Applied Intelligence (2023) 53:2967–2988
https://doi.org/10.1007/s10489-022-03550-z

Towards designing a generic and comprehensive deep reinforcement learning framework

Ngoc Duy Nguyen[1] · Thanh Thi Nguyen[2] · Nhat Truong Pham[3,4] · Hai Nguyen[5] · Dang Tu Nguyen[6] · Thanh Dang Nguyen[7] · Chee Peng Lim[1] · Michael Johnstone[1] · Asim Bhatti[1] · Douglas Creighton[1] · Saeid Nahavandi[1]

Accepted: 21 March 2022 / Published online: 19 May 2022

Abstract

Reinforcement learning (RL) has emerged as an effective approach for building an intelligent system, which involves multiple self-operated agents to collectively accomplish a designated task. More importantly, there has been a renewed focus on RL since the introduction of deep learning that essentially makes RL feasible to operate in high-dimensional environments. However, there are many diversified research directions in the current literature, such as multi-agent and multi-objective learning, and human-machine interactions. Therefore, in this paper, we propose a comprehensive software architecture that not only plays a vital role in designing a connect-the-dots deep RL architecture but also provides a guideline to develop a realistic RL application in a short time span. By inheriting the proposed architecture, software managers can foresee any challenges when designing a deep RL-based system. As a result, they can expedite the design process and actively control every stage of software development, which is especially critical in agile development environments. For this reason, we design a deep RL-based framework that strictly ensures flexibility, robustness, and scalability. To enforce generalization, the proposed architecture also does not depend on a specific RL algorithm, a network configuration, the number of agents, or the type of agents.

Keywords Deep learning · Human-machine interactions · Learning systems · Multi-agent systems · Reinforcement learning · Software architecture

1 Introduction

Reinforcement learning (RL) has attracted a great deal of research attention owing to its learning procedure that allows agents to directly interact with the environment. As a result, an RL agent can imitate human learning process to achieve a designated goal, *i.e.*, an agent can carry out trial-and-error learning (exploration) and draw on "experience" (exploitation) to improve its behaviours [1, 2]. Therefore, RL is used in a variety of domains, such as IT resources management [3], cyber-security [4], robotics [5–8], control systems [9–12], recommendation systems [13], stock trading strategies [14], bidding and advertising campaigns [15], and video games [16–18]. However, traditional RL methods and dynamic programming [19], which use a *bootstrapping* mechanism to approximate the objective function, cease to work in high-dimensional environments due to limitation in memory and computational load requirements. This *"curse of dimensionality"* issue creates a major challenge in RL principle.

Figure 1 depicts an RL problem by using a *Unified Modeling Language* (UML) [20] sequential diagram. Specifically, the problem includes two entities: a decision maker and an environment. The environment can be an artificial simulator or a wrapper of a real-world environment. While the environment is a *passive* entity, the decision maker is an *active* entity that periodically interacts with the environment. In the RL context, a decision maker and an agent are interchangeable, though they can be two identified objects from a software design perspective.

At first, the decision maker perceives a state s_t from the environment. Then it uses its internal model to select the corresponding action a_t. The environment interacts with the chosen action a_t by sending a numerical reward r_{t+1} to the

✉ Ngoc Duy Nguyen
n.nguyen@deakin.edu.au

Extended author information available on the last page of the article.

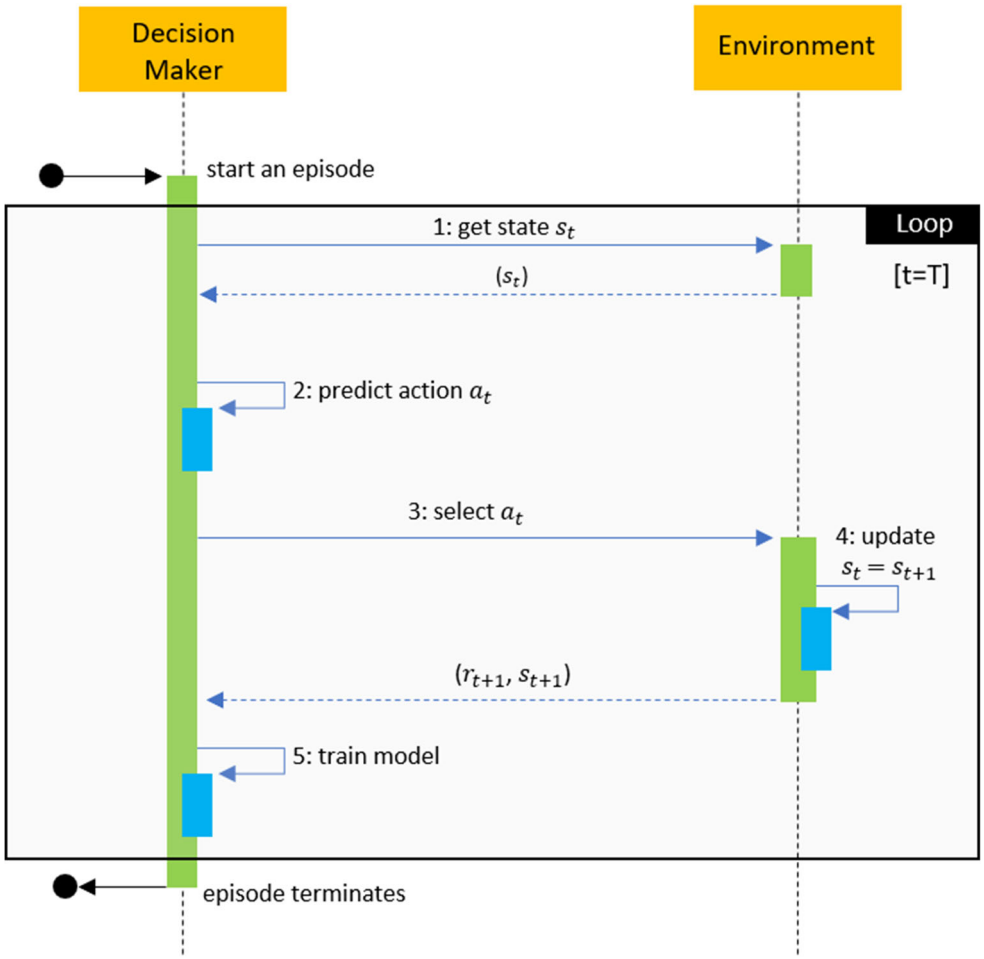

Fig. 1 A UML sequential diagram to describe an RL problem

decision maker. The environment also brings the decision maker to a new state s_{t+1}. Finally, the decision maker uses the current transition $\vartheta = \{a_t, s_t, r_{t+1}, s_{t+1}\}$ to update its decision model. This process is iterated until t equals T, where s_T denotes the terminal state of an episode. There are different methods to develop a decision model, such as fuzzy logic [21], genetic algorithms [22], or dynamic programming [23]. In this paper, however, we consider a deep neural network as the decision model.

The diagram in Fig. 1 infers that RL is capable of *online* learning because the model is updated continuously with incoming data. However, RL can be performed offline via a *batch learning* [24] technique. In particular, the current transition ϑ can be stored in an *experience replay* [25] and retrieved later to train the decision model. The goal of an RL problem is to maximize the expected sum of discounted reward R_t, *i.e.*,

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... + \gamma^{T-t-1} r_T,$$

where γ denotes the discounted factor and $0 < \gamma \leq 1$.

In 2015, Google DeepMind [26] announced a breakthrough in RL by combining it with deep learning to create an intelligent agent that can beat a professional human player in a series of 49 Atari games. The idea was to use a deep neural network with convolutional layers [27] to directly process raw images (states) of the game screen to estimate subsequent actions. The study was highly valued because it opened a new era of RL with deep learning, leading to partially solving the curse of dimensionality. In other words, deep learning offers a great complement for RL in a wide range of complicated applications. For instance, Google DeepMind created a program, AlphaGo, which beat the Go grandmaster, Lee Sedol, in the best-of-five tournament in 2016 [28]. AlphaGo is a full-of-tech AI based on Monte Carlo Tree Search [29], a hybrid network (policy network and value network), and a self-taught training strategy [30]. Other applications of deep RL can be found in self-driving cars [31, 32], helicopters [33], or even NP-hard problems such as *Vehicle Routing Problem* [34] and combinatorial graph optimization [35].

As stated above, deep RL is crucial owing to its appealing learning mechanism and widespread applications in the real world. In this study, we further delve into practical aspects of deep RL by analyzing challenges and solutions while

designing a deep RL-based system. Furthermore, we consider a real-world scenario where multiple agents, multiple objectives, and human-machine interactions are involved. Firstly, if we can take advantage of using multiple agents to accomplish a designated task, we can shorten the *wall time*, *i.e.*, the computational time to execute the assigned task. Depending on the task, the agents can be *cooperative* or *competitive*. In the cooperative mode, agents work in *parallel* or in *pipeline* to achieve the task [36]. In the case of competition, agents are scrambled, which basically raises the *resource hunting* problem [37]. However, in contrast to our imagination, competitive learning can be fruitful. Specifically, the agent is trained continually to place the opponent into a disadvantaged position and the agent is improved over time. As the opponent is also improved over time, this phenomenon eventually results in a *Nash equilibrium* [38]. Moreover, competitive learning promotes the self-taught strategy (*e.g.* AlphaGo) and a series of techniques such as the *Actor-Critic architecture* [39], opponent modeling [40], and *Generative Adversarial Networks* [41]. In this respect, we notice the problem of *moving target* [42] in multi-agent systems, which describes a scenario when the decision of an agent depends on other agents, thus the optimal policy becomes *non-stationary* [43].

Secondly, a real-world objective is often complicated as it normally comprises multiple sub-goals. It is straightforward if sub-goals are *non-conflicting* because they can be seen as a single composite objective. A more challenging case is when there are *conflicting objectives*. One solution is to convert a multi-objective problem into a single objective counterpart by applying *scalarization* through an application of a linear weighted sum for individual objective [44] or non-linear methods [45]. These approaches are categorized as *single-policy methods*. In contrast, *multi-policy methods* [46] seek multiple optimal policies at the same time. While the number of multi-policy methods is restricted, they are able to offer powerful solutions. For instance, the *Convex Hull Value Iteration* algorithm [47] computes a set of objective combinations to retrieve all deterministic optimal policies. To benchmark a multi-objective method, we can find or approximate a boundary surface, namely *Pareto dominance*, which presents the maximum performance of different weights (if scalarization is used) [48]. Recent studies have integrated multi-objective mechanisms into deep RL models [49, 50].

On the other hand, human-machine interaction is another key factor in designing a usable and useful deep RL-based system. A self-driving car, for example, should accept human intervention in emergency situations [31, 32]. Therefore, it is critical to ensure a certain level of safety while designing a hybrid system in which humans and machines can work together. Due to its importance, Google DeepMind and OpenAI have presented novel ways to encourage various innovations in recent years [51]. For instance, Christiano *et al.* [52] proposed a novel scheme that accepts human feedback during the training process. However, the method requires an operator to constantly observe the agent's behavior, which is an onerous and error-prone task. Recent work [53] provided a practical approach by introducing a behavioral control system. The system is used to control multiple agents in real time via human dictation. Table 1 summarizes key terminologies that are widely used in RL contexts (Table 1).

In summary, our study contributes to the following aspects:

- We present an overall picture of contemporary deep RL (Table 2). We consider state-of-the-art deep RL methods in three key aspects of pertaining to real-world applications such as multi-agent learning, multi-objective problems, and human-machine interaction. Thereafter, the paper offers a checklist for software managers, a guideline for software designers, and a technical reference for software programmers.
- We analyse challenges and difficulties in designing a deep RL-based system, contributing towards minimising possible mistakes during its development process. In other words, software designers can inherit the proposed design, foresee difficulties, and eventually expedite the entire development procedure of an RL-based system, especially in agile software development.
- We provide the source codes of our proposed framework in [54] (Table 3). Based on the template, RL beginners can design an RL method and implement it in real-world applications in a short time span. As a result, our work contributes toward promoting the use of deep RL to a wider community. A direct usecase of our study is to employ an educational framework that is used to demonstrate basic RL algorithms.

The paper has the following sections. Section 2 presents a survey on state-of-the-art deep RL methods in different research directions. Section 3 describes the proposed system architecture, which supports multiple agents, multiple objectives, and human-machine interactions. Concluding remarks are given in Section 4.

2 Literature review

2.1 Single-agent methods

The first advent of deep RL, namely a *Deep Q-Network* (DQN) [26], basically used a deep neural network to estimate values

Table 1 Key RL terminologies

Term	Description	Pros	Cons
Model-free RL	The environment is a black box. Agents mostly conduct a trial-and-error procedure to learn on its own. They use rewards to update their decision models	The algorithm does not need a model of the environment	Requires a large number of samples
Model-based RL	Agents construct a model that simulates the environment and use it to generate future episodes. By using the model, agents can estimate not only actions but also future states	Speed up learning and improve sample efficiency	Having an accurate and useful model is often challenging
Temporal difference learning	Use TD error to estimate the value function. For example, in Q-learning, $Q(s_t, a_t) = Q(s_t, a_t) + \beta(r_t + \gamma max_a Q(s_{t+1}, a))$	Fast convergence as it does not need to wait until the episode ends	Estimates can be biased
Monte-Carlo method	Estimate the value function by obtaining the average of the same values in different episodes $Q(s_t, a_t) = \lim_{N\to\infty} \sum_{i=1}^{N} Q(s_t^i, a_t^i))$	The values are non-biased estimates	Slow convergence and the estimates have high variances. Has to wait until episode ends to do updates
Continuous action space	The number of control actions is continuous	A policy-based method can be used	Cannot use a value-based method
Discrete action space	The number of control actions is discrete and finite	Both policy-based method and value-based method can be used	Intractable if the number of actions is large
Deterministic policy	The policy maps each state to a specific action	Reduce data sampling	Vulnerable to noise and stochastic environments
Stochastic policy	The policy maps each state to a probability distribution over actions	Better exploration	Requires a large amount of samples
On-policy method	Improve the current policy that the agent is using to make decisions	Safer to explore	May be stuck in local minimum solutions
Off-policy method	Learn the optimal policy (while samples are generated by the behavior policy)	Instability. Often used with an experience replay	Might be unsafe because the agent is free to explore
Fully observable environment	All agents can observe the complete states of the environment	Easier to solve than partially observable environments	The number of state can be large
Partially observable environment	Each agent only observes a limited observation of the environment	More practical in real-world applications	More difficult to solve as the agents require to remember past states

Table 2 Key deep RL methods in literature

Method	Description and Advantage	Technical Requirements	Drawbacks	Ref.
Value-based method				
DQN	Use a deep convolutional network to directly process raw graphical data and approximate the action-value function	• Experience replay • Target network • Q-learning	○ Excessive memory usage ○ Learning instability ○ Only for discrete action space	[55] [58] [56] [57]
Double DQN	Mitigate the maximization bias problem by using two separate networks: one for estimating value, one for selecting action.	• Double Q-learning	○ Inherit DQN's drawbacks	[58]
Prioritized Experience Replay	Prioritize important transitions so that they are sampled more frequently. Improve sample efficiency	• Importance sampling	○ Inherit DQN's drawbacks ○ Slower than non-prioritized experience replay (speed)	[55] [58]
Dueling Network	Separate the DQN architecture into two streams: one estimates state-value function and one estimates the advantage of each action	• Dueling network architecture • Prioritized replay	○ Inherit DQN's drawbacks	[58]
Recurrent DQN	Integrate recurrency into DQN. Extend the use of DQN in partially observable environments	• Long Short Term Memory	○ Inherit DQN's drawbacks	[58]
Attention Recurrent DQN	Highlight important regions of the environment during the training process	• Attention mechanism • Soft attention • Hard attention	○ Inherit DQN's drawbacks	[59]
Rainbow	Combine different techniques in DQN variants to provide the state-of-the-art performance on Atari domain	• Double Q-learning • Prioritized replay • Dueling network • Multi-step learning • Distributional RL • Noisy Net	○ Inherit DQN's drawbacks	[55]
Policy-based method				
A3C/A2C	Use actor-critic architecture to estimate directly the agent policy. A3C enables concurrent learning by allowing multiple learners to operate at the same time	• Multi-step learning • Actor-critic model • Advantage function • Multi-threading	○ Policy updates exhibit high variance	[56] [57] [58]

Table 2 (continued)

Method	Description and Advantage	Technical Requirements	Drawbacks	Ref.
UNREAL	Use A3C and multiple unsupervised reward signals to improve learning efficiency in complicated environments	• Unsupervised reward signals	○ Policy updates exhibit high variance	[60]
DDPG	Concurrently learn a deterministic policy and a Q-function in DQN's fashion	• Deterministic policy gradient	○ Support only continuous action space	[57] [58]
TRPO	Limit policy update variance by using the conjugate gradient to estimate the natural gradient policy. TRPO is better than DDPG in terms of sample efficiency	• Kullback-Leibler divergence • Conjugate gradient • Natural policy gradient	○ Computationally expensive ○ Large batch of rollouts ○ Hard to implement	[57] [58]
ACKTR	Inherit the A2C method Use Kronecker-Factored approximation to reduce computational complexity of TRPO ACKTR outperforms TRPO and A2C	• Kronecker-factored approximate curvature	○ Still complex	[57]
ACER	Integrate an experience replay into A3C Introduce a light-weight version of TRPO ACER outperforms TRPO and A3C	• Importance weight truncation & bias correction • Efficient TRPO	○ Excessive memory usage ○ Still complex	[57] [58]
PPO	Simplify the implementation of TRPO by using a surrogate objective function	• Clipped objective • Adaptive KL penalty coefficient	○ Require network tuning	[56] [57] [58]
TD3	Fixing the overestimation issue of DDPG by introducing clipped double-Q learning, policy update delays, and target policy smoothing	• Clipped objective • Deterministic policy gradient	○ Potential exploration issue	[61]
SAC	Optimize a stochastic policy using the maximum entropy framework	• Clipped objective • Stochastic policy optimization	○ Potential exploration issue	[62] [63]

Table 3 Demonstration codes of different use cases [54]

Use case	Description	Source Code
1. How to inherit an existing learner?	Develop a Monte-Carlo learner that inherits the existing Q-Learning learner	fruit/learners/mc.py
2. Develop a new environment	Create a Grid World [1] environment that follows the framework's interface	fruit/envs/games/grid_world/
3. Develop a multi-agent environment with human-agent interaction	Create a Tank Battle [53] game in which humans and AI agents can play together	fruit/envs/games/tank_battle/
4. Multi-objective environment and multi-objective RL	Use a multi-objective learner (MO Q-Learning) to train an agent to play Mountain Car [46]	fruit/samples/basic/multi_objectives_test.py
5. Multi-agent learner with human-agent interaction	Create a multi-agent RL method based on A3C [91] and apply it to Tank Battle	fruit/samples/tutorials/chapter_6.py
6. How to use a plugin?	Develop a TensorForce plugin, extract the PPO learner, and train an agent to play Cart Pole [1]	fruit/plugins/quick_start.py

of state-action pairs via a *Q-value function* (*a.k.a.*, *action-value function* or $Q(s, a)$). Thereafter, a number of variants based on DQN were introduced to improve the original algorithm. Typical extensions are *Double DQN* [64], *Dueling Network* [65], *Prioritized Experience Replay* [66], *Recurrent DQN* [67], *Attention Recurrent DQN* [59], and an ensemble method named *Rainbow* [68]. These methods use an experience replay to store historical transitions and retrieve them in batches to train the resulting network. Moreover, a separate *target network* can be used to mitigate the correlation of sequential data and prevent training network from overfitting.

Instead of estimating the action-value function, we can directly approximate the agent's policy $\pi(s)$. This approach is known as *policy gradient* or *policy-based* methods. *Asynchronous Advantage Actor-Critic* (A3C) [69] is one of the first policy-based deep RL methods in the literature. A3C comprises two networks: an actor network to estimate the agent policy $\pi(s)$ and a critic network to estimate the *state-value function* $V(s)$. Additionally, to stabilize the learning process, A3C uses an *advantage function*, *i.e.*, $A(s, a) = Q(s, a) - V(s)$. There is a synchronous version of A3C, namely A2C [69], which has the advantage of being simpler but with comparable or better performance. A2C mitigates the risk of multiple learners from overlapping when updating the weights of the global networks.

There have been a great number of policy gradient methods since the development of A3C. For instance, *UNsupervised REinforcement and Auxiliary Learning* (UNREAL) [70] used multiple unsupervised pseudo-reward signals simultaneously to improve the learning efficiency in complicated environments. Rather than estimating a stochastic policy, *Deterministic Policy Gradient* [71] (DPG) seeks for a deterministic policy, which significantly reduce data sampling. Moreover, *Deep Deterministic Policy Gradient* [72] (DDPG) combined DPG with DQN to enable learning of a deterministic policy in a continuous action space using an actor-critic architecture. To further stabilize the training process, a *Trust Region Policy Optimization* (TRPO) method [73] integrated *Kullback–Leibler divergence* [74] into the training procedure, leading to a complicated method. In 2017, Wu *et al.* [75] proposed *Actor-Critic using Kronecker-Factored Trust Region* (ACKTR), which applied Kronecker-factored approximation curvature into gradient update steps. Additionally, *Actor-Critic with Experience Replay* (ACER) [76] was introduced to offer an efficient off-policy sampling method based on A3C and an experience replay. To simplify the implementation of TRPO, ACKTR, and ACER, *Proximal Policy Optimization* (PPO) [77] is proposed to exploit a clipped "surrogate" objective function together with stochastic gradient ascent. Soft Actor-Critic (SAC) [62, 63] uses the maximum entropy reinforcement learning framework that aims to learn a stochastic policy that maximizes both the reward and the entropy. In other words, SAC learns an actor that succeeds at the task but is as random as possible. Concurrently, Twin-Delayed DDPG (TD3) [61] focuses on fixing the issue of value overestimation of DDPG by introducing three tricks: clipped double-Q learning, policy update delays, and target policy smoothing. Both SAC and TD3 are quite comparable, and they are currently considered state-of-the-art methods on continuous control domains. Some studies combined policy-based and value-based methods such as [78–80] or on-policy and off-policy methods such as [81, 82]. Table 2 presents a summary of key deep RL methods and their implementation. Based on specific application domains, software managers can select a suitable deep RL method to act as a baseline for the target system.

Recently, many studies have focused on efficient training and state-of-the-art performance in various tasks and settings

in the field of RL. For instance, D4PG (*Distributed Distributional Deep Deterministic Policy Gradients*) [83] was proposed to improve the performance of RL in a wide array of control tasks, such as robotics control with a finite number of discrete actions. To efficiently train an agent in a scalable RL system, Espeholt *et al.* introduced a fast and low-cost RL algorithm, namely SEED RL (*Scalable and Efficient Deep RL*), which could train numerous frames per second [84]. Moreover, a self-predictive representation learning algorithm [85] was proposed for RL to exploit data augmentation and future prediction objective. Using the algorithm, agents could learn future latent state representations with limited interactions.

2.2 Multi-agent methods

In multi-agent learning, there are two widely used schemes in the literature: *individual* and *mutual*. In the individual scheme category, each agent in the system can be considered as an independent decision maker and other agents as part of the environment. In this way, any deep RL methods in the previous subsection can be used in multi-agent learning. For instance, Tampuu *et al.* [86] used DQN to create an independent policy for each agent. Behavioural convergence of the involved agents was analysed with respect to cooperation and competition. Similarly, Leibo *et al.* [37] introduced a sequential social dilemma, which basically used DQN to analyze the agent's strategy in Markov games such as Prisoner's Dilemma, Fruit Gathering, and Wolfpack. However, the approach limits the number of agents due to computational complexity with the number of policies. To overcome this obstacle, Nguyen *et al.* developed a behavioural control system [53] for homogeneous agents to share the same policy. As a result, the method is robust and scalable. Another problem in multi-agent learning is the use of an experience replay, which amplifies the non-stationary problem due to asynchronous data sampling of different agents [43]. A lenient approach [42] can subdue the problem by mapping transitions into decaying temperature values, which basically controls the magnitude of updating different policies.

In the mutual scheme category, agents can "speak" with each other via a settled communication channel. While agents are often trained in a centralized manner, they eventually operate in a decentralized fashion [87]. In other words, a multi-agent RL problem can be divided into two sub-problems: a goal-directed problem and a communication problem. For instance, *Multi-agent DDPG* (MADDPG) [88] was proposed to employ DDPG in a multi-agent environment. Specifically, Foerster *et al.* [89] introduced two communication schemes based on the centralized-decentralized rationale: *Reinforced Inter-Agent Learning* (RIAL) and *Differentiable Inter-Agent Learning* (DIAL). While RIAL reinforces agent learning by sharing parameters, DIAL allows *inter-communication* between agents via a shared medium. Both methods, however, operate with a discrete number of communication actions. As opposed to RIAL and DIAL, *Communication Neural Net* (CommNet) [90] enabled communication by using a continuous vector. As a result, agents are trained to communicate by backpropagation. However, CommNet limits the number of agents due to computational complexity. To make it scalable, Gupta *et al.* [91] introduced a parameter sharing method to handle a large number of agents. However, the method only worked with homogeneous systems. Nguyen *et al.* [53] extended the study in [91] to heterogeneous systems by designing a behavioral control system. For further reading, comprehensive reviews on multi-agent RL can be found in [92, 93].

In recent years, many algorithms have been introduced for multi-agent RL (MARL). Shu and Tian proposed M^3RL (*Mind-aware Multi-agent Management RL*) to implement an optimal collaboration within agents by training a "super" agent to manage member agents [94]. The super-agent was trained in both policy learning and agent modeling, i.e., by combining imitation learning and RL. As a result, the super-agent could assign members to perform suitable tasks and maximise the overall productivity while minimizing payments for rewarding them with bonuses. Yang *et al.* presented CM3 (*Cooperative Multi-goal Multi-stage MARL*) using a novel multi-stage curriculum to learn both individual goal attainment and collaboration in MARL systems [95]. In CM3, an augmentation function was used to bridge value function and policy function across the multi-stage curriculum. Another approach, *Evolutionary Population Curriculum* (EPC) was proposed by Long *et al.* [96], which learned well-balanced policies in large-scale MARL systems. Additionally, in many real-world MARL systems, communication between agents is required to make sequential decisions in fully collaborative multi-agent tasks. Kim *et al.* presented a *SchedNet* for MARL systems to schedule inter-agent communication when the communication bandwidth is limited and the medium is shared among agents [97]. In cooperative MARL systems, common knowledge between the agents is critical to coordinate agent behaviours. Therefore, Schroeder de Witt *et al.* proposed MACKRL (*Multi-Agent Common Knowledge RL*) to learn a hierarchical policy tree [98]. MACKRL helped multiple agents to learn a decentralized policy by exploring and exploiting commonly available knowledge. However, exploration in MARL is a challenging problem. Christianos *et al.* proposed SEAC (*Shared Experience Actor-Critic*) for MARL to combine the gradient information of agents to share experience among agents in an actor-critic architecture [99]. Recently, to factorize the joint value function and overcome the limitation of value-based

MARL in terms of scalability, a *duplex dueling multi-agent Q-learning*, namely QPLEX [100] was proposed to fully support centralized training and decentralized execution in MARL systems.

In summary, it is critical to address the following factors in multi-agent learning as they significantly impact on the target software architecture:

- It is preferable to employ a centralized-decentralized rationale in a multi-agent RL-based system because the training process is time-consuming and computationally expensive. A working system requires hundreds to thousands of training sessions by searching through the hyper-parameter space to find an optimal solution.
- communication between agents can be *realistic* or *imaginary*. In realistic communication, agents "speak" with each other using an established communication protocol. However, there is no actual channel in imaginary communication. Agents are trained to collaborate using a specialized network architecture. For instance, OpenAI [88] proposes an actor-critic architecture where the critic is augmented with other agents' policy information. As a result, both methods can differentiate how to design an RL-based system.
- A partially observable environment has a great impact on designing a multi-agent system because each agent has its own unique perspective of the environment. Therefore, it is important to first carefully examine the environment and application type to avoid any malfunction in the design.

2.3 Meta-RL methods

Meta reinforcement learning or Meta-RL employs the principle of meta-learning in RL. The major difference from traditional RL is that the last reward r_{t-1} and the last action a_{t-1} are incorporated into the policy observation. The key components of Meta-RL consist of a model with memory (*e.g.*, RNN (*Recurrent Neural Networks*), LSTM (*Long short-term memory*)), meta-learning algorithm, and distribution of MDPs. Meta-RL aims to aid agents to adapt to new tasks by using a small amount of experience.

Wang *et al.* [101] presented a novel approach named Deep Meta-RL based on meta-learning with an RNN. In Deep Meta-RL, agents learn new tasks rapidly by acquiring the knowledge from previous experience. Additionally, learning from spare and under specified rewards is a challenging problem in RL, *e.g.*, when an agent is required to observe a complex state and provide sequential actions simultaneously. To overcome this problem, a meta reward learning algorithm leverage meta-learning and Bayesian strategy, which eventually optimises an auxiliary reward function [102]. Although deep Meta-RL algorithms aid agents to learn new tasks rapidly by a small amount of experience, the lack of a mechanism to explain task uncertainty in sparse-reward problems remains unsolved. Off-policy Meta-RL algorithm using probabilistic context variables, named PEARL (*Probabilistic Embeddings for Actor-critic Reinforcement Learning*) [103] was designed to improve meta-training efficiency. Another challenge in Meta-RL is the chicken-and-egg problem, which occurs when agents are required to explore and exploit relevant information in an end-to-end training. Liu *et al.* proposed a *decoupling exploration and exploitation* in Meta-RL named DREAM to overcome this canonical problem and avoid local optima [104]. However, existing Meta-RL algorithms are compromised when the rewards are sparse. To solve this problem, a meta-exploration namely *Hyper-sate Exploration* (HyperX) [105] was introduced to approximate exploration strategies based on a Bayesian RL model.

In MARL, agents interact with each other by cooperating or competing to maximize their return and information gain from all agents. A framework named IBRL (*Interactive Bayesian RL*) was used to find adaptive policies under uncertain scenarios with prior belief. However, the framework was not intractable in most settings and restricted to light-weight tasks or simple agent models. Therefore, Zintgraf *et al.* proposed a meta-learning deep IBRL for MARL to overcome this limitation [106]. Recently, many exploratory deep RL algorithms have been proposed based on task-agnostic objectives. However, it is necessary to learn effective exploration from prior experience. Gupta *et al.* presented MAESN (*Model Agnostic Exploration with Structured Noise*) using gradient-based meta-learning and a learned latent exploration space [107]. Another limitation in Meta-RL is that most existing Meta-RL methods can be sensitive with respect to distribution shift of a testing task, leading to performance degradation. In this respect, a model-based adversarial Meta-RL method [108] was proposed to overcome this issue.

2.4 RL challenges

In this subsection, we discuss the major challenges in designing a deep RL-based system and the corresponding solutions. To remain concise, the proposed framework is not hampered by the existing limitations, as it is straightforward to extend our proposed architecture to support these rectifications.

Catastrophic forgetting is a problem in *continual learning* and *multi-task learning*. Consider the scenario where a network is trained to learn the first task. In this case, the neural network gradually forgets the knowledge of the first task to adopt the new one. One solution is to use

regularization [109, 110] or a dense neural network [111, 112]. However, these approaches are only feasible with a limited number of tasks. Recent studies introduce more scalable approaches such as *Elastic Weight Consolidation* (EWC) [113] or *PathNet* [114]. While EWC finds a network configuration to yield the best performance in learning different tasks, PathNet uses a "super" neural network to learn the knowledge of different tasks in different paths.

Policy distillation [115] or *transfer learning* [116, 117] can be used to train an agent to learn individual tasks and collectively transfer the knowledge to a single network. Transfer learning is often used when the actual experiment is expensive and intractable. In this case, the network is trained with simulations and is deployed later in the target experiment. However, a *negative transfer* may occur when the performance of the learner is lower than the trainer. In this respect, the concept of *Hierarchical Prioritized Experience Replay* [116] was introduced to use high-level features of a task and selects important data from the experience replay to mitigate negative transfer. One recent study [118] proposed the principle of mutual learning to achieve a comparable performance between actual experiment and simulations.

Another obstacle in RL is dealing with a *long-horizon* environment with *sparse rewards*. In such tasks, the agent hardly receives any reward, and it can easily be trapped in local minimum solutions. One solution is to use *reward shaping* [119] that continuously instructs the agent to achieve the objective. The problem can also be divided into a hierarchical tree of sub-problems where the parent problem has a higher abstraction than that of the child problem (*Hierarchical RL*) [120]. To encourage self-exploration, intrinsic reward signals can be introduced to reinforce the agent to make a generic decision [121]. State-of-the-art methods of *intrinsic motivation* can be found in [122–124]. Andrychowicz *et al.* [125] proposed *Hindsight Experience Replay* to implicitly simulate *curriculum learning* [126] by creating imagined trajectories in experience replay with positive rewards. In this way, an agent can learn from failures and can automatically generalize a solution in successful cases.

A variety of RL-related methods have been proposed to make RL feasible in large-scale applications. One approach is to augment the neural network with a "memory" to enhance sample efficiency in complicated environments [127, 128]. Additionally, to enforce scalability, many distributed methods can be employed, such as *Distributed Experience Replay* [129], deep RL acceleration [130], and distributed deep RL [131]. In addition, *imitation learning* can be used together with *inverse RL* to accelerate training by directly learning from expert demonstration and extracting the expert's cost function [132].

2.5 Deep RL framework

In this subsection, we discuss the latest deep RL frameworks in the literature. We select the libraries based on different factors including Python-based implementation, clear documentation, reliability, and active community. Based on our analysis, software managers can select a suitable framework depending on the project requirements.

- *Chainer* – Chainer [58] is a powerful and flexible framework of neural networks. The framework is currently supported by IBM, Intel, Microsoft, and Nvidia. It provides an easy way to manipulate neural networks such as creating a customized network, visualizing a computational graph, and supporting a debug mode. It also implements a variety of deep RL methods. However, the Chainer architecture is complicated, which requires a great effort to develop a new deep RL method. The number of integrated environments is also limited, *e.g.*, Atari [133], OpenAI Gym [134], and Mujoco [135].
- *Keras-RL* – Keras-RL [136] is a friendly deep RL library, which is recommended for deep RL beginners. However, the library provides a limited number of deep RL methods and environments.
- *TensorForce* – TensorForce [137] is an ambitious project that targets both industrial applications and academic research. The library has the best modular architecture we have reviewed so far. Therefore, it is convenient to use the framework to integrate customized environments, modify network configurations, and manipulate deep RL algorithms. However, the framework has a deep software stack ("pyramid" model) that includes many abstraction layers, as shown in Fig. 2. This hinders novice users in prototyping a new deep RL method.
- *OpenAI Baselines* – OpenAI Baselines [57] is a high-quality framework of contemporary deep RL. In contrast to TensorForce, the library is suitable for researchers who want to reproduce original results. However, OpenAI Baselines is unstructured and incohesive. Moreover, the codebase is no longer maintained by OpenAI.
- *Stable-Baselines* and *Stable-Baselines3* - Stable-Baselines [138] is an attempt to port Tensorflow-implemented RL algorithms in OpenAI Baselines in Pytorch. It gradually grows into a reliable source for baseline RL algorithms. The project is now actively maintained under a different repository - Stable-Baselines3 [139]. This new codebase's advantage is the ease of modifying existing algorithms with modularized codes. Moreover, it is actively maintained; therefore, the authors are very responsive in answering questions.

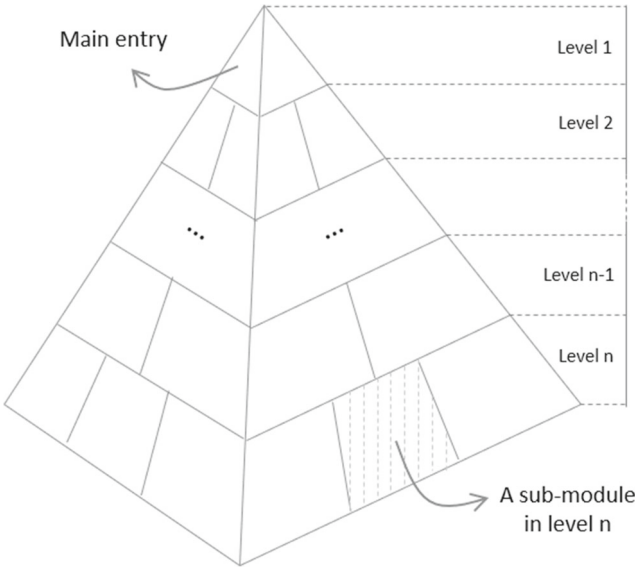

Fig. 2 A "pyramid" software architecture

- *RLLib* – RLLib [56] is a well-designed deep RL framework that allows deploying deep RL in distributed systems. The usage of RLLib is not friendly for RL beginners.
- *RLLab* – RLLab [140] provides diverse deep RL models including TRPO, DDPG, Cross-Entropy Method, and Evolutionary Strategy. While the library is friendly to use, it is not straightforward in terms of modifications.
- *PettingZoo* – PettingZoo [141] is a Python library, which supports a wide range of MARL environments and can be accessible for both university and non-expert researchers. However, it requires other languages to support games with more than 10,000 agents. Moreover, it does not support competitive games where different agents compete with each other.
- *MAgent* – MAgent [142] is a MARL platform that supports researchers to develop artificial collective intelligence at both individual agent and society levels. MAgent has limited algorithms and does not support continuous environments.
- *Acme* – A framework for distributed RL namely Acme is proposed by DeepMind [143]. Acme is a modular, lightweight tool that helps researchers to re-implement RL algorithms in both research and industrial environments. It also aids training of RL algorithms in both single-actor and distributed paradigms.
- *Megaverse* – Megaverse [144] is the first immersive 3D simulation framework for embodied agents and RL that supports multiple agents in immersive environments with more than 1,000,000 actions per second on a single 8-GPU node. The framework requires extensive computational resources to democratize deep RL research.
- *Tianshou* - Tianshou [145] is a modularized Pytorch codebase with friendly APIs. Besides supporting standard algorithms, this codebase supports memory-based agents needed to tackle partially observable MDPs (POMDPs).

In summary, most frameworks focus on the performance of deep RL methods. As a result, those frameworks limit code legibility, restricting RL users in terms of readability and modifications. In this paper, we propose a comprehensive framework that has the following properties:

- Allow new users, including novice developers, to prototype a deep RL method in a short period of time by following a modular design. As opposed to TensorForce, we limit the number of abstraction layers and avoid the pyramid structure.
- The framework is friendly with a simplified user interface. We provide an API based on three key concepts: policy network, network configuration, and learner.
- Scalability and generalization are realised in our framework, while supporting multiple agents, multiple objectives, and human-machine interactions.
- A concept of unification and transparency is introduced by creating plugins. Plugins are gateways that extract learners from other libraries and plug them into our proposed framework. In this way, users can interact with different frameworks using the same interface.

3 A prospective RL software architecture

In this section, we examine core components towards designing a comprehensive deep RL framework, which basically employs generality, flexibility, and interoperability. We aim to support a broad range of RL-related applications that involve multiple agents, multiple objectives, and human-agent interaction. We use the following pseudocode to describe a function signature:

- $\textbf{function\_name}([param1, param2, ...]) \rightarrow [return1, return2, ...]$ or $\{return1, return2, ...\}$ or A

where $\rightarrow$ denotes a return operation, A is a scalar value, [...] denotes an array, and {...} denotes a list of possible values of a single variable.

3.1 Environment

First, we create a unique interface for the *environment* to establish a communication channel between the framework

and agents. However, to reduce complexity, we put any human-related communication into the environment. As a result, human interaction assumes as a part of the environment and is hidden from the framework, *i.e.*, the environment provides two interfaces: one for the framework and one for human, as shown in Fig. 3. While the framework interface is often in programming level (functions), the human interface has a higher abstraction mostly in human understanding forms such as voice dictation, gesture recognition, or control system.

The environment framework interface provides the following functions:

- **clone**(): the environment can duplicate itself. The function is useful when an RL algorithm requires multiple learners simultaneously (*e.g.* A3C).
- **reset**(): reset the environment to its initial state. The function must be called after or before an episode.
- **step**($[a_1, a_2, ..., a_N]$) $\rightarrow$ $[r_1, r_2, ..., r_M]$: executes N specified actions of N agents in the environment. The function returns M rewards, each of which represents an objective function.
- **get_state**() $\rightarrow$ $[s_1, s_2, ..., s_N]$: retrieves the current states of the environment. If the environment is a partially observable MDP, the function returns N states, each presents the current state of an agent. However, if the environment is a fully observable MDP, we have $s_1 = s_2 = ... = s_N = s$.
- **is_terminal**() $\rightarrow$ {True, False}: checks whether an episode is terminated.
- **get_number_of_objectives**() $\rightarrow$ M: is a helper function that indicates the number of objectives in the environment.
- **get_number_of_agents**() $\rightarrow$ N: is a helper function that indicates the number of agents in the environment.

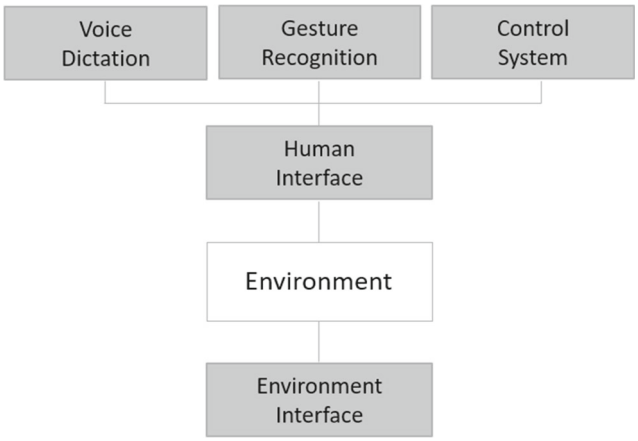

Fig. 3 A conceptual model of the environment with a human interface

In addition, it is important to consider the following questions during the design of an environment component, which has a significant impact on subsequent design stages:

- *Is it a simulator or a wrapper?* In the case of a wrapper, the environment is already developed and configured. Our task is then to develop a wrapper interface that can compatibly interact with the framework. In contrast to a wrapper, developing a simulator is complicated and requires expert knowledge. In real-time applications, we first develop a simulator in C/C++ (for better performance) and then create a Python wrapper interface (for ease of integration). In this case, we need to develop both simulator and wrapper.
- *Is it stochastic or deterministic?* A stochastic environment is more challenging to implement than a deterministic one. There are potential factors that contribute to randomness of the environment. Consider an example where a company intends to run a bicycle rental service, in which N bicycles are equally distributed into M potential locations. However, at a specific time, location A has many bicycles due to limited customers. As a result, bicycles in location A are delivered to other locations with higher demand. The company seeks development of an algorithm that can balance the number of bicycles in each place over time. This is a stochastic environment example. We can start building a simple stochastic model based on Poisson distribution to represent the bicycle demand in each place. We end up with a complicated model based on a set of observable factors such as rush hour, weekend, festival, etc. Depending on the stochasticity of the model, we can decide to use a model-based or model-free RL method.
- *Is it complete or incomplete?* A complete environment provides sufficient information at any time to construct a series of possible moves in the future (*e.g.* Chess or Go). Completeness in information helps the determination of an effective RL method, *e.g.*, a complete environment can be solved with a careful planning rather than a trial-and-error approach.
- *Is it fully observable or partially observable?* Observability of an environment is essential when designing a deep neural network. A partially observable environment requires recurrent layers or an attention mechanism to enhance the network capacity during training. As an example, a self-driving scenario is partially observable while a board game is fully observable.
- *Is it continuous or discrete?* As described in Table 1, this factor is important to determine the type of methods used, such as policy-based or value-based methods, as well as network configurations, such as actor-critic architectures.

- *How many objectives?* Real-world applications often have multiple objectives. If the importance weights between objectives can be identified initially, it is reasonable to use single-policy RL methods. Alternatively, a multi-policy RL method can prioritize the importance of an objective in real time.
- *How many agents?* A multi-agent RL-based system is more complicated than a single-agent counterpart. Therefore, it is essential to analyze the following factors of a multi-agent system before delving into the design: the number of agents, the type of agents, communication capabilities, cooperation strategies, and competitive potential.

3.2 Network

The *neural network* is a key module of our proposed framework, which includes a network configuration and a policy network, as illustrated in Fig. 4. A network configuration defines the deep neural network architecture (*e.g.* CNN (*Convolutional Neural Networks*) or LSTM), loss functions (*e.g.* Mean Square Error or Cross Entropy Loss), and optimization methods (*e.g.* Adam or SGD). Depending on the project's requirements, a configuration can be divided to different abstraction layers, where the lower abstraction layer is used as a mapping layer for the higher abstraction layer. In the lowest abstraction level (programming language), a configuration is implemented by a deep learning library, such as Pytorch [146] (with dynamic graph) or TensorFlow (with static graph). The next layer is to use a scripting language, such as *xml* or *json*, to describe the network configuration. This level is useful because it provides a faster and easier way to configure a network setting. For uses with limited knowledge in implementation details such as system analysts, an intuitive and easy-to-use graphical user interface is useful. However, there is a trade-off here: the higher abstraction layer achieves better usability and productivity but has a longer development cycle.

A policy network is a composite component with a number of network configurations. The dependency between a policy network and a configuration can be weak, *i.e.*, an *aggregation* relationship. The policy network objective is twofold. It provides a high-level interface that maintains connectivity with other modules in the framework, and it initializes the network, saves the network parameters into checkpoints, and restores the network parameters from checkpoints. The neural network interface provides the following functions:

- **create_network**() $\rightarrow$ $[\theta_1, \theta_2, ..., \theta_K]$: instantiates a deep neural network by using a set of network

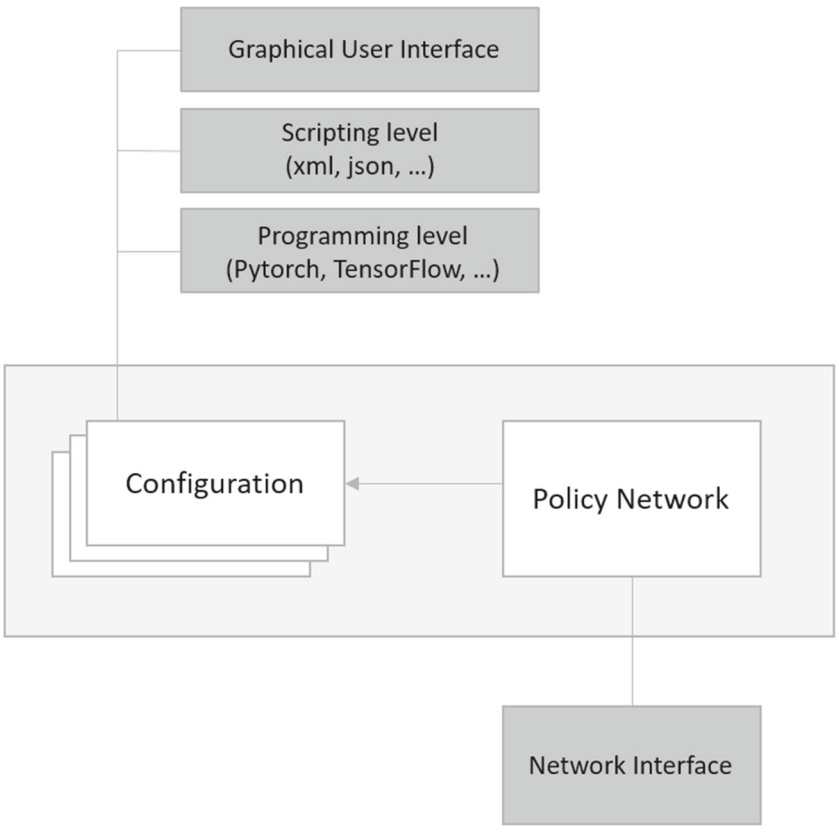

Fig. 4 A neural network module includes a network configuration and a policy network

configurations. The function returns the network parameters (references) $\theta_1, \theta_2, ..., \theta_K$.
- **save_model**(): saves the current network parameters into a checkpoint file.
- **load_model**([chk]): restores the current network parameters from a specified checkpoint file *chk*.
- **predict**($[s_1, s_2, ..., s_N]$) $\rightarrow$ $[a_1, a_2, ..., a_N]$: given the current states of N agents $s_1, s_2, ..., s_N$, the function uses the network to predict the next N actions $a_1, a_2, ..., a_N$.
- **train_network**([data_dict]): trains the network by using the given data dictionary. The data dictionary often includes the current states, current actions, next states, terminal flags, and miscellaneous information (global time step or objective weights) of N agents.

3.3 Learner

The last key module of our proposed framework is a *learner*, as shown in Fig. 5. While the environment module and the network module create the application shell, the learner acts as an engine that allows the system to operate properly. These three modules jointly create the backbone of the system. In particular, the learner uses the environment module to generate episodes. It manages the experience replay memory and defines the RL implementation details, such as multi-step learning, multi-threading, or reward shaping. The learner is often created together with a monitor. The monitor is used to manage multiple learners (if multi-threading is used) and collect any data from the learners during training, such as performance information for debugging purposes and post-evaluation reports. The learner collects necessary data, packs them into a dictionary before sending them to the network module for training.

Additionally, a *factory* pattern [147] can be used to hide the operation details between the monitor and the learner. As a result, the factory component promotes higher abstraction and usability through a simplified user API, as follows:

- **create_learner**([monitor_dict, learner_dict]) $\rightarrow$ obj: The factory creates a learner by using the monitor's data dictionary (such as batch size, the number of epochs, and the report frequency) and the learner's data dictionary (the number of threads, epsilon values, reward clipping thresholds, etc.).
- **train**(): trains the generated learner.
- **evaluate**(): evaluates the generated learner.

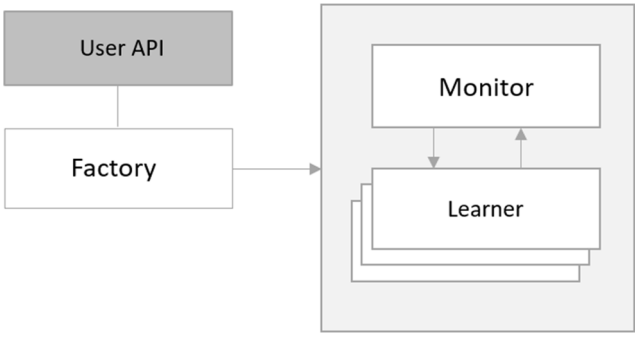

Fig. 5 A high-level design of a learner module

3.4 Plugin

Many RL methods are available in the literature, and it is impractical to implement all of them. However, we can reuse the implementation from existing libraries such as TensorForce, OpenAI Baselines, or RLLab. To ensure flexibility and interoperability, we introduce a concept of unification by using plugins. A plugin is a piece of program that extracts learners or network configurations from third party libraries and plugs them into our framework. As a result, the integrated framework provides a unique user API to support a variety of RL methods. In this way, users do not need to learn different libraries. The concept of unification is described in Fig. 6.

A plugin can also act as a conversion program that converts the environment interface of the library into the environment interface of another library. As a result, the proposed framework can work with any environment in third party libraries and vice versa. Therefore, a plugin should have the following functions:

- **convert_environment**([source_env]) $\rightarrow$ target_env: converts the environment interface from the source library to the environment interface defined in the target library.
- **extract_learner**([param_dict]) $\rightarrow$ learner: extracts the learner from the target library.
- **extract_configuration**([param_dict]) $\rightarrow$ config: extracts the network configuration from the target library.

3.5 Overall structure

Assembling everything together, we have a sequential diagram of the training process, as described in Fig. 7. The workflow divides the training process into smaller procedures. Firstly, the factory instantiates a specified learner (or a plugin) and sends its reference to the monitor. The monitor clones the learner into multiple learner threads. Each learner thread is executed until the number of epochs exceeds a predefined threshold, K. The second loop within the learner thread is used to generate episodes. In each episode, a learner thread perceives the current states of the environment and predicts the next actions using the policy network and configuration network. The next actions are applied to the environment. The environment returns the next states and a terminal flag. The policy network

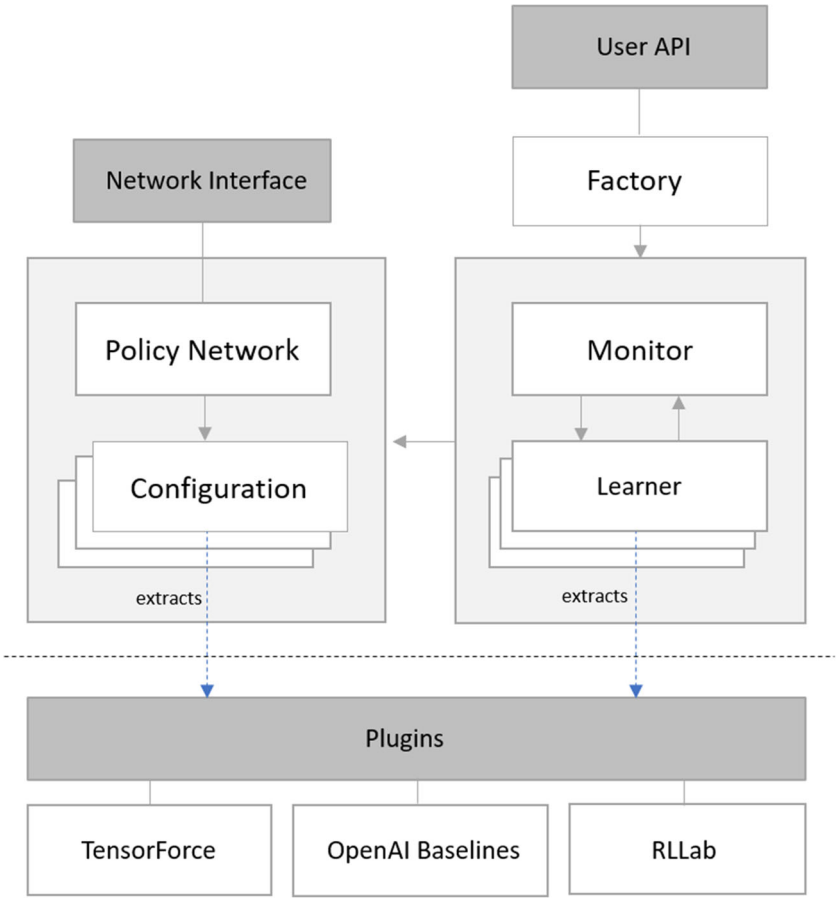

Fig. 6 A unification of different RL libraries by using plugins

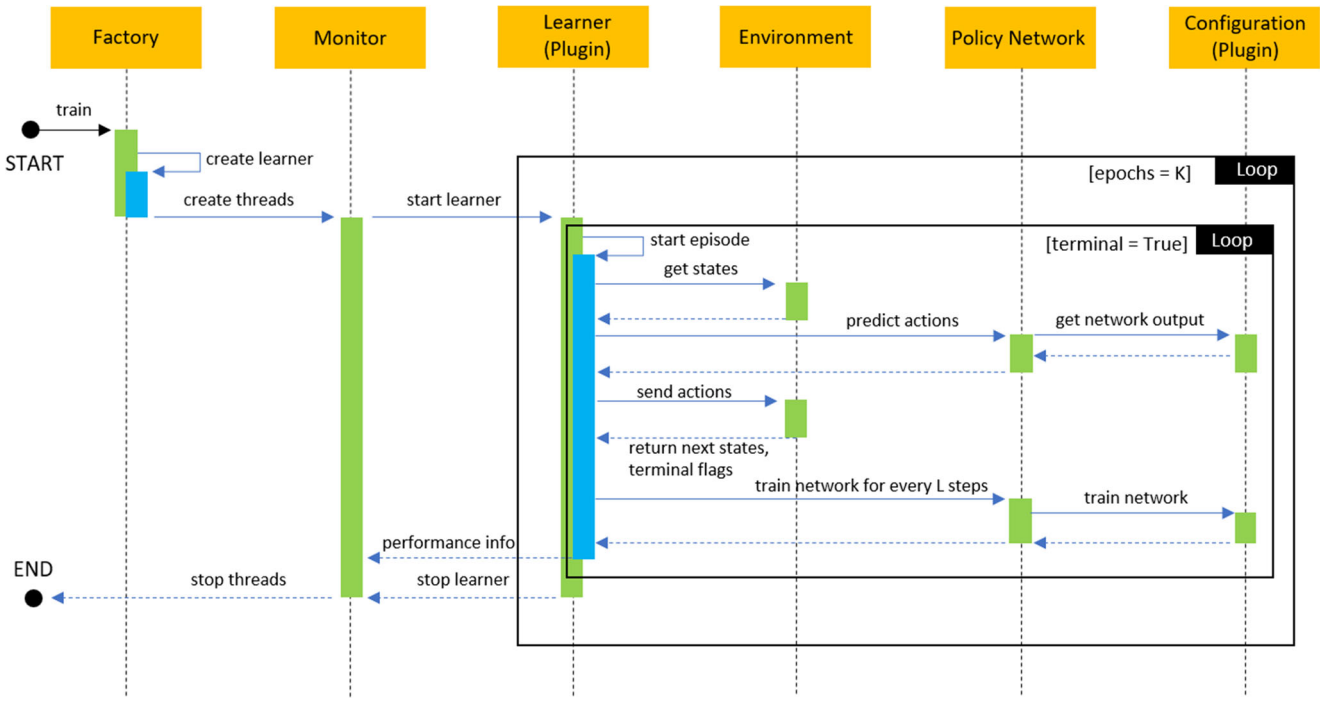

Fig. 7 A UML sequential diagram of the training process

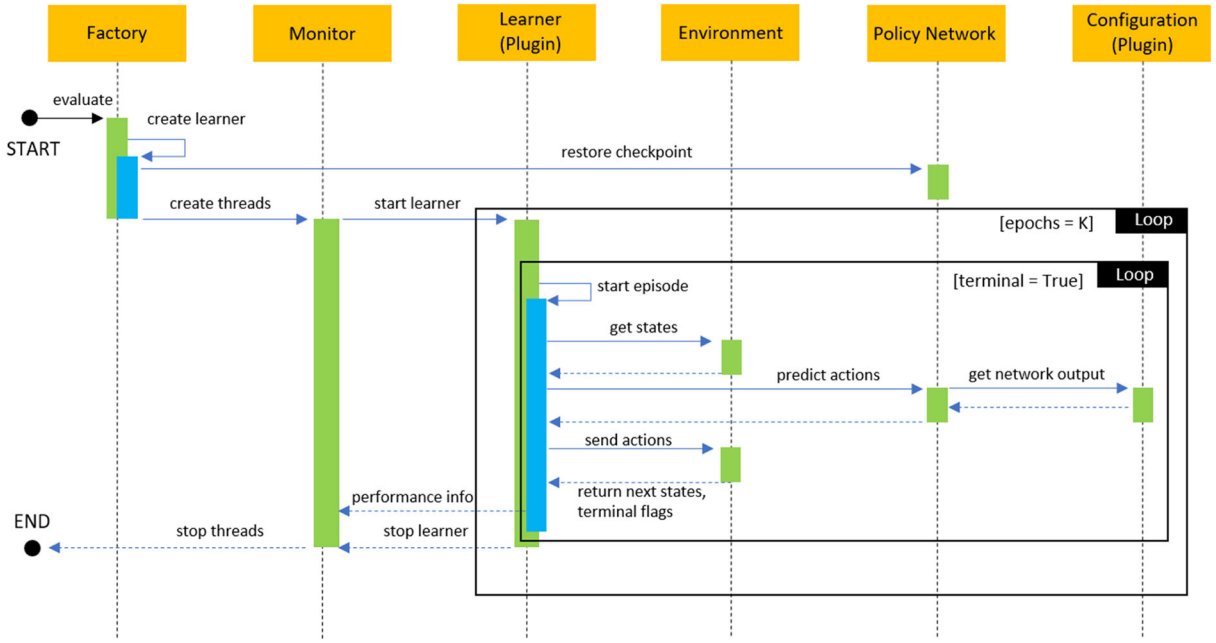

Fig. 8 A UML sequential diagram of the evaluation process

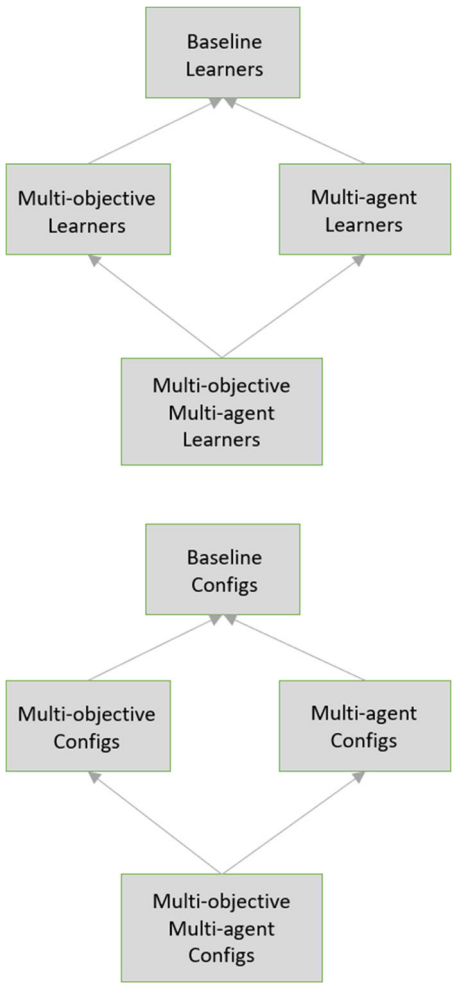

Fig. 9 An inheritance relationship between learners and configurations

is trained for every L-step. There are minor changes in the evaluation process, as shown in Fig. 8. Firstly, the policy network's parameters are restored from a specified checkpoint file when initializing the learner. Secondly, all training procedure calls are discarded when generating the episodes.

To enhance usability and reduce redundancy, it is advisable to implement the framework in *Object-Oriented Programming* (OOP). In this way, a new learner (configuration) can be easily developed by inheriting existing learners (configurations) in the framework, as shown in Fig. 9.

4 Conclusions

In this paper, we have presented a review on recent advances in the RL literature with respect to multi-agent learning, multi-objective learning, and human-machine interaction. We have also examined different deep RL libraries and analysed their limitations. Importantly, we have proposed a novel deep RL framework that offers usability, flexibility, and interoperability for developing RL-based systems. We have highlighted the key concerns so that software managers can avoid possible mistakes in designing an RL-based application.

The proposed framework served as a generic *template* to design and implement real-world RL-based applications. Because the framework is developed in OOP, it is beneficial to utilize OOP principles, such as inheritance, polymorphism, and encapsulation to expedite the development process. We have created a flexible software layer stack, where

the number of modules is minimal while maintaining a certain level of cohesion. As a result, the learning curve is not steep. By providing a simplified API, the framework is suitable for novice developers who are new to designing deep RL models, especially software engineers. The proposed framework acts as a bridge to connect different RL communities. Future developments of the framework include employing an educational RL platform in universities and a pilot program that uses the framework to demonstrate basic RL algorithms. A visual module is also developed to serve those purposes.

Appendix A: Documentation of the proposed framework

To keep the paper brief, we provide documentation of the proposed framework as online materials [148]. These include an installation guide, code samples, benchmark scores, tutorials, an API reference guide, a class diagram, and a package diagram. Table 3 lists the relevant demonstration codes of different use cases (codebase [54]).

Acknowledgements The authors wish to thank our colleagues in the Institute for Intelligent Systems Research and Innovation for their comments and helpful discussion. We truly appreciate Nguyen Chau, a principal IT product manager at Atlassian, who shared his expertise in the field to eradicate potential misunderstanding in this paper. We also thank Dr. Thanh Nguyen, University of Chicago, for being an active adviser in the design process. Finally, we are grateful to the RL community in providing crucial feedback during the project beta testing phase.

Author Contributions Ngoc Duy Nguyen wrote the paper and designed and developed the framework, other authors revised the paper and took part in the software development.

Funding Open Access funding enabled and organized by CAUL and its Member Institutions.

Declarations

Consent for Publication Yes

Conflict of Interests There is no conflict of interest.

References

1. Sutton RS, Barto AG et al (1998) Introduction to reinforcement learning. MIT press Cambridge, vol 135
2. Nguyen ND, Nguyen T, Nahavandi S (2017) System design perspective for human-level agents using deep reinforcement learning: A survey. IEEE Access 5:27091–27102
3. Mao H, Alizadeh M, Menache I, Kandula S (2016) Resource management with deep reinforcement learning. In: Proceedings of the 15th ACM workshop on hot topics in networks, pp 50–56
4. Nguyen TT, Reddi VJ (2021) Deep reinforcement learning for cyber security. IEEE Transactions on Neural Networks and Learning Systems, pp 1–17. https://doi.org/10.1109/TNNLS.2021.3121870
5. Fox D, Burgard W, Kruppa H, Thrun S (2000) A probabilistic approach to collaborative multi-robot localization. Autonomous robots 8(3):325–344
6. Wu X, Chen H, Chen C, Zhong M, Xie S, Guo Y, Fujita H (2020) The autonomous navigation and obstacle avoidance for usvs with anoa deep reinforcement learning method. Knowl-Based Syst 196:105201
7. Mülling K, Kober J, Kroemer O, Peters J (2013) Learning to select and generalize striking movements in robot table tennis. The International Journal of Robotics Research 32(3):263–279
8. Thuruthel TG, Falotico E, Renda F, Laschi C (2018) Model-based reinforcement learning for closed-loop dynamic control of soft robotic manipulators. IEEE Trans Robot 35(1):124–134
9. Li J, Yu T, Zhang X (2021) Emergency fault affected wide-area automatic generation control via large-scale deep reinforcement learning. Eng Appl Artif Intell 106:104500
10. Li J, Yu T, Yang B (2021) A data-driven output voltage control of solid oxide fuel cell using multi-agent deep reinforcement learning. Appl Energy 304:117541
11. Li J, Yu T, Zhang X (2022) Coordinated load frequency control of multi-area integrated energy system using multi-agent deep reinforcement learning. Appl Energy 306:117900
12. Li J, Yu T (2021) A new adaptive controller based on distributed deep reinforcement learning for pemfc air supply system. Energy Reports 7:1267–1279
13. Zheng G, Zhang F, Zheng Z, Xiang Y, Yuan NJ, Xie X, Li Z (2018) Drn: A deep reinforcement learning framework for news recommendation. In: Proceedings of the 2018 World Wide Web Conference, pp 167–176
14. Wu X, Chen H, Wang J, Troiano L, Loia V, Fujita H (2020) Adaptive stock trading strategies with deep reinforcement learning methods. Inf Sci 538:142–158
15. Jin J, Song C, Li H, Gai K, Wang J, Zhang W (2018) Real-time bidding with multi-agent reinforcement learning in display advertising. In: Proceedings of the 27th ACM International Conference on Information and Knowledge Management, pp 2193–2201
16. Xu P, Yin Q, Zhang J, Huang K (2021) Deep reinforcement learning with part-aware exploration bonus in video games. IEEE Transactions on Games
17. Ibarz J, Tan J, Finn C, Kalakrishnan M, Pastor P, Levine S (2021) How to train your robot with deep reinforcement learning: lessons we have learned. The International Journal of Robotics Research 40(4-5):698–721
18. Jaderberg M, Czarnecki WM, Dunning I, Marris L, Lever G, Castaneda AG, Beattie C, Rabinowitz NC, Morcos AS, Ruderman A et al (2019) Human-level performance in 3d multiplayer games with population-based reinforcement learning. Science 364(6443):859–865
19. Bellman RE (2010) Dynamic programming. Princeton University Press

20. Fowler M (2004) Uml distilled: a brief guide to the standard object modeling language. Addison-Wesley Professional
21. Ross TJ (2005) Fuzzy logic with engineering applications. John Wiley & Sons
22. Hausknecht M, Lehman J, Miikkulainen R, Stone P (2014) A neuroevolution approach to general atari game playing. IEEE Transactions on Computational Intelligence and AI in Games 6(4):355–366
23. Bertsekas DP (1995) Dynamic programming and optimal control. Athena Scientific
24. Duchi J, Singer Y (2009) Efficient online and batch learning using forward backward splitting. The Journal of Machine Learning Research 10:2899–2934
25. Adam S, Busoniu L, Babuska R (2011) Experience replay for real-time reinforcement learning control. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews) 42(2):201–212
26. Mnih V, Kavukcuoglu K, Silver D, Rusu AA, Veness J, Bellemare MG, Graves A, Riedmiller M, Fidjeland AK, Ostrovski G et al (2015) Human-level control through deep reinforcement learning. nature 518(7540):529–533
27. Krizhevsky A, Sutskever I, Hinton GE (2012) Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems 25:1097–1105
28. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M et al (2016) Mastering the game of go with deep neural networks and tree search. nature 529(7587):484–489
29. Browne CB, Powley E, Whitehouse D, Lucas SM, Cowling PI, Rohlfshagen P, Tavener S, Perez D, Samothrakis S, Colton S (2012) A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in games 4(1):1–43
30. Tesauro G (1994) Td-gammon, a self-teaching backgammon program, achieves master-level play. Neural computation 6(2):215–219
31. Sallab AhmadEL, Abdou M, Perot E, Yogamani S (2017) Deep reinforcement learning framework for autonomous driving. Electronic Imaging 2017(19):70–76
32. Shalev-Shwartz S, Shammah S, Shashua A (2016) Safe, multi-agent, reinforcement learning for autonomous driving. arXiv:1610. 03295
33. Ng AY, Coates A, Diel M, Ganapathi V, Schulte J, Tse B, Berger E, Liang E (2006) Autonomous inverted helicopter flight via reinforcement learning. In: Experimental robotics IX. Springer, pp 363–372
34. Nazari M, Oroojlooy A, Takáč M, Snyder LV (2018) Reinforcement learning for solving the vehicle routing problem. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. NIPS'18. Curran Associates Inc., Red Hook, NY, USA, p 9861?9871
35. Bello I, Pham H, Le QV, Norouzi M, Bengio S (2017) Neural combinatorial optimization with reinforcement learning. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings. OpenReview.net. https://openreview.net/forum?id=Bk9mxlSFx
36. Panait L, Luke S (2005) Cooperative multi-agent learning: The state of the art. Autonomous agents and multi-agent systems 11(3):387–434
37. Leibo JZ, Zambaldi V, Lanctot M, Marecki J, Graepel T (2017) Multi-agent reinforcement learning in sequential social dilemmas. In: Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems, pp 464–473
38. Wang X, Sandholm T (2002) Reinforcement learning to play an optimal nash equilibrium in team markov games. Advances in neural information processing systems 15:1603–1610
39. Peters J, Schaal S (2008) Natural actor-critic. Neurocomputing 71(7-9):1180–1190
40. He H, Boyd-Graber J, Kwok K, Daumé III H (2016) Opponent modeling in deep reinforcement learning. In: International conference on machine learning, PMLR, pp 1804–1813
41. Goodfellow I, Pouget-Abadie J, Mirza M, Xu B, Warde-Farley D, Ozair S, Courville A, Bengio Y (2014) Generative adversarial nets. Advances in neural information processing systems, vol 27
42. Palmer G, Tuyls K, Bloembergen D, Savani R (2018) Lenient multi-agent deep reinforcement learning. In: Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, pp 443–451
43. Tuyls K, Weiss G (2012) Multiagent learning: Basics, challenges, and prospects. Ai Magazine 33(3):41–41
44. Natarajan S, Tadepalli P (2005) Dynamic preferences in multi-criteria reinforcement learning. In: Proceedings of the 22nd international conference on Machine learning, pp 601–608
45. Roijers DM, Vamplew P, Whiteson S, Dazeley R (2013) A survey of multi-objective sequential decision-making. J Artif Intell Res 48:67–113
46. Van Moffaert K, Nowé A (2014) Multi-objective reinforcement learning using sets of pareto dominating policies. The Journal of Machine Learning Research 15(1):3483–3512
47. Barrett L, Narayanan S (2008) Learning all optimal policies with multiple criteria. In: Proceedings of the 25th international conference on Machine learning, pp 41–47
48. Vamplew P, Dazeley R, Berry A, Issabekov R, Dekker E (2011) Empirical evaluation methods for multiobjective reinforcement learning algorithms. Machine learning 84(1):51–80
49. van Seijen H, Fatemi M, Romoff J, Laroche R, Barnes T, Tsang J (2017) Hybrid reward architecture for reinforcement learning. In: Proceedings of the 31st International Conference on Neural Information Processing Systems, pp 5398–5408
50. Nguyen TT, Nguyen ND, Vamplew P, Nahavandi S, Dazeley R, Lim CP (2020) A multi-objective deep reinforcement learning framework. Eng Appl Artif Intell 96:103915
51. Amodei D, Olah C, Steinhardt J, Christiano P, Schulman J, Mané D (2016) Concrete problems in ai safety. arXiv:1606.06565
52. Christiano PF, Leike J, Brown TB, Martic M, Legg S, Amodei D (2017) Deep reinforcement learning from human preferences. In: Proceedings of the 31st international conference on neural information processing systems, pp 4302–4310
53. Nguyen ND, Nguyen T, Nahavandi S (2019) Multi-agent behavioral control system using deep reinforcement learning. Neurocomputing 359:58–68
54. Nguyen ND, Nguyen TT (2020) Fruit-api. GitHub. https://github.com/garlicdevs/Fruit-API
55. Castro PS, Moitra S, Gelada C, Kumar S, Bellemare MG (2018) Dopamine: A research framework for deep reinforcement learning. arXiv:1812.06110
56. Liang E, Liaw R, Nishihara R, Moritz P, Fox R, Gonzalez J, Goldberg K, Stoica I (2017) Ray rllib: A composable and scalable reinforcement learning library. arXiv:1712.09381, p 85
57. Dhariwal P, Hesse C, Klimov O, Nichol A, Plappert M, Radford A, Schulman J, Sidor S, Wu Y, Zhokhov P (2017) Openai baselines. GitHub. https://github.com/openai/baselines
58. Tokui S, Oono K, Hido S, Clayton J (2015) Chainer: a next-generation open source framework for deep learning. In: Proceedings of workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NIPS), vol 5, pp 1–6

59. Sorokin I, Seleznev A, Pavlov M, Fedorov A, Ignateva A (2015) Deep attention recurrent q-network. arXiv:1512.01693
60. Miyoshi K, Agarwal A, Toghiani-Rizi B (2017) Unreal. GitHub. https://github.com/miyosuda/unreal
61. Fujimoto S, Hoof H, Meger D (2018) Addressing function approximation error in actor-critic methods. In: International conference on machine learning, PMLR, pp 1587–1596
62. Haarnoja T, Zhou A, Abbeel P, Levine S (2018) Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: International conference on machine learning, PMLR, pp 1861–1870
63. Haarnoja T, Zhou A, Hartikainen K, Tucker G, Ha S, Tan J, Kumar V, Zhu H, Gupta A, Abbeel P et al (2018) Soft actor-critic algorithms and applications. arXiv:1812.05905
64. Van Hasselt H, Guez A, Silver D (2016) Deep reinforcement learning with double q-learning. In: Proceedings of the AAAI conference on artificial intelligence, vol 30
65. Wang Z, Schaul T, Hessel M, Hasselt H, Lanctot M, Freitas N (2016) Dueling network architectures for deep reinforcement learning. In: International conference on machine learning, PMLR, pp 1995–2003
66. Schaul T, Quan J, Antonoglou I, Silver D (2016) Prioritized experience replay. In: Bengio Y, LeCun Y (eds) 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings. 1511.05952
67. Hausknecht M, Stone P (2015) Deep recurrent q-learning for partially observable mdps. In: 2015 aaai fall symposium series
68. Hessel M, Modayil J, Van Hasselt H, Schaul T, Ostrovski G, Dabney W, Horgan D, Piot B, Azar M, Silver D (2018) Rainbow: Combining improvements in deep reinforcement learning. In: Thirty-second AAAI conference on artificial intelligence
69. Mnih V, Badia AP, Mirza M, Graves A, Lillicrap T, Harley T, Silver D, Kavukcuoglu K (2016) Asynchronous methods for deep reinforcement learning. In: International conference on machine learning, PMLR, pp 1928–1937
70. Jaderberg M, Mnih V, Czarnecki WM, Schaul T, Leibo JZ, Silver D, Kavukcuoglu K (2017) Reinforcement learning with unsupervised auxiliary tasks. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net. https://openreview.net/forum?id=SJ6yPD5xg
71. Silver D, Lever G, Heess N, Degris T, Wierstra D, Riedmiller M (2014) Deterministic policy gradient algorithms. In: International conference on machine learning, PMLR, pp 387–395
72. Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra D (2016) Continuous control with deep reinforcement learning. In: Bengio Y, LeCun Y (eds) 4th International conference on learning representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings. arXiv:1509. 02971
73. Schulman J, Levine S, Abbeel P, Jordan M, Moritz P (2015) Trust region policy optimization. In: International conference on machine learning, PMLR, pp 1889–1897
74. Van Erven T, Harremos P (2014) Rényi divergence and kullback-leibler divergence. IEEE Trans Inf Theory 60(7):3797–3820
75. Wu Y, Mansimov E, Grosse RB, Liao S, Ba J (2017) Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. Advances in neural information processing systems 30:5279–5288
76. Wang Z, Bapst V, Heess N, Mnih V, Munos R, Kavukcuoglu K, de Freitas N (2017) Sample efficient actor-critic with experience replay. In: 5th International conference on learning representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net. https://openreview.net/forum?id=HyM25Mqel
77. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. arXiv:1707.06347
78. Nachum O, Norouzi M, Xu K, Schuurmans D (2017) Bridging the gap between value and policy based reinforcement learning. In: Proceedings of the 31st international conference on neural information processing systems, pp 2772–2782
79. O'Donoghue B, Munos R, Kavukcuoglu K, Mnih V (2017) Combining policy gradient and q-learning. In: 5th International conference on learning representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net. https://openreview.net/forum?id=B1kJ6H9ex
80. Schulman J, Chen X, Abbeel P (2017) Equivalence between policy gradients and soft q-learning. arXiv:1704.06440
81. Gruslys A, Dabney W, Azar MG, Piot B, Bellemare MG, Munos R (2018) The reactor: A fast and sample-efficient actor-critic agent for reinforcement learning. In: 6th International conference on learning representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. OpenReview.net. https://openreview.net/forum?id=rkHVZWZAZ
82. Gu S, Lillicrap T, Ghahramani Z, Turner RE, Schölkopf B, Levine S (2017) Interpolated policy gradient: merging on-policy and off-policy gradient estimation for deep reinforcement learning. In: Proceedings of the 31st international conference on neural information processing systems, pp 3849–3858
83. Barth-Maron G, Hoffman MW, Budden D, Dabney W, Horgan D, TB D, Muldal A, Heess N, Lillicrap TP (2018) Distributed distributional deterministic policy gradients. In: 6th International conference on learning representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. OpenReview.net. https://openreview.net/forum?id=SyZipzbCb
84. Espeholt L, Marinier R, Stanczyk P, Wang K, Michalski M (2020) SEED RL: scalable and efficient deep-rl with accelerated central inference. In: 8th International conference on learning representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net. https://openreview.net/forum?id=rkgvXlrKwH
85. Schwarzer M, Anand A, Goel R, Hjelm RD, Courville AC, Bachman P (2021) Data-efficient reinforcement learning with self-predictive representations. In: 9th International conference on learning representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net. https://openreview.net/forum?id=uCQfPZwRaUu
86. Tampuu A, Matiisen T, Kodelja D, Kuzovkin I, Korjus K, Aru J, Aru J, Vicente R (2017) Multiagent cooperation and competition with deep reinforcement learning. PloS one 12(4):e0172395
87. Kraemer L, Banerjee B (2016) Multi-agent reinforcement learning as a rehearsal for decentralized planning. Neurocomputing 190:82–94
88. Lowe R, Wu Y, Tamar A, Harb J, Abbeel P, Mordatch I (2017) Multi-agent actor-critic for mixed cooperative-competitive environments. In: Proceedings of the 31st international conference on neural information processing systems, pp 6382–6393
89. Foerster JN, Assael YM, de Freitas N, Whiteson S (2016) Learning to communicate with deep multi-agent reinforcement learning. In: Proceedings of the 30th international conference on neural information processing systems, pp 2145–2153
90. Sukhbaatar S, Fergus R et al (2016) Learning multiagent communication with backpropagation. Advances in neural information processing systems 29:2244–2252
91. Gupta JK, Egorov M, Kochenderfer M (2017) Cooperative multi-agent control using deep reinforcement learning. In: International

conference on autonomous agents and multiagent systems, Springer, pp 66–83
92. Nguyen TT, Nguyen ND, Nahavandi S (2020) Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. IEEE transactions on cybernetics 50(9):3826–3839
93. Egorov M (2016) Multi-agent deep reinforcement learning. CS231n: convolutional neural networks for visual recognition, pp 1–8
94. Shu T, Tian Y (2019) M^3rl: Mind-aware multi-agent management reinforcement learning. In: 7th International conference on learning representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net. https://openreview.net/forum?id=BkzeUiRcY7
95. Yang J, Nakhaei A, Isele D, Fujimura K, Zha H (2020) CM3: cooperative multi-goal multi-stage multi-agent reinforcement learning. In: 8th International conference on learning representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net. https://openreview.net/forum?id=S1lEX04tPr
96. Long Q, Zhou Z, Gupta A, Fang F, Wu Y, Wang X (2020) Evolutionary population curriculum for scaling multi-agent reinforcement learning. In: 8th International conference on learning representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net. https://openreview.net/forum?id=SJxbHkrKDH
97. Kim D, Moon S, Hostallero D, Kang WJ, Lee T, Son K, Yi Y (2019) Learning to schedule communication in multi-agent reinforcement learning. In: 7th International conference on learning representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net. https://openreview.net/forum?id=SJxu5iR9KQ
98. Schroeder de Witt C, Foerster J, Farquhar G, Torr P, Boehmer W, Whiteson S (2019) Multi-agent common knowledge reinforcement learning. Advances in Neural Information Processing Systems 32:9927–9939
99. Christianos F, Sch¨afer L, Albrecht SV (2020) Shared experience actor-critic for multi-agent reinforcement learning. In: Larochelle H, Ranzato M, Hadsell R, Balcan M-F, Lin H-T (eds) Advances in neural information processing systems 33: annual conference on neural information processing systems 2020, NeurIPS 2020, December 6-12, 2020, virtual. https://proceedings.neurips.cc/paper/2020/hash/7967cc8e3ab559e68cc944c44b1cf3e8-Abstract.html
100. Wang J, Ren Z, Liu T, Yu Y, Zhang C (2021) QPLEX: duplex dueling multi-agent q-learning. In: 9th International conference on learning representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021. OpenReview.net. https://openreview.net/forum?id=Rcmk0xxIQV
101. Wang J, Kurth-Nelson Z, Soyer H, Leibo JZ, Tirumala D, Munos R, Blundell C, Kumaran D, Botvinick MM (2017) Learning to reinforcement learn. In: Gunzelmann G, Howes A, Tenbrink T, Davelaar EJ (eds) Proceedings of the 39th annual meeting of the cognitive science society, CogSci 2017, London, UK, 16-29 July 2017. cognitivesciencesociety.org. https://mindmodeling.org/cogsci2017/papers/0252/index.html
102. Agarwal R, Liang C, Schuurmans D, Norouzi M (2019) Learning to generalize from sparse and underspecified rewards. In: International conference on machine learning, PMLR, pp 130–140
103. Rakelly K, Zhou A, Finn C, Levine S, Quillen D (2019) Efficient off-policy meta-reinforcement learning via probabilistic context variables. In: International conference on machine learning, PMLR, pp 5331–5340
104. Liu EZ, Raghunathan A, Liang P, Finn C (2021) Decoupling exploration and exploitation for meta-reinforcement learning without sacrifices. In: International conference on machine learning, PMLR, pp 6925–6935
105. Zintgraf LM, Feng L, Lu C, Igl M, Hartikainen K, Hofmann K, Whiteson S (2021) Exploration in approximate hyper-state space for meta reinforcement learning. In: International conference on machine learning, PMLR, pp 12991–13001
106. Zintgraf L, Devlin S, Ciosek K, Whiteson S, Hofmann K (2021) Deep interactive bayesian reinforcement learning via meta-learning. In: Proceedings of the 20th international conference on autonomous agents and multiagent systems, pp 1712–1714
107. Gupta A, Mendonca R, Liu Y, Abbeel P, Levine S (2018) Meta-reinforcement learning of structured exploration strategies. Advances in Neural Information Processing Systems 31:5302–5311
108. Lin Z, Thomas G, Yang G, Ma T (2020) Model-based adversarial meta-reinforcement learning. In: Larochelle H, Ranzato M, Hadsell R, Balcan M-F, Lin H-T (eds) Advances in neural information processing systems 33: Annual conference on neural information processing systems 2020, NeurIPS 2020, December 6-12, 2020, virtual. https://proceedings.neurips.cc/paper/2020/hash/73634c1dcbe056c1f7dcf5969da406c8-Abstract.html
109. Girosi F, Jones M, Poggio T (1995) Regularization theory and neural networks architectures. Neural computation 7(2):219–269
110. Goodfellow IJ, Mirza M, Da X, Courville AC, Bengio Y (2014) An empirical investigation of catastrophic forgeting in gradient-based neural networks. In: Bengio Y, LeCun Y (eds) 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings. 1312.6211
111. Thrun S, Pratt L (1998) Learning to learn: Introduction and overview. In: Learning to learn. Springer, pp 3–17
112. Rusu AA, Rabinowitz NC, Desjardins G, Soyer H, Kirkpatrick J, Kavukcuoglu K, Pascanu R, Hadsell R (2016) Progressive neural networks. arXiv:1606.04671
113. Kirkpatrick J, Pascanu R, Rabinowitz N, Veness J, Desjardins G, Rusu AA, Milan K, Quan J, Ramalho T, Grabska-Barwinska A et al (2017) Overcoming catastrophic forgetting in neural networks. Proceedings of the national academy of sciences 114(13):3521–3526
114. Fernando C, Banarse D, Blundell C, Zwols Y, Ha D, Rusu AA, Pritzel A, Wierstra D (2017) Pathnet: Evolution channels gradient descent in super neural networks. arXiv:1701.08734
115. Rusu AA, Colmenarejo SG, Gülçehre C, Desjardins G, Kirkpatrick J, Pascanu R, Mnih V, Kavukcuoglu K, Hadsell R (2016) Policy distillation. In: Bengio Y, LeCun Y (eds) 4th International conference on learning representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings. 1511.06295
116. Yin H, Pan SJ (2017) Knowledge transfer for deep reinforcement learning with hierarchical experience replay. In: Thirty-first AAAI conference on artificial intelligence
117. Parisotto E, Ba LJ, Salakhutdinov R (2016) Actor-mimic: Deep multitask and transfer reinforcement learning. In: Bengio Y, LeCun Y (eds) 4th International conference on learning representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings. 1511.06342
118. Wulfmeier M, Posner I, Abbeel P (2017) Mutual alignment transfer learning. In: Conference on robot learning, PMLR, pp 281–290
119. Grześ M, Kudenko D (2010) Online learning of shaping rewards in reinforcement learning. Neural Netw 23(4):541–550
120. Barto AG, Mahadevan S (2003) Recent advances in hierarchical reinforcement learning. Discrete event dynamic systems 13(1):41–77

121. Kulkarni TD, Narasimhan K, Saeedi A, Tenenbaum J (2016) Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. Advances in neural information processing systems 29:3675–3683
122. Burda Y, Edwards H, Pathak D, Storkey AJ, Darrell T, Efros AA (2019) Large-scale study of curiosity-driven learning. In: 7th International conference on learning representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net. https://openreview.net/forum?id=rJNwDjAqYX
123. Pathak D, Agrawal P, Efros AA, Darrell T (2017) Curiosity-driven exploration by self-supervised prediction. In: International conference on machine learning, PMLR, pp 2778–2787
124. Ostrovski G, Bellemare MG, Oord A, Munos R (2017) Count-based exploration with neural density models. In: International conference on machine learning, PMLR, pp 2721–2730
125. Andrychowicz M, Wolski F, Ray A, Schneider J, Fong R, Welinder P, McGrew B, Tobin J, Abbeel P, Zaremba W (2017) Hindsight experience replay. In: Proceedings of the 31st international conference on neural information processing systems, pp 5055–5065
126. Bengio Y, Louradour J, Collobert R, Weston J (2009) Curriculum learning. In: Proceedings of the 26th annual international conference on machine learning, pp 41–48
127. Santoro A, Faulkner R, Raposo D, Rae J, Chrzanowski M, Weber T, Wierstra D, Vinyals O, Pascanu R, Lillicrap T (2018) Relational recurrent neural networks. Advances in Neural Information Processing Systems 31:7299–7310
128. Parisotto E, Salakhutdinov R (2018) Neural map: Structured memory for deep reinforcement learning. In: 6th International conference on learning representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. OpenReview.net. https://openreview.net/forum?id=Bk9zbyZCZ
129. Horgan D, Quan J, Budden D, Barth-Maron G, Hessel M, van Hasselt H, Silver D (2018) Distributed prioritized experience replay. In: 6th International conference on learning representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings. OpenReview.net. https://openreview.net/forum?id=H1Dy---0Z
130. Stooke A, Abbeel P (2018) Accelerated methods for deep reinforcement learning. arXiv:1803.02811
131. Liang E, Liaw R, Nishihara R, Moritz P, Fox R, Goldberg K, Gonzalez J, Jordan M, Stoica I (2018) Rllib: Abstractions for distributed reinforcement learning. In: International conference on machine learning, PMLR, pp 3053–3062
132. Ho J, Ermon S (2016) Generative adversarial imitation learning. Advances in neural information processing systems 29:4565–4573
133. Bellemare MG, Naddaf Y, Veness J, Bowling M (2013) The arcade learning environment: An evaluation platform for general agents. J Artif Intell Res 47:253–279
134. Brockman G, Cheung V, Pettersson L, Schneider J, Schulman J, Tang J, Zaremba W (2016) Openai gym. arXiv:1606.01540
135. Todorov E, Erez T, Tassa Y (2012) Mujoco: A physics engine for model-based control. In: 2012 IEEE/RSJ international conference on intelligent robots and systems, IEEE, pp 5026–5033
136. Plappert M (2016) keras-rl. GitHub. https://github.com/keras-rl/keras-rl
137. Kuhnle A, Schaarschmidt M, Fricke K (2017) Tensorforce: a tensorflow library for applied reinforcement learning. Web page. https://github.com/tensorforce/tensorforce
138. Hill A, Raffin A, Ernestus M, Gleave A, Kanervisto A, Traore R, Dhariwal P, Hesse C, Klimov O, Nichol A, Plappert M, Radford A, Schulman J, Sidor S, Wu Y (2018) Stable baselines. GitHub. https://github.com/hill-a/stable-baselines
139. Raffin A, Hill A, Gleave A, Kanervisto A, Ernestus M, Dormann N (2021) Stable-baselines3: Reliable reinforcement learning implementations. J Mach Learn Res 22(268):1–8. http://jmlr.org/papers/v22/20-1364.html
140. Duan Y, Chen X, Houthooft R, Schulman J, Abbeel P (2016) Benchmarking deep reinforcement learning for continuous control. In: International conference on machine learning, PMLR, pp 1329–1338
141. Terry JK, Black B, Jayakumar M, Hari A, Sullivan R, Santos L, Dieffendahl C, Williams NL, Lokesh Y, Horsch C et al (2020) Pettingzoo: Gym for multi-agent reinforcement learning. arXiv:2009.14471
142. Zheng L, Yang J, Cai H, Zhou M, Zhang W, Wang J, Yu Y (2018) Magent: A many-agent reinforcement learning platform for artificial collective intelligence. In: Proceedings of the AAAI conference on artificial intelligence, vol 32
143. Hoffman M, Shahriari B, Aslanides J, Barth-Maron G, Behbahani F, Norman T, Abdolmaleki A, Cassirer A, Yang F, Baumli K et al (2020) Acme: A research framework for distributed reinforcement learning. arXiv:2006.00979
144. Petrenko A, Wijmans E, Shacklett B, Koltun V (2021) Megaverse: Simulating embodied agents at one million experiences per second. In: International conference on machine learning, PMLR, pp 8556–8566
145. Weng J, Chen H, Yan D, You K, Duburcq A, Zhang M, Su H, Zhu J (2021) Tianshou: A highly modularized deep reinforcement learning library. arXiv:2107.14171
146. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L et al (2019) Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems 32:8026–8037
147. Ellis B, Stylos J, Myers B (2007) The factory pattern in api design: A usability evaluation. In: 29th International conference on software engineering (ICSE'07), IEEE, pp 302–312
148. Nguyen ND, Nguyen TT (2020) Fruitlab. https://fruitlab.org/

Affiliations

Ngoc Duy Nguyen[1] · Thanh Thi Nguyen[2] · Nhat Truong Pham[3,4] · Hai Nguyen[5] · Dang Tu Nguyen[6] · Thanh Dang Nguyen[7] · Chee Peng Lim[1] · Michael Johnstone[1] · Asim Bhatti[1] · Douglas Creighton[1] · Saeid Nahavandi[1]

Thanh Thi Nguyen
thanh.nguyen@deakin.edu.au

Nhat Truong Pham
phamnhattruong.st@tdtu.edu.vn

Hai Nguyen
hainguyen@ccs.neu.edu

Dang Tu Nguyen
dtnguyen@amazon.com

Thanh Dang Nguyen
thanhnd@uchicago.edu

Chee Peng Lim
chee.lim@deakin.edu.au

Michael Johnstone
michael.johnstone@deakin.edu.au

Asim Bhatti
asim.bhatti@deakin.edu.au

Douglas Creighton
douglas.creighton@deakin.edu.au

Saeid Nahavandi
saeid.nahavandi@deakin.edu.au

1 Institute for Intelligent Systems Research and Innovation, Deakin University, Geelong, 3220, Victoria, Australia

2 School of Information Technology, Deakin University, Melbourne, 3002, Victoria, Australia

3 Division of Computational Mechatronics, Institute for Computational Science, Ton Duc Thang University, Ho Chi Minh City, 72915, Vietnam

4 Faculty of Electrical and Electronics Engineering, Ton Duc Thang University, Ho Chi Minh City, 72915, Vietnam

5 Khoury College of Computer Sciences, Northeastern University, Boston, 02115, MA, USA

6 Amazon Web Services, Seattle, 98144, WA, USA

7 Center for Translational Data Science, University of Chicago, Street, Chicago, 60615, IL, USA