# **Difference Between Artificial Intelligence vs Machine Learning vs Deep Learning**

Artificial Intelligence is the mechanism to incorporate human intelligence into machines through a set of rules (algorithm).

ML is an application or subset of AI.

* The major aim of ML is to allow the systems to learn by themselves through experience without any kind of human intervention or assistance.

Deep learning is a sub-part of ML which makes use of Neural Networks

Deep Reinforcement Learning

DRL is a subfield of AI and ML which combines deep learning and reinforcement learning.

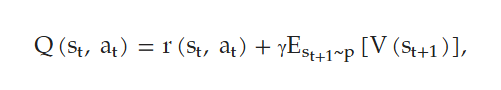
DL is a technique that uses artificial neural networks to learn from data, with the neural network serving as a predictive model that extracts complex features from data and performs predictions based on that data.

DRL uses deep learning to represent the policy and the action value function of the agent.

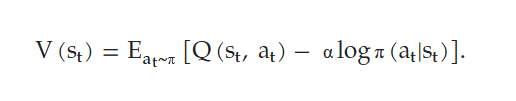
* Policy network returns the probability distribution of actions based on the environment's observations and states.
* Action value function evaluates the action taken in a given state and represents the results as a numerical value.

Soft Actor Critic

When an action a\_t is taken in state s\_t, SAC computes the soft Q-value as:



The A-function estimates the expected return of taking action a\_t in state s\_t, plus the expected value of the next state\_s\_t+1



How good is it to be in state s\_t, considering all possible actions from the current policy.

Deep Reinforcement Learning and Brief Survey

Deep learning is enabling RL to scale to problems that were previously intractable.

* Learning to play video games directly from pixels
* Control policies for robots to be learned directly from camera inputs in the real world.

Value Functions

Value function methods are based on estimating the value of being in a given state.

Introduction

A primary goal of AI is to produce fully autonomous agents that interact with their environments to learn optimal behaviors.

* Principled mathematical framework for experience-driven autonomous learning is RL

Approaches prior to DRL were inherently limited to low dimensional problems

* Memory complexity
* Computation complexity
* Sample complexity

The powerful function approximation and representation learning properties of deep neural networks has provided new tools in overcoming these challenges.

* DNN can automatically find compact low-dimensional representations of high-dimensional data (e.g., images, text, and audio)

Deep learning enables RL to scale to decision-making problems that were previously intractable. i.e , settings with high dimensional state and action spaces.

The Curse of Dimensionality - refers to the various challenges (such as computation complexity) that arise when applying data analysis and machine learning methods to high dimensional datasets.

As the dimensionality of a dataset increases, the volume of the high dimensional space grows exponentially. The available data can become very sparse in relation to higher dimensions volumes, where essentially all the data points become further and further apart. This makes machine learning tasks that rely on identifying feature interactions and similarities in groups of data, like clustering , extremely hard and often computationally intractable.

[Reinforcement Learning: A Survey](https://www.jair.org/index.php/jair/article/view/10166/24110)

Introduction

Reinforcement Learning is the problem faced by an agent that must learn behavior through trial-and-error interactions with a dynamic enviroment.

The agent’s behavior, *B*, should choose actions that increase the long-run discounted, accumulated reward.

* Learn to do this over time by systemic trial and error, guided by a wide variety of algorithms.
  + Agents job is to find policy π(at|st), mapping states to actions, that maximizes some long-run measure of reinforcement.

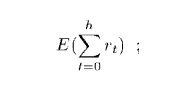
Different from supervised learning in several ways

* No presentation of input output pairs.
  + After choosing an action, the agent is told the immediate reward and the subsequent state, it is not told which action would have been the best in it’s long term interests.
* On-line performance is important: the evaluation of the system is concurrent with learning.

1.2 Models of Optimal Behavior

The *finite-horizon* model

* At a given moment in time, the agent should optimize it’s expected reward for the next *h* time steps:



Where r\_t is the scalar reward received *t* steps into the future.

The *infinite-horizon model*

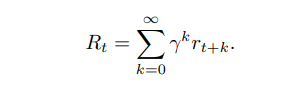
[DEEP REINFORCEMENT LEARNING: AN OVERVIEW](https://arxiv.org/pdf/1810.06339)

***Introduction/Background***

Problem Setup

At each time step t,

1. Agent receives a state s\_t in a state space S
2. Agent selects action a\_t from actions space A, following policy π(at|st)
3. Agent receives scaler reward r\_t
4. Environment transitions to the next state s\_t+1 according to environment dynamics, and state transition probabiltiy P(st+1|st, at).
5. Return discounted,accumulated reward with the discount factor γ ∈ (0, 1].

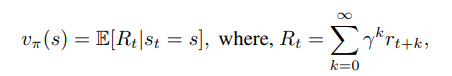


When an RL problem satisfies the Markov property, it is formulated as a Markov decision process (MDP), defined by the 5-tuple (S, A,P, R, γ)

Value Function

A value function is a prediction of the expected, accumulative, discounted, future reward, measuring how good each state, or state-action pair is.

The state value:



Is the expected return for following policy π from state s.

The action value,

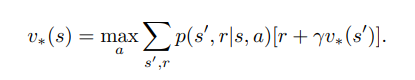


is the expected return for selecting action a in state s and then following policy π.

An optimal state value:



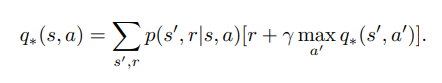
Is the maximum state value achievable by any policy for state s, which decomposes into the Bellman equation:



An optimal action value function:



Is the maximum action value achievable by any policy for state s and action a, which decomposes into the Bellman equation:

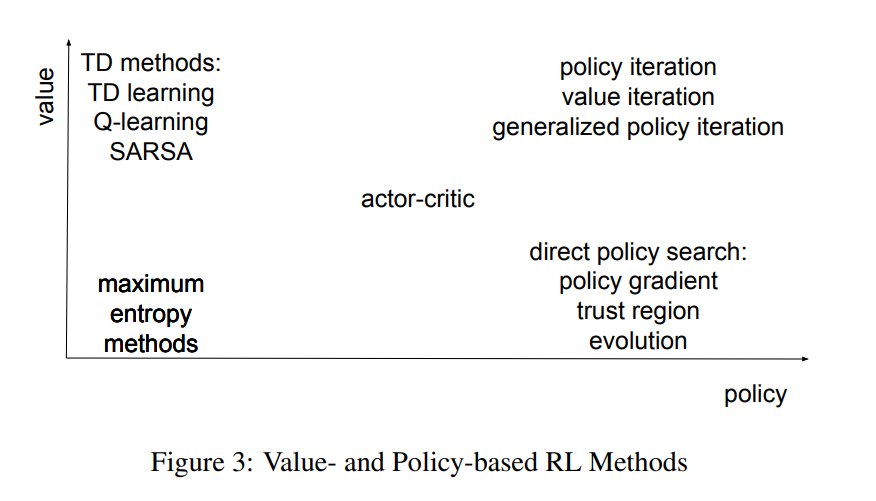


Dynamic Programming

Deep RL

Deep Reinforcement Learning methods are those where deep neural networks are used to represent the state or observation, and/or to approximate any of the following components of reinforcement learning: value function, vˆ(s; θ) or qˆ(s, a; θ), policy π(a|s; θ), and model (state transition function and reward function)\

***Part I: Core Elements***



Value Function

A value function is a prediction of the expected, accumulative, discounted, future reward, measuring the goodness of each state, or each state-action pair.

* Temporal Difference (TD) Learning
* Q-Learning

Once we have an optimal value function, we may derive an optimal policy

Deep Q-Learning

Before Deep Q-Network (DQN), it was well known that RL is unstable or even divergent when action value function is approximated with a nonlinear function like neural networks.

DQN makes several contributions:

1. Stabilizing the training of action value function approximation with deep neural networks.
2. Designing an end-to-end RL approach, w/ only the pixels and the game score as inputs (minimal domain knowledge is required).
3. Training a flexible network with the same algorithm, network architecture and hyperparameters to perform well on many different tasks

Policy

A policy maps a state to an action, or a distribution over actions, and policy optimization is to find an optimal mapping. Value-based methods optimize value functions first, then derive optimal policies. Policy-based methods directly optimize an objective function, usually cumulative rewards.

On policy methods are sample inefficient, using data only once, with estimations based on trajectories for the current policy.

Off-policy methods, like Q-learning, can learn from any trajectories from any policies, e.g., expert demonstrations, from the same environment.

Policy Gradient

Policy gradients are popular methods in RL, optimizing policies directly. Policies may be deterministic or stochastic.

Actor-Critic

An actor critic algorithm learns both a policy and state value function, and the value function is used for bootstrapping.

***Part II: Important Mechanisms***

Attention is a mechanism to focus on the salient parts. (Can be an approach for memory addressing)

Memory provides long term data storage.

Robotics

Classical application area for reinforcement learning.

Challenges for reinforcement learning applications to robotics include dimensionality, realworld examples, under-modeling (models not capturing all details of system dynamics) and model uncertainty, and reward and goal specification.

Sim-to-Real

It is easier to train a robot in simulation than in reality.

* RL algorithms are sample intensive
* Exploration may cause risky policies to the robot/environment

However, a simulator cannot precisely reflect reality.

Sim-to-Real (Special type of transfer learning) is a term used to describe the techniques used to bridge the gap between simulation and reality. (Critical and Challenging Task in Robotics).

[Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor](https://arxiv.org/pdf/1801.01290)

Model-free deep RL algorithms typically suffer from two major challenges: very high sample complexity and brittle convergence properties.

* Necessitate meticulous hyperparameter tuning.
* Limit applicability of such methods to complex, real-world domains.

Soft Actor-Critic; An off-policy actor-critic deep RL algorithm based on the maximum entropy reinforcement learning framework.

* Actor aims to maximize expected reward while also maximizing entropy.

The soft actor-critic algorithm incorporates three key ingredients: an actor-critic architecture with separate policy and value function networks, an off-policy formulation that enables reuse of previously collected data for efficiency, and entropy maximization to enable stability and exploration.

Related Work

Actor-critic algorithms are typically derived starting from policy iteration, which alternates between *policy evaluation* - computing the value function of a policy - and *policy improvement* - using the value function to obtain a better policy.

It is typically impractical to run either steps to convergence, instead the value function and policy are optimized jointly.

* Policy is referred to as the actor
* Value function is referred to as the critic

Preliminaries

Notation

We consider an infinite-horizon Markov decision process (MDP), defined by the tuple (S, A, p, r)

* S: State Space, continuous
* A: Action Space, continuous
* p: S x S x A -> [0, INF), the probability density of the next state, given the current state and action.
* R: S x A -> [r\_min, r\_max], bound reward emitted upon each transition.

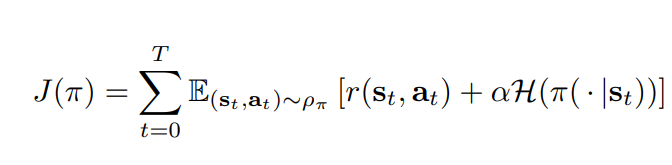
ρ*π(st) and ρ*π(st, at) to denote the state and state-action marginals of the trajectory distribution induced by a policy π(at|st).

Maximum Entropy Reinforcement Learning

Standard RL maximizes the expected sum of rewards,



We consider a more general maximum entropy objective which favors stochastic policies by augmenting the objective with the expected entropy of the policy over ρπ(st):



* α : temperature parameter, determines the relative importance of the entropy term in the reward.

[Design and Challenges of Radiopharmaceuticals](https://www.sciencedirect.com/science/article/pii/S0001299819300704)

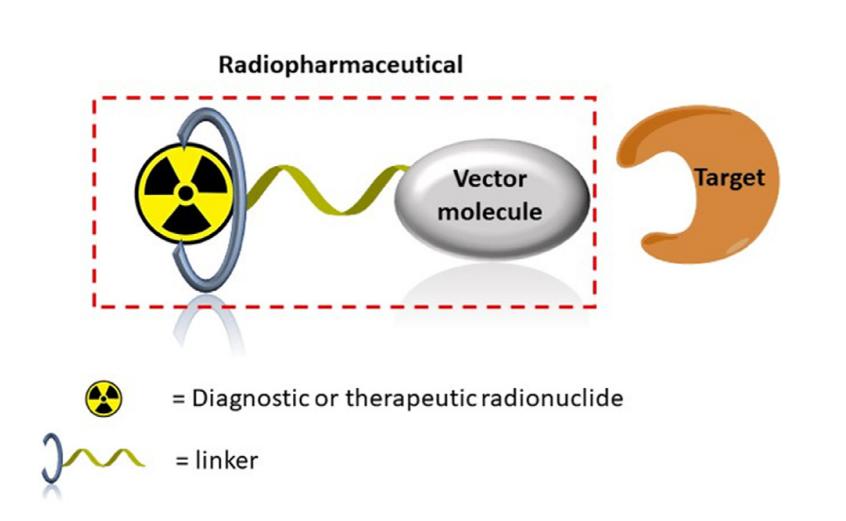
Radioactive label is used diagnostically as an emitter of EM radiation, of which detection allows quantification of the concentration of the radiopharmaceutical.

* In vivo distribution of the radionuclide can be determined using SPECT or PET scans.

Alternatively can be used for therapeutic applications where the ionizing radiation emitted upon decay of the radionuclide is used to destroy cells.

Radiopharmaceutical for Diagnosis and Therapy: The Radionuclide Vector Concept

In general, a radiopharmaceutical consists of three components: a vector molecule, a radionuclide for diagnostic or therapeutic applications and a linker in between.



The radionuclide provides the radiation component, the vector molecule specifically targets biomolecules expressed in tissues or cells, and the linker forms a stable chemical connection between the radionuclide and the vector molecule

Radionuclide

Radionuclides are elements w/ an excess of nuclear energy, making them unstable.

* Excess of energy in the nucleus of the unstable element can result in emission of either particles (a, b+/), and/or EM radiation (gamma ray photons [g]) and as a secondary effect X-rays, conversion electrons and Auger electrons.

Radionuclides occur naturally or are artificially made using cyclotrons, particle accelerators, or are generated by decay of the radionuclides and obtained from radionuclide generators.

A rapidly expanding number of radionuclides with a broad variety of half-lives, emission types, and energies are routinely produced.

* A wide range of radiopharmaceuticals makes it possible to carefully pick the best matching radionuclide for a given applicaton.

Production

The production of a radiopharmaceutical consists of

1. Production of the radionuclide
2. Incorporation of the radionuclide in the radiopharmaceutical
3. Purification and reformulation

# [**Development of novel radionuclides for medical applications**](https://analyticalsciencejournals.onlinelibrary.wiley.com/doi/full/10.1002/jlcr.3578)

[Optuna: A Next-generation Hyperparameter Optimization Framework](https://dl.acm.org/doi/pdf/10.1145/3292500.3330701)

Sampling Methods on Dynamically Constructed Parameter Space

There are generally two sampling methods; *relational sampling* which exploits correlations among the parameters and *independent sampling* that samples each parameter independently.

The Optuna framework features both, and it can handle various independent sampling methods including TPE as well as relational sampling methods like CMA-ES.

*Relational sampling in define-by-run frameworks.*

* An advantage of the old *define-and-run* optimization design is that the program is given knowledge of the concurrence relations among the hyperparameters from the beginning of the optimization process.
* Implementing optimization methods that take concurrence relations among parameters is nontrivial when search spaces are being dynamically constructed.
  + To solve this problem, Oputna identifies trial results that are informative about the concurrence relations. Allows the framework to identify underlying concurrence relations after some number of independent samplings, then use the identified relations to conduct user-selected relations sampling algorithms like CMA-ES and GP-B).

[Tree-Structured Parzen Estimator: Understanding Its Algorithm Components and Their Roles for Better Empirical Performance](https://arxiv.org/pdf/2304.11127)

Tree-structured Parzen estimator (TPE) is a widely used Bayesian optimization method in these frameworks.

Notations

[IsaacLab](https://isaac-sim.github.io/IsaacLab/main/index.html)

IsaacLab; A unified and modular framework for robot learning that aims to simplify common workflows in robot research (i.e reinforcement learning, learning from demonstrations, and motion planning).

Core Objectives:

* Modularity: Easily customize and add new environments, robots, and sensors.
* Agility: Adapt to the changing needs of the community.
* Openness: Remain open-sourced to allow the community to contribute and extend the framework.
* Batteries-included: Include a number of environments, sensors, and tasks that are ready to use.

Isaac Lab Ecosystem

Isaac Lab is built on top of Isaac Sim.

* Designed to be modular and extensible
* Aims to simplify common workflows in robots research

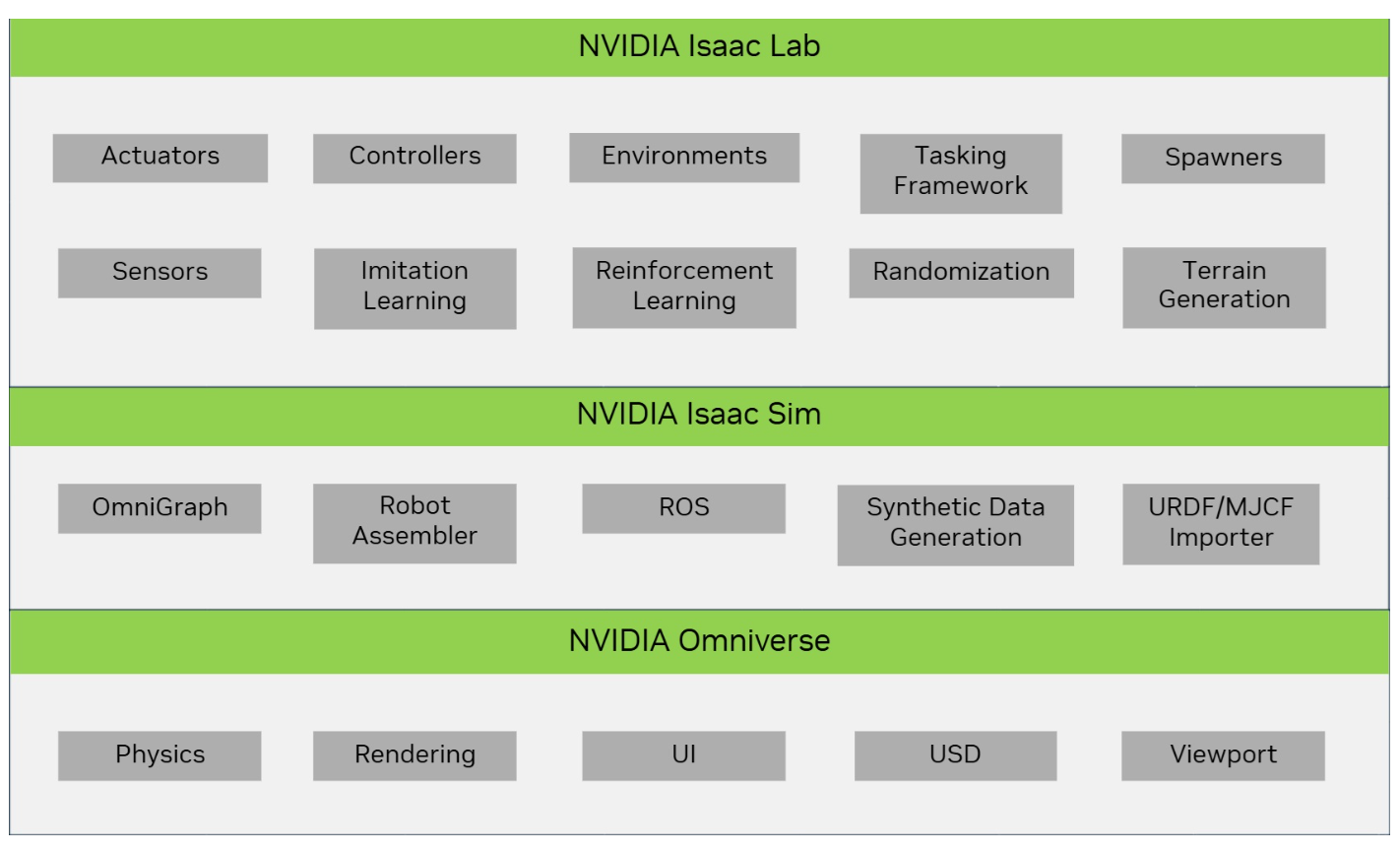
Although Isaac Lab includes pre-built environments, sensors, and tasks, the main goal is to provide an open sourced, unified, and easy-to-use interface for developing and testing custom environments and robot learning algorthims.

Working in Isaac Lab Requires:

* Isaac Sim
  + URDF/MJCF importers
  + Simulation Managers
  + ROS features
  + NVIDIA Omniverse platform
    - Advanced physics simulation (PhysX)
    - Photorealistic rendering technologies
    - Universal Scene Description (USD) for scene creation

On top of inheriting capabilities from Isaac Sim, Isaac Lab adds a number of new features that pertain to robot learning research.

* Actuator dynamics in simulation
* Procedural terrain generation
* Support to collect data from human demonstrations



Where does IsaacLab fit in the Isaac ecosystem?

Isaac Gym provides a high performance GPU-based physics simulation for robot learning.

* Built on top of PhysX
  + Support GPU-accelerated simulation of rigid bodies
  + Python API to directly access physic simulation data
* Despite the success of Isaac Gym it is not designed to be a general purpose simulator for robotics. It does not include;
  + Interaction between deformable and rigid objects
  + High-fidelity rendering
  + Support for ROS

With the release of Isaac Sim, NVIDIA is building a general purpose simulator for robotics and has integrated the functionalities of Isaac Gym.

Simulation Setup and Configuration

USD assets can be annotated with UsdPhysicsSchema and PhysX Schema to give them physical properties for simulation.

Colliders

Geometric prims may be used as colliders by applying UsdPhysics.CollsionAPI

Rigid Bodies

Rigid bodies are used to represent objects that can move without deforming.

* Typically affected by external forces (i.e gravity, collisions, and other constraints such as joints.)
* If kinematic, motion can be fully controlled by the application to influence other objects
* Represented in the USD stage hierarchy as a prim of type UsdGeom.Xformable, with the applied UsdPhysics.RigidBodyAPI.
  + The prim with UsdPhysics.RigidBodyAPI and all descendants move as one rigid object.

Another way to think about dynamic vs kinematic rigid bodies:

* A dynamic rigid obyd has its transform written to by the simulation.
* A kinematic rigid body’s transform is read by the simulation

Joints

Joins govern the relative motion of pairs of rigid bodies.

Task Design Workflows

A Task is defined by an environment w/ specific interfaces for observation to and actions from a specific agent (robot).

* Environment provides agent w/ current observations
* Environment executes agents’s actions by updating the simulation forward time.

Managing actions, observations, rewards, ect…. across a vectorized GPU simulation as required by RL can be a daunting task. Isaac Lab provides Manger-based and direct RL interfaces:

Manager-based

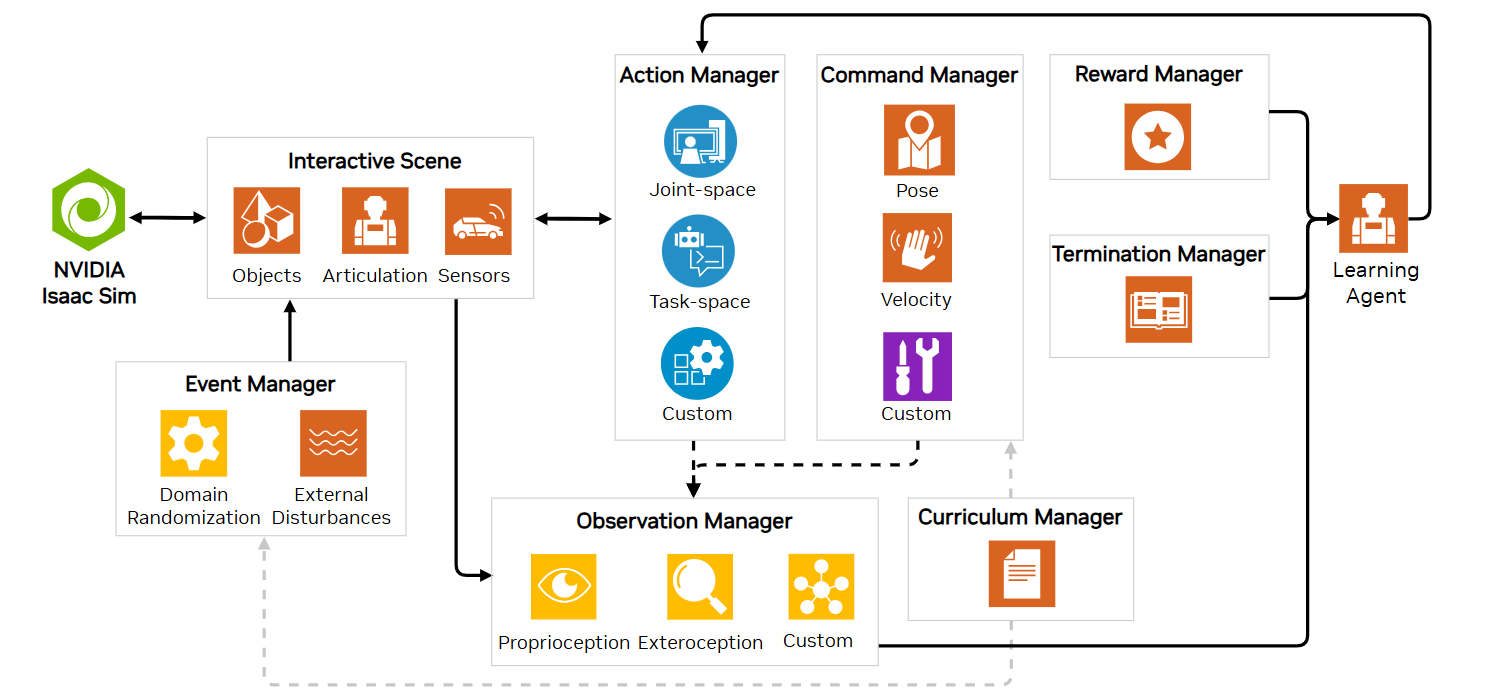
* Environment is decomposed into individual components (managers) that handle different aspects of the environment.
* User defines configuration classes for each component
* Environment is responsible for coordinating managers and calling their functions.

Direct

* User defines a single class that implements the entire environment directly w/o need for separate managers.

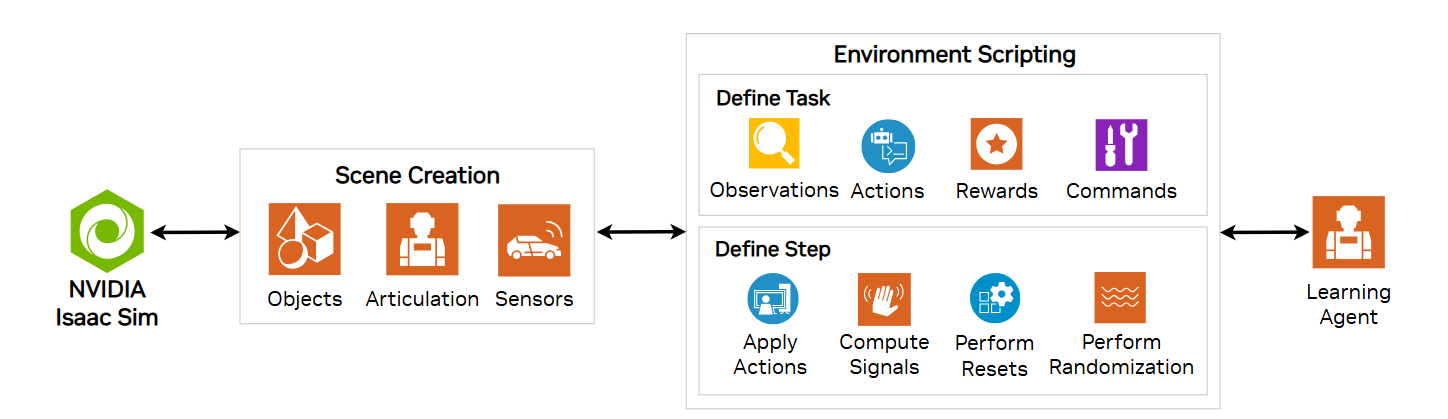
Manager-based workflow is more modular and allows different components of the environment to be swapped out easily.

* Useful for prototyping environment/experimenting w/ different configurations



Direct workflow is more efficient and allows for more fine-grained control over environment logic.

* Useful for optimizing the environment for performance
* Implementing complex logic that is difficult to decompose into separate components.



Actuators

An articulated system comprises of actuated joints (DOF).

In physical system actuation typically through

* electric /hydraulic motors (active components)
* Springs (passive components)

These components can introduce non-linear characteristics.

In simulation, the joints are either position, velocity, or torque-controlled.

For position and velocity control, the physics engine internally implements a spring-damp (PD) controller which computes the torques applied on the actuated joints.

In torque-control, commands are set directly as the joint efforts.

These methods mimic an ideal behavior of the joint mechanism, but do not truly model how the drives work in the physical world.

* Isaac Lab provides a mechanism to inject external models to compute the joint commands that would represent the physic robot’s behavior.

There are two main categories of actual model that are supported;

1. Implicit: corresponds to the ideal simulation mechanism (provided by physics engine).
2. Explicit: corresponds to external drive models (implemented by user).

Actuator models themselves are computation blocks that take as inputs the disiered joint commands and output the joint commands to apply into the simulator.

* Do not contain any knowledge about the joint they are acting on

Motion Generators

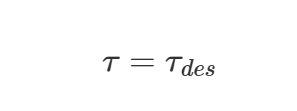
Robot tasks are typically defined in task-space in terms of desired end-effector trajectory, while control actions are executed in the joint-space.

* Successful execution of interaction tasks using motion control requires an accurate model of both robot manipulator as well as its envoirment.
* Planning errors caused by mismatch can be overcome by introducing a complaint behavior during interaction.

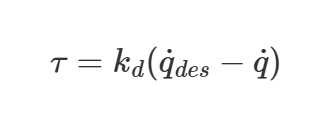
Compliance is achievable through robot’s structure, however Isaac Lab implements controller designs that focus on active interaction control. These are broadly categorized into:

1. Impedance Control: indirect control method where motion deviations caused during interaction relates to contact force as a mass-spring -damper system with adjustable parameters (stiffness and damping).
2. Hybrid force/motion control: active control method which controls motion and force along unconstrained and constrained task directions respectively.

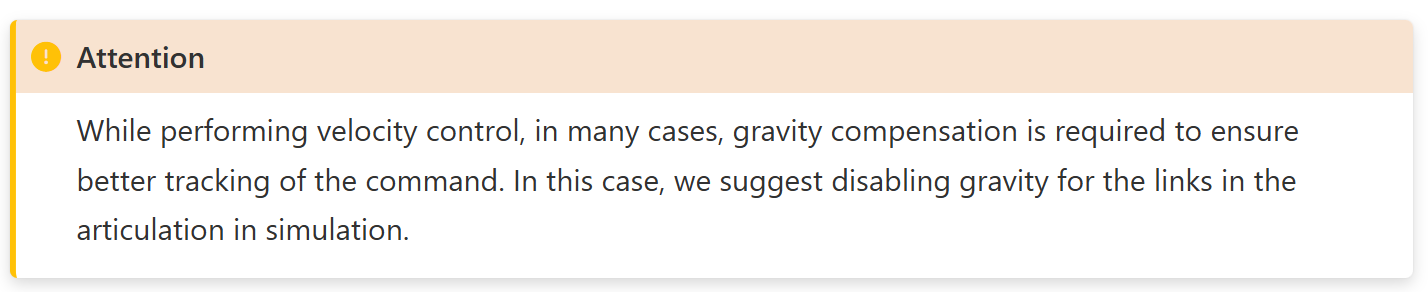
In torque control mode, the input actions are directly set as feed-forward joint torque commands, i.e at every time step,



In velocity control mode, a proportional control law is required to reduce the error between the current and desired joint velocities. Based on input actions, the joint torques commands are computed as:



Where k\_d are the gains parsed from the configuration.



Hydra Configuration System

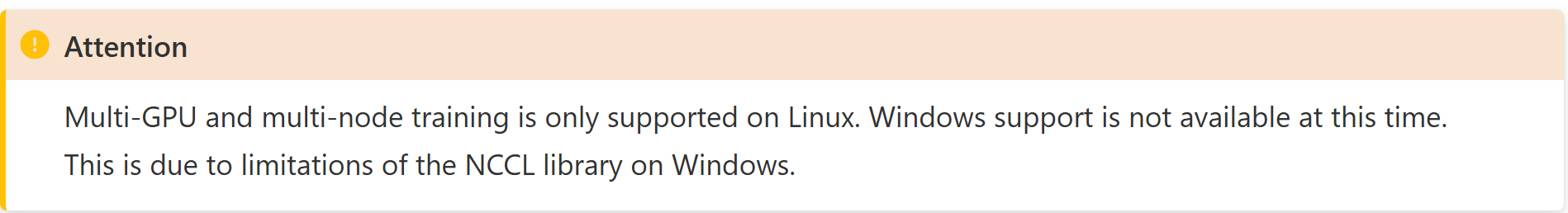
Isaac Lab supports the Hydra configuration system to modify the task’s configuration using command line argument.

* Can be useful to automate experiments and perform hyperparameter tuning.
* E.x python scripts/reinforcement\_learning/sb3/train.py --task=Isaac-Cartpole-v0 --headless env.actions.joint\_effort.scale=10.0 agent.seed=2024

Multi-GPU and Multi-Node Training

Isaac Lab supports multi-GPU and mutli-node reinforcement learning.

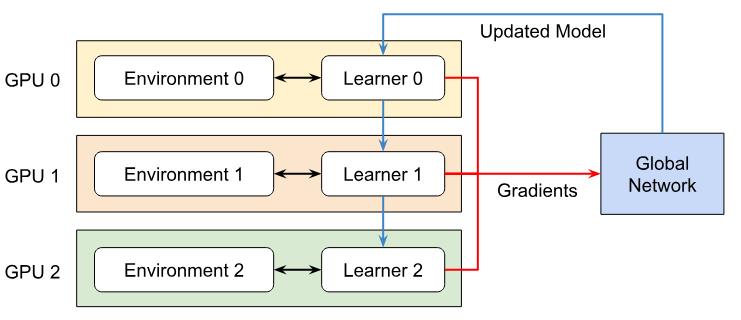
* Currently only available for RL-Games, RSL-RL, and skrl libraries



It may be desirable to scale up training across multi GPUs. This is possible in Isaac Lab through use of the PyTorch distributed framework or the JAX distributed module respectivley.

The torch.distributed() API is used to launch multiple process of training, where the # of processes must be less than or equal to the number of GPUs available.

* Each process runs on a dedicated GPU and launches its own instance of Isaac Sim and Isaac Lab enviorment.
* Each process collects its own rollouts during the raining process and has its own copy of the policy network.
* Gradients are aggregated across the process and broadcast black to the process at the end of the epoch.



To train across multiple nodes/machines, it is required to launch an individual process on each node.

Ray Job Dispatch and Tuning

Isaac Lab supports [Ray](https://docs.ray.io/en/latest/index.html) for streamlining, dispatching multiple training jobs (in parallel and in series), and hyperparameter tuning, both on local and remote configurations.

Reproducibility and Determinism

Given the same hardware and Isaac Sim (PhysX) version, the simulation produces identical results for scenes w/ rigid bodies and articulations.

* However simulation results can vary across different hardware configurations due to floating point precision and rounding errors.
* PhysX does not guarantee determinism for any scene with non-rigid bodies.

Isaac Lab provides a deterministic simulation that ensures consistent simulation results across different runs; at construction of the environment, the random seed is set to a fixed value using the set\_seed() method.

LLM Generated Reward Functions

NVIDIA’s Eureka!, has resulted in a pipeline for generating and tuning Reinforcement Learning (RL) reward functions using an LLM.

***Robot Operating System (ROS)***

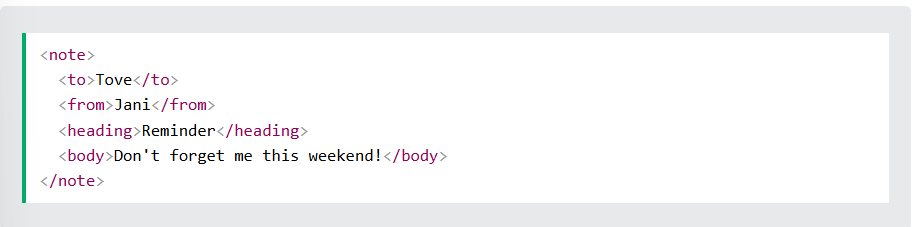
***XML File***

eXtensibile Markup Language

* Markup language much like HTML
* Designed to store and transport data
* Designed to be self-descriptive

XML Does Not DO Anything

e.x



The XML does not DO anything. XML is just information wrapped in tags.

The Difference Between XML and HTML

XML and HTML were designed w/ different goals

* XML was designed to carry data - w/ focus on what data is
* HTML was designed to display data - with focus on how data looks
* XML tags are not predefined like HTML tags are

XML simplifies data sharing, data transport, platform changes, and data availability.

***XACRO File***

Xacro is an XML macro language.

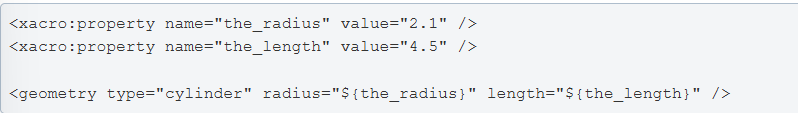
* w/ xacro, you can construct shorter and more readable XML files by using macros that expand to larger XML expressions.
* Most useful when working with large XML documents such as robot descriptions.

Property and Property Blocks

Properties are named values that can be inserted anywhere into the XML document.

Property blocks are named snippets of of XML that can be inserted anywhere that XML is allowed.

Both use the property tag to define values.



Math expressions

Within dollared-braces (${}), you can also write simple math expressions.



Cond

***Unified Robotics Description Format (URDF)***

The Unified Robot Description Format (URDF) is an XML specification to describe a robot.

* The main limitation at this point is that only tree structures can be represented, ruling out all parallel robots.
* Specification assumes the robot consists of rigid links connected by joints; flexible elements are not supported.

The description of a robot consists of a set of link elements, and a set of joint elements connecting the links togethter.

<robot> element

The root element in a robot description file must be a robot, with all other elements encapsulated within.

<sensor> element

The sensor element describes basic properties of a visual sensor (i.e camera/ray sensor)

<link> element

The link element describes a rigid body w/ an interior, visual features, and collision properties.

<transmission> element

The transmission element is an extension to the URDF robot description model that is used to describe the relationship between an actuator and a joint.

* Model concepts such as gear ratios and parallel linkages

<Joint> element

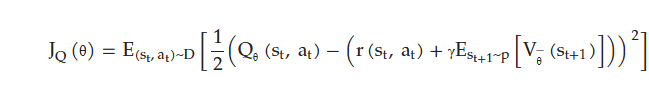
The joint element describes the kinematics and dynamics of the joint and also specifies the safety limits of the joint.

***USD File Format***

Bellman equation, is a technique in dynamic programming which breaks an optimization problem into a sequence of simpler subproblems.

* The “value” of a decision problem at a certain point in time is written in terms of the payoff from some initial choices and the value of the remaining decision problem that results from those choices.

The objective function of the critic network is defined as the MSE between the Q value approximated using the critic network and the target Q value, and is formulated in the following form



The KL Divergence (KLD) between the policy entropy value and Q value is used as the objective function of the actor network.

