frontiers | Frontiers in Neuroinformatics

TYPE Methods
PUBLISHED 09 March 2023
DOI 10.3389/fninf.2023.1134405

OPEN ACCESS

EDITED BY
Shankar Subramaniam,
University of California, San Diego,
United States

REVIEWED BY
Sebastien Miellet,
University of Wollongong, Australia
Mirko Zanon,
University of Trento, Italy

*CORRESPONDENCE
Sen Cheng
✉ sen.cheng@rub.de

RECEIVED 30 December 2022
ACCEPTED 17 February 2023
PUBLISHED 09 March 2023

CITATION
Diekmann N, Vijayabaskaran S, Zeng X, Kappel D, Menezes MC and Cheng S (2023) CoBeL-RL: A neuroscience-oriented simulation framework for complex behavior and learning. *Front. Neuroinform.* 17:1134405. doi: 10.3389/fninf.2023.1134405

CoBeL-RL: A neuroscience-oriented simulation framework for complex behavior and learning

Nicolas Diekmann[1,2], Sandhiya Vijayabaskaran[1], Xiangshuai Zeng[1,2], David Kappel[1], Matheus Chaves Menezes[3] and Sen Cheng[1]*

[1]Faculty for Computer Science, Institute for Neural Computation, Ruhr University Bochum, Bochum, Germany, [2]International Graduate School of Neuroscience, Ruhr University Bochum, Bochum, Germany, [3]Laboratory of Artificial Cognition Methods for Optimisation and Robotics, Federal University of Maranhão, São Luís, Brazil

Reinforcement learning (RL) has become a popular paradigm for modeling animal behavior, analyzing neuronal representations, and studying their emergence during learning. This development has been fueled by advances in understanding the role of RL in both the brain and artificial intelligence. However, while in machine learning a set of tools and standardized benchmarks facilitate the development of new methods and their comparison to existing ones, in neuroscience, the software infrastructure is much more fragmented. Even if sharing theoretical principles, computational studies rarely share software frameworks, thereby impeding the integration or comparison of different results. Machine learning tools are also difficult to port to computational neuroscience since the experimental requirements are usually not well aligned. To address these challenges we introduce CoBeL-RL, a closed-loop simulator of complex behavior and learning based on RL and deep neural networks. It provides a neuroscience-oriented framework for efficiently setting up and running simulations. CoBeL-RL offers a set of virtual environments, e.g., T-maze and Morris water maze, which can be simulated at different levels of abstraction, e.g., a simple gridworld or a 3D environment with complex visual stimuli, and set up using intuitive GUI tools. A range of RL algorithms, e.g., Dyna-Q and deep Q-network algorithms, is provided and can be easily extended. CoBeL-RL provides tools for monitoring and analyzing behavior and unit activity, and allows for fine-grained control of the simulation *via* interfaces to relevant points in its closed-loop. In summary, CoBeL-RL fills an important gap in the software toolbox of computational neuroscience.

KEYWORDS

spatial navigation, spatial learning, hippocampus, place cells, grid cells, simulation framework, reinforcement learning

1. Introduction

The discovery of place cells in the hippocampus (O'Keefe and Dostrovsky, 1971) and grid cells in the medial enthorinal cortex (Hafting et al., 2005) have advanced our knowledge about spatial representations in the brain. These discoveries have also spurred the development of computational models to complement electrophysiological and behavioral experiments, and to better understand the learning of such representations

and how they may drive behavior (Bermudez-Contreras et al., 2020). However, the computations performed during spatial navigation have received far less attention. In recent years, reinforcement learning (RL) (Sutton and Barto, 2018) has been of increasing interest in computational studies as it allows for modeling complex behavior in complex environments. Reinforcement learning describes the closed-loop interaction of an agent with its environment in order to maximize rewarding behavior or to minimize repulsive situations. A common way of solving the RL problem consists of inferring the value function, a mapping from state-action pairs to expected future reward, and selecting those actions which yield the highest values. The brain is thought to support this value-based type of RL (Schultz et al., 1997), and RL has been used to explain both human (Redish et al., 2007; Zhang et al., 2018) and animal behavior (Bathellier et al., 2013; Walther et al., 2021) (see Botvinick et al., 2020 for a recent review).

Concurrent with advances in understanding neural spatial representations, our theoretical understanding of reinforcement learning has also improved significantly in recent years. One major innovation was the combination of RL with deep neural networks (DNNs) (Mnih et al., 2015). These Deep RL models are now among the best-performing machine learning techniques, reaching higher performances than humans on complex tasks like playing Go (Mnih et al., 2015) or video games (Silver et al., 2016, 2017).

These advances in RL methods offer the opportunity to study the behavior, learning, and representations that emerge, when complex neural network models are trained on tasks similar to those used in biological experiments. The use of virtual reality in animal experiments (Pinto et al., 2018; Koay et al., 2020; Nieh et al., 2021) would even allow for the direct comparison of *in-vivo* and *in-silico* behavior. First attempts in this direction have recently shown that Deep RL models develop spatial representations similar to those found in enthorinal cortex (Banino et al., 2018; Cueva and Wei, 2018) and hippocampus (Vijayabaskaran and Cheng, 2022).

In many computational studies, models are custom built for specific experiments and analyses. While several software frameworks for simulations have been developed, they are often specialized for a certain type of task (Eppler et al., 2009; Leibo et al., 2018). Furthermore, the majority of frameworks are developed primarily within the context of machine learning and for eventual practical applications (Beattie et al., 2016; Chevalier-Boisvert et al., 2018; Liang et al., 2018), which makes their adaptation for neuroscience a daunting task. The high heterogeneity of models with respect to their technical design, implementation and requirements severely complicates their integration. A common RL framework would provide an opportunity to share and combine work across different fields and levels of abstraction.

A number of RL simulation frameworks have been introduced previously that target machine learning or industrial applications. For example, DeepMind Lab provides a software interface to a first-person 3D game platform to develop general artificial intelligence and machine learning systems (Beattie et al., 2016). RLlib is a Python-based framework that provides a large number of virtual environments and toolkits for building and performing RL simulations (Liang et al., 2018). The main targets of RLib are industrial applications in domains such as robotics, logistics, finance, etc. One unique advantage of RLlib is that it offers architectures for large-scale distributed RL simulations to massively speed up training. The framework also supports Tensorflow and Pytorch. RLib has been used by industry leaders, but does not target scientific studies in neuroscience. The Minimalistic Gridworld Environment (MiniGrid) provides an efficient gridworld environment setup with a large variety of different types of gridworlds, targeting machine learning (Chevalier-Boisvert et al., 2018). MiniGrid differs from our Gridworld interface in that it provides isometric top view image observations instead of abstract states. MAgent comprises a library for the efficient training of multi-agent systems (Zheng et al., 2018; Terry et al., 2020) and includes implementations of common Deep RL algorithms like DQN.

On the other hand, there are efforts that are targeted toward application in neuroscience. For instance, SPORE provides an interface between the NEST simulator, which is optimized for spiking neuron models (Eppler et al., 2009), and the GAZEBO[1] robotics simulator (Kaiser et al., 2019). SPORE targets robotics tasks, and the framework is not easy to use for analysis of network behavior or emerging connectivity in the network. The framework also cannot be easily combined with deep Q-learning. Psychlab aims at comparing the behaviors of artificial agents and human subjects by recreating the experimental setups used commonly in psychological experiments (Leibo et al., 2018). These setups include visual search, multiple object tracking, and continuous recognition. The setups are recreated inside the DeepMind Lab virtual environment (Beattie et al., 2016). RatLab is a software framework for studying spatial representations in rats using simulated agents (Schönfeld and Wiskott, 2013). The place code model is based on the hierarchical Slow Feature Analysis (SFA) (Schönfeld and Wiskott, 2015). RatLab supports a number of different spatial navigation tasks and can be flexibly extended. It is not well-suited to be integrated with reinforcement learning methods.

Here, we introduce the *"Closed-loop simulator of **Co**mplex **Be**havior and **L**earning based on **R**einforcement **L**earning and deep neural networks*," or *CoBeL-RL* for short. The framework is based on the software that had been developed for studying the role of the hippocampus in spatial learning (Walther et al., 2021; Vijayabaskaran and Cheng, 2022) and further extends and unifies its functionality. The CoBeL-RL framework offers a range of reinforcement learning algorithms, e.g., Q-learning or deep Q-network, and environments commonly used in behavioral studies, e.g., T-maze or Morris water maze. In contrast to other libraries, virtual environments across different levels of abstraction are supported, e.g., gridworld and 3D simulation of physical environments. Furthermore, our framework provides tools for the monitoring and analysis of generated behavioral and neuronal data, e.g., place cell analysis.

2. Methods

The CoBeL-RL framework provides the tools necessary for setting up the closed-loop interaction between an agent and an

1 http://gazebosim.org/

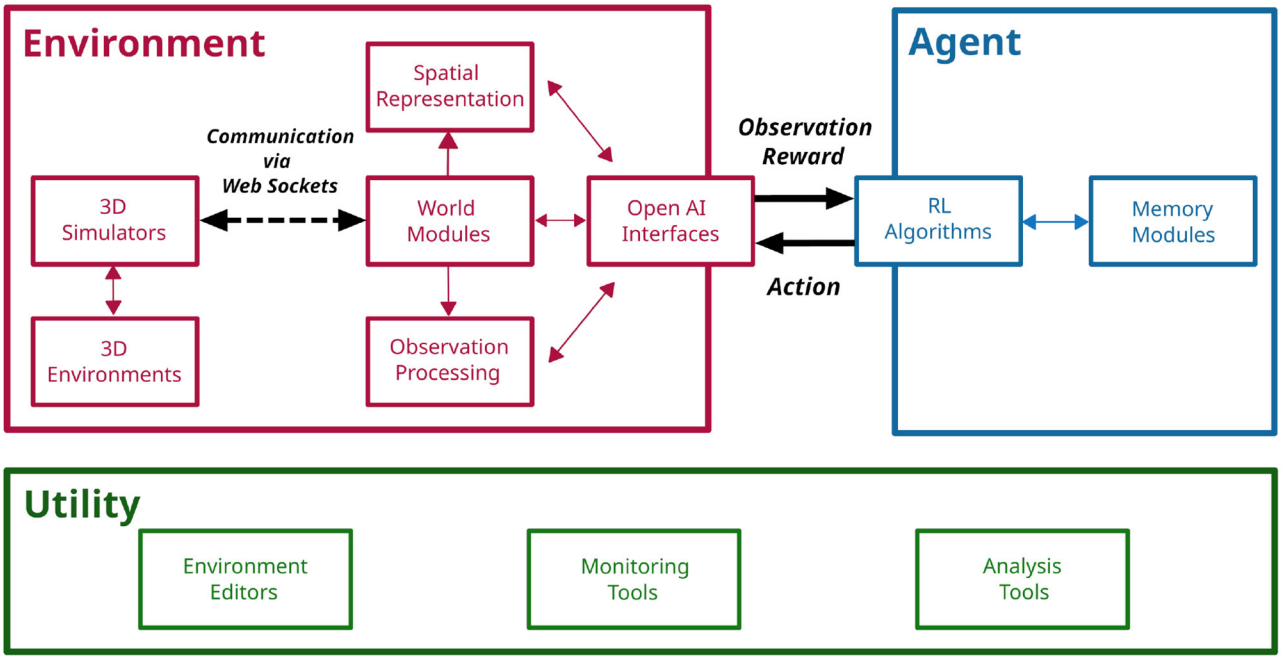

FIGURE 1
The CoBeL-RL framework. Illustration of CoBeL-RL's modules and their roles. Modules can be grouped into three categories: Modules, which handle the simulation of the virtual environment and the flow of information (Environment), implement different RL algorithms (Agent), and implement supporting functionalities (Utility) such as monitoring and data analysis. The interactions between an agent and its environment (Sutton and Barto, 2018) is implemented with the help of the module "OpenAI Interfaces" and provides observation and reward information to the agent, and sends actions to be executed to the environment.

environment (Sutton and Barto, 2018). CoBeL-RL focuses on the simulation of trial-based experimental designs, that is, the learning of a task is structured into separate trials, similar to behavioral experiments. Single trials are (usually) defined by the completion of the task or reaching a timeout condition, i.e., time horizon. Each trial is further differentiated into agent-environment interactions referred to as steps. Each step yields an experience tuple (s_t, a_t, r_t, s_{t+1}), which the agent can learn directly from or store in a memory structure for later learning. s_t is the agent's state at time t, a_t is the action selected by the agent, r_t is the reward received for selecting action a_t, and s_{t+1} is the agent's new state.

CoBeL-RL is separated into modules that can be classified into three categories (Agent, Environment, and Utility), which provide models of the RL agent, models of the environment, and utility functionalities, respectively (Figure 1). These three categories of modules will be explained in detail in the following sections. The majority of the framework is written in Python 3 (Rossum, 1995), while a few components are written in other programming languages as required by the software to which they interface. Tensorflow 2 (Abadi et al., 2015) serves as the main library for implementing Deep Neural Network models, and Keras-RL2 (Tensorflow 2 compatible version of Keras-RL; Plappert, 2016) as the base for the framework's deep Q-network (DQN) agent (Mnih et al., 2015). The interaction between RL agents and RL environments is facilitated through Open AI Gym (Brockman et al., 2016). PyQt5 is used for visualization.

2.1. Agent modules

RL agents are implemented *via* the Agent modules that define the behavior including learning, exploration strategies, and memory (see Figure 1). All RL agents inherit from a common abstract RL agent class requiring them to implement functions for training and testing, as well as the computation of predictions for a given batch of observations. Furthermore, all RL agents implement callbacks which are executed at the start and end of a trial or step to allow for fine-grained control during simulations. Trial information, e.g., number of trial steps, reward, etc., is collected by RL agents and passed to the callbacks. All callback classes inherit from a common callback class, and custom callback functions can be defined by the user and passed as a dictionary to the RL agent.

RL agents in CoBeL-RL can use experience replay (Lin, 1992, 1993; Mnih et al., 2015) as part of the learning algorithm. To do so, agents are linked with different memory modules used internally as buffers for experience replay. Experience tuples are stored in the memory modules providing the possibility of building up a history of experiences, which are used for training. Agents and memory modules can be combined freely to study the effects of different replay models.

Four agents are currently available and described briefly in the following.

2.1.1. DQN agents

The framework's baseline DQN agent encapsulates Keras-RL2's DQN implementation. It uses a small fully connected DNN by default and follows an epsilon-greedy policy. Since the original implementation is trained for a given number of steps instead of trials, the aforementioned callbacks are used to allow training for a given number of trials. This agent's callback class inherits from both the Keras callback class as well as the common callback class. Furthermore, versions of the DQN which implement Prioritized Experience Replay (Schaul et al., 2016) (PER-DQN) and learn an environmental model are also available.

2.1.2. Dyna-Q agents

The Dyna-Q model (Sutton and Barto, 2018) is implemented as a static tabular agent, that is, the agent's Q-function is represented as an array of size $|S| \times |A|$ where $|S|$ is the number of environmental states and $|A|$ is the number of available actions. Due to its tabular nature, this agent, and those that derive from it, can only be used in conjunction with discrete static environments that represent states as abstract indices, such the gridworld interface. The agent's environmental model is encapsulated as a separate memory module and similarly represents the environment in a tabular fashion. The memory module stores and retrieves experiences. For action selection either an epsilon-greedy (default) or softmax policy can be chosen. Furthermore, an optional action mask can be used to remove actions from the action selection that do not result in a state change. The agent's Q-function is updated each step online and *via* experience replay (Lin, 1992). Experience replay can be performed after each step, after each trial, or disabled. In addition, an implementation of the Prioritized Memory Access (Mattar and Daw, 2018) model is also provided.

2.1.3. Dyna-DQN and DSR agents

CoBeL-RL further offers hybrid agents which are derived from the Dyna-Q agent and use a DQN to represent their Q-function which we refer to as Dyna-DQN. The DQN part of the hybrid does not rely on Keras-RL2 and can be implemented separately in Tensorflow 2 or PyTorch (Paszke et al., 2019). An abstract network class serves as an interface between the agent and the separately implemented network. A set of observations which correspond to the different discrete environmental states can be passed to the agent. If no observations are defined, a one-hot encoding of the environmental states is generated which serve as observations. Additionally, a hybrid agent that implements a version of Deep Successor Reinforcement Learning (DSR) (Kulkarni et al., 2016) is also provided which we refer to as Dyna-DSR. Similar to the DQN agents, the Dyna-DSR uses a small DNN by default to represent the Q-function. However, unlike the DSR, the Dyna-DSR does not learn a separate feature representation of its observations. Instead, reward and successor representation models are trained directly on the observations. We refer to the representation learned by Dyna-DSR as the deep successor representation (Deep SR).

2.1.4. Model-free episodic control agents

While all three Q-learning-based agents introduced above update the Q-function incrementally, Model-Free Episodic Control (MFEC) (Blundell et al., 2016) agents are designed for fast learning by repeating the best action they have performed in a specific state in the past. MFEC is therefore well-suited for modeling one-shot learning (Wiltgen et al., 2006; Kosaki et al., 2014). The Q-function of MFEC agents is represented by an array of growing size where a state-action pair and its corresponding Q-value is inserted if the pair is encountered by the agent for the first time. The Q-value is updated in a one-shot manner, which simply keeps the best accumulative reward encountered so far. During inference, the agent searches the array, finds the most similar state to the current state and retrieves the Q-values upon which action selection decisions are made. To further improve computational efficiency, all unique, high-dimensional states are first projected to a low dimensional space and then stored by using a KD-tree data structure (a K-dimensional tree is a binary tree where every node is a k-dimensional point; Bentley, 1975), so that the search of closest neighbors to a given state (measured under Euclidean distance) becomes efficient.

2.2. Environment modules

The implementation of the RL environments is split across different modules, and the agents interact with them through Open AI Gym interfaces. Roughly, CoBeL-RL provides two types of environments: simple environments that are directly implemented within an interface, e.g., a gridworld, and complex environments that are rendered by a game engine, such as Unity (Unity Technologies, 2005), Godot (Linietsky and Manzur, 2007), or Blender Game Engine (Blender Online Community, 2018). The latter involves additional modules for interaction with game engines (3D Simulators), processing of observations (observation modules) and navigation (spatial representation modules).

2.2.1. Interface module

All interfaces inherit from the Open AI Gym interface (Brockman et al., 2016) and implement step and reset functions. CoBeL-RL offers a wide range of different interfaces.

The gridworld interface represents gridworld environments in a tabular fashion. Environmental transitions are determined *via* a state-action-state transition matrix of size $|S| \times |A| \times |S|$. $|A| = 4$ to realize the movements on the grid (up, down, left, right). The reward function and terminal states are represented as tables of size $|S|$, where the reward function is real valued and the terminal states are binary encoded. The set of possible starting states is represented as a list of state indices.

A simple 2D environment is implemented by the 2d interface and allows interaction either by moving in the four cardinal directions or as a differential wheeled robot (i.e., the agent can move either of its wheels). In the former case the agent's state is represented as 2d coordinates and in the latter case orientation is also included. The environment itself has no obstacles with the exception of walls that delineate the area in which the agent can

navigate. The environment can contain multiple reward locations. Trials end when the agent reaches a reward, and optionally when it hits a wall.

The discrete interface implements tabular environments similar to the gridworld interface but is compatible with more complex environments. Like the gridworld interface, it requires the definition of a state-action-state transition matrix, a reward function, a table containing terminal states, and a list of starting states. Unlike the gridworld interface, it provides observations, which can be defined for each state, instead of state indices. If no observations are defined, a one-hot encoding is generated for each state.

The baseline interface provides a simple interface implementation for deep Q-learning agents. The baseline interface defines the interaction of the baseline DQN agent with the spatial representation module, the observation module and the simulated 3D environment. The number of actions available to the agent is determined by the spatial representation module. The following sections detail the functionality of spatial representation, observation, and 3D simulator modules.

2.2.2. Spatial representation module

The spatial representation module allows the agent to navigate on a simplified spatial representation of the environment, rather than continuously through space. Currently, this module constructs a topological graph of the environment, with nodes and edges defining the connectivity. The topological graph can either be manually defined by the user when working with Blender by placing nodes and edges directly in the Blender Game Engine (Blender Online Community, 2018) (BGE), or can be automatically constructed using the grid graph module. CoBeL-RL provides implementations of simple rectangular and hexagonal graphs, and other graph types, such as a Delaunay triangulation, can be easily implemented if needed.

The spatial representation module can also be used to directly define the action space of the agent and specify how the actions are mapped to transitions on the graph. The default topological graphs implement two modes of transitions—without rotation, which only allows transitions to neighboring nodes, and with rotations, which allows both translational and rotational movements on the graph.

2.2.3. Observation module

The observation module provides functionality for the pre-processing of observations retrieved by the environment before they are sent to the RL agent.

The simplest observation module retrieves the agent's position in x-y coordinates as well as its current heading direction, which can be used by the RL agent alongside or instead of visual observations. The observation module also pre-processes visual observations by resizing them to a user-defined size and normalizing the pixel values in the range $[0, 1]$, such that they can be passed to the agent. Furthermore, observations can be corrupted with different types of noise, e.g., Gaussian noise, to better capture the imprecision of biological observations. Two or more observation modules can also be flexibly combined to simulate multisensory observations. For multisensory simulations, the individual observation modules are stored in a dictionary. The observations are then passed to the RL agent through the interface module. For deep RL agents, the keys of the dictionary can be used to indicate the input layer of the neural network to which those observations should be passed, allowing the use of complex network structures to process the observations.

2.2.4. 3D simulators

The 3D Simulators module implements the communication between the CoBeL-RL framework and game engines that are used for simulation and rendering. The sending of commands and retrieval of data is handled *via* web sockets. CoBeL-RL supports three different game engines for simulation and rendering: Blender Game Engine (BGE) (Blender Online Community, 2018), Unity (Unity Technologies, 2005), and Godot (Linietsky and Manzur, 2007).

The baseline BGE Simulator module communicates *via* three separate web sockets. The control socket is used to send commands and retrieve control relevant data, e.g., object identifiers. Commands are encoded as strings which contain the command name as well as parameter values in a comma-separated format. Similarly, retrieved values arrive as strings in a comma-separated format. Observational data like visual observations and sensory data are retrieved *via* the video socket and data socket, respectively. During module initialization, a new Blender process is launched and a user-defined Blender scene is opened. Important commands provided by the module include the changing of an agent's position and orientation, as well as the control of light sources. Observational data is automatically retrieved when the agent is manipulated, but can also be retrieved manually.

The Unity Simulator module is based on the *Unity ML-Agents Toolkit* (Juliani et al., 2020), which provides various interfaces for the communication between Unity and Python. The toolkit is mainly divided into two parts, one for the Unity side and another one for the Python side. The former is written in *C#*, which is the programming language for developing Unity games. Data transmission is handled by web sockets in the *Unity ML-Agents Toolkit*. Users do not need to set up the sockets themselves, but rather use the APIs provided by the Toolkit. Both string and float variables can be exchanged between the two sides.

The Godot Simulator module is built on the Godot engine and follows the same communication scheme as the BGE Simulator module. However, instead of Python, *GDScript* is used by Godot. Furthermore, instead of using comma-separated strings for communication most data is encoded in the JSON format. The Godot Simulator module supports the same commands and functionality as the BGE Simulator module and additionally allows for the switching of environments without the need for restarting the 3D simulator.

2.3. Utility modules

CoBeL-RL's utility modules consist of the analysis module and misc module. Simulation variables, e.g., behavior and learning progress, can be monitored using the analysis module's various

monitor classes. Gridworld tools and an environment editor can be found in the misc module.

2.3.1. Analysis module

The analysis module contains code for different types of analysis and monitoring tools.

CoBeL-RL provides monitors which store variables of interest during the course of training: the number of steps required to complete a trial (escape latency), the cumulative reward received in each trial, the responses emitted by the agent on each trial or step, and the agent's position at each step (trajectory). Additionally, for Deep RL agents, the activity of layer units can be tracked with the response monitor. The different monitors can be added to any RL agent and are updated with trial information from callback calls.

The analysis tools enable the computation of spatial activation maps of network units at a desired resolution. The maps can be generated during the course of training or at the end of training by artificially moving the agent on a rectangular grid in the environment and recording the activity of the units of interest at the grid points. Based on the spatial activation maps, CoBeL-RL also provides a method to identify units that show special spatial firing properties, such as place-cell-like units and vector-like units (Vijayabaskaran and Cheng, 2022).

2.3.2. Misc module

The misc module provides tools that support the setting up of simulations and environments.

The gridworld tools provide functionality for the creation of gridworld environments. Gridworlds can be generated by either manually defining the relevant variables, e.g., size, reward function, starting states, etc., or by using templates for specific instances of gridworld environments, e.g., an open field with a single goal location. The gridworld tools generate the variables required by the gridworld interface, such as the state-action-state transition matrix, reward function, etc., and store them in a dictionary. A visual editor for gridworlds, which builds on the gridworld tools is provided in the gridworld GUI.

The Unity editor tools allow the user to quickly design the structure of a discrete, maze-like environment including external walls, obstacles, an agent and rewards, and import the generated file to the Unity editor to create a virtual environment. The tools are written in Java (version 17.0.1) and include a GUI as back-end functionalities, which generate the `.yaml` file required by the Unity editor. The user can choose from a large number of textures, and has the option of adding and deleting materials.

2.4. Additional details about offline rendering experiment

In Section 3.3, we show two examples of simulations using the BGE and Unity simulators. For the online/offline rendering experiment in Section 3.3, we use a T-maze environment with a Gridworld topology. The scene consisted of 3,061 vertices and 2,482 faces, including a visual representation of the agent that was partly visible in every frame and a single spherical light source. Maze walls had image textures and used diffuse (Lambert) and specular (CookTorr) shading. Backface culling was active for the complete scene. Simulations were run on a 16 core Intel(R) Xeon(R) W-1270P CPU with clock speed 3.80 GHz and 32 GB RAM and a NVIDIA Quadro RTX(R) 5000 GPU. The agent was trained for five trials on the random pellet chasing task to encourage exploration of the entire maze, with a maximum trial duration of 400 time steps. For the offline rendering, observation images were pre-rendered and stored for all possible agent locations and later retrieved when needed. Simulation time for pre-rendering and storage was excluded from the mean frame time analysis.

3. Results

3.1. CoBeL-RL

The CoBeL-RL framework is a highly modular software platform to conduct spatial navigation and learning experiments with virtual agents. Figure 2 illustrates one of its realizations to simulate a T-maze experiment where an agent has to visit the right arm of the maze to receive a reward. The representation of this environment was split into two parts: the visual appearance of the environment and the topology. The former was built using the Unity simulator, which renders RGB images and sends them to the observation module. These image observations can be resized, or normalized to conform to parameter ranges of pixel values and are then sent to the OpenAI Gym Interface. The topology module generates a topological graph of the environment automatically and sends the spatial information of the artificial agent (blue square in Figure 2) to the Gym interface as well. The reward function was defined inside the Gym interface to decide what kind of behavior is reinforced. For example, if the reward (green ball, Figure 2) is placed in a fixed location in the maze in every trial, then a simple goal-directed navigation task is simulated.

As part of the Gym Interface, a DQN agent implemented using Keras/Tensorflow receives the processed image observation, the instant reward and then takes an action, deciding whether the artificial agent should move forward, turn left, etc. The DQN agents can utilize memory replay to speed up learning and to study the role of short-term memory and episode replay in spatial navigation tasks (Zeng et al., 2022). It is possible to turn the memory on or off, limit its size, or change the statistics of how memories are replayed to model the effect of manipulating episodic memory (Diekmann and Cheng, 2022). The action signal is further passed from the Gym interface to the topology graph, where the position and orientation of the artificial agent is computed and sent to the Unity simulator to update the virtual environment, which closes the feedback loop between agent and environment. At the same time, the performance, i.e., escape latency, cumulative response curve, etc., of the DQN agent is monitored live and displayed to the user.

3.2. Fast and easy creation of environments

We evaluated CoBeL-RL on a set of environments that model biological experiments with rodents. CoBeL-RL provides multiple

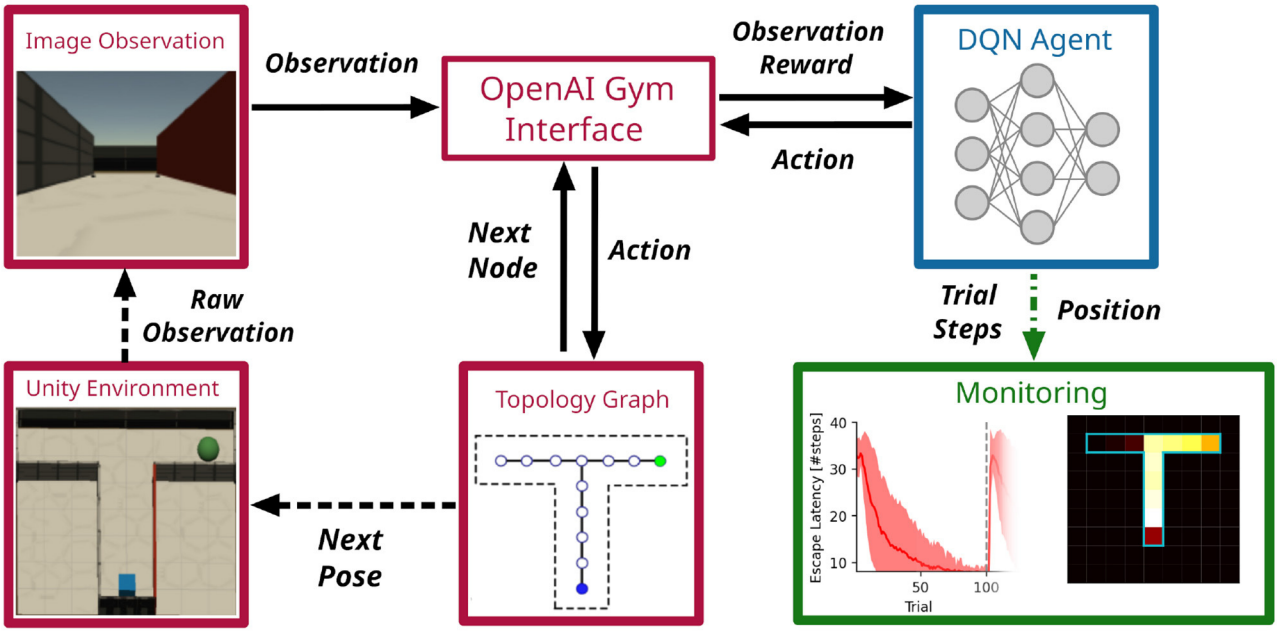

FIGURE 2
The CoBeL-RL framework in use. Example of a simulation loop employing a DQN agent in a virtual 3D environment with visual stimuli. Simulation relevant variables are passed between CoBeL-RL's modules directly (solid arrows) and *via* web sockets between CoBeL-RL and the 3D simulator (dashed arrows). The monitor modules rely on callback to record variables of interest (dashed green arrow).

options to set up a virtual environment. The requirements for the set-up differ based on the type of virtual environment: Gridworld environments require the definition of a transition graph, while 3D environments further require the modeling and texturing of the environment's geometry. In addition to these fully manual methods, CoBeL-RL provides simple visual editors for the quick creation of maze environments at different abstraction levels, i.e., 2D gridworld environments and 3D environments in the Unity Game Engine. As an example we demonstrate how both of them can be used to set up a simple T-maze environment in the following sections.

3.2.1. Gridworld editor

The Gridworld editor provides an intuitive graphical interface for the creation of gridworld environments. After first selecting the height and width of a new gridworld, the editor presents a grid of states which can be interacted with using the mouse (Figure 3A). States can be selected *via* a mouse left click and their properties edited, e.g., the reward associated with a state. Transitions can be edited by double clicking on edges between adjacent states, thereby toggling the transition probability between the states in both directions between zero and one. For more fine grained control, state-action transitions can be manually defined upon selecting the advanced settings option in the properties of a currently selected state. Gridworlds created with the editor are stored using Python's *pickle* module and directly used with CoBeL-RL's gym gridworld and gym discrete interfaces. The gridworld itself is represented as a state-action-state transition matrix, reward function, starting set (i.e., the states at which an agent may start), and terminal set (i.e., the states at which a trial terminates). Additionally, metrics like the number of states, state coordinates, and a list of invalid transitions are also stored.

3.2.2. Unity editor

The Unity Editor can be used to easily build 3D maze environments, which can then be employed as the training and testing environments for the RL agents in CoBeL-RL. An interactive canvas covered with grids in the GUI (Figure 3B) allows the user to simply draw the positions and dimensions of the maze walls as well as the locations of obstacles, an agent and reward(s). The grid size corresponds to one unit length in Unity and the dimension of the entire 2D grids are adjustable. Therefore, the user is able to design mazes of any size and shape. The editor has included a library of different texture materials. After the structure of the maze has been defined, the user can choose to either manually select a material from the library for each object in the maze, or let the editor randomly assign a texture for that same object. The files for all materials are placed in a folder, and the user has the option to add or delete any material from within the GUI. Finally, the Unity scene files can be directly imported into the Unity Editor and start communicating with the CobeL-RL framework *via* the frontend interface for Unity.

3.3. Efficient distributed simulation using CoBeL-RL

CoBeL-RL can be used in a closed-loop simulation to simulate the interaction of the agent and its environment online. Using the renderer online in a closed-loop setup has the advantage of being

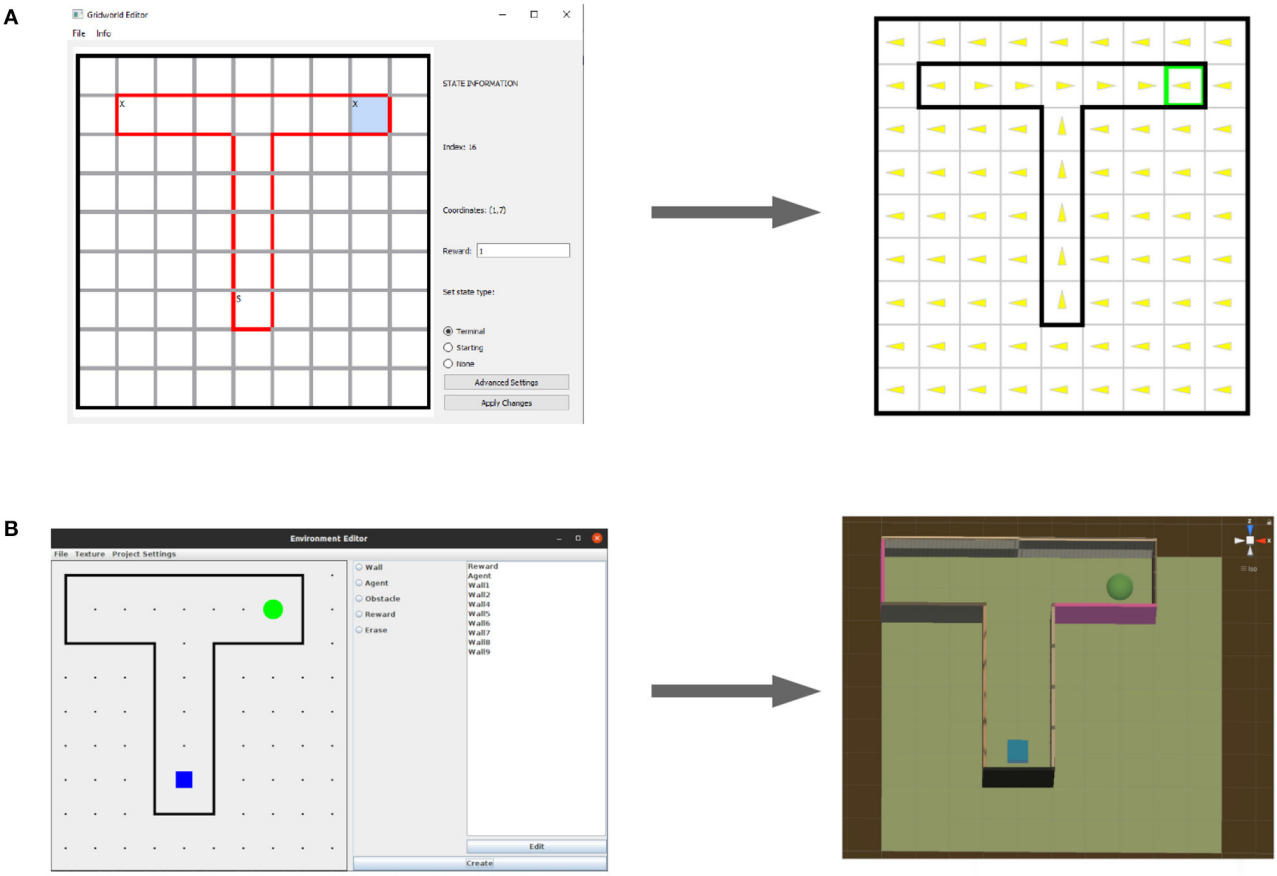

FIGURE 3
World editors. **(A)** The gridworld editor (left) can be used to quickly and intuitively create new environments for simulation (right). **(B)** The editor for 3D Unity environments (left) generates 3D Unity scenes (right).

very flexible and reflecting changes in the environment directly. However, often the environment tested for simple RL agents only consists of a small set of possible observations. For instance, in the Gridworld example in Figure 2, the agent can be located in one of only 12 different positions. Even if multiple configurations, like heading direction or different visual cues are allowed for every location, the total number of possible image observations will still be small compared to the number of learning iterations. In such situations, it may be beneficial to resort to an offline rendering strategy, i.e., to render and store the image observations once at the beginning of the simulation and then retrieve the observations from memory instead of rendering them anew in every time step. This mode is crucial when simulations are run on a remote compute cluster that does not have the required software packages installed to run the renderer locally. CoBeL-RL provides the functionality to switch between online and offline rendering with little changes to the simulation code. In this section we provide a comparison between these two modes in terms of the simulation runtime.

For a numerical comparison, we used a simple T-maze environment that was rendered using Blender and a foraging task to encourage exploration using the MFEC agent (Blundell et al., 2016) (see Section Methods for details). For online rendering, Blender was queried to retrieve image observations at every simulation time step. For the offline rendering case, this was only done once for all possible image observations at the beginning of the simulation. Images were then stored in the main memory for later retrieval during the simulation. Frame time was measured by invoking the system clock at the beginning of every simulation time step. As expected, average frame times over the entire simulation were much higher for online than for offline rendering (Figure 4).

The speedup gained by offline rendering is likely to depend on a number of factors, of which we explore the image size, the number of CPU cores, game engine, and RL algorithm. First, the simulation time is expected to strongly depend on the size of the image observations, we tested different sizes between 20x80 and 75x300 pixels and confirmed that the relative overhead for rendering varied greatly with the image size (Figure 4A). Specifically, we found that for small input image sizes the overhead of rendering with Blender was substantial. For instance, for the smallest tested image sizes of 20 × 80, online rendering frame time was 16.9 ms, more than eleven times the frame time for offline rendering, which was 1.5 ms. For larger input images this difference was significantly less pronounced, e.g., for the 75 × 300 pixel images, mean frame time was 34.1 and 24.2 ms for online and offline rendering, respectively, a factor of only ca. 1.4. Second, somewhat surprisingly increasing the number of CPU

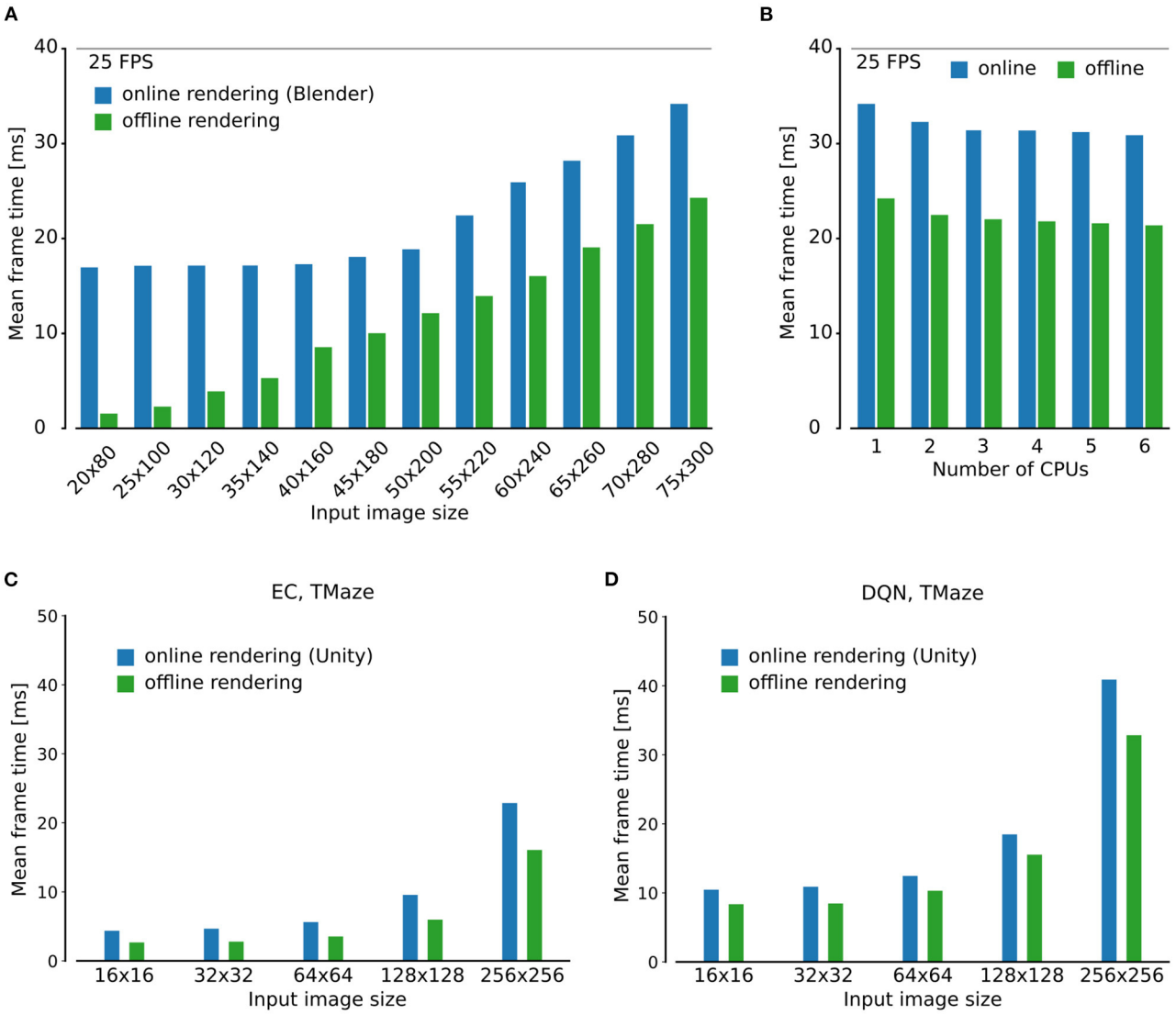

FIGURE 4
Analysis of simulation speed. **(A)** Comparison of simulation time of a single step using the online and offline rendering method with Blender for observations of different size. **(B)** Scaling of run time with number of CPUs. **(C, D)** As in **(A)** but for the Unity rendering instead of Blender, using the MFEC **(A)** and DQN **(B)** agent.

cores beyond two did not have a strong impact on the simulation speed for both online and the offline rendering (Figure 4B). We hypothesize that this is due to the low level of parallelism that can be achieved in the small models that we studied here. Third, scaling of frame time with image size was qualitatively similar for the Unity renderer (Figures 4C, D). Finally, we compared two different RL agents, MFEC and DQN, where the latter is more computationally expensive than the former because of the need for GPU computing. The relative overhead for rendering increases for both agents when the image size expands. On the other hand, although the total frame time for the DQN is larger than that for MFEC, the relative overhead for rendering is comparable between the two agents for all image sizes (Figures 4C, D). This result shows that the contribution of the agent and the renderer to the total simulation time, are modular and do not much interfere with one another.

In summary, we find that offline rendering is especially beneficial if small image observations are used since the overhead for rendering dominates the simulation in this situation. Frame time speedups are achieved with offline rendering for all frame sizes, but this method is only suitable if the total number of possible image observations is small.

3.4. Analysis of behavior with CoBeL-RL

To analyze the behavior of the simulated agents, relevant parameters have to be recorded and evaluated. To do so, CoBeL-RL provides various monitors with which relevant variables can be recorded, visualized, and evaluated during training (Figure 5). Monitors are available for tracking trial reward, escape latency, and cumulative responses. For the last monitor, the specific response

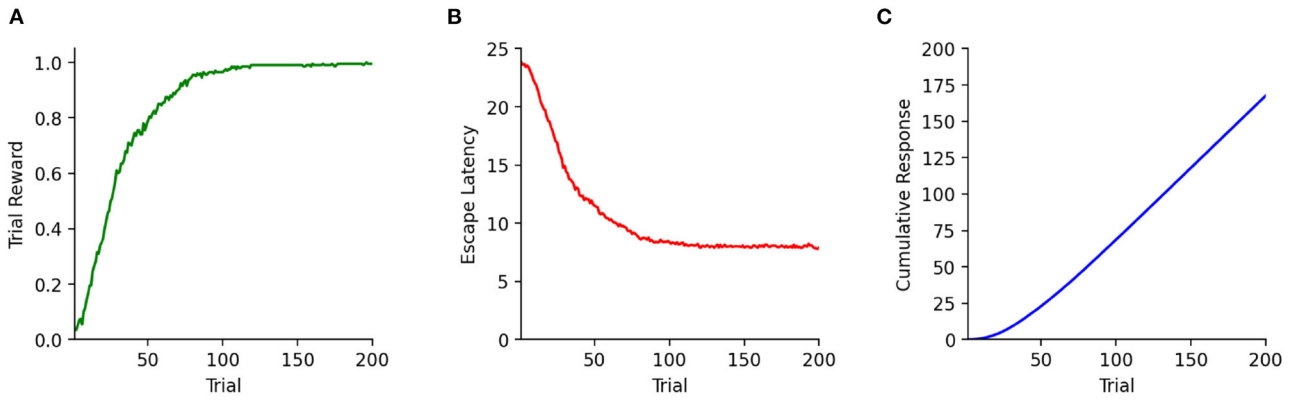

FIGURE 5
Commonly needed monitors in CoBeL-RL. **(A)** The reward monitor tracks the cumulative reward collected during each trial. **(B)** The escape latency (in number of trial steps) can be tracked with the escape latency monitor. **(C)** The response monitor can track the cumulative responses emitted by the agent. Per default responses are coded with one when a reward was collected and zero otherwise. The coding of responses can be customized by using the simulation loop's callback functionality.

coding can be defined by the user. The default coding is one when reward was collected in a trial, and zero if not. These three behavioral performance monitors are compatible with all agents. Below we demonstrate the use of CoBeL-RL's performance monitors in two example experimental paradigms.

The analysis of responses emitted by an agent is important to understand the reinforcement of behaviors in classical and instrumental conditioning. We therefore simulated a simple extinction learning paradigm: The agent was placed in a T-maze environment and rewarded for choosing the right arm during the first 100 trials. Then reward was moved to the left arm for the remaining 100 trials. Responses are visualized by the cumulative response curves, where each trial with the choice of the right arm was encoded as one and the choice of the left (or neither) arm as zero, for a DQN agent in a complex 3D environment with visual stimuli (Figure 6A) and for a tabular Dyna-Q agent in a simple gridworld (Figure 6B).

Another example is latent learning in the Blodgett maze (Blodgett, 1929), a composition of 6-Unit Alley T-maze (Figure 7A), which we simulated using the Dyna-DSR agent in CoBeL-RL (see Section Methods for details). Agents were tested in two settings: latent learning, in which the environment was devoid of reward for the first 100 trials (exploration phase; −100 to −1 trials) and reward was introduced for the remaining trials (Figure 7B, purple line.), and direct learning, in which there was no exploration phase and reward was present from the first trial in the maze (Figure 7B, orange line). Latent learning agents learned to reach the goal state faster than direct learning agents after the reward was introduced. These results qualitatively reproduce experimental findings (Blodgett, 1929; Reynolds, 1945; Tolman, 1948).

3.5. Analysis of neural representations with CoBeL-RL

An important goal of CoBeL-RL is to allow the analysis of not only behavioral measures of the reinforcement learning agents, but also the computations and representations that emerge in the deep neural network to support behavior. To this end, CoBeL-RL provides response monitors which can track network representations of Deep RL agents. In the latent learning simulations above, we analyzed the learned Deep SR of the *start* state to examine what the agent had learned. At the end of the training phase (Figure 7C, left), the learned SR mainly represents states close to the start state, simply reflecting the environment's topology. In contrast, by the end of the simulation (Figure 7C, right) the learned SR represents the path to the goal state.

CoBeL-RL also includes analysis tools to analyze spatial representations, including the ability to compute spatial activity maps, classify them, and identify units in the network that have place fields. To demonstrate this functionality, we built a simple Blender environment (Figure 8A) and trained the agent to solve a goal finding task on a hexagonal topology graph (Figure 8B). In each trial, the agent had to navigate to the unmarked goal node (indicated in green) from a random starting position. The agent had 12 available actions—six translations and six rotations. The translations allowed the agent to move to any of the six neighboring nodes on the topological graph, and the rotations turned the agent in place to face a neighboring node. We generated spatial activity maps by placing the agent in all locations of a 25 × 25 grid that was embedded in the environment, which can be set flexibly depending on the desired resolution in CoBeL-RL, and recorded the activation of nodes in the DQN network. Place field maps of all units preceding the output action selection layer of the Deep Q-Network were recorded every 800 trials during the training process. Recordings were obtained for all six heading direction of the agent and further processed to identify units that exhibit place-cell-like firing (Figure 8C)—see Section 2 for a detailed description. As would be expected, if place-field-like representations supported spatial navigation, the number of neurons that exhibit place fields increased during training (Figure 8D). In general, the spatial activation maps can be constructed either at set points during the training process to understand how they evolve with learning, as in this example, or after the training is completed.

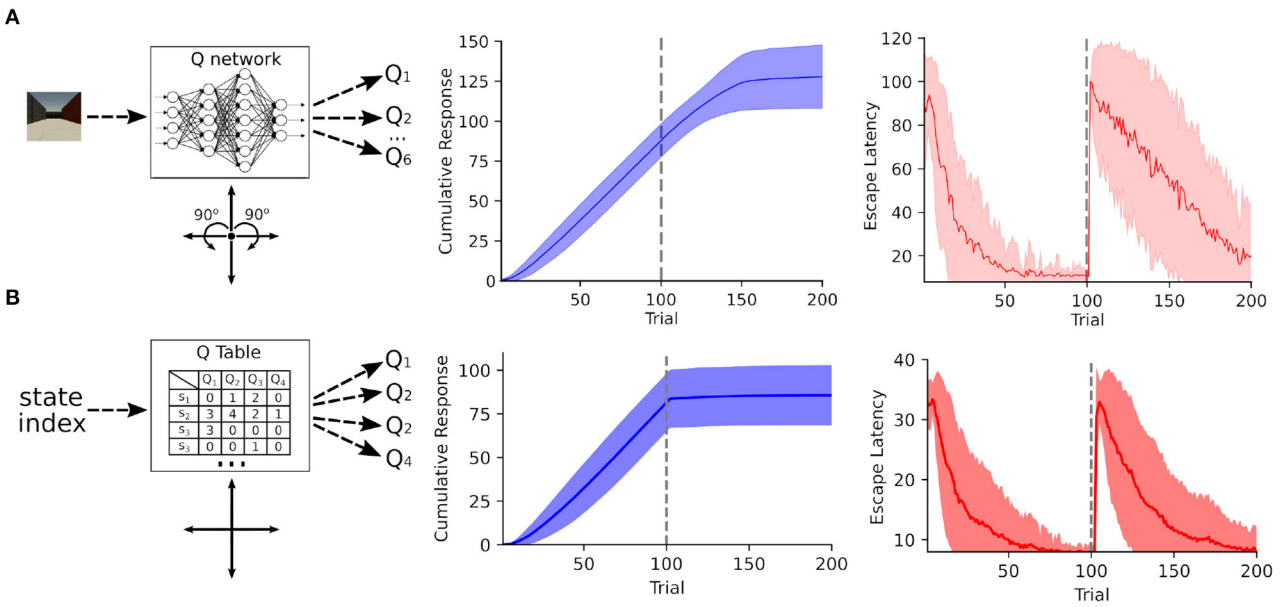

FIGURE 6
Analysis of behavior in an extinction learning paradigm. **(A)** A DQN agent was trained on a simple extinction paradigm within a complex 3D Unity environment (left). The agent has six actions at its disposal: movement in either of the four cardinal directions and rotating either left or right by 90°. The cumulative response (center) reveals that the agent reliable picks the rewarded right arm for the first 100 trials. After the right arm was no longer rewarded (trials 101–200), the previously learned behavior initially persists and then gradually extinguishes. The escape latency curves (right) show that agent quickly learned to reach the right arm. After the reward switched the agent slowly adapted to the new reward location. **(B)** Same as **(A)** but recorded from a Dyna-Q agent (left) in a gridworld environment. Unlike the DQN agent the Dyna-Q agent can only move in the four cardinal directions. Similar to the DQN agent the Dyna-Q agent reliably picks the rewarded right arm for the first 100 trials. However, due to the Q-function's tabular representation the previously learned behavior extinguishes rapidly.

3.6. Modeling of non-spatial tasks with CoBeL-RL

So far we focused on the modeling of spatial tasks since CoBeL-RL mainly supports spatial environments. However, non-spatial tasks, e.g., alternative forced choice tasks (Ratcliff et al., 2003; Brünken et al., 2004), visual search tasks (Sheliga et al., 1994; Reavis et al., 2016) and classical conditioning (Ernst et al., 2019; Batsikadze et al., 2022) are ubiquitous in neuroscience. To demonstrate that CoBeL-RL can easily be used to accommodate such non-spatial tasks we model a simple decision task. We first set up a custom interface for CoBeL-RL which stores a set of stimuli and a set of choice trials. One-hot vectors serve as stimuli in our simulation. Each choice trial C_i consists of two stimuli which are contained within the set of stimuli. One stimulus represents the correct choice and is rewarded, the other stimulus represents the incorrect choice and is not rewarded (Figure 9A). During each trial of a training session the interface picks one of the choice trials at random. Both stimuli are presented together in a randomized order (Figure 9B). We train 100 DQN agents for 50 trials each on the task. The reward attained we record with CoBeL-RL's reward monitor. By inspecting the average reward attained on each trial we see that the agents initially perform at chance level and then quickly learn the task within a couple of trials (Figure 9C). As this simulation serves only as a demonstration of CoBeL-RL's extendability we used very simple stimuli. However, they can be replaced by complex visual stimuli and the interface can be extended with more complex task rules.

4. Discussion

In this paper, we introduced CoBeL-RL, a RL framework oriented toward computational neuroscience, which provides a large range of environments, established RL models and analysis tools, and can be used to simulate a variety of behavioral tasks. Already, a set of computational studies focusing on explaining animal behavior (Walther et al., 2021; Zeng et al., 2022) as well as neural activity (Diekmann and Cheng, 2022; Vijayabaskaran and Cheng, 2022) have employed predecessor versions of CoBeL-RL. The framework has been expanded and refined since these earlier studies. Here, we provided additional details about the simulation framework and how it can be extended and used in future work.

4.1. Flexibility and extensibility

As discussed in Section 2, CoBeL-RL provides multiple RL agents with the option of integrating additional agents not included in the default implementation. Since RL in general refers to a class of learning problems, several learning algorithms and architectures that solve the problem fall under its umbrella. This naturally raises the question of which agent is best suited to model a given task. While there is no straightforward way to tell, the goals of the computational modeling can help narrow the search for an appropriate agent. For instance, a deep RL agent such a DQN (and not a tabular agent) would be appropriate if we wanted to use

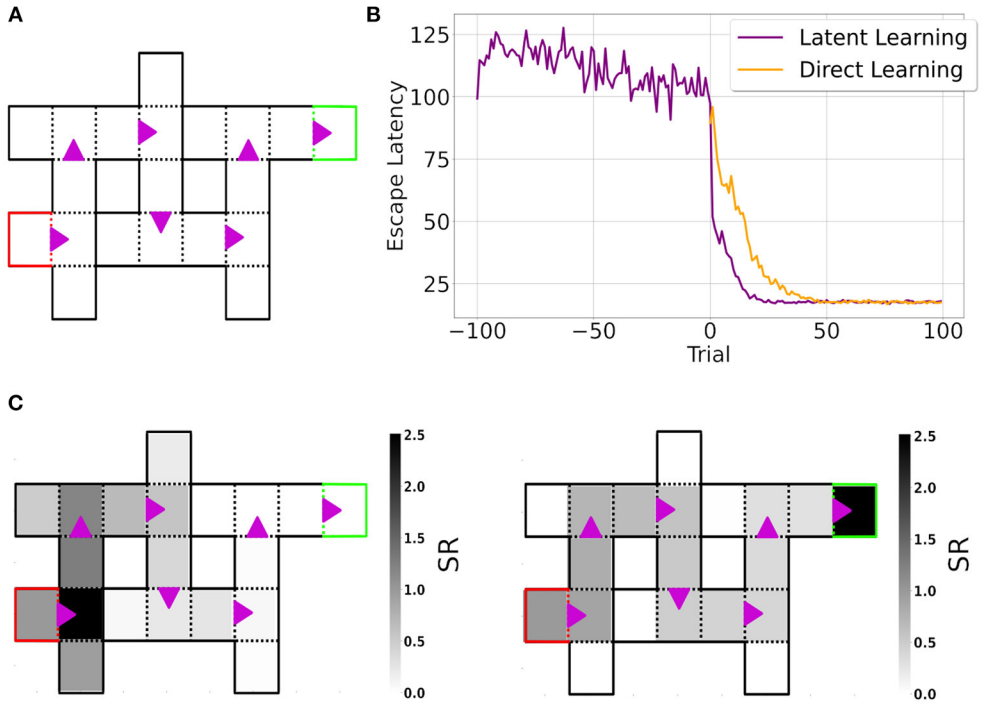

FIGURE 7
Simulation of latent learning using CoBeL-RL. **(A)** Gridworld of the Blodgett maze. The red and green squares represent the start and goal states, respectively. Purple arrows indicate transitions that are only allowed in the indicated direction. **(B)** Escape latency of agents trained in latent learning (purple) and direct learning (orange) settings. For better comparison, the curve for the direct learning setting was shifted to the point of reward introduction of the latent learning setting. Curves represent means over 30 simulations per agent. **(C)** Learned Deep SR of the start state at the end of the training phase (left) and at the end of the simulation (right). At the end of the training phase, the learned SR mainly represents states close to the start state. In contrast, by the end of the simulation, the learned SR represents the path to the goal state. The color bar represents the learned cumulative discounted (state) occupancy.

complex visual stimuli directly as observations, or if generalization to new situations is particularly important. A deep RL agent is once again appropriate if we want to study the neural representations in the agent. On the other hand, a tabular agent might be preferred if one's main interest is modeling behavioral statistics due to its interpretability and decreased computational load.

Other factors may influence the choice of agent. Some models suggest that model-based RL, in which the agent learns and plans on a model of the environment, is better suited to modeling cognitive processes that require some level of abstraction and planning, whereas model-free RL is better suited to modeling automatic, behavioral response learning (Chavarriaga et al., 2005; Khamassi and Humphries, 2012). Within CoBeL-RL, the DQN and MFEC agents are model-free, while the Dyna-Q, Dyna-DQN, and DSR agents represent a middle-ground between model-free and model-based RL. Additionally, many RL algorithms use some form of memory and experience replay, which might also be of interest to those looking to model hippocampal replay. The different replay mechanisms in RL agents, and how they might relate to replay in the hippocampus are reviewed extensively in Cazé et al. (2018).

We discuss only a few of the possible factors that influence the choice of agent here, however, the use of RL to model experiments in neuroscience and behavioral psychology has been reviewed in Subramanian et al. (2022). Botvinick et al. (2020) review Deep RL algorithms and their implications for neuroscience, and Bermudez-Contreras et al. (2020) focus specifically on RL models of spatial navigation.

CoBeL-RL can also be easily integrated with other existing RL simulation frameworks for machine learning. For example many gridworld environments, such as MiniGrid (Chevalier-Boisvert et al., 2018), are based on OpenAI Gym and could therefore be easily integrated with CoBeL-RL.

4.2. Limitations and future developments

CoBeL-RL provides an efficient software framework for simulating closed-loop trial-based learning, which is well matched for modeling behavioral experiments. Nevertheless, experiments that do not have a strict trial-based structure or well-defined time horizon could be modeled in CoBeL-RL as well by controlling the training loop *via* CoBeL-RL's callback functionality, although such simulations would be less efficient. In addition, CoBeL-RL's monitoring tools would require adaptation to account for the potentially undefined time horizon and lack of trial structure.

The experimental paradigms discussed here only include one agent. However, the effect of another agent's presence on behavior (Boesch and Tomasello, 1998; Krützen et al., 2005) and neural activity (Rizzolatti and Craighero, 2004; Mukamel et al., 2010) has also been of great interest. Currently, CoBeL-RL does not support simulations that involve multiple agents and at first glance it is not

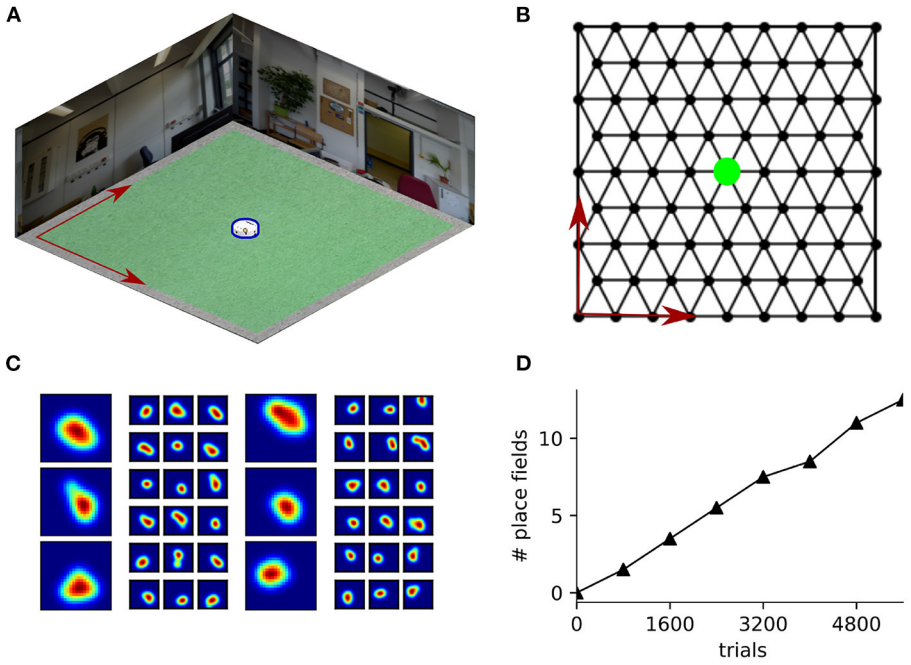

FIGURE 8
Analysis of spatial representations using CoBeL-RL. **(A)** Blender environment used for simulating a navigation task. The agent is marked in blue. **(B)** Topology graph on which the agent navigates. The location of the unmarked goal is indicated in green. **(C)** Example place cells emerging in the simulation. The large maps show the place field activity averaged over all head directions and the adjacent six smaller maps show the corresponding activity at each head direction, spaced 60° apart. **(D)** Evolution in number of place fields with learning trials.

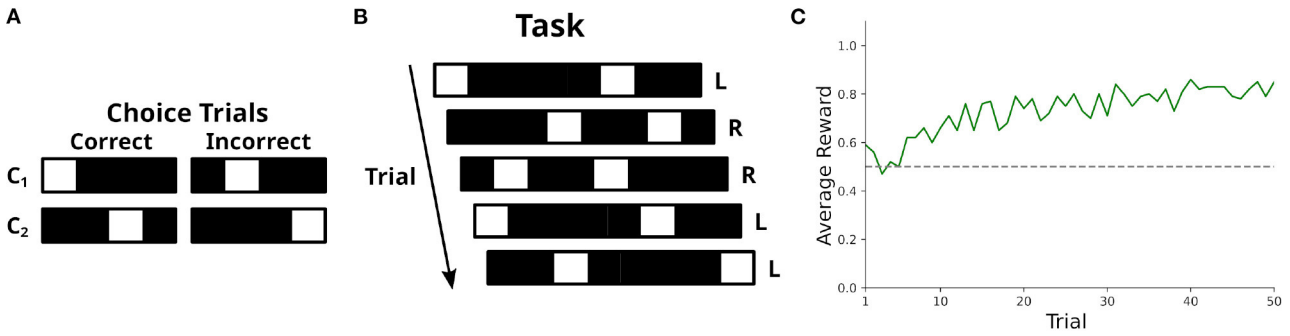

FIGURE 9
Modeling of non-spatial tasks using CoBeL-RL. **(A)** Definition of the possible choice trials. Each choice trial consists of two stimuli: the correct one, which is rewarded, and the incorrect one, which is not rewarded. **(B)** Illustration of the decision task. In each trial, a random choice trial is presented and the positions of the stimuli are randomized. The correct choice for the trial is indicated as left (capital L) or right (capital R). **(C)** Learning performance of a DQN agent in the task measured *via* the average reward received. Performance is at chance level (indicated by the dashed horizontal gray line) in the beginning but quickly improves within the first couple of trials.

clear how they could fit in the trial-based training loop. However, such multi-agent simulations could be facilitated by extending CoBeL-RL with an abstract supervisor agent which can encapsulate an arbitrary number of agents. The training loop of this abstract supervisor agent could then control the training of the agents it supervises. The behavior of the agents could be easily monitored by creating separate instances of the monitoring classes for each agent.

CoBeL-RL was initially developed to model tasks and address questions in the field of spatial navigation. Because of this the environments and interfaces implemented mainly accommodate spatial navigation tasks. Nonetheless, CoBeL-RL is fairly adaptable and can be used to model non-spatial tasks as we have demonstrated in the Results. Furthermore, many of CoBeL-RL's monitor classes do not track exclusively spatial variables, i.e., reward monitor, response monitor and representation monitor, and can therefore be applied to non-spatial tasks as well. CoBeL-RL may also offer a convenient way of implementing non-spatial tasks that require 3D environments, e.g., visual search tasks, visual fixation tasks or tasks involving the manipulation of objects. The supported 3D simulators also allow for the change of an agent's orientation, thereby facilitating the implementation of visual search and fixation tasks. The manipulation of simulation

objects is implemented to some extent, e.g., change of object pose and material, and can be further developed to allow for the manipulation of arbitrary simulation objects. Since CoBeL-RL implements agent-environment interaction with the wide-spread gym interface many existing implementations of non-spatial tasks should be easily integrable.

The analysis of the selectivity of neuronal activity at single neuron and network levels has contributed greatly to our understanding of cognitive function and representations learned by the brain (Watkins and Berkley, 1974; De Baene et al., 2008; Decramer et al., 2019; Packheiser et al., 2021; Vaccari et al., 2022). Similarly, understanding the representations learned by Deep RL agents and how they relate to the current task has been of great interest early on (Mnih et al., 2015), and they have proven to be a useful tool in understanding the emergence of spatial representations, e.g., grid cells (Banino et al., 2018) and place cells (Vijayabaskaran and Cheng, 2022), and units encoding for other task-relevant variables (Wang et al., 2018; Cross et al., 2021), e.g., time cells (Lin and Richards, 2021). We showed how CoBeL-RL can be used to record and analyze the network activity of Deep RL agents as they learn a task, and we did so at the level of both single and multiple units, i.e., analysis of place cells and the deep successor representation. Additionally, CoBeL-RL has been used to understand the emergence of other spatial representations, e.g., head direction modulated cells, and their dependence on navigational strategy employed (Vijayabaskaran and Cheng, 2022). The initial version of the framework was used to analyze representational changes resulting from the learning of context-specific behavior in an extinction learning paradigm (Walther et al., 2021). CoBeL-RL currently only provides a small repertoire for the analysis of network activity with a focus on spatial representations. Analysis tools could potentially be expanded to also include unit selectivity analysis. Such analysis would synergize well with the already existing, and potential future, behavioral monitors. Another type of analysis which would benefit the framework is representational similarity analysis (RSA) which has been a useful tool to compare brain activity with internal representations of computational models (Kriegeskorte, 2008).

Currently, CoBeL-RL largely relies on the Tensorflow-2-based implementations of Deep RL agents provided by Keras-RL2. This strong reliance on Tensorflow 2 could be detrimental to the appeal and longevity of CoBeL-RL: Multiple programming libraries for the implementation of DNNs exist (Al-Rfou et al., 2016; Paszke et al., 2019) and change over time. Agents would have to be re-implemented for different libraries and updated whenever a library's behavior changes. We address these problems with the use of an abstract network class which serves as an interface between Deep RL agents and specific network implementations. Currently, network classes for Tensorflow 2 and PyTorch are supported.

As a framework, CoBeL-RL is continuously developed and extended. While current efforts focus on simulations in virtual environments, CoBeL-RL can be connected to physical robots like the Khepera-IV (Tharin et al., 2019). Agents can be trained efficiently in simulation and then take control of the physical robot. Pretraining an agent in simulations before letting it control a real robot has been shown to work in other settings (James and Johns, 2016; Tzeng et al., 2017).

4.3. Conclusion

In conclusion, CoBeL-RL is an RL framework oriented toward computational neuroscience that fills a gap in the landscape of simulation software, which currently focuses mostly on machine learning, a specific task paradigm, or certain type of model. Importantly, CoBeL-RL provides a set of tools which simplify the process of setting up simulations through its environment editors. This is the case especially in the context of 3D simulations since otherwise their creation would require the user to acquire a wide range of additional skills, e.g., 3D modeling, game engine programming, etc. CoBeL-RL hence greatly reduces the overhead of setting up closed-loop simulations which are required to understand the computational issues that animals face in behavioral tasks and the solutions that they generate.

Data availability statement

The datasets presented in this study can be found in online repositories. The names of the repository/repositories and accession number(s) can be found at: https://github.com/sencheng/CoBeL-RL; https://github.com/sencheng/CoBeL-RL-Paper-Simulations.

Author contributions

ND, SV, and SC contributed to conception and design of the framework. ND, SV, XZ, DK, and MM performed and analyzed the simulations and contributed to the code. All authors contributed to the first draft of the manuscript, edited, and approved the submitted version.

Funding

This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation), project numbers 419037518 (FOR 2812, P2) and 316803389 (SFB 1280, A14).

Acknowledgments

We thank Dr.-Ing. Thomas Walther for the conceptualization and development of CoBeL-RL's initial version.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Available online at: tensorflow.org

Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., Ballas, N., et al. (2016). Theano: A Python framework for fast computation of mathematical expressions. *arXiv [Preprint]*. arXiv:1605.02688. doi: 10.48550/arXiv.1605.02688

Banino, A., Barry, C., Uria, B., Blundell, C., Lillicrap, T., Mirowski, P., et al. (2018). Vector-based navigation using grid-like representations in artificial agents. *Nature* 557, 429–433. doi: 10.1038/s41586-018-0102-6

Bathellier, B., Tee, S. P., Hrovat, C., and Rumpel, S. (2013). A multiplicative reinforcement learning model capturing learning dynamics and interindividual variability in mice. *Proc. Natl. Acad. of Sci. U.S.A.* 110, 19950–19955. doi: 10.1073/pnas.1312125110

Batsikadze, G., Diekmann, N., Ernst, T. M., Klein, M., Maderwald, S., Deuschl, C., et al. (2022). The cerebellum contributes to context-effects during fear extinction learning: a 7T fMRI study. *NeuroImage* 253:119080. doi: 10.1016/j.neuroimage.2022.119080

Beattie, C., Leibo, J. Z., Teplyashin, D., Ward, T., Wainwright, M., Küttler, H., et al. (2016). DeepMind Lab. *arXiv [Preprint]*. arXiv:1612.03801. doi: 10.48550/arXiv.1612.03801

Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 509–517. doi: 10.1145/361002.361007

Bermudez-Contreras, E., Clark, B. J., and Wilber, A. (2020). The neuroscience of spatial navigation and the relationship to artificial intelligence. *Front. Comput. Neurosci*. 14:63. doi: 10.3389/fncom.2020.00063

Blender Online Community (2018). *Blender is the Free and Open Source 3D Creation Suite*. Amsterdam: Blender Foundation; Blender Institute.

Blodgett, H. C. (1929). *The Effect of the Introduction of Reward Upon the Maze Performance of Rats*. University of California Publications in Psychology, 114–134.

Blundell, C., Uria, B., Pritzel, A., Li, Y., Ruderman, A., Leibo, J. Z., et al. (2016). Model-free episodic control. *arXiv [Preprint]*. arXiv:1606.04460. doi: 10.48550/arXiv.1606.04460

Boesch, C., and Tomasello, M. (1998). Chimpanzee and human cultures. *Curr. Anthropol*. 39, 591–614. doi: 10.1086/204785

Botvinick, M., Wang, J. X., Dabney, W., Miller, K. J., and Kurth-Nelson, Z. (2020). Deep reinforcement learning and its neuroscientific implications. *Neuron* 107, 603–616. doi: 10.1016/j.neuron.2020.06.014

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., et al. (2016). OpenAI Gym. *arXiv [Preprint]*. arXiv:1606.01540. doi: 10.48550/arXiv.1606.01540

Brünken, R., Plass, J. L., and Leutner, D. (2004). Assessment of cognitive load in multimedia learning with dual-task methodology: auditory load and modality effects. *Instruct. Sci*. 32, 115–132. doi: 10.1023/B:TRUC.0000021812.96911.c5

Cazé, R., Khamassi, M., Aubin, L., and Girard, B. (2018). Hippocampal replays under the scrutiny of reinforcement learning models. *J. Neurophysiol*. 120, 2877–2896. doi: 10.1152/jn.00145.2018

Chavarriaga, R., Strösslin, T., Sheynikhovich, D., and Gerstner, W. (2005). A computational model of parallel navigation systems in rodents. *Neuroinformatics* 3, 223–242. doi: 10.1385/NI:3:3:223

Chevalier-Boisvert, M., Willems, L., and Pal, S. (2018). *Minimalistic Gridworld Environment for Openai Gym*. Available online at: https://github.com/maximecb/gym-minigrid

Cross, L., Cockburn, J., Yue, Y., and O'Doherty, J. P. (2021). Using deep reinforcement learning to reveal how the brain encodes abstract state-space representations in high-dimensional environments. *Neuron* 109, 724–738.e7. doi: 10.1016/j.neuron.2020.11.021

Cueva, C. J., and Wei, X. -X. (2018). Emergence of grid-like representations by training recurrent neural networks to perform spatial localization. *arXiv [Preprint]*. arXiv:1803.07770. doi: 10.48550/arXiv.1803.07770

De Baene, W., Ons, B., Wagemans, J., and Vogels, R. (2008). Effects of category learning on the stimulus selectivity of macaque inferior temporal neurons. *Learn. Mem*. 15, 717–727. doi: 10.1101/lm.1040508

Decramer, T., Premereur, E., Uytterhoeven, M., Van Paesschen, W., van Loon, J., Janssen, P., et al. (2019). Single-cell selectivity and functional architecture of human lateral occipital complex. *PLoS Biol*. 17:e3000280. doi: 10.1371/journal.pbio.3000280

Diekmann, N., and Cheng, S. (2022). A model of hippocampal replay driven by experience and environmental structure facilitates spatial learning. *bioRxiv [Preprint]*. doi: 10.1101/2022.07.26.501588

Eppler, J., Helias, M., Muller, E., Diesmann, M., and Gewaltig, M.-O. (2009). PyNEST: a convenient interface to the NEST simulator. *Front. Neuroinform*. 2:8. doi: 10.3389/neuro.11.012.2008

Ernst, T. M., Brol, A. E., Gratz, M., Ritter, C., Bingel, U., Schlamann, M., et al. (2019). The cerebellum is involved in processing of predictions and prediction errors in a fear conditioning paradigm. *eLife* 8:e46831. doi: 10.7554/eLife.46831

Hafting, T., Fyhn, M., Molden, S., Moser, M.-B., and Moser, E. I. (2005). Microstructure of a spatial map in the entorhinal cortex. *Nature* 436, 801–806. doi: 10.1038/nature03721

James, S., and Johns, E. (2016). 3D simulation for robot arm control with deep Q-learning. *arXiv [Preprint]*. arXiv:1609.03759. doi: 10.48550/arXiv.1609.03759

Juliani, A., Berges, V.- P., Teng, E., Cohen, A., Harper, J., Elion, C., et al. (2020). Unity: a general platform for intelligent agents. *arXiv [Preprint]*. arXiv:1809.02627. doi: 10.48550/arXiv.1809.02627

Kaiser, J., Hoff, M., Konle, A., Vasquez Tieck, J. C., Kappel, D., Reichard, D., et al. (2019). Embodied synaptic plasticity with online reinforcement learning. *Front. Neurorobot*. 13:81. doi: 10.3389/fnbot.2019.00081

Khamassi, M., and Humphries, M. D. (2012). Integrating cortico-limbic-basal ganglia architectures for learning model-based and model-free navigation strategies. *Front. Behav. Neurosci*. 6:79. doi: 10.3389/fnbeh.2012.00079

Koay, S. A., Thiberge, S., Brody, C. D., and Tank, D. W. (2020). Amplitude modulations of cortical sensory responses in pulsatile evidence accumulation. *eLife* 9:e60628. doi: 10.7554/eLife.60628

Kosaki, Y., Lin, T.-C. E., Horne, M. R., Pearce, J. M., and Gilroy, K. E. (2014). The role of the hippocampus in passive and active spatial learning. *Hippocampus* 24, 1633–1652. doi: 10.1002/hipo.22343

Kriegeskorte, N. (2008). Representational similarity analysis - connecting the branches of systems neuroscience. *Front. Syst. Neurosci*. 2:4 doi: 10.3389/neuro.06.004.2008

Krützen, M., Mann, J., Heithaus, M. R., Connor, R. C., Bejder, L., and Sherwin, W. B. (2005). Cultural transmission of tool use in bottlenose dolphins. *Proc. Natl. Acad. Sci. U.S.A*. 102, 8939–8943. doi: 10.1073/pnas.0500232102

Kulkarni, T. D., Saeedi, A., Gautam, S., and Gershman, S. J. (2016). Deep successor reinforcement learning. *arXiv [Preprint]*. arXiv:1606.02396. doi: 10.48550/arXiv.1606.02396

Leibo, J. Z., d'Autume, C. de M., Zoran, D., Amos, D., Beattie, C., Anderson, K., et al. (2018). Psychlab: A psychology laboratory for deep reinforcement learning agents. *arXiv [Preprint]*. arXiv:1801.08116. doi: 10.48550/arXiv.1801.08116

Liang, E., Liaw, R., Moritz, P., Nishihara, R., Fox, R., Goldberg, K., et al. (2018). RLlib: Abstractions for distributed reinforcement learning. *arXiv [Preprint]*. arXiv:1712.09381. doi: 10.48550/arXiv.1712.09381

Lin, D., and Richards, B. A. (2021). Time cell encoding in deep reinforcement learning agents depends on mnemonic demands. *bioRxiv [Preprint]*. doi: 10.1101/2021.07.15.452557

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Mach. Learn*. 8, 293–321. doi: 10.1007/BF00992699

Lin, L. -J. (1993). *Reinforcement learning for robots using neural networks* (PhD Thesis). Carnegie Mellon University, Pittsburgh, PA, United States.

Linietsky, J., and Manzur, A. (2007). *Godot Engine*. Godot.

Mattar, M. G., and Daw, N. D. (2018). Prioritized memory access explains planning and hippocampal replay. *Nat. Neurosci*. 21, 1609–1617. doi: 10.1038/s41593-018-0232-z

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature* 518, 529–533. doi: 10.1038/nature14236

Mukamel, R., Ekstrom, A. D., Kaplan, J., Iacoboni, M., and Fried, I. (2010). Single-neuron responses in humans during execution and observation of actions. *Curr. Biol.* 20, 750–756. doi: 10.1016/j.cub.2010.02.045

Nieh, E. H., Schottdorf, M., Freeman, N. W., Low, R. J., Lewallen, S., Koay, S. A., et al. (2021). Geometry of abstract learned knowledge in the hippocampus. *Nature* 595, 80–84. doi: 10.1038/s41586-021-03652-7

O'Keefe, J., and Dostrovsky, J. (1971). The hippocampus as a spatial map. Preliminary evidence from unit activity in the freely-moving rat. *Brain Res.* 34, 171–175. doi: 10.1016/0006-8993(71)90358-1

Packheiser, J., Donoso, J. R., Cheng, S., Güntürkün, O., and Pusch, R. (2021). Trial-by-trial dynamics of reward prediction error-associated signals during extinction learning and renewal. *Prog. Neurobiol.* 197:101901. doi: 10.1016/j.pneurobio.2020.101901

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., et al. (2019). "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems* (Vancouver, BC), 8024–8035. doi: 10.48550/arXiv.1912.01703

Pinto, L., Koay, S. A., Engelhard, B., Yoon, A. M., Deverett, B., Thiberge, S. Y., et al. (2018). An accumulation-of-evidence task using visual pulses for mice navigating in virtual reality. *Front. Behav. Neurosci.* 12:36. doi: 10.3389/fnbeh.2018.00036

Plappert, M. (2016). *keras-rl*. Available online at: https://github.com/keras-rl/keras-rl

Ratcliff, R., Cherian, A., and Segraves, M. (2003). A comparison of macaque behavior and superior colliculus neuronal activity to predictions from models of two-choice decisions. *J. Neurophysiol.* 90, 1392–1407. doi: 10.1152/jn.01049.2002

Reavis, E. A., Frank, S. M., Greenlee, M. W., and Tse, P. U. (2016). Neural correlates of context-dependent feature conjunction learning in visual search tasks: visual feature conjunction learning. *Hum. Brain Mapp.* 37, 2319–2330. doi: 10.1002/hbm.23176

Redish, A. D., Jensen, S., Johnson, A., and Kurth-Nelson, Z. (2007). Reconciling reinforcement learning models with behavioral extinction and renewal: implications for addiction, relapse, and problem gambling. *Psychol. Rev.* 114, 784–805. doi: 10.1037/0033-295X.114.3.784

Reynolds, B. (1945). A repetition of the blodgett experiment on 'latent learning'. *J. Exp. Psychol.* 35:504. doi: 10.1037/h0060742

Rizzolatti, G., and Craighero, L. (2004). The mirror-neuron system. *Annu. Rev. Neurosci.* 27, 169–192. doi: 10.1146/annurev.neuro.27.070203.144230

Rossum (1995). *Python Reference Manual*. Rossum.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). Prioritized experience replay. *arXiv [Preprint]*. arXiv:1511.05952. doi: 10.48550/arXiv.1511.05952

Schönfeld, F., and Wiskott, L. (2013). RatLab: an easy to use tool for place code simulations. *Front. Comput. Neurosci.* 7:104. doi: 10.3389/fncom.2013.00104

Schönfeld, F., and Wiskott, L. (2015). Modeling place field activity with hierarchical slow feature analysis. *Front. Comput. Neurosci.* 9:51. doi: 10.3389/fncom.2015.00051

Schultz, W., Dayan, P., and Montague, P. R. (1997). A neural substrate of prediction and reward. *Science* 275, 1593–1599. doi: 10.1126/science.275.5306.1593

Sheliga, B., Riggio, L., and Rizzolatti, G. (1994). Orienting of attention and eye movements. *Exp. Brain Res.* 98, 507–522. doi: 10.1007/BF00233988

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., et al. (2016). Mastering the game of go with deep neural networks and tree search. *Nature* 529, 484–489. doi: 10.1038/nature16961

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., et al. (2017). Mastering the game of go without human knowledge. *Nature* 550, 354–359. doi: 10.1038/nature24270

Subramanian, A., Chitlangia, S., and Baths, V. (2022). Reinforcement learning and its connections with neuroscience and psychology. *Neural Netw.* 145, 271–287. doi: 10.1016/j.neunet.2021.10.003

Sutton, R. S., and Barto, A. G. (2018). *Reinforcement Learning: An Introduction. Adaptive Computation and Machine Learning Series, 2nd Edn*. Cambridge, MA: The MIT Press.

Terry, J. K., Black, B., and Jayakumar, M. (2020). *Magent*. Available online at: https://github.com/Farama-Foundation/MAgent. GitHub repository.

Tharin, J., Lambercy, F., and Carron, T. (2019). *Khepera IV User Manual*. K-Team.

Tolman, E. C. (1948). Cognitive maps in rats and men. *Psychol. Rev.* 55:189. doi: 10.1037/h0061626

Tzeng, E., Devin, C., Hoffman, J., Finn, C., Abbeel, P., Levine, S., et al. (2017). Adapting deep visuomotor representations with weak pairwise constraints. *arXiv [Preprint]*. arXiv:1511.07111. doi: 10.48550/arXiv.1511.07111

Unity Technologies (2005). *Unity*. Unity Technologies.

Vaccari, F. E., Diomedi, S., Filippini, M., Hadjidimitrakis, K., and Fattori, P. (2022). New insights on single-neuron selectivity in the era of population-level approaches. *Front. Integr. Neurosci.* 16:929052. doi: 10.3389/fnint.2022.929052

Vijayabaskaran, S., and Cheng, S. (2022). Navigation task and action space drive the emergence of egocentric and allocentric spatial representations. *PLoS Comput. Biol.* 18:e1010320. doi: 10.1371/journal.pcbi.1010320

Walther, T., Diekmann, N., Vijayabaskaran, S., Donoso, J. R., Manahan-Vaughan, D., Wiskott, L., et al. (2021). Context-dependent extinction learning emerging from raw sensory inputs: a reinforcement learning approach. *Sci. Rep.* 11:2713. doi: 10.1038/s41598-021-81157-z

Wang, J. X., Kurth-Nelson, Z., Kumaran, D., Tirumala, D., Soyer, H., Leibo, J. Z., et al. (2018). Prefrontal cortex as a meta-reinforcement learning system. *Nat. Neurosci.* 21, 860–868. doi: 10.1038/s41593-018-0147-8

Watkins, D. W., and Berkley, M. A. (1974). The orientation selectivity of single neurons in cat striate cortex. *Exp. Brain Res.* 19, 433–446. doi: 10.1007/BF00234465

Wiltgen, B. J., Sanders, M. J., Anagnostaras, S. G., Sage, J. R., and Fanselow, M. S. (2006). Context fear learning in the absence of the hippocampus. *J. Neurosci.* 26, 5484–5491. doi: 10.1523/JNEUROSCI.2685-05.2006

Zeng, X., Wiskott, L., and Cheng, S. (2022). The functional role of episodic memory in spatial learning. *bioRxiv [Preprint]*. doi: 10.1101/2021.11.24.469830

Zhang, R., Zhang, S., Tong, M. H., Cui, Y., Rothkopf, C. A., Ballard, D. H., et al. (2018). Modeling sensory-motor decisions in natural behavior. *PLoS Comput. Biol.* 14:e1006518. doi: 10.1371/journal.pcbi.1006518

Zheng, L., Yang, J., Cai, H., Zhou, M., Zhang, W., Wang, J., et al. (2018). "MAgent: A many-agent reinforcement learning platform for artificial collective intelligence," in *Proceedings of the AAAI Conference on Artificial Intelligence* (New Orleans, LA: AAAI). doi: 10.1609/aaai.v32i1.11371