15]. Furthermore, multi-task agents have been trained in complex simulated environments including 2D and 3D games [25–27].

Combining RL with deep learning (DL) significantly enhances learning sophisticated representations. Representation learning frameworks such as ResNet [35], ViT [36], ConvNext [37], and Swin Transformers [38] promote efficient RL training. Self-supervised learning methods (DINO [39], SimCLR [40], SimSiam [41], SWAV [42]) further enhance RL agent generalization. Decision Transformer [6] integrates transformer-based models into RL, treating decision-making as sequence modeling. Hybrid RL+DL approaches combining model-based dynamics with neural policies enable precise drone maneuvers [22]. Object detection models such as Faster R-CNN [43] and YOLO [44] have been leveraged within RL for tasks involving precise object localization. Applications integrating visual-language signals [20], memory modules [19], and multi-modal sensor fusion [21] have been crafted using a RL+DL approach.

3.3. Learning inside game engines

During the past few years, there has been an increasing interest in integrating learning algorithms -in particular reinforcement learning- inside game engines, to facilitate the scripting of complex NPC and enemies inside video games. One of the first to successfully provide a working toolkit for this purpose was Unity with the ML Agents framework [45]. Following the success of ML Agents, other game engines like Unreal Engine and Godot started supporting RL agent training with realistic environments and advanced physics simulations. Godot's RL Agents [46] enables seamless RL integration across various scenarios. Unreal Engine has developed specialized tools such as Learning Agents [10] and Neural Network Engine (NNE) [11], facilitating in-engine deep RL training and real-time inference. Similarly, Unity recently announced their own Inference Engine, for running neural network inference inside the engine. Recent studies utilized Unreal Engine extensively, such as dynamic crowd simulation [8], human-like behavior modeling in Fortnite [13], and integrated visual RL training pipelines [9]. The major drawback of all in-game learning implementations is that they mostly allow the training of RL-only networks, with a separate toolkit for inference. With our solution, we propose to unify inference and training in a single package, allowing the user to use pre-trained frozen backbones as feature extractors for more complex tasks (i.e. visual navigation guided by object detection or segmentation). The REP framework specifically highlights a multi-task approach within Unreal Engine 5, demonstrating generalizability across robotics, navigation, and gaming tasks by combining physical and visual observations in complex open-world environments.

3.4. Multi-modal RL and digital twins

Recent research explores reinforcement learning agents that fuse multiple sensory modalities – visual, proprioceptive and semantic signals – to achieve robust behavior across domains. For character

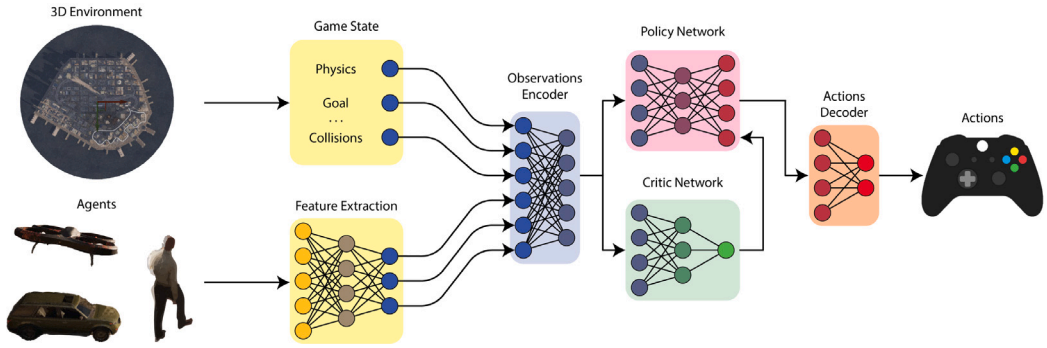

Fig. 2. Simplified schematic of our **REP** architecture. The goal here is to map physical and visual observations from multiple kinds of agents from a big, open world, into a meaningful set of actions. We do so by first creating an encoding of the visual observations using a DNN encoder. Then, we concatenate the normalized physical observations (i.e. game state) with the newly created visual observations. We then train the agents to optimize a policy that produces the best set of actions for a given reward function, using the PPO algorithm. At inference time, the same diagram applies, but the computation of the visual features runs in-engine using NNE [11] for optimal performance.

animation and crowd simulation, Kwiatkowski et al. [5] survey deep RL methods that leverage multimodal observations such as pixels, depth maps, skeletal joints and language instructions to control virtual characters and large crowds. Multi-modal RL has also been applied to navigate crowds in dynamic environments while preserving group coherence and social norms. Beyond simulation, digital-twin frameworks are emerging as a bridge between simulated training and real-world deployment, especially in very specific scenarios such as soft robot arms [47]. Industry reports highlight that digital twins provide a foundation for reinforcement learning–driven decision support and automation [12]. Our work follows this direction by integrating Unreal Engine with Eclipse Ditto to synchronize our simulated agents with their physical counterparts during both training and inference.

4. Observations and rewards

Differently from actions, rewards and observations can be agent-agnostic. In this section, we describe the pool of possible observations and associated rewards that the agents in REP can be associated with. However, these are not fixed and can be extended to fit even more challenging scenarios. An overview of our architecture can be seen in Fig. 2. For a more exhaustive discussion of the underlying PPO training pipeline, including the hyper-parameters configuration and formal specification of the training procedure, please refer to [9].

4.1. Observations

For each agent $a \in A$, where A is the pool of agents, and for every discrete timestep t, we can define a set of possible observations:

- Agent velocity: $v_t^a = \frac{p_t^a - p_{t-1}^a}{\Delta t}$, where p_t^a is the normalized world position of agent a at timestep t and Δt is expressed as the elapsed time in seconds between timestep $t-1$ and timestep t.
- Goal location: $g_t^a = g_t - p_t^a$, represents the relative coordinates of the desired goal with respect to the current position of the agent. In the case of spline-based trajectories, goal locations can be expressed as relative positions of the closest S intermediate control points $g_t^S = g_s - p_t^a \quad \forall s \in S$.
- Goal direction: $d_t^a = \frac{R_t^a(g_t - p_t^a)}{\|R_t^a(g_t - p_t^a)\|}$, the directional versor of the goal position expressed in the agent's local coordinate frame. This coordinate frame is defined by the rotation matrix R_t^a (the agent's forward, right, and up vectors). A similar observation d_t^S can be expressed for the closest S spline points g_s.
- Rays observations: $r_{t,n\in N}^a$, distances measured by N simulated rays emanating from the agent position p_t^a in predetermined directions. These rays detect obstacles and other environmental features, returning distances to the nearest intersection points, or a maximum sensing distance if no intersection occurs.
- Visual features: ϕ_t^a, features extracted from visual input, such as RGB camera images or semantic segmentation outputs. These may include information regarding nearby agents, static and dynamic obstacles, terrain details, or contextual information from the environment.

Other simple observations, such as the world agent location and acceleration, can be easily defined in a similar fashion.

4.2. Rewards

For each agent $a \in A$ and timestep t, we can define a set R of possible rewards, each weighted by a corresponding scaling factor λ (e.g. λ^{goal}):

- Distance to goal: $r_t^{\text{goal}} = -|g_t - p_t^a|$. In the spline scenario, the reward can be defined as the mean negative distance to the closest S spline control points:

$$r_t^{\text{spline}} = -\frac{1}{|S|}\sum_{s=1}^{|S|}|g_s - p_t^a|$$

- Desired velocity: $r_t^{\text{velocity}} = v_t^a \cdot d_t^a$, which rewards the agent for moving at the desired velocity along the direction towards the goal. For spline-based trajectories, d_t^a can be the local direction along the spline segment.
- Trajectory smoothness: $r_t^{\text{smoothness}} = \frac{f_t^a \cdot f_{t-1}^a}{\|f_t^a\|\|f_{t-1}^a\|}$, defined by the dot product between the current forward vector f_t^a and the previous forward vector f_{t-1}^a, rewarding smooth transitions in direction.
- Collision avoidance: $r_t^{\text{collision}} = -(T - \min_{n\in N}(r_{t,n}^a))$, rewarding the agent based on the minimum ray observation distance up to a threshold T, promoting safer distances from obstacles.
- Feature matching: $r_t^{\text{feature}} = -\text{LPIPS}(\phi_t^a, \phi_t^{\text{ref}})$, rewarding visual feature similarity to a reference feature set ϕ_t^{ref} using Learned Perceptual Image Patch Similarity (LPIPS) [48].

The complete reward for a certain timestep t is simply defined by a weighted mean of the chosen rewards:

$$r = \sum_{i=1}^{|R|}\lambda_i r_i$$

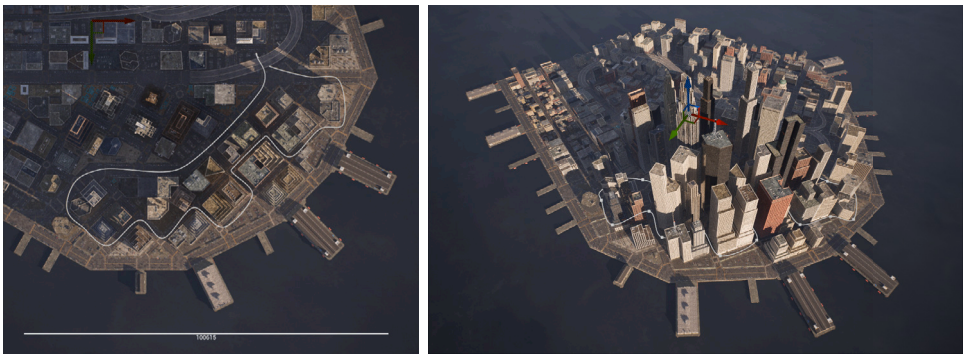

Fig. 3. Top and perspective view of a 2.5 Km spline for a drone mission scenario. In REP, similar splines can be defined for the humanoid and car agents as well.

Table 2
Observations and rewards for the considered scenarios, in increasing task difficulty order. The "Complete Scenario" is the most complex one, since it combines both visual and physical observations for multiple kinds of agents.

Scenario	Observations	Rewards
Spline Navigation	v_t^a, g_t^S, d_t^S	$r_t^{\text{spline}}, r_t^{\text{smoothness}}$
Dynamic Goal	g_t^a, d_t^a, v_t^a	$r_t^{\text{goal}}, r_t^{\text{velocity}}$
Dynamic Obstacles	$r_{t,n\in N}^a, ag_t^a, d_t^a, v_t^a$	$r_t^{\text{collision}}, r_t^{\text{goal}}, r_t^{\text{velocity}}$
Visual Navigation	ϕ_t^a	$r_t^{\text{collision}}, r_t^{\text{goal}}, r_t^{\text{velocity}}$
Complete Scenario	$\phi_t^a, r_{t,n\in N}^a, v_t^a$	$r_t^{\text{collision}}, r_t^{\text{goal}}, r_t^{\text{velocity}}, r_t^{\text{smoothness}}$

5. Scenarios

In this section, we will describe the five training scenarios that we considered for our experiments. They are presented in an increasing complexity order, with the goal of keeping the action space fixed, with straightforward observations and progressively difficult rewards, adding visual observations during the latest phases. No curriculum learning was used when switching from a scenario to the next one, although feasible in practice. A general overview of the actions, observations and rewards for each scenario can be seen in Tables 2 and 3.

5.1. Spline navigation

In this scenario, the agents are tasked to follow a predefined spline path at a certain velocity and with smooth movements. The splines are hand-crafted, which means that they are likely not optimal in terms of shape. The agents need to stay as close to the spline as possible, while simultaneously matching a certain velocity. This ultimately results in the creation of new, optimized splines once the agents are trained. Humanoid and vehicle agents rely on 2D splines for their movement, while drones (Fig. 3) can move along 3D splines, making the task more difficult.

5.2. Dynamic goals

In this scenario, we abandon the static spline, and instead define a dynamic goal, changing its location on the map during training time. The agents need to reach that goal in the shortest amount of time possible. By introducing this small change, we immediately obtain tangible improvements in terms of generalization. The agents can now move towards any goal efficiently, despite not considering possible obstacles in between.

5.3. Dynamic obstacles

For the aforementioned problem, we now need to introduce dynamic obstacles into the environment. By having both dynamic goals and obstacles, the agents can now navigate very complex scenarios, thanks to the introduction of collision rays that make them sense both the proximity of obstacles and the visibility of the goal.

In fact, whenever the goal is not visible, the agents learn to roam around looking for it until they find it. This is possible due to the introduction of a memory module in the PPO implementation, as explained in [8]. An overview of the training process for this scenario can be seen in Fig. 4, while we present a comparison of the features proposed by our method with those belonging to similar approaches in Table 4.

5.4. Visual navigation

In this scenario, agents are assigned the task of rapidly reaching a moving goal while avoiding dynamic obstacles, using only visual observations. Relying on visual observations alone is a very challenging task because the visual features need to be rich enough to encode multiple kinds of information, such as depth, segmentation, and more semantic information.

As shown in Fig. 5, we train different kinds of agents with this shared visual encoding as the only observation, to perform different navigation tasks. In Section 6 we show how to properly train these kinds of networks to obtain satisfying results, without using the game state.

5.5. Complete scenario

By combining the observations of the previous scenarios, we can merge both physical and visual observations into a unified observation array, enabling even more difficult tasks, such as recognizing and reaching a moving goal, navigating optimally through a complex environment. In these kinds of scenarios, the physical observations can be the agent-centric ones (own velocity, position, orientation, acceleration...), while the world-related ones (location of other objects and entities, depth, segmentation maps...) can be obtained by sensors, such as onboard cameras and lidars.

This is especially important when talking about digital twins, for which the real counterpart does not have access to many of the game state observations. For this reason, we integrated Eclipse Ditto[1] into our solution, to enable the training of proper digital twin intelligent agents within REP. All the features and attributes of the twin are encoded by specific JSON files, then the training carries on as explained before.

[1] Eclipse Ditto, https://eclipse.dev/ditto/

Table 3
Different kinds of agents that we considered for REP, in order of action space complexity. While vehicles and pawns usually move in a 2D plane (at least in terms of actions), a drone has a much more complex action space.

Agent type	Navigation domain	Action space
Vehicle	Ground-based	Steering, Throttle, Brake
Pawn	Ground-based	Move Forward/Backward, Turn Left/Right
Drone	Flight-based	Pitch, Roll, Yaw, Throttle (Ascend/Descend)

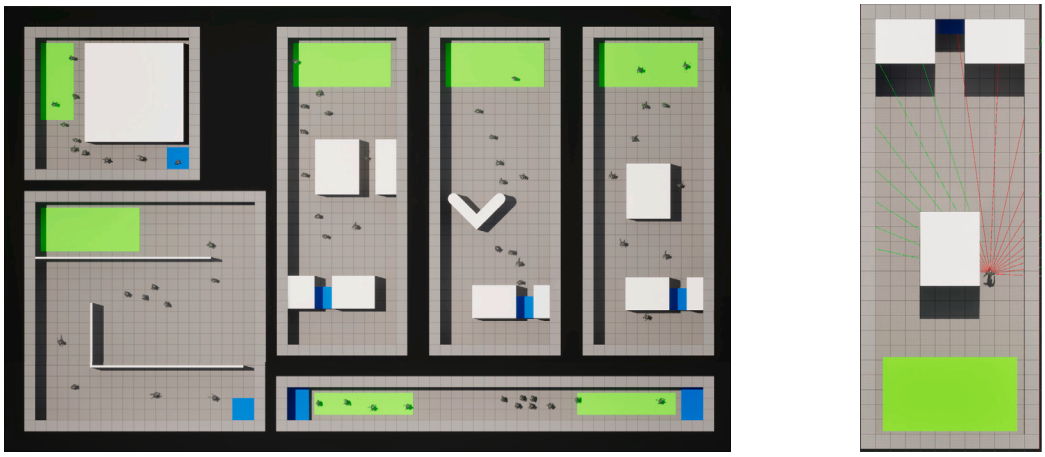

Fig. 4. A top-view example of some possible configurations of our training environment with dynamic goals and obstacles, as detailed in [8]. Each of the five maps comprises: (i) a light green spawn area, (ii) a light blue dynamic goal area, (iii) multiple dynamic obstacles and walls with arbitrary size and location (in white). Agents are instructed to navigate each procedurally generated environment according to the rewards detailed in Table 2.

Table 4
Comparison with deterministic and learning-based methods for crowd dynamics. Methods with (*) use pathfinding, so they cannot be considered real-time. Our method is the only one that does not require any map or goal knowledge.

	Requires training	Real-time	Dynamic goals	Dynamic obstacles	Without map knowledge	Without goal knowledge	Run-time behavior customizability	Collision avoidance
SFM [49]	✗	✓	✓	✓	✓	✗	Low	Explicit
R-SFM [50] *	✗	✗	✓	✓	✗	✗	Low	Explicit
CFSM [51] *	✗	✗	✓	✓	✗	✗	Low	Explicit
GCFSM [52] *	✗	✗	✓	✓	✗	✗	Low	Explicit
Continuum Crowds [53]	✗	✓	✓	✓	✗	✗	Medium	Implicit
Unicrowd [54]	✗	✓	✓	✗	✗	✗	Medium	Explicit
Mass AI (RVO) [55]	✗	✓	✗	✗	✗	✓	Medium	Explicit
TRACE [56]	✓	✗	✓	✓	✗	✗	High	Implicit
Ours	✓	✓	✓	✓	✓	✓	High	Implicit

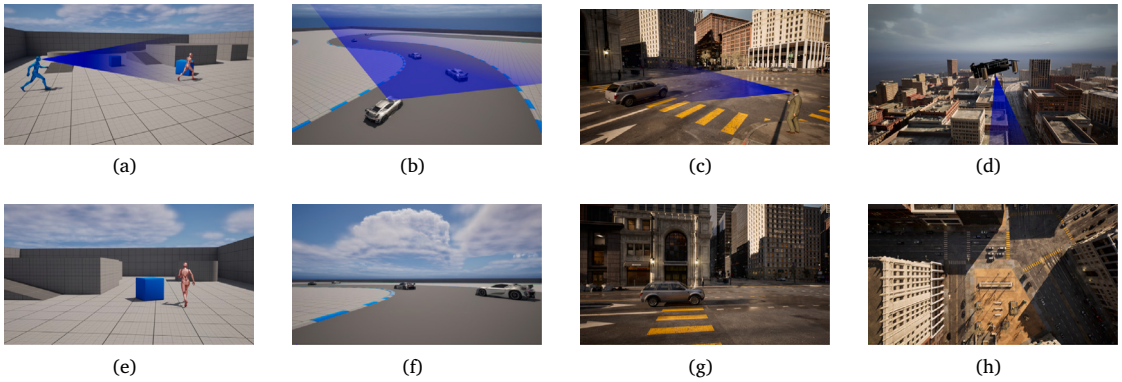

Fig. 5. REP visual observations system applied to multiple dynamic and unbounded scenes. **Top row**: different types of training scenarios (hide and seek, car race, urban navigation, drone coverage) and visual observation cones (blue). **Bottom row**: corresponding visual observations to be encoded by the neural network into observation vectors. Our method allows us to manage different kinds of rich, dynamic scenes even when using visual observations only, as explained in [9].

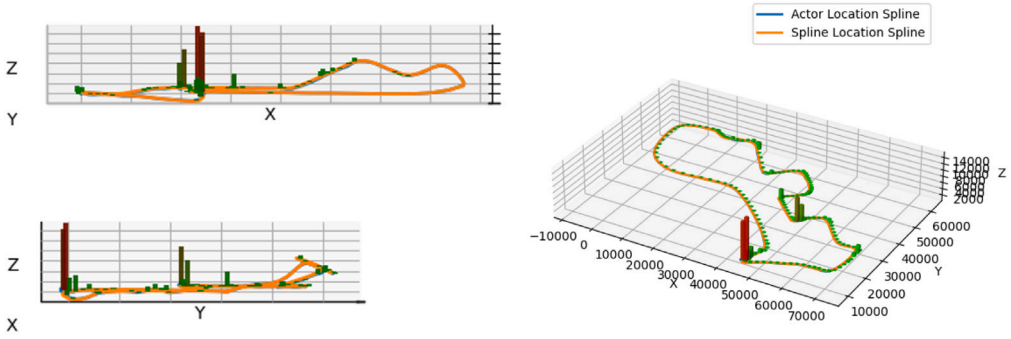

Fig. 6. Horizontal and vertical extent of the Spline Navigation scenarios. Vertical bars identify the localized inverse value of the dot product between the drone flying direction and the spline direction.

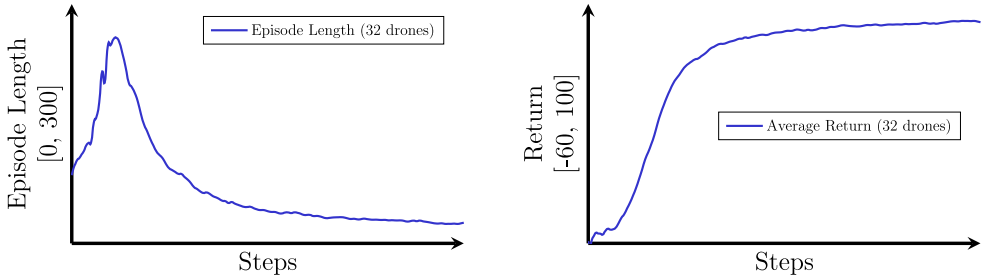

Fig. 7. Episode length and average return at training time for the spline scenario when using 32 drones.

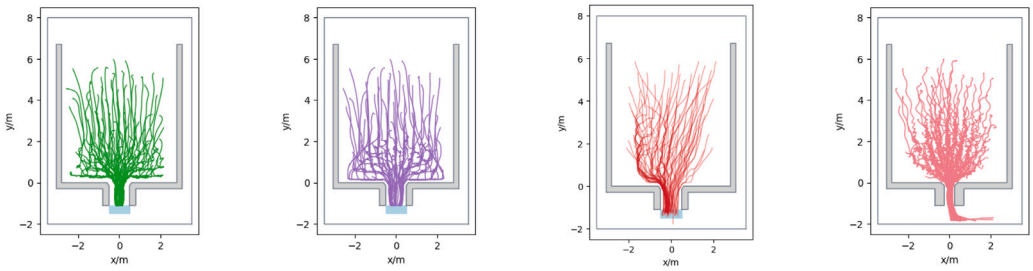

Fig. 8. Green: R-SFM, Purple: CFSM, Red: Ours, Pink: Ground truth. Comparison with real ground truth trajectories in a bottleneck scenario (agents move top-to-bottom). The agents in our simulation behave more like real humans, not walking towards walls to suddenly change direction, but instead slowing down and organizing in a more realistic queue formation.

Table 5
Configuration of the Spline Navigation scenario trained on an urban environment with 32 drones. The challenges of this scenario are due to the length of the spline, as well as its vertical extent. Despite the challenge, the trained drones stay close to the hand-drawn spline, choosing the best trade-off between distance and speed.

Spline length	2.5 km (0.62 miles)
Spline vertical extent	70 m (229.66 ft)
Termination distance from spline	8 m (26.24 ft)
Avg distance from spline	1.49 m (4.88 ft)
Avg speed along spline	60 km/h (37.28 mph)
Mean dot product with spline direction	0.9789

Once trained, at inference time the twin agent can communicate with the real one via a bidirectional channel, receiving observations from the real world and outputting a set of actions coming from the in-engine simulation, all in real-time.

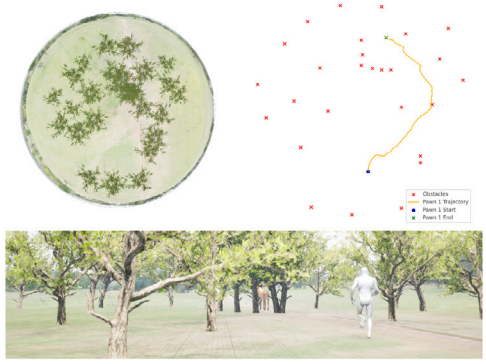

Fig. 9. Qualitative results of the Visual Navigation scenario [9].

6. Results

In this section, we provide the most interesting results that we obtained when training different agents on the provided scenarios, from the simpler to the more complex ones. We mostly divide them into game state-based and vision-based, claiming that mixing the two kinds of observation usually leads to results comparable to the game state-only scenarios in terms of raw optimization, but more aligned with

Table 6
Quantitative results of the visual navigation scenario using different backbones [9]. Numbers indicate Average Episode Length.

Pre-training	Observation type	Architecture	No Obstacles		Obstacles		Average
			No BG	BG	No BG	BG	
Self-supervised	**Features**	DINO (ViT-B/16)	51,89	66,37	81,96	121,20	80,35
Contrastive	**Features**	SWAV (ResNet50)	30,45	41,60	69,69	54,68	49,11
Supervised	**Features**	ResNet34	44,04	61,99	64,06	71,37	60,36
		ConvNext-T	46,34	60,45	60,81	41,81	52,35
		SWINv2-T/16	19,90	25,77	63,45	33,52	35,66
	BBoxes	YOLOv8n	28,10	78,70	38,28	136,5	70,40
		Faster R-CNN	47,30	38,05	49,72	32,11	41,80
		YOLOv8m	13,26	21,11	33,39	56,28	31,01

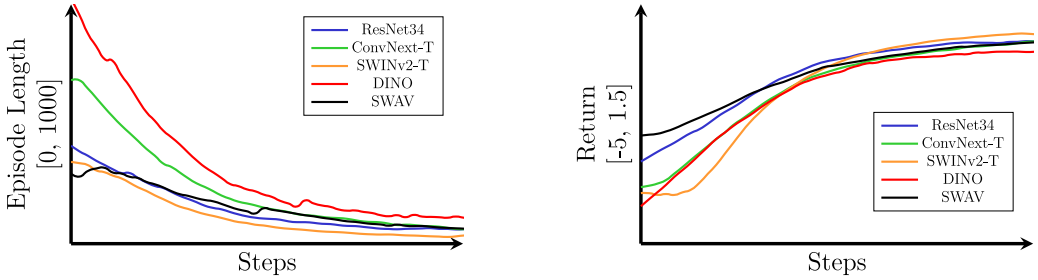

Fig. 10. Convolutional vs Transformer vs Self-supervised (DINO) vs Contrastive (SWAV) visual features as observations in the Visual Navigation scenario. The results show that most of the backbones can extract meaningful features that can be used for vision-based navigation, with SWINv2 obtaining the best results because of its window-based multi-scale approach.

real-world behavior in terms of the performed actions, thanks to the visual cues.

Baseline comparisons. To provide a fair comparative evaluation, we implemented simple baselines for each scenario. In addition to the REP agent, we trained (i) a hand-crafted waypoint follower that greedily moves towards the goal without collision avoidance, and (ii) an RL agent that uses only game-state observations (positions, velocities and ray sensors) without any visual features. These baselines are described in the supplementary material and their quantitative results are summarized there for brevity. Across all scenarios, the game-state-only RL baseline performs well when environments contain static goals and few obstacles but suffers degraded performance when dynamic goals or vision-based tasks are introduced. The hand-crafted follower consistently yields the lowest returns and success rates because it cannot adapt to dynamic obstacles. Our hybrid REP agent outperforms both baselines in terms of average return and success rate, particularly in the Complete Scenario where visual cues enable more natural and robust navigation. A detailed ablation showing the effect of adding visual observations to the game-state-only model is also included in the supplementary material.

For the spline scenario, we configured the experiment as shown in Table 5, with 32 drones and a training time of just a couple of hours on a single Nvidia RTX 4090 GPU. Similar training times also apply for the other scenarios.

In Fig. 6 we show that the trained drone perfectly aligns its moving direction with the spline direction in most cases, except when very sudden changes of direction occur. In those cases, the trained policy gives more importance to smoothing out the trajectory, without sacrificing the velocity component. However, even in those cases the alignment is still pretty good, as can be seen in the supplementary video on the website.

In Fig. 7 we show an example of training curves for the dynamic goal scenario. The agents easily learn to maximize the cumulative reward and return, while at the beginning the episode length increases, until the agents learn how to avoid the penalty for reaching out-of-map locations.

We demonstrate that our method generalizes better to the real world by better replicating real world human trajectories in a challenging bottleneck scenario. In Fig. 8 we show that the trajectories inferred by our method closely follow the ground truth, while traditional methods such as the social force model [57] and variations [50,51] often fail to replicate human behavior.

Regarding the vision part of our framework, we provide extensive ablation studies on the choice of the backbone for visual observation encoding, as seen in Table 6 and Fig. 10. In Fig. 9 we show a qualitative representation of how a pedestrian agent can navigate a visually complex environment, with the task of finding a single deer, hidden in a forest full of trees that occlude its vision. In this scenario, the agent needs to avoid collisions with the trees, and it relies solely on visual features as observations.

We observed that a very nice side effect of relying on vision (without *cheating*, i.e. providing the agent with any explicit direction or location vector of the deer) gives rise to more plausible trajectories, with the agent usually roaming around before establishing visual contact with the goal.

Combining both visual and physical observations leads to the best results in terms of quality of motion and planning. More importantly, the results are comparable to those that rely on the game state alone, but with the advantage of a more *natural* planning, typical of visual-only scenarios, such as the one in Fig. 9.

7. Conclusions

We presented REP: Render, Encode, Plan, a novel framework for in-engine training of mixed RL+DL intelligent embodied agents inside Unreal Engine 5. We demonstrate the effectiveness of our method under multiples scenarios with multiple configurations of agents, actions, observations and rewards. We believe that frameworks like REP can promote the creation of truly intelligent agents that can use both physical and visual observations to carry on arbitrarily complex tasks. By integrating REP with Eclipse Ditto, we enable the creation of digital twins of real-world agents, unlocking even more applications in multiple areas of interest.

CRediT authorship contribution statement

Daniele Della Pietra: Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Conceptualization. **Nicola Garau:** Writing – review & editing, Writing – original draft, Software, Methodology, Investigation, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

Funded by the European Union - Next Generation EU, Mission 4 Component 2 - CUP E63C22000970007.

Data availability

The website link with all the needed data is available at https://mmlab-cv.github.io/REP/.

References

[1] Brooks R. A robust layered control system for a mobile robot. IEEE J Robot Autom 1986;2(1):14–23.
[2] Colledanchise M, Ögren P. Behavior trees in robotics and AI: An introduction. CRC Press; 2018.
[3] Millington I. AI for games. CRC Press; 2019.
[4] Shao K, Tang Z, Zhu Y, Li N, Zhao D. A survey of deep reinforcement learning in video games. 2019, arXiv preprint arXiv:1912.10944.
[5] Kwiatkowski A, Alvarado E, Kalogeiton V, Liu CK, Pettré J, van de Panne M, et al. A survey on reinforcement learning methods in character animation. In: Computer graphics forum, vol. 41, (no. 2):Wiley Online Library; 2022, p. 613–39.
[6] Chen L, Lu K, Rajeswaran A, Lee K, Grover A, Laskin M, et al. Decision transformer: Reinforcement learning via sequence modeling. Adv Neural Inf Process Syst 2021;34:15084–97.
[7] Achiam J, Adler S, Agarwal S, Ahmad L, Akkaya I, Aleman FL, et al. Gpt-4 technical report. 2023, arXiv preprint arXiv:2303.08774.
[8] Della Pietra D, Garau N, Conci N, Granelli F. Dynamic crowd routing: RL-driven crowd dynamics. In: 2024 IEEE 25th international workshop on multimedia signal processing. IEEE; 2024, p. 1–6.
[9] Pietra DD, Garau N, Conci N, Granelli F. Peek-a-bot: learning through vision in Unreal Engine. The Eurographics Association; 2024.
[10] Epic Games. Unreal Engine learning agents. 2025, URL https://dev.epicgames.com/community/learning/courses/kRm/unreal-engine-learning-agents-5-5.
[11] Epic Games. Unreal Engine Neural Network Engine (NNE). 2024, URL https://dev.epicgames.com/community/learning/courses/e7w/unreal-engine-neural-network-engine-nne.
[12] PricewaterhouseCoopers. The perfect match – Digital twins and reinforcement learning. 2024, URL https://www.pwc.de/en/digitale-transformation/the-perfect-match-digital-twins-and-reinforcement-learning.html. [Accessed 19 August 2025].
[13] Farhang AR, Mulcahy B, Holden D, Matthews I, Yue Y. Humanlike behavior in a third-person shooter with imitation learning. In: 2024 IEEE conference on games. IEEE; 2024, p. 1–4.
[14] Nair AV, Pong V, Dalal M, Bahl S, Lin S, Levine S. Visual reinforcement learning with imagined goals. Adv Neural Inf Process Syst 2018;31.
[15] Shah R, Kumar V. Rrl: Resnet as representation for reinforcement learning. 2021, arXiv preprint arXiv:2107.03380.
[16] Tirumala D, Wulfmeier M, Moran B, Huang S, Humplik J, Lever G, et al. Learning robot soccer from egocentric vision with deep reinforcement learning. 2024, arXiv preprint arXiv:2405.02425.
[17] Ma L, Chen J, et al. Using RGB image as visual input for mapless robot navigation. 2019, arXiv preprint arXiv:1903.09927.
[18] Choi Y, Oh S. Image-goal navigation via keypoint-based reinforcement learning. In: 2021 18th international conference on ubiquitous robots. IEEE; 2021, p. 18–21.
[19] Mezghan L, Sukhbaatar S, Lavril T, Maksymets O, Batra D, Bojanowski P, et al. Memory-augmented reinforcement learning for image-goal navigation. In: 2022 IEEE/RSJ international conference on intelligent robots and systems. IEEE; 2022, p. 3316–23.
[20] Hwang M, Jeong J, Kim M, Oh Y, Oh S. Meta-explore: Exploratory hierarchical vision-and-language navigation using scene object spectrum grounding. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2023, p. 6683–93.
[21] Zheng D, Huang S, Zhao L, Zhong Y, Wang L. Towards learning a generalist model for embodied navigation. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2024, p. 13624–34.
[22] Kaufmann E, Loquercio A, Ranftl R, Müller M, Koltun V, Scaramuzza D. Deep drone acrobatics. 2020, arXiv preprint arXiv:2006.05768.
[23] Hwangbo J, Sa I, Siegwart R, Hutter M. Control of a quadrotor with reinforcement learning. IEEE Robot Autom Lett 2017;2(4):2096–103.
[24] Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, et al. Playing atari with deep reinforcement learning. 2013, arXiv preprint arXiv:1312.5602.
[25] Lample G, Chaplot DS. Playing FPS games with deep reinforcement learning. In: Proceedings of the AAAI conference on artificial intelligence, vol. 31, (no. 1). 2017.
[26] Ha D, Schmidhuber J. Recurrent world models facilitate policy evolution. Adv Neural Inf Process Syst 2018;31.
[27] Jaderberg M, Czarnecki WM, Dunning I, Marris L, Lever G, Castaneda AG, et al. Human-level performance in 3D multiplayer games with population-based reinforcement learning. Science 2019;364(6443):859–65.
[28] Berner C, Brockman G, Chan B, Cheung V, Debiak P, Dennison C, et al. Dota 2 with large scale deep reinforcement learning. 2019, arXiv preprint arXiv:1912.06680.
[29] Vinyals O, Babuschkin I, Czarnecki WM, Mathieu M, Dudzik A, Chung J, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. Nature 2019;575(7782):350–4.
[30] Todorov E, Erez T, Tassa Y. Mujoco: A physics engine for model-based control. In: 2012 IEEE/RSJ international conference on intelligent robots and systems. IEEE; 2012, p. 5026–33.
[31] Brockman G. OpenAI gym. 2016, arXiv preprint arXiv:1606.01540.
[32] Song Y, Steinweg M, Kaufmann E, Scaramuzza D. Autonomous drone racing with deep reinforcement learning. In: 2021 IEEE/RSJ international conference on intelligent robots and systems. IEEE; 2021, p. 1205–12.
[33] Ho J, Ermon S. Generative adversarial imitation learning. Adv Neural Inf Process Syst 2016;29.
[34] Pathak D, Agrawal P, Efros AA, Darrell T. Curiosity-driven exploration by self-supervised prediction. In: International conference on machine learning. PMLR; 2017, p. 2778–87.
[35] He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016, p. 770–8.
[36] Dosovitskiy A, Beyer L, Kolesnikov A, Weissenborn D, Zhai X, Unterthiner T, et al. An image is worth 16x16 words: Transformers for image recognition at scale. 2020, arXiv preprint arXiv:2010.11929.
[37] Liu Z, Mao H, Wu C-Y, Feichtenhofer C, Darrell T, Xie S. A convnet for the 2020s. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2022, p. 11976–86.
[38] Liu Z, Lin Y, Cao Y, Hu H, Wei Y, Zhang Z, et al. Swin transformer: Hierarchical vision transformer using shifted windows. In: Proceedings of the IEEE/CVF international conference on computer vision. 2021, p. 10012–22.
[39] Oquab M, Darcet T, Moutakanni T, Vo H, Szafraniec M, Khalidov V, et al. Dinov2: Learning robust visual features without supervision. 2023, arXiv preprint arXiv:2304.07193.
[40] Chen T, Kornblith S, Norouzi M, Hinton G. A simple framework for contrastive learning of visual representations. In: International conference on machine learning. PMLR; 2020, p. 1597–607.
[41] Chen X, He K. Exploring simple siamese representation learning. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2021, p. 15750–8.
[42] Caron M, Misra I, Mairal J, Goyal P, Bojanowski P, Joulin A. Unsupervised learning of visual features by contrasting cluster assignments. Adv Neural Inf Process Syst 2020;33:9912–24.
[43] Ren S, He K, Girshick R, Sun J. Faster r-cnn: Towards real-time object detection with region proposal networks. Adv Neural Inf Process Syst 2015;28.
[44] Jocher G, Chaurasia A, Qiu J. Ultralytics YOLO. 2023, URL https://github.com/ultralytics/ultralytics.
[45] Juliani A, Berges V-P, Teng E, Cohen A, Harper J, Elion C, et al. Unity: A general platform for intelligent agents. 2020, arXiv preprint arXiv:1809.02627.
[46] Beeching E, Debangoye J, Simonin O, Wolf C. Godot reinforcement learning agents. 2021, arXiv preprint arXiv:2112.03636.
[47] Jitosho R, Lum TGW, Okamura A, Liu K. Reinforcement learning enables real-time planning and control of agile maneuvers for soft robot arms. In: Conference on robot learning. PMLR; 2023, p. 1131–53.
[48] Zhang R, Isola P, Efros AA, Shechtman E, Wang O. The unreasonable effectiveness of deep features as a perceptual metric. In: CVPR. 2018.

[49] Helbing D, Molnar P. Social force model for pedestrian dynamics. Phys Rev E 1995;51(5):4282.
[50] Schrödter T, The PedPy Development Team. PedPy - Pedestrian trajectory analyzer (v1.1.0). Zenodo; 2024, http://dx.doi.org/10.5281/zenodo.10814490.
[51] Tordeux A, Chraibi M, Seyfried A. Collision-free speed model for pedestrian dynamics. In: Traffic and granular flow'15. Springer; 2016, p. 225–32.
[52] Xu Q, Chraibi M, Tordeux A, Zhang J. Generalized collision-free velocity model for pedestrian dynamics. Phys A 2019;535:122521.
[53] Treuille A, Cooper S, Popović Z. Continuum crowds. ACM Trans Graph 2006;25(3):1160–8.
[54] Bisagno N, Stefani AL, Garau N, De Natale F, Conci N. UniCrowd simulator: Visual and behavioral fidelity for the generation of crowd datasets. In: 2024 IEEE international conference on image processing. IEEE; 2024.
[55] Van den Berg J, Lin M, Manocha D. Reciprocal velocity obstacles for real-time multi-agent navigation. In: 2008 IEEE international conference on robotics and automation. IEEE; 2008, p. 1928–35.
[56] Rempe D, Luo Z, Bin Peng X, Yuan Y, Kitani K, Kreis K, et al. Trace and pace: Controllable pedestrian animation via guided trajectory diffusion. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2023, p. 13756–66.
[57] Alahi A, Goel K, Ramanathan V, Robicquet A, Fei-Fei L, Savarese S. Social lstm: Human trajectory prediction in crowded spaces. In: Proceedings of the IEEE conference on computer vision and pattern recognition. 2016, p. 961–71.