Isaac Lab: A GPU-Accelerated Simulation Framework for Multi-Modal Robot Learning

NVIDIA[1]

Abstract

We present Isaac Lab, the natural successor to Isaac Gym, which extends the paradigm of GPU-native robotics simulation into the era of large-scale multi-modal learning. Isaac Lab combines high-fidelity GPU parallel physics, photorealistic rendering, and a modular, composable architecture for designing environments and training robot policies. Beyond physics and rendering, the framework integrates actuator models, multi-frequency sensor simulation, data collection pipelines, and domain randomization tools, unifying best practices for reinforcement and imitation learning at scale within a single extensible platform. We highlight its application to a diverse set of challenges, including whole-body control, cross-embodiment mobility, contact-rich and dexterous manipulation, and the integration of human demonstrations for skill acquisition. Finally, we discuss upcoming integration with the differentiable, GPU-accelerated Newton physics engine, which promises new opportunities for scalable, data-efficient, and gradient-based approaches to robot learning. We believe Isaac Lab's combination of advanced simulation capabilities, rich sensing, and data-center scale execution will help unlock the next generation of breakthroughs in robotics research.

Isaac Lab is Open Source. Code and documentation are available here: https://github.com/isaac-sim/IsaacLab

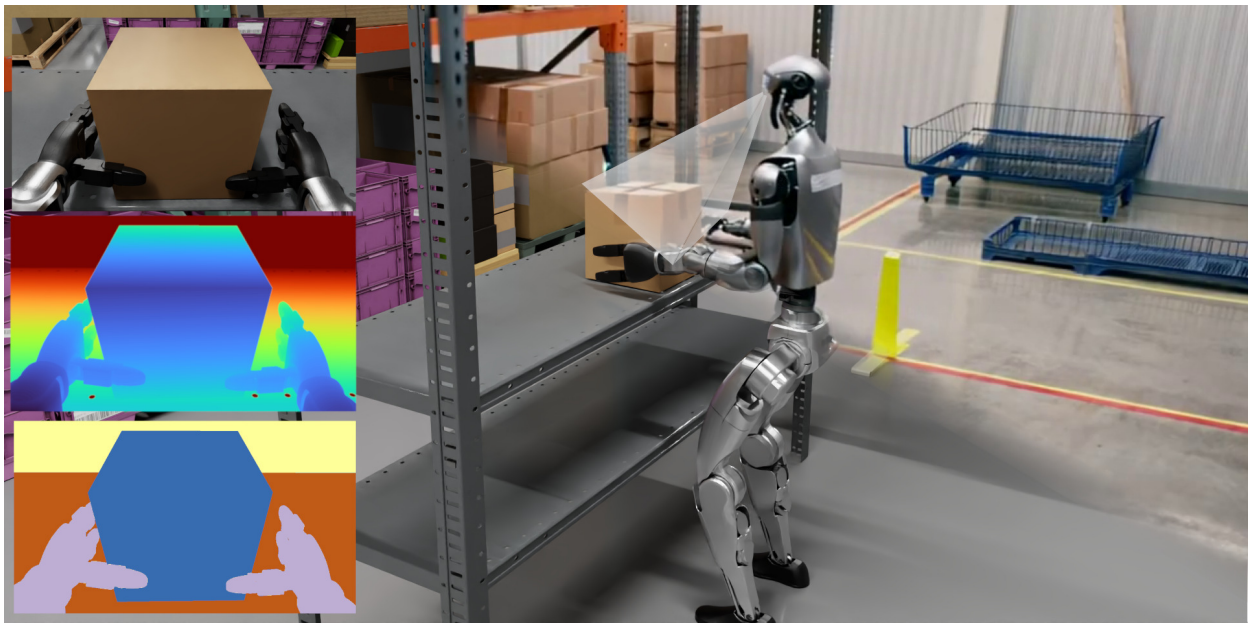


Figure 1: Isaac Lab supports diverse robotic applications with exteroceptive observation inputs. It provides a user-friendly API for experimentation and includes features to facilitate sim-to-real transfer. The framework also supports multiple learning paradigms, including reinforcement learning and imitation learning.

[1]A detailed list of contributors and acknowledgments can be found in App. A of this paper.

1. Introduction

The development of robust and intelligent robotic systems increasingly depends on the ability to evaluate their performance in complex real-world environments. While the physical world remains the definitive testbed, the acquisition of physical interaction data with robots is expensive, time-consuming, and often necessitates specialized instrumentation. These limitations are especially acute in rare but safety-critical situations. Events such as high-speed collisions, hardware malfunctions, or navigation in unpredictable human environments are difficult to reproduce and pose significant risks to equipment and human safety. Moreover, real-world data collection is inherently biased toward normative conditions, leaving robotic systems insufficiently prepared for atypical or extreme situations. Simulation provides a compelling alternative by offering controlled, reproducible, and risk-free environments in which robotic systems can be developed and evaluated rigorously. High-fidelity simulators extend these advantages by modeling physics, sensors, and environmental complexity with greater realism. This enables large-scale data collection, systematic stress testing, and the development of algorithms that transfer more effectively to real-world systems.

The emergence of GPU-accelerated, physics-based simulators has democratized robotics research by making scalable training feasible on consumer-grade hardware. Traditional CPU-based simulators (Coumans and Bai, 2016–2023; Lee et al., 2018; Todorov et al., 2012) often struggle to meet the computational demands of high-fidelity physics, complex sensor models, and large-scale parallelization. Scaling such simulations typically requires clusters with high-core CPUs, which are costly and less widely available. In contrast, modern GPU-based simulators (Makoviychuk et al., 2021; Tao et al., 2025; Zakka et al., 2025) exploit massive parallelism to efficiently execute a larger number of concurrent environments, dramatically accelerating the training of complex robotic policies. By running the agent-environment interaction loop entirely on the GPU, these frameworks avoid inefficiencies associated with frequent CPU-GPU data transfers. This approach is particularly advantageous for on-policy Reinforcement Learning (RL), which benefits from large batch sizes during training. Beyond improving efficiency, GPU-based simulation lowers the entry barrier for researchers, allowing training and development of sophisticated robotic systems without access to specialized supercomputer resources.

A landmark contribution in this space came from NVIDIA Isaac Gym (Makoviychuk et al., 2021), which demonstrated for the first time that end-to-end RL for complex robotic tasks could be performed entirely on a single GPU. Isaac Gym uses NVIDIA PhysX, a GPU-accelerated physics engine capable of simulating high-fidelity rigid body dynamics at massive scales. By exposing the physics simulation results directly as PyTorch tensors, Isaac Gym provides a GPU-native pipeline that reduces training times from days to hours. Since its release, the framework has been used successfully in a large number of projects, such as locomotion (Agarwal et al., 2023; Rudin et al., 2022), whole-body control (Fu et al., 2023; He et al., 2024), in-hand manipulation (Allshire et al., 2022; Handa et al., 2023), dexterous grasping (Lum et al., 2024; Wang et al., 2023), and industrial assembly (Narang et al., 2022). In doing so, it has established a new standard for scalable, high-performance robotic simulation and laid the foundation for subsequent GPU-based simulation frameworks.

Isaac Lab is the natural successor of Isaac Gym, carrying forward the paradigm of GPU-native robotics simulation into the era of large-scale multi-modal learning. Built on NVIDIA Isaac Sim, Isaac Lab combines RTX rendering for photorealistic, scalable visuals with PhysX for high-fidelity physics simulation. It uses Universal Scene Description (USD) as the core data layer for structured world authoring, simplifying the design of complex sensor-rich scenes. The physics engine adds numerous enhancements over the one in Isaac Gym, such as filtered contact reporting, mimic joint systems, closed-loop kinematic chains, deformable objects (cloth and soft bodies), and coupled solvers for rigid and deformable bodies. High-throughput GPU-accelerated rendering supports large-scale generation of RGB, depth, and segmentation data, facilitating policy training and sim-to-real transfer using exteroceptive information. Together, these capabilities scale efficiently across multi-GPU and multi-node setups.

Based on the design from Mittal et al. (2023), Isaac Lab provides users with more than just the outputs of

the underlying physics and rendering engines. Since the adoption of Isaac Gym, various paradigms have emerged to facilitate robot learning and sim-to-real transfer. Often, these practices have been independently re-implemented across projects, leading to significant duplication of effort. Isaac Lab addresses this challenge by unifying these practices within a modular and extensible framework for robotics research. Key features include integration of non-linear actuator models, multi-frequency sensor simulation, interfaces for low-level controllers, and tools for procedural environment generation and domain randomization. The framework also supports custom sensors beyond rendering, such as raycast-based LiDAR, height scan, and visuo-tactile sensors. At its core, Isaac Lab designs a *manager-based* API that organizes environment design into reusable and composable components, allowing consistent workflows across diverse research projects. However, the use of this API is optional, and researchers can also structure their simulation environments with simple single-script setups if preferred. In addition, Isaac Lab offers data collection pipelines to record expert demonstrations, access to a wide range of RL libraries, and a large suite of robotic environments (shown in Figure 12).

This technical report presents the core capabilities and features of Isaac Lab, providing insight into its design decisions and implementation. It examines the core technologies underlying Isaac Lab (USD for scene authoring, PhysX for high-fidelity physics, and RTX rendering for photorealistic rendering), highlighting their importance for advancing simulation. Building on this foundation, the report details Isaac Lab's unique enhancements, including custom sensors, actuator models, motion generation pipelines, teleoperation devices, and the environment design framework. It further describes the various learning workflows supported by Isaac Lab and showcases the wide range of robotic applications, from locomotion and navigation to contact-rich manipulation, each benefiting from the framework's capabilities and modularity. Finally, the report concludes with future directions, including the Newton engine (Newton Contributors), and a roadmap for Isaac Lab as a platform for next-generation robotics research.

Key contributions of Isaac Lab

- **Modular and scalable framework:** Built on NVIDIA Omniverse, enabling high-fidelity, GPU-accelerated simulation for complex robots and tasks.
- **Advanced sensor simulation:** Supports tiled RTX rendering, Warp-based custom sensors, and physics-based data for rich observation spaces.
- **Seamless teleoperation and data collection:** Integrates spacemouse, VR headsets, and other devices for large-scale demonstration capture.
- **Extensive environment suite:** Provides diverse, ready-to-use environments for reinforcement learning, imitation learning, and sim-to-real research.

2. Core Simulation Infrastructure

2.1. USD for Robotics

The robotics simulation ecosystem remains highly fragmented, with developers managing diverse data sources that include CAD models, kinematic and dynamic descriptions, sensor parameters, and more. Additionally, simulators often rely on multiple specialized attributes for physics and rendering, which further increases the complexity of asset data. Existing tools illustrate these challenges. Gazebo (Koenig and Howard, 2004) utilizes the flat and inflexible XML-based Simulation Description Format (SDF). This format limits the ability to generate large, photorealistic worlds and collaborate on scene variations. It only defines descriptions at the component level, making complex subsystem behaviors, such as closed-loop kinematic chains, difficult to represent. Other widely used XML-based robotics formats, such as URDF and MJCF, face similar limitations. Modern game engines such as Unity and Unreal are increasingly used in robotics. They provide photorealistic rendering, integrated physics, and better scene authoring capabilities. However, these tools were originally designed for entertainment and use paradigms that differ from traditional robotics workflows. AirSim (Shah

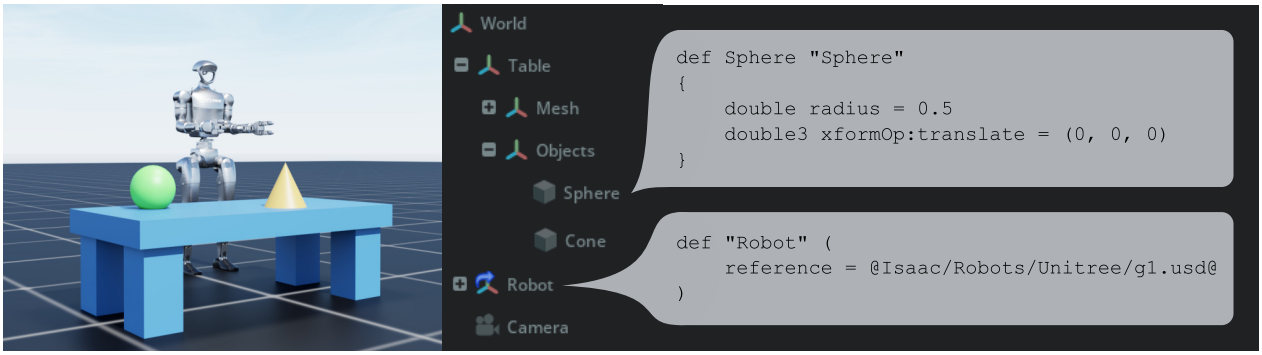


Figure 2: Isaac Lab uses OpenUSD to unify complex robotics simulations. Hierarchical scene graphs organize robots, objects, and sensors, while simulation-specific schemas define visual and collider elements, physical properties, semantic IDs, and sensor configurations. References and instancing support scalable, parallelized simulation of large, complex scenes required for robot learning.

et al., 2017) attempts to lower the barrier to entry for robotics researchers, but its approach of compiling simulations into a monolithic "game" introduces significant limitations: Any modifications to the simulation often require returning to the underlying game engine. These limitations highlight the need for a flexible unifying format that can integrate diverse data types, support complex subsystem modeling, and facilitate collaborative workflows for digital content creation.

Addressing these challenges, Open Universal Scene Description (OpenUSD) (Pixar Animation Studios, 2016) is an open-source format for robust and scalable authoring of complex 3D scenes composed of numerous elements. It provides a comprehensive set of tools and Python/C++ APIs to define, organize, and edit 3D data. USD represents a 3D scene as a hierarchical scene graph (or *stage*), with data arranged in namespaces of primitives (or *prims*). Each prim can contain child prims and have attributes or properties, allowing transformations and properties to be inherited from parent prims. USD's *schema*-based system helps define structured properties for geometry, materials, physics, and more, while *layering* and non-destructive composition facilitate multiple collaborators working on a scene simultaneously without overwriting changes. Additionally, *references* and *instancing* let users combine multiple USD files and reuse repeated elements efficiently. These features make USD a flexible, scalable, and collaborative foundation that naturally ties into robotics simulation workflows. An example of a USD scene leveraging referencing is demonstrated in Figure 2.

The Alliance for OpenUSD (AOUSD) is an open, non-profit organization that aims to advance 3D data interoperability through OpenUSD. Within this effort, a key development for robotics is the USDPhysics schema. The USDPhysics schema extends OpenUSD with a standardized way to describe physical properties such as rigid bodies, collisions, joints, and materials. By providing a common representation for physics simulation, it enables robotics and simulation tools to share and interpret scenes consistently across different engines and workflows. While these definitions are designed to generalize across multiple simulation backends, OpenUSD also allows a straightforward extension with engine-specific schemas. For example, the PhysxSchema provides parameters used in NVIDIA PhysX, while the MjcPhysics schema, currently under development in collaboration between NVIDIA and Google DeepMind, extends USD for MuJoCo.

Beyond physics, OpenUSD also provides complementary schemas that enrich how scenes can be represented and exchanged across domains. The Semantics schema annotates prims with categorical labels, enabling tasks in perception and learning. Camera prims allow the description of virtual sensors directly within a scene, ensuring that viewpoints and sensor models are preserved consistently across tools. Similarly, Material schemas capture surface properties like textures, reflectance, and friction in a standardized way, bridging the needs of both rendering and physics. These schemas unify geometry, dynamics, semantics, sensing, and appearance within a single scene description. This integrated representation overcomes the limitations of existing robotics formats

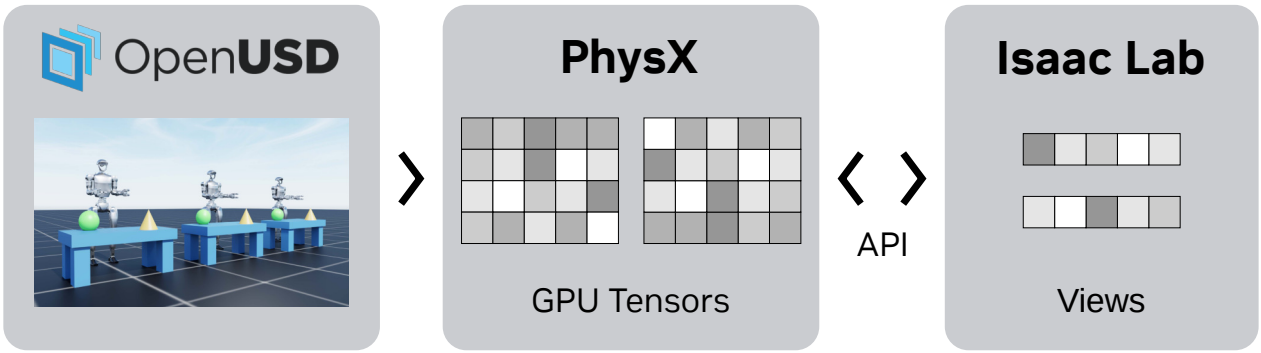


Figure 3: Integration of USD with PhysX in OmniPhysics. Scene data from USD is parsed into PhysX, which constructs GPU tensors representing internal simulation states. Unlike IsaacGym, where users accessed these states directly, OmniPhysics manages them through the View API for improved usability.

such as MJCF (physics-focused, limited scene richness) and URDF (kinematics/dynamics with Gazebo-specific tags for sensors).

In Isaac Lab, USD serves as the foundation for robotics simulation. While USD's domain-agnostic design provides flexibility, it necessitates domain-specific conventions to maintain structured scene graphs and develop reliable APIs. Robotics uses meters and a Z-up convention in the USD stage, where other domains, such as computer graphics, follow loosely defined conventions. Prims must follow specific hierarchy rules. For instance, two prims with rigid body properties cannot be nested under each other. Additional considerations, such as instancing, are necessary to ensure efficient stage creation for large-scale simulation scenes. To address these requirements, Isaac Lab provides USD converters for widely used formats, including URDF, MJCF, and meshes (e.g., OBJ, DAE). It also offers high-level wrappers around USD APIs, allowing users to configure attributes and create prims via simple configuration objects. These wrappers handle stage-modification nuances, such as cloning prims efficiently for large-scale scenes and altering USD properties for domain randomization. Additionally, thanks to ongoing SimReady asset creation efforts, Isaac Lab includes various photorealistic and physically accurate robot and object assets that are ready for use in simulation.

2.2. Physics Simulation

The **NVIDIA PhysX SDK** is an open-source, multi-physics simulation engine designed to meet the demanding needs for robotics and industrial applications. With the release of PhysX 5, the engine incorporates capabilities from the former NVIDIA Flex library, including Finite Element Method (FEM)–based soft-body dynamics, as well as Position-Based Dynamics (PBD) for liquids and inflatable objects. It supports a wide range of simulation types, from rigid and articulated bodies to deformable bodies such as cloth, fluids, and soft bodies. These simulation objects can interact with each other through two-way coupling between specialized solvers. As an example, the FEM cloth solver exchanges impulses with the Featherstone-based articulation solver to support efficient and accurate mixed-physics interactions. Building on this foundation, PhysX has continued to evolve towards robotics use cases, collaborating with robotics engineers and researchers to develop domain-specific features, including emulation of force–torque load cells, advanced actuator and friction models for an articulation's joints, and collision handling improvements such as Signed Distance Field (SDF) methods, which are particularly valuable for high-precision, non-convex geometries encountered in robotic assembly tasks.

PhysX can run on both CPU and GPU, providing flexibility for different simulation scenarios. For high-performance, large-scale robot learning workflows, GPU execution delivers the parallelism and throughput necessary to scale efficiently. PhysX's Direct-GPU API provides direct read and write access to simulation state and control data in GPU memory. The resulting CUDA tensors can then be processed efficiently using user-defined GPU kernels for downstream applications, such as computing observations for robotics learning

Algorithm 1 OmniPhysics Workflow in Isaac Lab

```
Prepare USD stage                                   ▷ All modifications possible through USD APIs
Start simulation                                    ▷ Parse USD stage and initialize PhysX objects
Create physics simulation views                     ▷ Define views via USD prim path patterns
Access simulation state using Tensor API views      ▷ Read/write simulation state efficiently
if direct GPU API disabled then
    Use standard USD APIs with OmniPhysics monitoring   ▷ Fallback to standard USD workflow
else
    Access PhysX exclusively via Tensor API views   ▷ USD read/write suppressed for performance
end if
```

workflows. This end-to-end GPU pipeline eliminates the performance bottlenecks from CPU–GPU data transfers seen in traditional simulators. The benefit of this GPU-first simulation approach was first demonstrated for RL in Isaac Gym (Makoviychuk et al., 2021), where training times for complex robotic tasks were reduced from days to hours. It is important to note that only the simulation state and control can currently be accessed directly on the GPU device. Simulation parameters, such as friction coefficients, rigid-body masses, and joint properties, must still be set via the PhysX CPU APIs due to current design constraints.

NVIDIA Omniverse Physics (OmniPhysics) serves as the integration layer for the PhysX simulation engine within NVIDIA Omniverse. It extends the OpenUSD framework by introducing user-defined data types under PhysxSchema, which enables the representation and control of physics simulations. OmniPhysics parses these attributes from USD, maps them into PhysX, runs the simulation, and writes the simulation output back to USD. It also supports state updates by monitoring changes in USD and updating the active simulation accordingly. In Isaac Lab, where high-performance and large-scale simulation scenes are common, the USD read–write cycle during simulation is often a performance bottleneck and is therefore bypassed. Instead, the simulation data is accessed through OmniPhysics Tensor APIs, which internally rely on PhysX Direct GPU APIs. For workflows that require rendering, such as vision-in-the-loop training, OmniPhysics also maintains efficient synchronization between the simulation state and the renderer.

Algorithm 1 summarizes the OmniPhysics workflow in Isaac Lab. The user sets up the USD stage through USD APIs. Isaac Lab provides configurable interfaces and APIs to facilitate programmatic creation of prims. These interfaces support spawning prims of different types, modifying their USD attributes, and efficiently duplicating a prototype scene to multiple environment instances on the USD stage. Once the stage is ready for simulation and the simulation is *played*, OmniPhysics parses the USD stage to create a set of PhysX simulation objects (*e.g.* robots, rigid objects, or static colliders) that mirror the USD scene. To efficiently scale a prototype environment to thousands of instances, OmniPhysics provides replication APIs that duplicate the prototype's PhysX simulation objects across many parallel training environments. During this replication process, a mapping is also established between the PhysX objects and their USD prim paths, which is subsequently used to construct **Tensor API** views.

The OmniPhysics **Tensor API** presents simulation data as batched, device-resident arrays organized into views, as shown in Figure 3. Instead of operating each physics actor individually, users can interact with a collection of actors through a `SimulationView`, which links the chosen tensor framework (*i.e.* NumPy, PyTorch, or NVIDIA Warp) to the physics simulation backend. A simulation view can then be used to create specialized views for different types of physics objects, including rigid bodies (`RigidBodyView`) and articulated systems (`ArticulationView`). Views are defined using USD prim path pattern matching. For instance, if a prototype scene contains a robot prim at `"/World/envs/env_0/Robot"`, and this scene is cloned N times, then all robots across the N environments can be collected into a single articulation view using the pattern `"/World/envs/*/Robot"`. The resulting view exposes simulation data as arrays with the first dimension corresponding to the number of robots N.

Figure 4: Photo-realistic rendering in Isaac Lab using the Omniverse RTX renderer, demonstrating high-quality ray tracing with complex physically-based materials authored using NVIDIA's MDL. The rendering showcases realistic effects such as reflections and refractions, resulting in visually rich and high-quality scenes.

In Isaac Lab, the Tensor API-based view classes provide efficient access to underlying simulation data for user-facing asset classes, such as `Articulation` and `RigidObject`, as well as physics-based sensor classes like `ContactSensor` and `IMU`. These views enable GPU-accelerated batched operations and serve as the foundation for additional asset- and sensor-level functionalities. For example, the `Articulation` class extends the OmniPhysics `ArticulationView` to support user-defined actuator models, store default physical properties for domain randomization, and implement efficient buffering mechanisms to minimize redundant read-write operations. These API-level features are described in detail in Section 3.

2.3. Rendering

The **Omniverse RTX** renderer simulates RGB cameras and synthetic ground truth sensors, including depth, surface normals, and semantic segmentation, using physically based ray tracing. It supports real-time and offline path tracing, direct lighting, and denoising on top of hardware-accelerated ray tracing. To improve rendering efficiency, the RTX renderer leverages Deep Learning Super Sampling (DLSS), which upscales images from a lower internal resolution using high-quality temporal super-resolution, making it well-suited for high-resolution perception tasks. The RTX renderer allows users to selectively disable DLSS in very low-resolution scenarios where there is insufficient visual data for the upscaler to make use of. DLSS only affects RGB outputs; other ground truth data, such as depth and segmentation, are always rendered at their native resolution.

An important aspect of high-quality rendering is applying the right materials to USD prims. Material schemas for USD prims are authored using NVIDIA's Material Definition Language (MDL). MDL provides a flexible and powerful framework for describing complex, physically-based materials (e.g., OmniPBR, OmniGlass) with realistic details such as reflections, refractions, and surface patterns. These materials enable high-quality visual results in rendered scenes, as illustrated in Figure 4. Additionally, to support semantic segmentation, USD prims can also be annotated with semantic information, such as object class or instance identifiers. This metadata allows per-pixel instance and class labels to be rendered alongside RGB images, which is fully compatible with tiled rendering in large-scale, multi-camera setups.

To support thousands of cameras in parallel simulation environments, Isaac Lab uses the tiled rendering pipeline of the RTX renderer. This method batches multiple cameras into a single render pass by spatially arranging them as tiles within the GPU framebuffer. Each camera preserves its own intrinsics and pose, and the deterministic tiling layout enables efficient reconstruction of per-environment tensors without incurring costly host–device

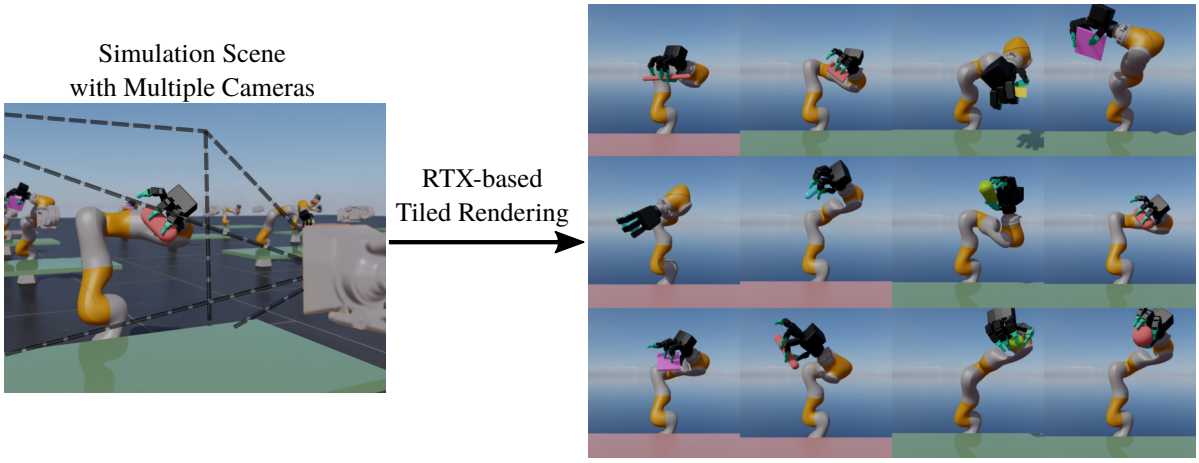


Figure 5: Tiled rendering of multiple simulated environments. Each environment has a separate camera, and their outputs are spatially tiled into a single GPU frame-buffer. The deterministic layout allows efficient reconstruction of per-environment observations without costly host–device transfers.

transfers. This design is essential for large-scale data generation workloads, such as vision-in-the-loop RL, as it scales sensor throughput linearly with GPU resources, minimizes latency, and synchronizes observations across environments for policy training (see Figure 5). Although active sensors such as LiDARs and radars are already supported by the Omniverse RTX renderer, their integration with tiled rendering is forthcoming. As an alternative, the `RayCaster` sensor uses NVIDIA Warp operations for ray-casting, as described in Section 3.2.3.

The RTX renderer can also perform 3D Gaussian rendering via Omniverse NuRec, supporting realistic reconstructions of the real-world scenes using approaches such as 3D Gaussian Splats (3DGS) by Kerbl et al. (2023) and 3D Gaussian Unscented Transforms (3DGUT) by Wu et al. (2025). These Gaussian primitives integrate seamlessly with ray-traced geometry in the RTX renderer, allowing robots and synthetic objects to operate within photorealistic, reconstructed environments. This approach improves visual realism and facilitates policy transfer without requiring hand-crafted assets. Figure 6 shows an early example of this integration from Liu et al. (2025).

Figure 6: 3D Gaussian rendering combined with mesh rendering, with shadows from the mesh affecting the Gaussian scene.

Isaac Lab uses the RTX renderer to implement the `TiledCamera` sensor class, which batches the rendering output for learning pipelines. It supports specifying and retrieving camera poses in multiple conventions, such as those used in ROS and computer graphics. The class also ensures that sensor data is updated at a specified frequency to match real-world sensors. Since the RTX renderer settings affect performance, Isaac Lab provides users with presets that trade off between quality and speed, and exposes rendering parameters through configuration objects for further customization. Furthermore, Isaac Lab employs Replicator API to randomize MDL materials on prims and scene lighting. Together, these features enable scalable, high-fidelity rendering with multi-camera observations while supporting realistic variations in textures and illumination.

3. Isaac Lab Design and Features

Building on the core technologies introduced in Section 2, Isaac Lab brings state-of-the-art simulation capabilities to robot learning researchers. While these general-purpose technologies expose numerous low-level states and properties, this flexibility can create a steep learning curve that alienates non-expert users. Isaac Lab

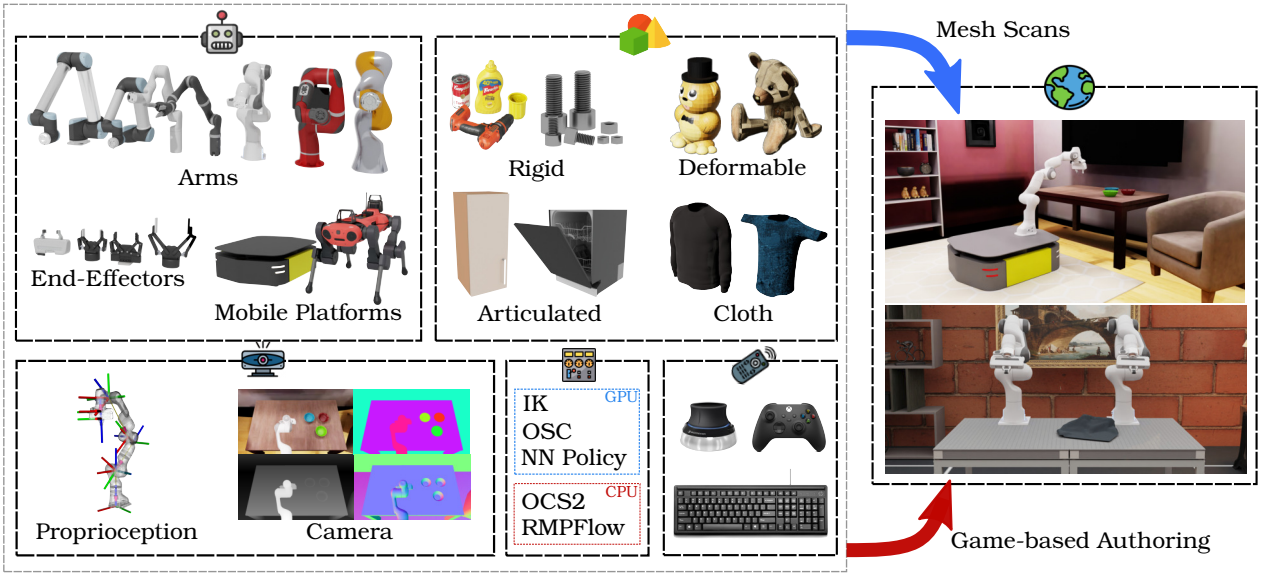


Figure 7: Overview of Isaac Lab features. These include support for diverse asset types (articulated robots, rigid and deformable objects), sensor modalities (proprioception, RGB/depth images, height scans), controllers (IK, RMPFlow), and teleoperation devices (keyboard, spacemouse). Additionally, the environment design interface allows adding USD scenes created via custom mesh scans, game-based authoring, or programmatic generation.

aims to lower this entry barrier through a modular, integrated framework that leverages the latest advances in physics and rendering. Its interfaces are specialized for robot learning, simplifying environment design, and facilitating deployment to physical robots. The framework adopts a bottom-up design philosophy, starting with modeling complex actuator dynamics, asynchronous sensing and control, realistic sensor noise, and environmental uncertainties, and building upward to high-level robot learning interfaces and task abstractions (Figure 7). Unified abstractions for different robot and object types, support for actuator and noise models to aid sim-to-real transfer, and integration with peripherals for data collection further streamline robotics research and development.

3.1. Assets

Assets encompass the physical elements within the environment, including terrains, robots, and objects, which can be represented as rigid bodies, articulated systems, and deformable objects. The rigid object and articulation framework provides a robust foundation for simulating both individual rigid bodies and articulated structures such as robotic arms, mobile manipulators, and floating-base systems. Rigid objects are modeled as non-deformable entities, represented by their collision geometry and inertial properties, ensuring stable and efficient simulation of contact dynamics. Rigid objects can be created through various methods, including importing custom USD files or converting common mesh formats such as OBJ, STL, and FBX directly into USD.

Articulated systems extend the rigid representation by connecting rigid links through joints, supporting a wide variety of configurations, including fixed-base manipulators and free-floating robots. Articulations can be incorporated into environments through USD representations or by utilizing built-in converters for URDF and MJCF assets. The framework offers detailed access to joint states (positions, velocities, torques) and supports advanced control modes such as position, velocity, and torque control. Key features include updating buffers lazily for improved simulation performance, joint control actuator models, and compatibility with kinematic-only or fully dynamic operations. This design makes it suitable for high-fidelity robotics research, including motion planning, robot learning, control benchmarking, and real-world hardware integration.

The deformable object class provides a sophisticated framework for simulating soft and deformable materials,

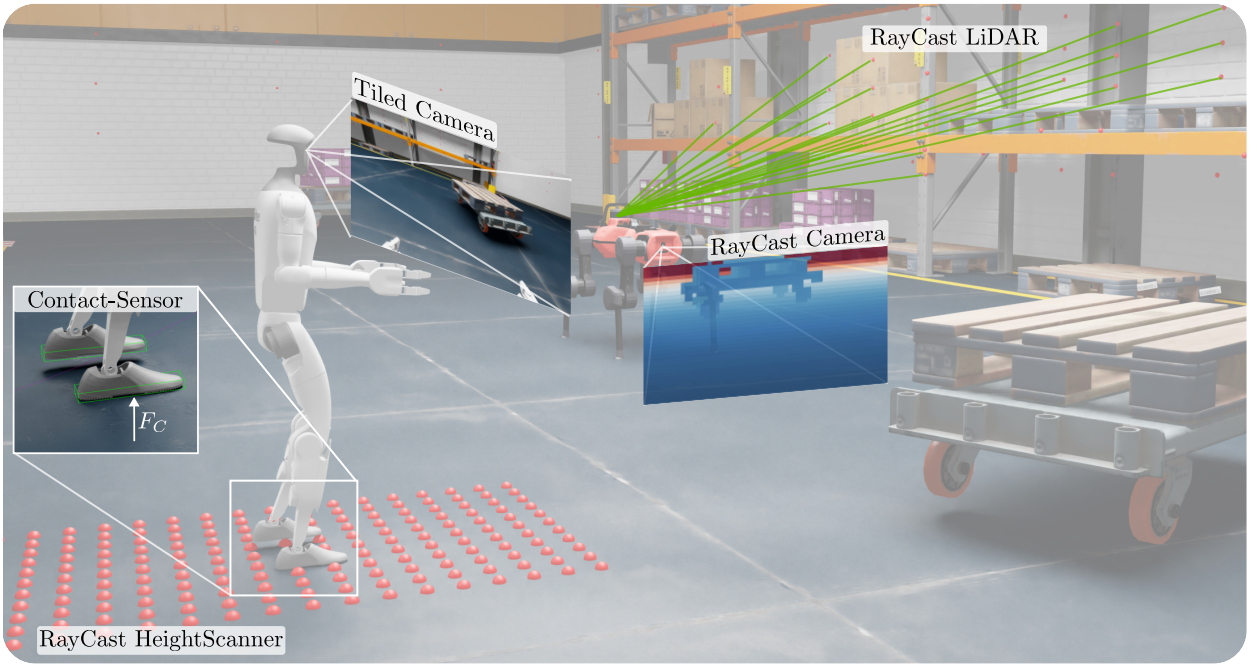


Figure 8: Overview of the Isaac Lab sensor suite, including physics-based sensors (IMUs, frame transformers, contact sensors), parallelized camera implementations, and warp-based raycasting for geometric perception. The figure shows example outputs in a photorealistic scene.

including deformable tissues for medical applications, food items handled by service robots in home environments, and ropes and cables for industrial settings. It leverages the finite element method, discretizing objects into tetrahedral meshes composed of two key components: a simulation mesh that accurately models deformation physics, and a collision mesh dedicated to contact detection. Unlike rigid bodies, deformables do not rely on a single transform but update their state through mesh point attributes. One key feature is its support for partial kinematic control, enabling targeted manipulation of specific object nodes while allowing the rest of the object to respond naturally through full physical simulation. It also provides access to deformation gradients, stress tensors, and element-wise rotations, offering deep insights into the object's dynamic behavior. Furthermore, it enables flexible material binding with customizable physical properties, making it ideal for advanced soft-body simulations in robotics.

3.2. Sensors

Sensors are fundamental for robot learning, as they enable agents to perceive both their own physical state and the surrounding environment. Isaac Lab provides three major sensor classes: *physics-based*, *rendering-based*, and *warp-based* sensors, as shown in Figure 8. While physics-based sensors are generally attached to articulations or rigid bodies, rendering- and warp-based sensors can also be configured as external observers (e.g., third-person cameras). All sensors are unified under a common interface, simplifying instantiation, configuration, and runtime integration. Moreover, the sensor implementations are optimized to run in parallelized environment settings as demonstrated in competitive throughput benchmarks in Section 4.1.

3.2.1. Physics-based

While many physical signals can be extracted from the articulation and rigid object data classes, capturing these signals often requires the use of dedicated physical sensors. In particular, Frame Transformers, Inertial Measurement Units (IMUs), and Contact Sensors can be used to collect relevant physical data across these modalities.

The **Frame Transformer** sensor, while not a direct analog to a physical sensor, provides a convenient method for computing the poses of multiple target frames relative to a specified source frame. Notably, it batches transformation computations, significantly reducing computational overhead compared to performing transformations individually. Users can also define offsets for both the source and target frames, enabling precise specification of transformations relative to key reference points — such as a body's center of mass or known positions of motion capture markers.

An **IMU** sensor provides rich state information about moving bodies, whether they are connected in an articulation or on their own as a single rigid body. In the real world, simple IMUs traditionally collect angular velocity via a gyroscope and linear acceleration via an accelerometer. Others can collect orientation information via a magnetometer or estimation of the direction of gravity. Advanced units may come with onboard sensor fusion to provide other components of the inertial state. The IMUs in Isaac Lab provide linear and angular components of pose, velocity, and acceleration. Accelerations are computed via finite difference, which can introduce noise — especially at low physics rates. Following real-world sensors, the IMUs in Isaac Lab provide measurements in the sensor's local frame relative to the world and can be placed on any rigid body with an arbitrary offset. To simulate real-world behavior, modifiers such as observation noise and signal drift (see Section 3.2.5) can be applied. Additionally, a built-in measurement provides the projection of world gravity in the sensor frame.

The **Contact Sensor** captures interactions between rigid bodies with colliders. In its basic form, it reports the net normal contact forces acting on the assigned bodies, which can be used to measure data such as ground reaction forces for locomotion or grasping forces for manipulation. For more fine-grained measurements, users can specify filter bodies to separate forces between different contact pairs. The sensor also offers optional outputs, including temporal information such as contact lengths and the intervals between them, as well as the average point of contact between the body the sensor is attached to and the filter bodies. Since contacts between rigid bodies are inherently discrete and depend on the observation rate, the sensor can maintain a short history of contact events. Exposing this history to policies can provide richer feedback and improve learning performance.

3.2.2. Rendering-based

Rendering-based sensors emulate real-world cameras, which remain among the most informative sensors for robotic perception. Isaac Lab supports both **Pinhole** and **Fisheye** camera models, with outputs including RGB, depth, semantic labels, instance segmentation, surface normals, and motion vectors. Cameras can be configured using standard parameters (e.g., focal length, aperture) or intrinsic matrices. Since orientation conventions vary across frameworks, Isaac Lab supports multiple frame definitions: *world* (x-forward, z-up), *ROS* (z-forward, -y-up), and *OpenGL* (-z-forward, y-up). For photorealistic rendering, Isaac Lab relies on RTX-based pipeline, as described in Section 2.3.

Each camera sensor generates a *render product*, which first uses path-tracing to generate a rough image and then undergoes post-processing steps such as denoising and anti-aliasing to produce high-quality outputs. The **USD-Camera** implementation assigns one render product per environment, enabling highly accurate images that capture illumination, shadows, and reflections. This is well-suited for data generation tasks where photorealism is critical and parallelization is less of a concern. In contrast, learning tasks often require frequent image generation across thousands of environments, where individual render products become a computational bottleneck. To address this, the **Tiled-Camera** offers a parallelized implementation that aggregates all camera data into a single tiled render product. While this approach may slightly reduce photorealistic quality due to post-processing being optimized for single images rather than tiled layouts, it provides significant speedups. Ongoing improvements in tiled post-processing aim to further close this quality gap.

3.2.3. Warp-based

Distance queries are a key component in many robotic tasks and policy learning pipelines. In real systems, such measurements are typically provided by LiDAR sensors or derived from environment height maps (Miki et al., 2022; Rudin et al., 2022). In Isaac Lab, we implement these queries through the **RayCaster** sensor, which leverages NVIDIA Warp (Macklin, 2022) to achieve lightweight and highly parallelized geometric computations on the GPU.

The RayCaster can be configured with flexible raycast patterns to emulate a variety of sensors, including height scanners, solid-state LiDARs, and rotating LiDARs. Rays can be cast against arbitrary dynamic meshes, enabling users to include or exclude specific actors in the scene. Internally, scene geometry is stored as Warp meshes, and their poses are synchronized through PhysX views to support dynamic environments. The ray origins and directions, defined by the chosen pattern, together with the current mesh transforms, are then passed to Warp kernels for efficient GPU-based raycasting. All raycast results are computed at a single time instance, eliminating temporal distortion effects that can occur in real-world rotating sensors where individual rays are captured at different timestamps.

Providing a pinhole raycast pattern enables a **RayCasterCamera**, which integrates with the standard camera interface to provide depth images. RGB and semantic outputs are currently under development. Compared to USD- or Tiled-Cameras, raycast-based sensors prioritize efficiency over photorealism by omitting rendering effects, making them particularly well-suited for large-scale geometric training scenarios.

3.2.4. Visuo-Tactile

Tactile sensor simulation consists of modeling (1) sensor–object contact interactions and (2) transduction of these interactions into measurable signals. These processes are implemented in Isaac Lab for vision-based tactile sensors, where contact interactions are transduced into camera images, as demonstrated in Figure 9 (Akinola et al., 2025). The implementation consists of two GPU-parallelized phases. In the first phase, soft contact dynamics are approximated with a compliant contact model. Deformation is not explicitly modeled, but resistive forces are captured using stiffness and damping parameters, which regulate softness and velocity decay during sensor-object interaction. In the second phase, the contact interaction is used to derive visuotactile RGB images and contact force fields, where the latter characterizes distributed normal and shear forces across the sensor. Depth images are rendered using tiled cameras and mapped to RGB space (Si and Yuan, 2022), while a penalty-based model is applied to compute the force distributions (Xu et al., 2022). Together, this implementation enables substantial performance gains relative to existing simulation frameworks by combining GPU parallelization, approximate but efficient contact modeling, and tiled image rendering.

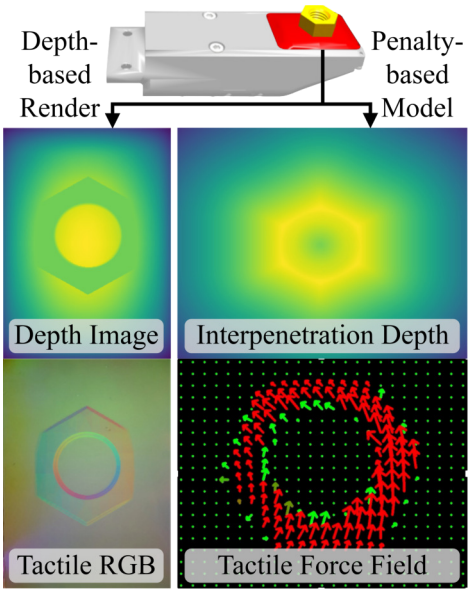


Figure 9: A fast visuo-tactile simulation module that generates tactile images and force fields, based on a soft-contact model between rigid bodies and the sensor.

3.2.5. Simulation Frequencies and Noise Models

Isaac Lab supports flexible sensor update frequencies. This avoids the unrealistic assumption of synchronized sensing and control, and better approximates real-world conditions where sensors operate at different rates and experience communication delays. Standard noise types (e.g., Gaussian, uniform, salt-and-pepper) are included by default as a way to augment observation data for robust training and emulation of real-world systems, improving sim-to-real transfer. Isaac Lab further introduces **Modifiers** to augment observations with a wider variety of perturbations. While they can be used to apply simple operations like noise, bias, or scaling,

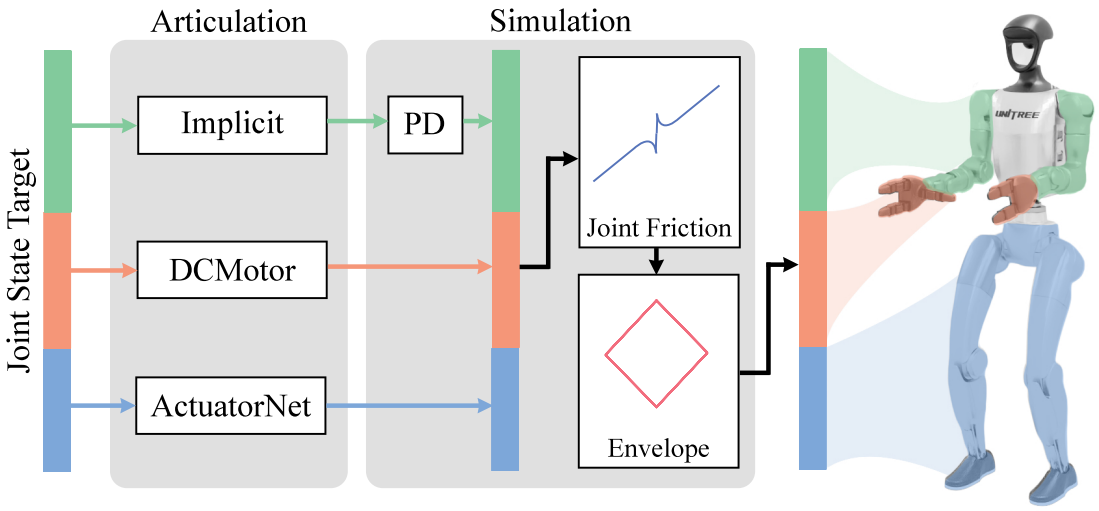


Figure 10: Custom actuator integration in Isaac Lab. Different robot joints can use different actuator models. Implicit actuators rely on the simulator's PD controller, while explicit actuators (*e.g.* neural networks) process commands directly to generate joint efforts that are set into the simulator.

Modifiers can also be used to apply stateful augmentations like discrete filters and integrators. Moreover, Modifiers are chainable, meaning they can be combined in arbitrary order, allowing for reuse and customization. These flexible APIs of Isaac Lab allow for a straightforward integration of tailored noise models without having to adjust the individual sensor.

3.3. Actuators

In Isaac Lab, actuators are the interface between desired joint actions and articulation motion. Actuators provide a control loop over desired motion and joint intrinsic model definitions. All actuators provide interfaces for defining joint friction using two different models. The first model is a simple Coulomb friction model with a constant coefficient of friction. The second model provides a stiction model with a static slip threshold, dynamic friction, and viscous friction coefficients. Actuators also have an armature value that represents the motor inertia and is used to affect the dynamic response of the actuator. Limits for both velocity and effort can also be provided to better approximate real-world limitations. Actuators are separated into two different classes: implicit and explicit actuators. They can be configured separately across different joints or joint groups in an articulation (Figure 10).

3.3.1. Implicit Actuators

Implicit actuators utilize the Nvidia PhysX joint PD controllers. These controllers can be used to track both position and velocity by specifying the stiffness and damping terms of the PD controller. A feedforward effort can also be provided from the desired control action. Implicit actuators can utilize the velocity and effort limits to apply inequality constraints to the simulation solver. This can introduce issues with solvers if they become too restrictive. These joint-level controllers are typically more stable than explicit actuators, as they are iteratively solved in step with physics. They also tend to be more accurate at low sampling rates than explicit actuators.

3.3.2. Explicit Actuators

Explicit actuators operate in a similar manner as the implicit actuators, but they better approximate the discrete implementation of joint controllers on hardware, especially at higher physics sampling rates. Given a desired joint action, explicit actuators compute the applied effort that is sent to the physics solver. This introduces the inherent numerical challenges of discrete-time control. Ways to mitigate this include traditional control

stability analysis, as well as actuator armature. Explicit actuators can utilize the solver limits for velocity and effort, but can also apply limits that are used by the explicit effort calculations. A variety of different explicit actuators are available in Isaac Lab, including:

- **Ideal PD** actuators are the simplest form of explicit actuators. Similar to the implicit actuator, the Ideal PD actuator implements a PD feedback controller with feedforward effort, using specified stiffness and damping parameters. However, the Ideal PD actuator operates in a discrete-time formulation. It can also apply an effort limit, clipping the computed torque before passing it to the solver.
- **DC Motor** actuators utilize the same control and intrinsic parameters as the Ideal PD actuator, but provide an additional velocity-dependent effort limit model that approximates a four-quadrant DC motor torque speed curve. They do not simulate the electro-mechanical dynamics of a DC motor.
- **Delayed PD** actuators build upon the Ideal PD actuator by providing a mechanism for simulating communication delay in the control system by buffering desired actions by a configurable number of simulation steps. These delays are important to simulate when approximating the communication delays on real distributed systems. The Spot application described in (Section 6.1) is an example use case of the Delayed PD actuator.
- **Remotized PD** actuators are used to simulate actuators with joint position-dependent effort limits. Some robots have remote actuators that utilize linkages to transfer power to distal joints. This type of actuation results in the nonlinear transformation ratios and position-dependent effort limits that are emulated by the Remotized Actuator. An example use of this actuator is the Spot knee actuator (Section 6.1).
- **Neural Net** Actuators utilize a trained neural network for joint control rather than the PD control loop of the Ideal PD actuator. These models can be trained on hardware data to better emulate the dynamic response of the actuator in the real world. They are often coupled with the DC motor torque-speed curve effort limits. Isaac Lab has examples for both Long Short-Term Memory LSTM actuators (Rudin et al., 2022) and Multi-layer Perceptron (MLP) actuators (Hwangbo et al., 2019) models.

3.4. Controllers

Controllers in Isaac Lab represent a class of robotic tools that generate desired joint-level commands (position, velocity, effort) from a higher-level input. These higher-level inputs typically involve task or joint space desired motion. The role of a controller is to determine the desired joint-level actions required to complete the higher-level command. Controllers in Isaac Lab can be organized into a few key categories: inverse kinematics, force control, and motion planners. These controllers are typically integrated into the actions of an MDP formulation of the robot learning tasks.

3.4.1. Inverse Kinematics

The inverse kinematics (IK) controllers in Isaac Lab convert desired task-space motion into the required joint space motion using two main methods. First is the differential IK controller that utilizes the kinematic Jacobian to compute changes in joint position from a desired change in end-effector pose. The second is an implementation of the Pink library that utilizes a quadratic programming method to determine desired joint velocities from a desired end-effector velocity.

The **Differential IK** controller converts desired changes in task-space pose to changes in joint position by utilizing the notion of differential kinematics defined by the kinematic Jacobian, by using the inverted Jacobian matrix. The Jacobian inverse has a flaw that occurs when reaching kinematic singularities. The Differential IK controller allows for different ways of handling this singularity. Options include the Moore-Penrose pseudo-inverse, adaptive singular value decomposition, the Jacobian transpose approximation, and the damped pseudo-inverse. Configurable functionality for the Differential IK controller includes controlling the position or pose of the end-effector and handling the relative or absolute desired position or pose.

Isaac Lab integrates the **Pink** library, a Python framework for task-based differential inverse kinematics that

uses Pinocchio for forward kinematics and quadratic programming (QP) solvers for optimization. Unlike traditional analytical IK methods, Pink allows for computing motions that steer the robot toward achieving multiple simultaneous tasks through a weighted objective formulation. Building on this foundation, we have implemented a reactive IK controller that achieves real-time performance. A key advantage of Pink lies in its extensible task-based architecture, which allows users to easily introduce custom objectives into its underlying QP solver. Tasks are defined by residual functions that describe desired behaviors (e.g., end-effector positioning, collision avoidance, joint limits). While conflicts between competing tasks are resolved through normalized cost weighting, this approach trades off performance and accuracy of each task. Isaac Lab extends Pink's task library with the NullSpacePostureTask, designed for high degree-of-freedom robotic systems. This task regularizes redundant joints that do not contribute to the primary control objective, such as maintaining a preferred arm posture during end-effector positioning. The implementation provides controlled null-space behavior while preserving task-space accuracy, improving motion quality for articulated systems with kinematic redundancy.

3.4.2. Force Control

Force control is a useful way to enable robots to interact with environments and utilize proprioception to regulate how forces are imparted on the environment and objects. Isaac Lab provides implementation of two force control methods: the joint impedance controller and the operational space controller.

The **Joint Impedance** controller provides an interface to control the joint position of an articulation given a desired impedance. This happens individually at the joint level using a similar PD control to the Ideal PD joint actuator, but it additionally provides interfaces for calculating feedforward efforts for both inertial and gravity compensation. This controller also provides the ability to have fixed impedance, variable stiffness, and variable impedance versions.

The **Operational Space** controller provides an interface for controlling the impedance of a robot at the operational or task space of the robot. This controller comes with a generalized interface for functionality like hybrid force-motion control, dynamics decoupling, gravity compensation, variable impedance, and null space control for redundant manipulators.

3.4.3. Motion Planning

The **cuRobo** (Sundaralingam et al., 2023) integration in Isaac Lab enables fast, GPU-parallelized collision-aware motion planning. At its core is cuRobo's `MotionGen`, which combines inverse kinematics, collision checking, and trajectory optimization, with optional graph-based planning for global motion. This provides low-latency planning for dynamic scenes and manipulation tasks such as those in workflows like Section 5.5.2. Within Isaac Lab, the planner is initialized from the robot configuration and world description, running on a dedicated CUDA device. Tensor handling is managed internally to ensure consistency between cuRobo computation and Isaac Lab tensors. A brief warmup call primes kernels and caches to reduce first-plan delay.

3.5. Teleoperation Support

Teleoperation remains an integral part of robotics. Its uses span direct user control, controller and policy evaluation, and demonstrations for robot learning. Teleoperation in Isaac Lab can be performed via a range of hardware devices and applications in extended reality.

3.5.1. Teleoperation Devices

Isaac Lab provides support for various teleoperation devices that allow users to control a diverse set of robots. A keyboard device interface is provided and enables delta pose control over robotic arms and grippers. Users can map additional keys to custom callback functions to design specialized workflows, such as controlling the base height/velocity of a humanoid. For smoother human control, Isaac Lab also supports spacemouse teleoperation devices. The use of a spacemouse can enable higher quality human demonstrations for data collection for imitation learning tasks, as well as provide simultaneous multi-axis movement.

3.5.2. Extended Reality (XR) Device Teleoperation

Teleoperation of bimanual or humanoid dexterous manipulation tasks can be challenging to nearly infeasible with traditional devices such as keyboards and spacemice. To enable these tasks, Isaac Lab includes support for extended reality (XR) devices which allow users to perform dexterous control bimanual/humanoid robots using an Apple Vision Pro (AVP) headset. The user's hand joints detected by the AVP are mapped to the end-effector pose, which is then used in inverse kinematics (IK) to compute the corresponding joint angles and arm positions. To provide a natural and intuitive user experience, the IK formulation incorporates two tasks: (1) a waist task, which infers waist joint positions from hand motion, and (2) a null-space task to maintain the arm in nominal (neutral) positions as described in Section 3.4.1.

In contrast to prior XR teleoperation systems that directly stream stereoscopic video from the robot's perspective to the operator's display (Cheng et al., 2024; Zhang et al., 2025), our approach leverages NVIDIA's CloudXR framework (NVIDIA) to enhance user comfort through low-latency streaming, supported by GeForce Now technology. CloudXR incorporates re-projection algorithms to mitigate the perceptual effects of residual streaming delays, thereby providing stable and visually comfortable image qualities. An additional benefit is the platform's support for augmented reality (AR) overlays, which allows operators to maintain contextual awareness of their environment during teleoperation.

3.6. Procedural Generation

3.6.1. Terrain Generation

Isaac Lab supports multiple approaches for creating simulation environments: (i) *procedural generation* through scripts, (ii) importing scanned meshes (Straub et al., 2019), (iii) interactive editing via the Isaac Sim game-like GUI, and (iv) hybrid combinations of thereof. The script-based interface provides fine-grained control over environment difficulty and enables systematic terrain randomization for benchmarking robustness. This interface leverages Trimesh (et al.) to generate primitive shapes and arbitrary meshes, which can be flexibly combined into complex structures (e.g., pyramids or composite terrains). It also supports heightfields, which are automatically converted into meshes to represent rough terrain surfaces. To ensure efficient simulation, all generated structures are optimized to minimize mesh complexity while preserving geometric fidelity, thereby improving performance in the underlying PhysX engine. Figure 11 showcases various types terrain assets available in Isaac Lab.

3.6.2. Integration with External Asset Generation Frameworks

Scene graph generation frameworks such as Scene Synthesizer (Eppner et al., 2024) and Infinigen (Raistrick et al., 2024) can be used to programmatically design cluttered 3D environments (such as homes and kitchens) and exported to USD for use in Isaac Lab. These frameworks have a library of predefined objects and seemlessly combine them with artist-generated datasets like Objaverse (Deitke et al., 2023) to generate USD scenes. Training loco-manipulation policies on such procedural scenes in Isaac Lab results in better robustness and generalization. For instance, (Sleiman et al., 2024) showed sim-to-real transfer of a door opening policy by a quadrupedal mobile manipulator, while (Luo et al., 2025) demonstrated generalizable pick-and-place skills for a humanoid robot trained in a large number of procedural kitchen environments.

3.6.3. Multi-Asset Spawning and Randomization

After defining a terrain, Isaac Lab allows assets to be instantiated in the scene and cloned across parallel environments. Within each environment, multiple assets can be spawned simultaneously, and both geometric and visual properties can be randomized. Isaac Lab further supports swapping assets across environments, provided they belong to the same asset class. For example, in a manipulation task, the target object may vary between a cup, a glass, or cutlery, enabling the policy to generalize across different shapes while pursuing the same objective. For vision-based policies, Isaac Lab integrates domain randomization techniques to improve sim-to-real transfer. Here, visual properties such as textures, materials, and colors can be procedurally varied

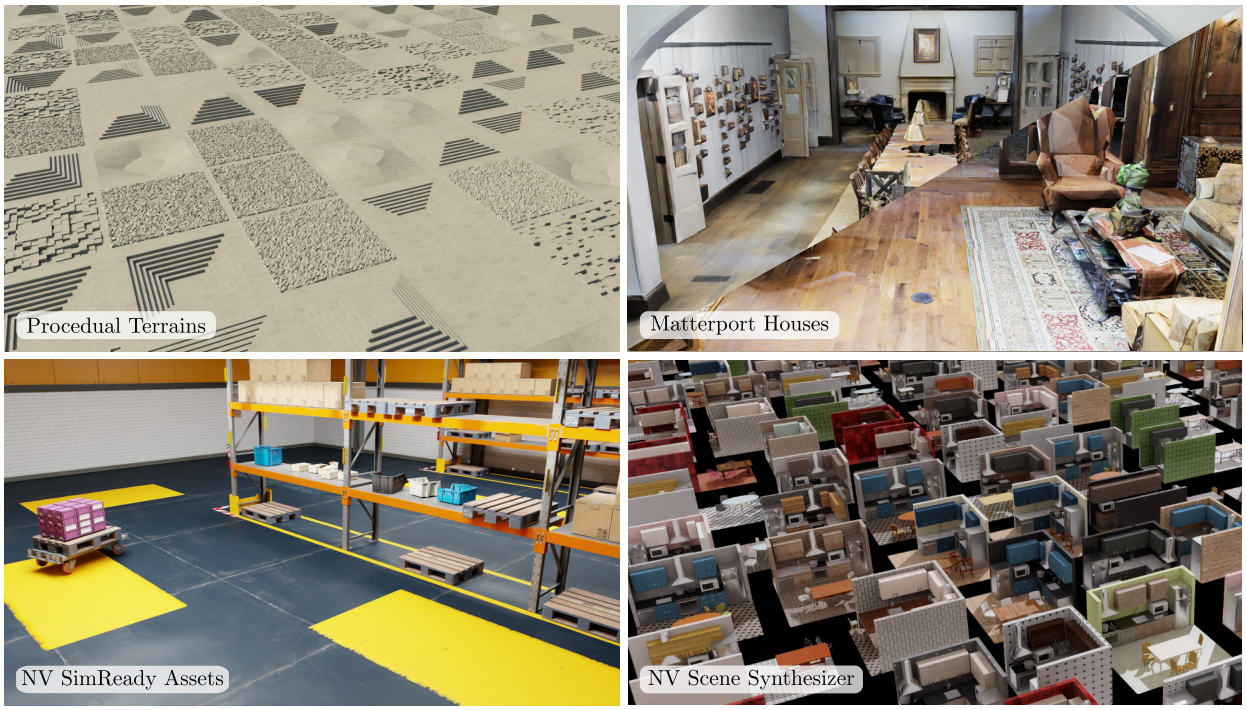


Figure 11: Overview of different terrains that can be used with Isaac Lab. The procedural terrains are generated using Trimesh and allow for large, randomized terrains. At the same time, USD terrains from Matterport (Chang et al., 2017), NV SimReady Assets, or the NV Scene Synthesizer can be imported into Isaac Lab.

across assets and environments. This combination of geometric variation and visual randomization promotes robustness by exposing policies to a broader distribution of task-relevant scenarios.

3.7. Task Framework and Environments

Isaac Lab supports two primary paradigms for constructing and executing robot learning workflows: the manager-based workflow and the direct workflow. These workflows differ in their level of abstraction, modularity, and intended use cases. This provides flexibility for users ranging from those building minimal, high-performance environments to those developing complex, structured robotic systems.

The manager-based workflow provides a more structured and modular design, encapsulating robot components into reusable subsystems (e.g., actions, observations, rewards, commands). In contrast, the direct workflow offers an interface that allows users to interact directly with the simulation, physics, and learning components without enforcing a specific organizational structure. It is particularly well-suited for performance-sensitive training pipelines, where minimal overhead and tight integration with GPU resources are critical. Algorithm 2 outlines the `step` function logic for both workflows, demonstrating the sequence of computations to step an environment.

3.7.1. Manager-Based Workflow

The manager-based workflow in Isaac Lab is a structured way to build environments by decomposing the MDP into reusable managers — observations, actions, rewards, terminations, commands, curricula, events, and recording — so each unit has a single responsibility and a clear interface. This design targets ML/RL practitioners: you interact with MDP concepts directly while the framework handles simulation plumbing (scene updates, decimation), vectorization, per-env resets, and Gym-compatible spaces derived from the active terms. As a result, environments remain readable and extensible, and switching on/off terms or swapping configurations becomes a configuration change rather than a refactor.

Algorithm 2 Environment Step Function

input: action
output: observations, rewards, terminations, timeouts, extras
procedure STEP(action)
    `// pre-physics step`
    Process actions (*e.g.* clipping, affine transformation)
    `// physics step`
    for each substep in environment decimation **do**
        Apply processed actions to simulation buffers (via actuator models)
        Apply pre-simulation events (if any)
        Advance simulation (without rendering)
        if render interval reached **and** `is_rendering` **then**
            Render simulation
        end if
        Update scene buffers
    end for
    `// post-physics step`
    Update episode counters
    Compute terminations and timeouts
    Compute rewards
    `// environment reset`
    `reset_env_ids` ← environments to reset
    if `reset_env_ids` not empty **then**
        Update curriculum
        Apply reset events to reset the environments
        Reset episode counter and internal buffers (*e.g.* state history)
        Update simulator kinematic state
        if sensors present **and** rerender configured **then**
            Render simulation
        end if
    end if
    `// command and observation update`
    Update commands
    Apply interval events (if any)
    Compute observations
    return observations, rewards, terminations, timeouts, extras
end procedure

Beyond enabling flexible MDP definitions, Isaac Lab ships a catalog of tested, independent term functions with explicit inputs/outputs. Practitioners can add or ablate terms without touching unrelated code paths, accelerating idea testing, ablation studies, and continuous development. The manager API formalizes where each computation lives and exposes logging hooks to attribute signals at term-level granularity (e.g., reward contributions, termination causes), improving training analysis and reproducibility. While a manager layer can introduce modest overhead compared with a hand-tuned direct workflow, most of that overhead is CPU orchestration and kernel-launch latency rather than physics itself — precisely the kind of cost that CUDA Graphs are designed to reduce by bundling many GPU operations into a single launch. We foresee manager term graphed executions as a plausible roadmap to bootstrap the MDP calculation runtime, thus narrowing any remaining gap.

3.7.2. Direct Workflow

The direct workflow in Isaac Lab is designed to expose fine-grained control over simulation and learning pipelines with minimal abstraction overhead. It allows users to instantiate simulation environments, robots, and tasks directly through procedural APIs, providing immediate access to low-level data structures such as joint states, contact forces, and sensor outputs.

This workflow is particularly aligned with the end-to-end training paradigm introduced in Isaac Gym (Makoviy-

chuk et al., 2021), where simulation and learning are tightly coupled on the GPU. Users can execute physics steps, compute rewards, apply control actions, and collect observations within a single, optimized environment class. The simplicity and performance of the direct workflow make it ideal for benchmarking algorithms and performance-sensitive workflows.

Despite its simplicity, the direct workflow remains extensible. Users can define custom tasks, observations, and reward functions, and integrate third-party learning frameworks or data logging tools as needed. It serves as a strong foundation for users who require maximum performance and flexibility, especially in scenarios where control and learning are prioritized over system modularity.

3.7.3. Environments

Isaac Lab provides robot learning environments across four families, namely classic, locomotion, manipulation, and navigation, with many tasks offered in both direct and manager-based variants. Figure 12 showcases some examples of environments available in Isaac Lab.

Across categories, the environments were chosen for their sim-to-real relevance, their advanced simulation features (e.g., SDF, contact sensing, tiled rendering), and their clear skill structure spanning dexterity, locomotion, and hierarchical control. Re-implemented benchmarks share unified APIs and data layouts to minimize glue code and maximize comparability and accessibility for the community to recreate, study, or expand.

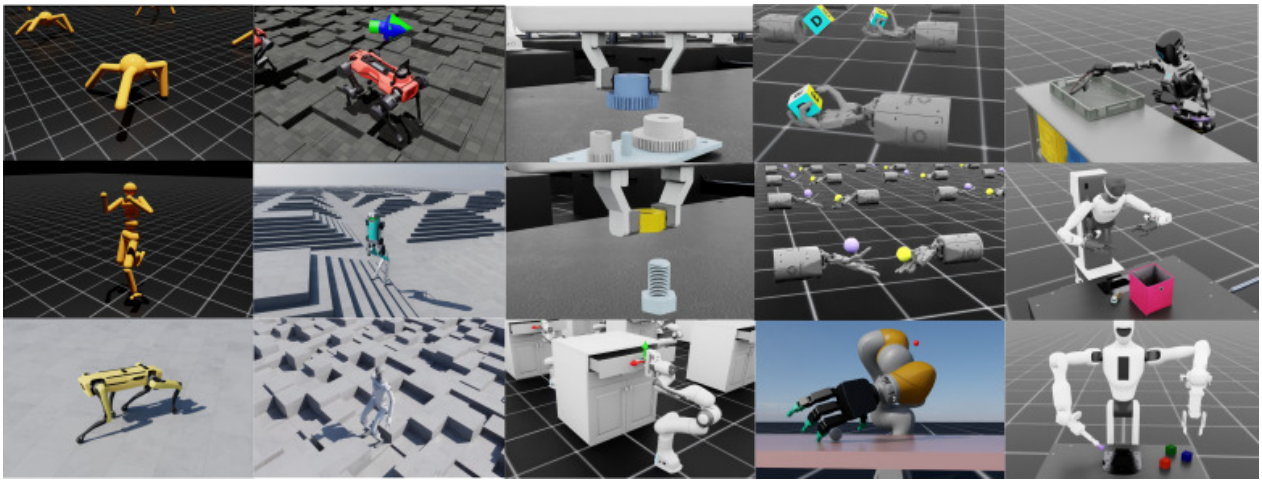


Figure 12: Suite of environments in Isaac Lab. They illustrate a variety of simulation capabilities, including contact-rich interactions, multiple sensor modalities for observations, and support for diverse robotic morphologies, such as humanoids, quadrupeds, fixed-arm robots, and dexterous hands.

Classic. "Hello World" tasks (Ant, Humanoid, Cartpole) are quick-start baselines for bring-up, regression, and throughput testing. Their simplicity allows for rapid prototyping before integration into more complex environments.

Locomotion. Quadruped velocity-tracking and terrain curricula follow massively parallel training recipes and evaluation protocols inspired by prior work (Rudin et al., 2022), which we expanded to 11 different robot morphologies, including a1, g1, h1, go1/2, Anymal-B/C/D, Cassie, Digit, and Spot. Notably, this implementation has shown robust sim to real for Anymal and Spot.

Manipulation. Dexterous hand–arm tasks (e.g., reorientation, grasping) (Petrenko et al., 2023; Singh et al., 2025) are re-created, alongside contact-rich assembly benchmarks that leverage SDF-based contact modeling and force-aware strategies (Narang et al., 2022; Noseworthy et al., 2025; Tang et al., 2024). The suite covers both single-object dexterity and multi-part assembly, with options for tactile/force sensing.

Navigation. Goal-conditioned navigation tasks target hierarchical RL setups where we show how to compose a high-level command policy with a low-level locomotion policy into one environment.

Multi-Agent Environments Isaac Lab includes and supports the creation of custom environments for solving general physical-based Multi-Agent Reinforcement Learning (MARL) tasks, in which multiple learning agents coexist and interact within a shared environment. Depending on the nature of the problem and the interests of the agents, tasks can be defined with different settings, such as competition (agents play against each other), cooperation (agents work together to achieve a common goal), or a combination of the two.

The exposed API is available for direct workflows and is based on PettingZoo (Terry et al., 2021) Parallel API. The Parallel API is a standard interface for MARL in which all involved agents observe the environment and take action simultaneously at each step. Future releases will explore PettingZoo's Agent Environment Cycle (AEC) game model, in which agents act sequentially and environments are updated after each agent makes a decision.

4. Simulation Performance

4.1. Environment Throughput

We evaluate throughput for complex learning tasks with different sensor setups. All benchmarks are executed in headless mode on three GPU platforms: *L40* (48 GB), *RTX Pro 6000* (96 GB), and *GeForce 5090* (32 GB). These cover a range of numbers and generations of RTX cores and VRAM sizes. In addition, we study the scaling behavior under distributed training across multiple GPUs on the *RTX Pro 6000*. As a performance metric, we report frames per second (FPS), defined for environment learning throughput as:

$$FPS = \frac{\#\text{ of environment steps}}{\text{simulation time} + \text{learning time}} \tag{1}$$

Apart from the GPU, environment throughput can also be dependent on single-core CPU performance due to bottlenecks in some parts of PhysX simulation and the main training loop. The *L40* server uses two AMD EPYC 7763 64-Core Processors, the *RTX Pro 6000* server uses two AMD EPYC 9554 64-Core Processors, and finally the *GeForce 5090* workstation uses a single 8-core AMD 9800X3D processor with single-core performance more than double that of the *L40* server.

We first benchmark state-based environments, where simulation cost is dominated by PhysX computation. We then analyze perceptive environments, where rendering becomes the bottleneck. Finally, we compare performance across the manager-based and direct workflows introduced in Section 3.7.1 and Section 3.7.2.

4.1.1. State-based environments

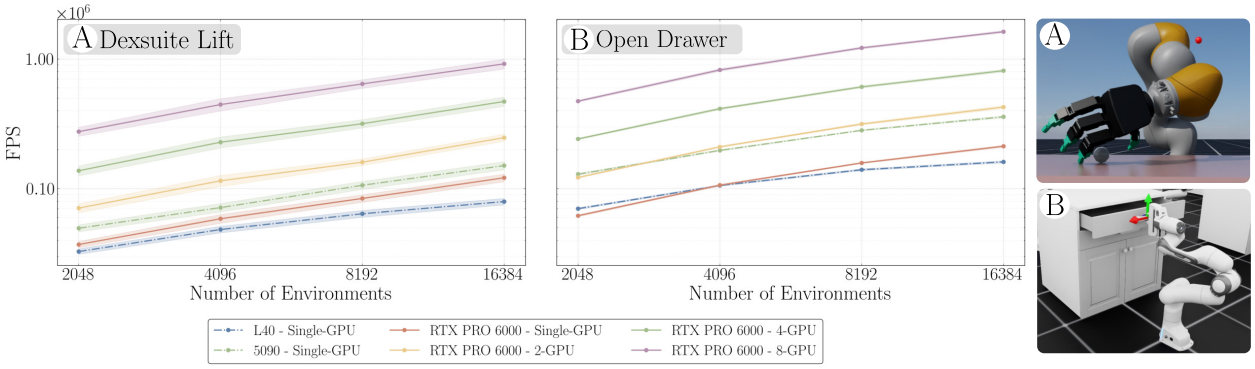


Figure 13: Log-scale throughput comparison for state-based manipiulation tasks on three GPU platforms, including distributed training with two, four, and eight GPUs. Shown are (A) the DexrAH lift task and (B) the Franka arm cabinet drawer opening task.

For state-based training relying on proprioceptive information only, we benchmark two manipulation tasks - the DextrAH (Lum et al., 2024) lifting teacher task with no perception, and the Franka arm cabinet drawer opening task.

As shown in Figure 13, the systems with newer Blackwell GPUs have improved performance across both environments compared to the previous generation *L40*. Distributed training provides further gains, scaling almost perfectly linearly as the number of GPUs increases. With eight GPUs and 16384 environments, the DextrAH teacher task reaches over 900,000 frames per second in training, while the Franka cabinet task reaches over 1.6 million frames per second.

It is notable that the workstation *GeForce 5090* system comes close in performance to the two GPU *RTX Pro 6000* server in the Franka cabinet drawer task, but is only about 25% faster in the DextrAH task. These differences are primarily due to the combination of CPU bottlenecks and extremely fast single-core CPU performance in the 9800X3D CPU compared to the throughput-oriented server CPUs. Eliminating these bottlenecks is an important goal for future releases of Isaac Lab and the Newton physics engine.

4.1.2. Perceptive Environments

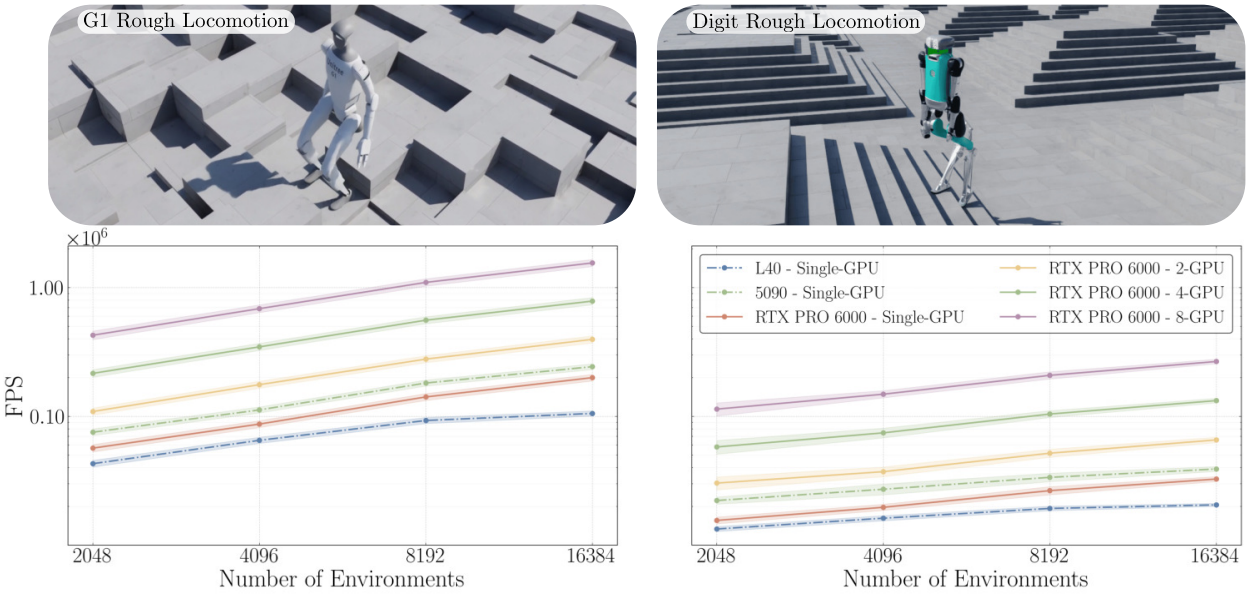


Figure 14: Log-scale throughput comparison for perceptive locomotion tasks across GPU models and distributed training setups. Both tasks include a RayCaster-based height scanner ($1.6\,\mathrm{m} \times 1.2\,\mathrm{m}$, resolution $0.1\,\mathrm{m}$). Additionally, the Digit task highlights the simulation of closed-loop kinematic chains in Isaac Lab.

In perceptive tasks, the simulation speed depends on the combination of PhysX and rendering speeds, where the latter typically dominates computational demands. We benchmark three perceptive tasks: two rough terrain locomotion tasks with the Unitree G1 humanoids and Agility Robotics Digit humanoid, and the end-to-end perception-based version of DextrAH for dexterous manipulation (Singh et al., 2025). The two locomotion tasks use a height-scanner of size $1.6m \times 1.2m$ implemented using the *RayCaster* sensor. For the DextrAH environment, we compare two rendering approaches: the *Tiled-Camera* (Section 3.2.2) and the Warp-based *RayCasterCamera* (Section 3.2.3). The Digit locomotion environment also showcases the simulation of closed-loop kinematic chains. Benchmarks are run on the same GPU platforms as for the state-based learning tasks.

As shown in Figure 14 and Figure 15, throughput improves substantially with newer GPUs (*RTX Pro 6000* and *GeForce 5090*) and with distributed training, mirroring the trends observed in state-based environments. However, throughput remains lower overall compared to state-based tasks due to the additional rendering cost.

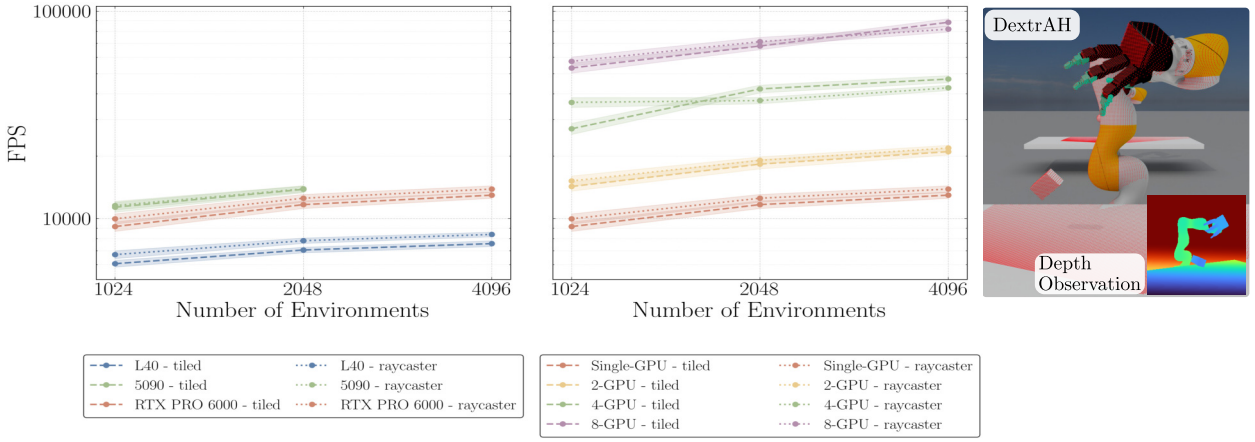


Figure 15: Log-scale throughput comparison for perceptive learning task in dexterous manipulation (DextrAH). Results are shown for the Tiled and Raycaster-based camera at 64x64 resolution across GPU models (left) and distributed training setups (right). All multi-GPU training is tested on *RTX Pro 6000* systems.

The *GeForce 5090* system, with its high single-core performance CPU, achieves higher throughput at smaller environment counts, while the *RTX Pro 6000* system narrows the gap as the number of environments increases. Owing to its larger VRAM capacity, the RTX Pro 6000 also supports a greater degree of parallelization, enabling more environments to be simulated concurrently. When comparing sensor implementations, the RayCasterCamera demonstrates superior performance on a single GPU. In contrast, with multiple GPUs and large environment counts, the Tiled-Camera achieves better parallelization, eventually surpassing the RayCasterCamera.

4.1.3. Workflow Comparison

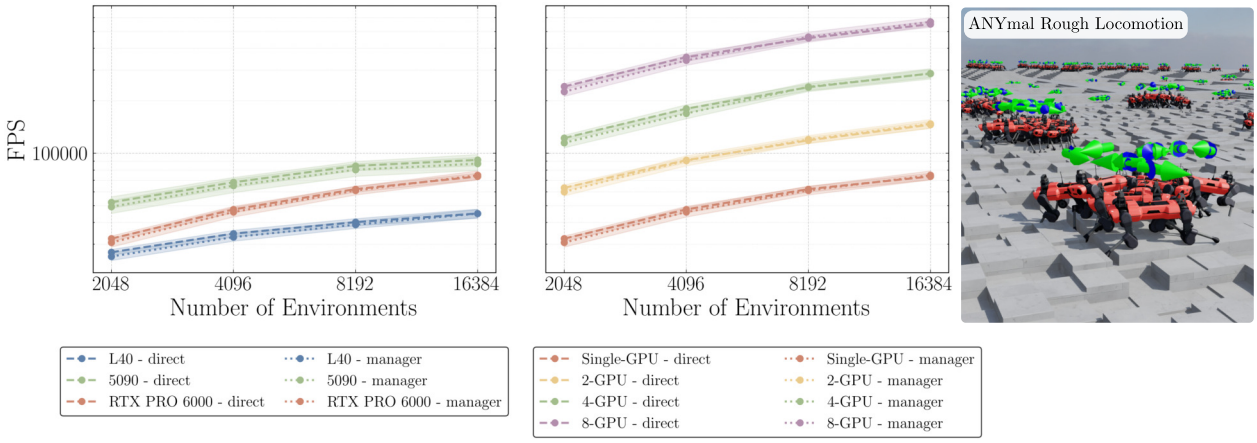
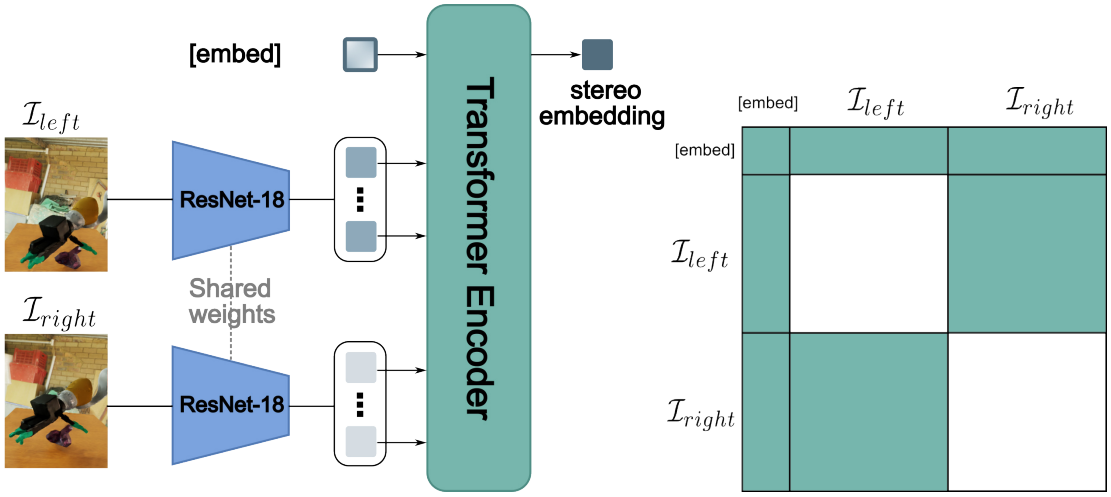


Figure 16: Log-scale throughput comparison for the ANYmal locomotion task using the manager-based and direct workflows.

We further evaluate the performance of the manager-based and direct workflows introduced in Section 3.7.1 and Section 3.7.2. As shown in Figure 16, the direct workflow generally achieves slightly higher throughput, on average 3.53% on a single *RTX PRO 6000*. However, the performance gap is small and becomes negligible for larger environment counts and in tasks where perception dominates the computational cost. This indicates that the manager-based workflow offers comparable efficiency while providing additional structure and flexibility for complex training setups.



(a) Stereo Encoder (b) Cross-Attention Mask

Figure 17: Singh et al. (2025) train a student policy with stereo RGB images. **(a)** The stereo encoder uses a pre-trained ResNet-18 (with the last two layers removed) to encode each image independently into a high-dimensional vector. Each vector is projected and split into 128 tokens. Tokens from both images, along with a learnable `[embed]` token, are passed into a two-layer transformer that performs cross-attention. The output from the `[embed]` token is processed through an MLP, producing the final stereo embedding vector. **(b)** The turquoise regions illustrate cross-attention between the tokens. Each image's tokens attend to the other image's tokens and the shared `[embed]` token, which attends to all tokens.

5. Learning Techniques and Workflows

5.1. Reinforcement Learning

Isaac Lab adheres to the Gymnasium API (Towers et al., 2024), a widely adopted interface for defining RL environments. By conforming to the `gymnasium.Env` specification, Isaac Lab environments can be used directly with any library that supports Gym-compatible environments. At the same time, many RL frameworks implement custom specifications for handling batched data from parallelized environments. To address this, Isaac Lab is designed with extensibility in mind to integrate custom solutions with minimal engineering effort. Out-of-the-box, Isaac Lab provides built-in support for SKRL (Serrano-Munoz et al., 2023), RSL-RL (Schwarke et al., 2025) and RL-Games (Makoviichuk and Makoviychuk, 2022), Stable-Baselines3 (SB3) (Raffin et al., 2021), and Ray (Moritz et al., 2018). These libraries complement different aspects of the RL workflow and highlight Isaac Lab's interoperability.

Learning effective control policies directly from visual inputs is a central challenge in modern robotics and reinforcement learning. While traditional approaches rely on low-dimensional state representations, many real-world applications require policies capable of interpreting and acting upon high-dimensional sensory observations, such as RGB or depth images. Isaac Lab supports this line of research by providing fast high-fidelity rendering and privileged state access. It supports visual domain randomization, exposing policies to diverse lighting conditions, textures, colors and backgrounds, and provides examples using pre-trained visual backbones such as Theia (Shang et al., 2024) and ResNet (He et al., 2016).

In the following, we describe two complementary paradigms for incorporating perception into reinforcement learning: *teacher-student distillation*, where a perception-based student imitates a privileged state-based teacher, and *end-to-end perception-in-the-loop training*, where policies are learned directly from raw sensory inputs.

5.1.1. Teacher-Student Distillation

Training end-to-end pixels-to-action policies directly with RL presents unique challenges, as the agent must simultaneously learn perception and control. A widely adopted solution is the teacher-student distillation approach (Chen et al., 2020; Lee et al., 2020), in which a teacher policy, trained via RL on state-based (privileged) observations, guides a student policy that receives partial information (such as rendered RGB images) corresponding to the teacher's states. The student is trained using an online imitation learning method, such as DAGGER (Ross et al., 2011), which is particularly effective as it incrementally aggregates data from the student's own trajectories while leveraging the teacher for corrective actions.

The distillation framework can be applied to students receiving different types of observations. For example, Lee et al. (2020) trains a student with proprioception-based observations, while Lum et al. (2024) uses depth-based observations. More recently, Singh et al. (2025) train a student policy with RGB images, using high-fidelity rendering in Isaac Lab. Student architectures tend to be simpler for proprioception- and depth-based observations. In contrast, training an RGB-based student is more challenging, as the network must learn invariance to textures and lighting, infer 3D structure, and identify objects of interest directly from RGB inputs. To facilitate this, it is often helpful to initialize the student with a pre-trained encoder, as shown in Figure 17.

5.1.2. End-to-End Perception-in-the-Loop

While distillation enables training an RGB-driven student policy by imitating a state-based teacher, there are compelling reasons to also train policies end-to-end directly from images. First, end-to-end training can exploit rich spatial and shape cues that low-dimensional state vectors cannot capture. Second, imitation-based distillation introduces an information gap due to input mismatch: the teacher observes privileged state information, while the student sees only partial observations. This can lead to a performance drop, as the student must reconstruct unobserved states from images, a challenge that becomes particularly pronounced under high camera occlusions.

Singh et al. (2025) show the first sim-to-real system trained end-to-end for multi-fingered hands using Isaac Lab. Training end-to-end RL policies can also give rise to emergent active perception behaviors. Luo et al. (2025) use Isaac Lab to train a humanoid agent relying solely on egocentric vision and proprioception to perform various loco-manipulation tasks. Similarly, Yang et al. (2025) train an attention-based RL policy for long-range navigation. The learned agents in these works exhibit search and exploratory behaviors, actively moving around to gather visual information relevant to the task, demonstrating active perception strategies that emerge naturally from end-to-end training.

5.2. Population-Based Training

While RL remains the predominant approach for training policies in simulation, the stochastic nature of the training process and the large hyperparameter space collectively introduce high variance in training results, particularly for systems with high Degrees-of-Freedom (DoF) systems and tasks with sparse rewards. As a result, some runs may progress rapidly, while others stagnate with little or no improvement. Petrenko et al. (2023) addresses this problem by using genetic algorithms to mutate the hyperparameter space, promoting diversity and exploration in the learning agent. Built on top of Population-Based Training (PBT) (Jaderberg et al., 2017), DexPBT assigns each worker an independent RL training process with its own set of hyperparameters. Periodically, the best-performing workers, with their weights and hyperparameters, replace the worst-performing ones, as illustrated in Figure 18.

Isaac Lab includes an implementation of DexPBT with RL-Games for the dexterous manipulation environments in DextrAH (Lum et al., 2024; Singh et al., 2025), as shown in Figure 19. It reproduces the 6D reposing task from the original DexPBT work using 8 workers, each with 1–2 GPUs, and converges in approximately 16 hours on NVIDIA OVX L40 hardware. The PBT framework in Isaac Lab is configurable and generalizable to other environments, enabling scalable multi-node RL training.

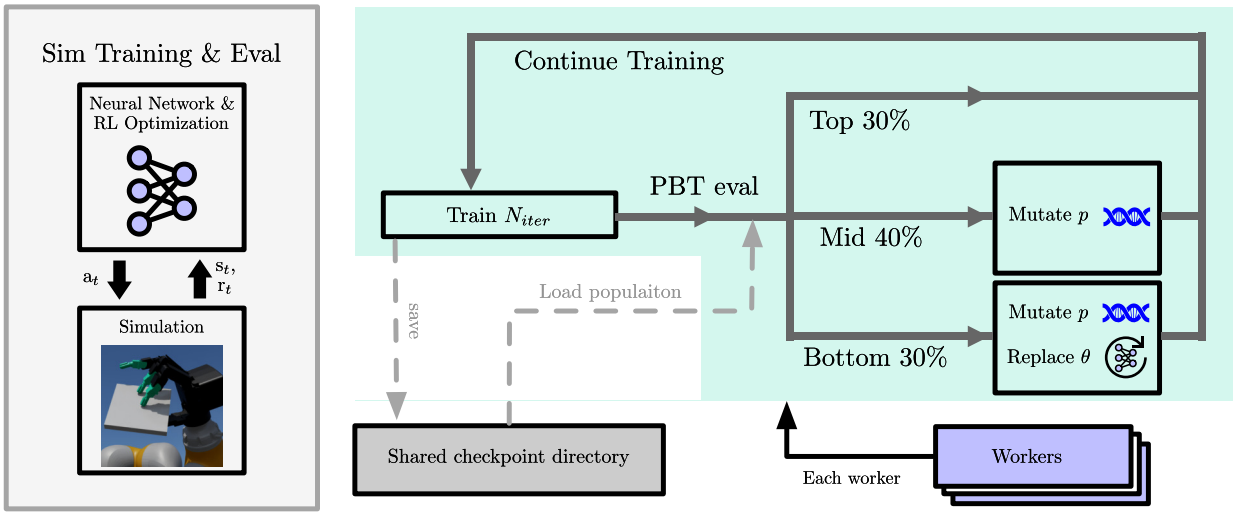


Figure 18: An overview of the PBT framework for RL in Isaac Lab. Each worker runs an independent RL training process with its own hyperparameters. Periodically, top-performing workers share their weights and hyperparameters to replace those of bottom-performing workers, while genetic mutations introduce diversity in the hyperparameter space. This iterative process improves exploration and stability in RL training.

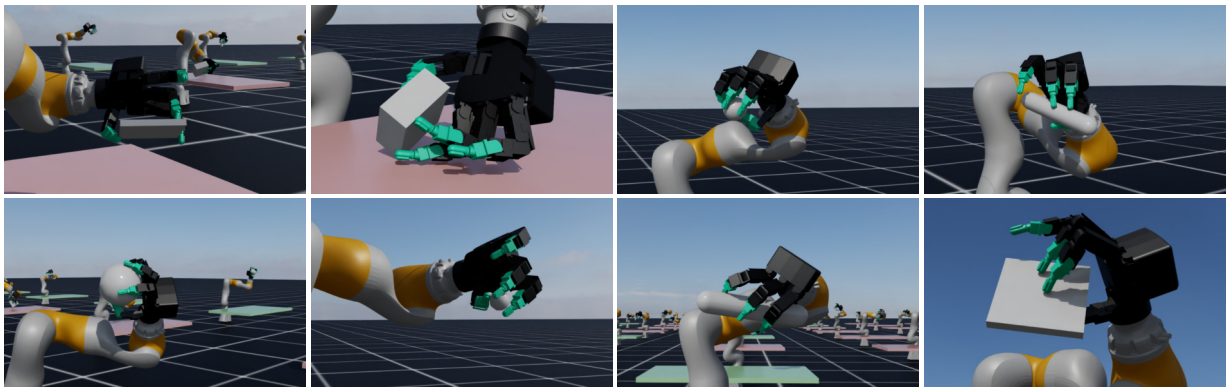


Figure 19: The 6-D object reposing task from DexPBT (Petrenko et al., 2023), performed with an Allegro hand mounted on a KUKA arm. Results are reproduced in Isaac Lab using 8 workers, each with 1–2 GPUs.

5.3. Domain Randomization

Domain Randomization (DR) is a critical component of sim-to-real transfer, where the system parameters are randomized in simulation to promote policy generalization and robustness on deployment. For physics, these include friction, armature, gravity, and mass (Peng et al., 2017), while for rendering they include texture, material, and lighting (Sadeghi and Levine, 2016; Tobin et al., 2017). Such parameters may be difficult to estimate accurately or may vary over time in the real world. By randomizing them over broad distributions during training, policies acquire invariance to these parameters, improving transfer to reality. A key requirement for DR is the ability to modify parameters on the fly, providing diverse training scenarios to the learning agent.

Isaac Lab supports randomization of both physics and rendering parameters. Mesh-related attributes, such as scale and collider type, can only be randomized before the simulation begins playing (Algorithm 1). Most other physics parameters can be randomized at runtime. As discussed in Section 2.2, due to current PhysX design limitations, only simulation state is accessible directly on the GPU, while simulation parameters (*e.g.* masses, friction, contact offsets, and joint armature) must be modified through the CPU API. Rendering parameters such as visual materials, lighting intensities, light source locations, and background textures can also be randomized at runtime, though they currently rely on CPU-based USD APIs. Figure 20 illustrates RGB renderings in Isaac

Figure 20: Top: Camera renderings from different environment instances in simulation, adapted from Singh et al. (2025). Bottom: Examples of data augmentations applied to these renderings before passing them to the learning agent.

Lab with randomized textures and lighting, alongside standard computer vision augmentation techniques.

Furthermore, Isaac Lab also supports **Automatic Domain Randomization (ADR)** (OpenAI et al., 2019) through a configurable curriculum framework that adaptively adjusts the difficulty of the environment based on the performance of the agent. The dexsuite examples in Isaac Lab provide reference ADR configurations for RL training. The framework automatically manages difficulty progression, continuously challenging the learning agent without overwhelming it, thereby promoting more robust and generalizable policies.

5.4. Imitation Learning

While RL has demonstrated clear benefits when scaling with large amounts of simulation data, designing reward functions for many robotic tasks remains a challenge. Imitation Learning (IL) offers an alternative by allowing agents to learn directly from expert demonstrations rather than relying solely on trial-and-error exploration. Simulation provides a safer, more scalable, and cost-effective environment for collecting training data for IL than the real world. Additionally, recent advances in sim-to-real techniques, including the use of generative models such as NVIDIA Cosmos, offer promising approaches for training policies entirely on synthetic data and transferring them effectively to real-world environments.

Isaac Lab supports IL through integration with the widely used **RoboMimic** framework by Mandlekar et al. (2022). Demonstration data can be collected through human teleoperation or synthetically generated using **Isaac Lab Mimic** (described in Section 5.5) . All demonstrations are stored in the standardized HDF5 format based on the RoboMimic schema. Isaac Lab includes a suite of reference training and evaluation scripts that work out of the box with RoboMimic, offering a streamlined starting point for experimentation. It also provides a workflow for converting existing HDF5-based datasets into the **LeRobot** format (Cadene et al., 2024), thereby facilitating access to the extensive ecosystem of open-source models and tools available on the Hugging Face Hub. This conversion transforms row-oriented HDF5 structures into a highly optimized, columnar Parquet format suitable for time-series data, while simultaneously encoding camera frames into MP4 video files.

5.5. Synthetic Data Generation

Isaac Lab Mimic focuses on synthetically generating a large number of robot demonstrations from a limited set of human demonstrations. The workflow segments a human demonstration into object-centric subtasks, applies rigid transformations to each segment, and recombines them into new demonstrations (Jiang et al., 2025; Mandlekar et al., 2023). By transforming and stitching trajectories, Mimic adapts these demonstrations

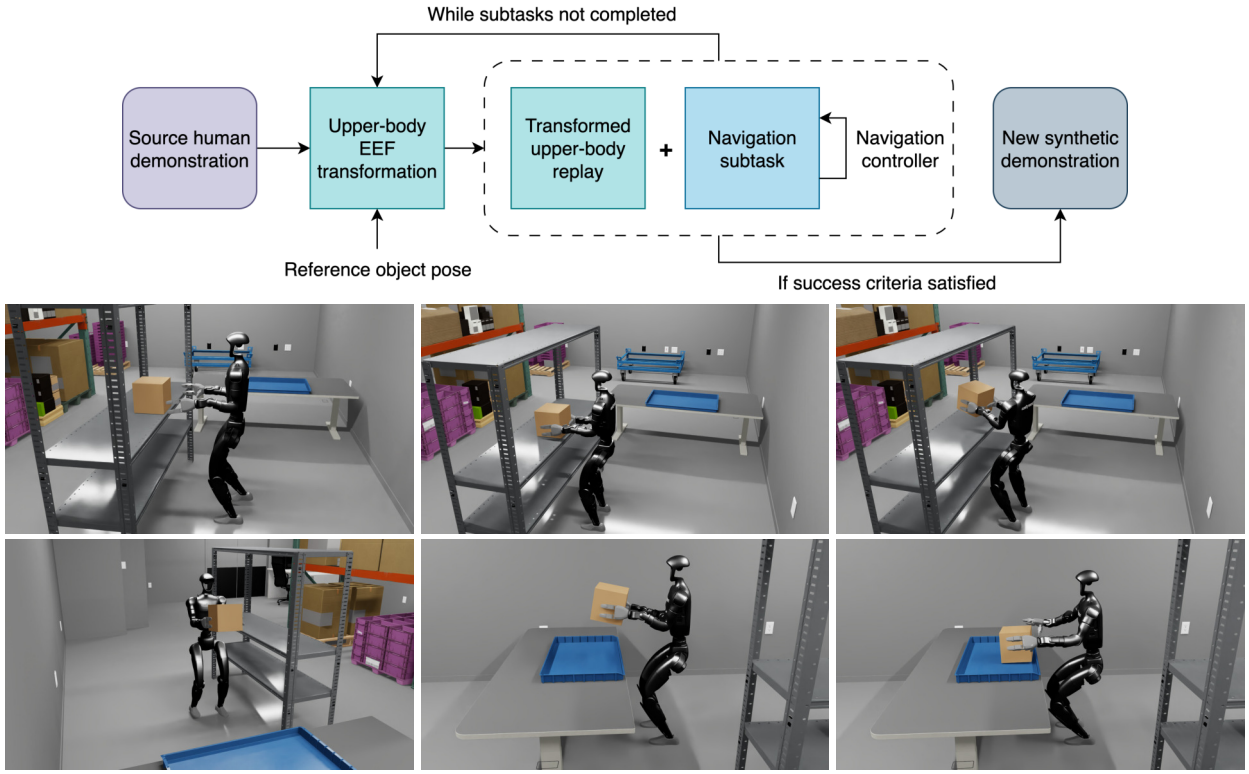


Figure 21: Top: The generation pipeline for loco-manipulation synthetic data using Isaac Lab Mimic. Bottom: A loco-manipulation task consisting of multiple subtasks for manipulation and navigation, such as walking to the shelf, picking up the box, turning towards the table, walking to the table, squatting down, and dropping the box. This example is included as part of Isaac Lab Mimic workflow.

so that the robot can successfully execute tasks even when the robot or objects occupy poses different from those in the original demonstration, significantly expanding the diversity and coverage of the training dataset.

To make a manager-based environment design compatible with the Mimic workflow, users implement the required integration interfaces, including functions to convert robot poses to actions (and vice versa), retrieve the robot end-effector pose, and extract the gripper state from environment actions. Once an environment is Mimic-compatible, it can generate an effectively unbounded number of synthetic demonstrations from as few as a single human demonstration. Furthermore, Isaac Lab Mimic supports parallelized environment execution for data generation, substantially increasing throughput and reducing overall demonstration generation time.

5.5.1. Loco-Manipulation Data Generation

An example of the Isaac Lab Mimic workflow is learning loco-manipulation tasks through imitation learning. Figure 21 provides an overview of the data generation pipeline that employs a decoupled whole-body controller for a humanoid, similar to Ben et al. (2025). By combining a whole-body controller with navigation, we synthesize task demonstrations for loco-manipulation, where coordinated locomotion and manipulation enable robots to move (*e.g.* walk or squat) while simultaneously interacting with objects (*e.g.* grasping, pushing, pulling). This coupling supports complex sequences such as picking up an object from a table, traversing space, and placing the object elsewhere.

The system augments demonstrations with randomized pickup and drop-off locations for boxes, and varied positioning of obstacles. It enhances the data collection pipeline by segmenting manipulation into pick-and-place phases interleaved with locomotion. Using this pipeline, we generate abundant augmented loco-manipulation data from manipulation-only human demonstrations, allowing humanoid robots to learn inte-

grated loco–manipulation skills. The provided interface in Isaac Lab remains flexible, allowing users to apply different embodiments, including humanoids and mobile manipulators, with their choice of controllers.

5.5.2. SkillGen-based Dataset Augmentation

SkillGen (Garrett et al., 2024) is an automated demonstration generation system in Isaac Lab Mimic that produces high-quality, collision-aware robot demonstrations at scale. It combines human-provided subtask segments with GPU-accelerated motion planning (described in Section 3.4.3) to create diverse feasible trajectories that adapt to new object placements and scene layouts. By automating transit motions between annotated skills, it reduces manual data collection while improving dataset consistency and validity.

The Isaac Lab integration of SkillGen offers simple controls for scalable dataset creation. Users can select planner-backed generation with a single flag, configure the number of trials, number of parallel environments, and compute devices, and adjust the planner's parameters to balance speed, success rates, and motion quality. The resulting datasets support IL workflows in Isaac Lab.

6. Application to Robotics Research

In the following, we provide an overview of the various robotic research areas that have benefited from the capabilities of Isaac Lab. As the framework continues to mature, this section will be expanded to include the latest advancements and applications. We focus this survey primarily on work conducted by closely collaborating academic and industrial partners, including NVIDIA, ETH Zürich, and the Robotics and AI Institute (RAI).

6.1. Locomotion

Legged locomotion has long posed a central challenge in robotics, due to the difficulty of maintaining balance, coordinating complex movements, and adapting to uncertain or dynamic environments. Reinforcement learning enabled a significant breakthrough (Hwangbo et al., 2019), allowing robots to learn locomotion policies with a level of robustness that was difficult to achieve with traditional, model-based techniques (Lee et al., 2020; Miki et al., 2022). Because this learning process requires vast amounts of data, simulation has become an indispensable component for developing controllers safely and efficiently (Rudin et al., 2022), with successful deployment across several robotic platforms, as depicted in Figure 22.

The RAI Institute released the first open-source end-to-end training pipeline for Boston Dynamics' Spot using Isaac Lab, along with a method to close the sim-to-real gap via evolutionary strategies (Miller et al., 2025), achieving zero-shot transfer from simulation to hardware with running speeds up to 5.2 m/s, nearly three times the default limit. Their approach closely models hardware-specific dynamics, including actuator delays and joint torque limits, using custom actuator classes in Isaac Lab to enable robust and dynamic real-world performance. Wen et al. (2025) optimized a style-imitation objective with constraints to learn locomotion for quadruped and humanoid from imperfect demonstrations. Arm et al. (2025) explored how to develop and validate a locomotion controller and a base pose controller in gravity environments from lunar gravity to a hypothetical super-Earth. Cathomen et al. (2025) proposed an unsupervised skill discovery method to learn various locomotion skills for a quadruped without explicit task-specific rewards. Li et al. (2025) created an open-source multi-agent soccer environment and use multi-agent reinforcement learning (MARL) to train decentralized policies for sophisticated team play behavior. Arnold et al. (2025) used Isaac Lab to train an RL controller for a wheeled quadruped capable of transportation of materials over challenging terrains.

Although locomotion can be learned with purely proprioceptive information, incorporating exteroceptive feedback from sensors such as LiDARs or cameras can greatly increase the capabilities of learned controllers. For example, Rudin et al. (2025) trained a quadruped robot to climb obstacles and traverse unstructured terrain using depth images as input to an end-to-end visuomotor policy. It first applies reinforcement learning on a simplified terrain representation, then distills the policy with depth-image observations, and finally performs RL fine-tuning. The required algorithms are tightly integrated with Isaac Lab through the RSL-RL

Figure 22: Platforms using Isaac Lab for learning robust and agile locomotion. Top to right: (a) Boston Dynamics Spot (Miller et al., 2025), (b) Magnecko (Arm et al., 2025), (c) LEVA (Arnold et al., 2025), (d) ANYmal (Rudin et al., 2025), and (e) RAI UMV (RAI Institute, 2025).

library (Schwarke et al., 2025). Bjelonic et al. (2025) investigate bridging the sim-to-real gap for multiple legged robots by employing evolutionary algorithms for system identification, tuning simulation parameters to better match real-world robot trajectories.

Beyond legged locomotion, Isaac Lab has also been applied to wheeled platforms, such as the Ultra Mobility Vehicle (UMV), developed by the RAI Institute (Figure 23). UMV enhances the bicycle form with a custom jumping mechanism, enabling both efficient wheeled locomotion and dynamic legged behaviors, such as hopping, flipping, and obstacle clearance. Achieving these behaviors requires accurate modeling of wheel–ground dynamics and the use of domain randomization to capture the uncertainties of high-energy impacts. Isaac Lab supports both aspects, allowing UMV to deploy RL policies with robust sim-to-real performance. This demonstrates how Isaac Lab facilitates high-performance mobility research on robotic systems that extend beyond conventional legged designs (RAI Institute, 2025).

6.2. Whole-body control

Whole-Body Control (WBC) considers the robot as a single, integrated system, enabling simultaneous coordination of locomotion, manipulation, and environmental interaction across all available degrees of freedom. This holistic approach allows robots to perform complex, multi-objective tasks—such as walking while carrying objects, stabilizing against external forces, or adjusting posture in real time to maintain balance—making WBC essential for deployment in unstructured and dynamic environments. However, achieving effective WBC in practice presents significant challenges. It requires resolving multiple, often competing objectives (*e.g.* balance vs. manipulation accuracy), handling kinematic and dynamic constraints, and managing the interactions between subsystems such as arms, legs, and torso. Additionally, WBC must be robust to modeling inaccuracies, sensor noise, and latency in perception and actuation. Despite these complexities, recent advances have accelerated progress in the field. Learning-based methods, motion retargeting from human demonstrations, and task prioritization strategies are increasingly integrated into modern WBC pipelines. Isaac Lab has further enabled scalable development and evaluation of WBC policies, offering high-fidelity simulation, physics realism,

Figure 23: Left: The ANYmal robot interacting with the rocks by using the robot's foot (Arm et al., 2024). Right: Boston Dynamics eAtlas robot to perform a wide array of athletic skills, sourced from diverse human motion data, including video and motion capture (RAI Institute, 2025).

and seamless deployment on real-world hardware.

Building on the idea of leveraging humanoid motion datasets for general-purpose control, He et al. (2025) propose a policy distillation framework that unifies different control modes under a single WBC policy. A remake of the HOVER framework with Isaac Lab (Biswas et al., 2025) enhances the original implementation with extensive Sim2Sim and Sim2Real evaluations, demonstrating the effectiveness of Isaac Lab training in comparison to the initial implementation in IsaacGym. Sleiman et al. (2024) applied DeepMimic-style training to long-horizon loco-manipulation tasks, where a quadrupedal robot with an arm learns to interact with articulated objects like dishwashers and spring-loaded doors. Stolle et al. (2024) introduced a perceptive pedipulation policy that uses elevation maps to perform local collision avoidance while tracking foot positions, allowing the robot to safely navigate around dynamic obstacles using contact filtering in Isaac Lab.

On the control side, Portela et al. (2025) developed methods for robust end-effector pose tracking on rough terrains, while Vijayan et al. (2025) proposed a multi-critic setup for smoother whole-body control through robust end-effector twist tracking. Dadiotis et al. (2025) developed a learning-based controller for a mobile manipulator to move an unknown object to a desired position and yaw orientation through a sequence of pushing actions. The proposed controller for the robotic arm and the mobile base motion is trained using a constrained RL formulation.

Isaac Lab has also been used to develop RL policies for humanoid platforms such as Boston Dynamics' Atlas (Boston Dynamics, 2025; RAI Institute, 2025). Leveraging human motion capture and animation data, the trained policies enable a diverse set of whole-body movements, including army crawling, shoulder rolls, handstand forward rolls, and breakdance motions (Figure 23). This work showcases rich multi-contact behaviors and demonstrates Isaac Lab's ability to support RL pipelines that extend humanoid control beyond conventional locomotion. Together, these works push humanoid capabilities beyond locomotion into rich loco-manipulation and robust interaction in unstructured environments.

6.3. Navigation

Navigation remains a central challenge in robotics, which requires an integration of perception, planning, and control across diverse embodiments and environments. Isaac Lab has rapidly become a preferred framework for learned navigation research, as demonstrated by a growing body of recent works spanning wheeled, legged, aerial, and even aquatic and space robots, as well as different learning paradigms.

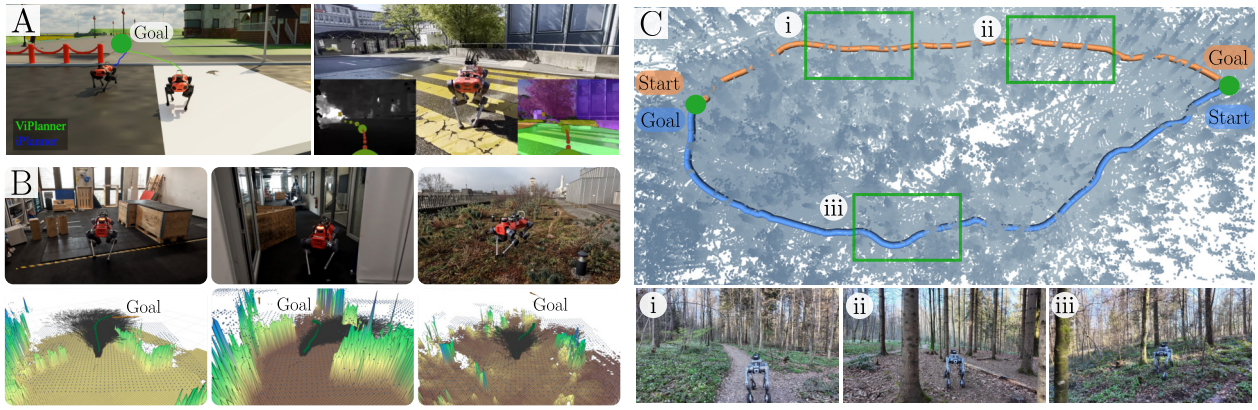


Figure 24: Sim-to-real navigation with Isaac Lab: (a) ViPlaner (Roth et al., 2024) learns an end-to-end policy from depth and semantic images. (b) A perceptive Forward Dynamics Model (Roth et al., 2025) trained in simulation and deployed with a sampling-based planner. (c) An RL navigation policy with a novel memory unit for spatio-temporal reasoning (Yang et al., 2025).

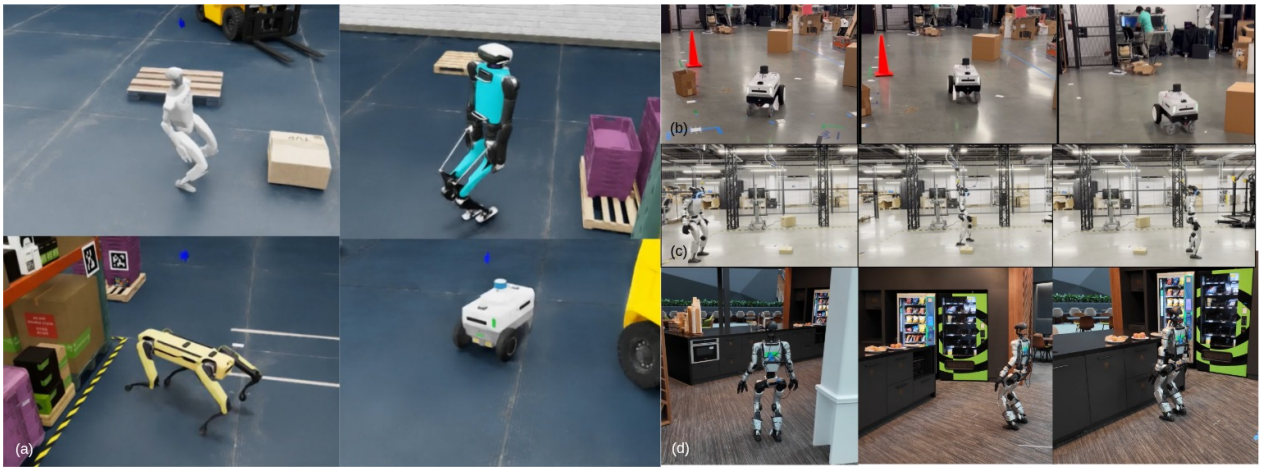


Figure 25: COMPASS (Liu et al., 2025) trains residual RL policies in diverse simulated scenes for cross-embodiment mobility. It enables sim-to-real transfer to a mobile wheeled platform , quadruped and a humanoid. Fine-tuning GR00T N1.5 on COMPASS distillation datasets further demonstrates zero-shot sim-to-real navigation.

Navigation requires policies to have a precise understanding of the environment and the skills of the embodiment. For the former, Isaac Lab's sensor ecosystem (Section 3.2) provides physically and photometrically realistic modalities at scale, making it well-suited for vision-based navigation. This has enabled works such as ViPlanner (Roth et al., 2024), which trains an end-to-end semantic local planner purely with data generated in Isaac Lab (Figure 24), and NaVILA (Cheng et al., 2025), which extends navigation to vision-language-action models for legged robots. By combining sensors with the accurate PhysX simulation of Isaac Lab, researchers at ETH have developed RL based navigation policies with a novel recurrent memory architecture for long-range navigation (Yang et al., 2025). Another line of navigation research focuses on learning forward-dynamics models for safe motion planning, which can be established using the accurate dynamics provided within Isaac Lab (Roth et al., 2025).

A unifying factor across these contributions is Isaac Lab's ability to support *multi-platform training* through the manager-based workflow (Section 3.7.1), where switching between environment scenes and robot embodiments requires only minimal reconfiguration while preserving realistic physical behavior. Building on this foundation,

a central goal in robot learning is to develop generalizable navigation policies that transfer across embodiments and adapt efficiently to real-world conditions. COMPASS (Liu et al., 2025), a novel workflow for developing cross-embodiment mobility policies by integrating imitation learning, residual RL, and policy distillation, has demonstrated the effectiveness of RL fine-tuning and strong zero-shot sim-to-real performance using Isaac Lab (see Figure 25). COMPASS can also provide navigation specialist policies for generating large-scale synthetic datasets in Isaac Lab to train advanced VLA foundation models (*e.g.* Gr00t N1.5). By leveraging COMPASS, the USD scenes generated from the real-to-sim NuRec pipeline, such as these example assets, can be used to further reduce the sim-to-real gap.

6.4. Industrial Manipulation

Industrial manipulation frequently involves contact-rich interactions, where robots must maintain sustained physical contact with objects and precisely regulate forces and motions under uncertainty. In contrast to simple pick-and-place operations, such tasks demand careful handling of friction, compliance, and alignment, and are essential for assembly processes like peg insertion, gear meshing, threaded-fastener mating, and snap-fit assembly. Isaac Lab provides simulation capabilities, robotic assembly environments, and algorithms for simulating and learning skills for precise contact-rich manipulation.

The Factory environments combine SDF-based contact generation, a contact reduction technique, and a Gauss–Seidel solver for efficient contact simulation (Narang et al., 2022). Policies trained in these environments can achieve zero-shot sim-to-real transfer with 83–99% success rates (Tang et al., 2023). The AutoMate environments extend these capabilities towards more challenging geometries and generalizable assembly by combining reinforcement and imitation learning and train multi-task policies (Tang et al., 2024). The environments include 100 simulation-compatible assembly assets, specialist policies for ∼80 tasks, and a distilled generalist policy for 20 tasks, all achieving around 80% success rates both in simulation and in the real world. See Figure 26.

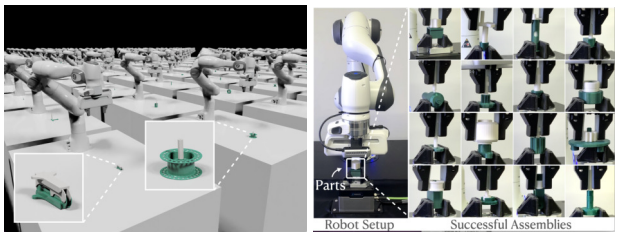


Figure 26: Assembly Environments in Isaac Lab (Tang et al., 2023). Left: Simulation. Right: Real World.

SRSA (Guo et al., 2025) enables continual learning by selecting and fine-tuning pre-trained skills from a task library based on task similarity, while MatchMaker (Wang et al., 2025) procedurally expands this library for scalable training. FORGE (Noseworthy et al., 2025) introduces force-aware environments with features such as adaptive force regulation, thresholding, and randomized dynamics to enable robust execution of precision tasks like snap-fit insertion and gear meshing under uncertainty. Complementing these approaches, TacSL (Akinola et al., 2025) uses visuotactile sensing to handle occlusion and lighting variation, enabling precise contact-rich manipulation with reliable sim-to-real transfer, supported by Isaac Lab's tactile simulation tools.

In addition to assembly tasks, grasping is also a foundational task in robotic manipulation, requiring precise coordination between sensing, planning, and control to enable robots to reliably interact with objects in diverse and often unstructured environments. Isaac Lab advances this capability by providing high-fidelity simulation, accurate contact modeling, and GPU-accelerated data generation pipelines. vMF-Contact (Shi et al., 2024) introduced a novel probabilistic architecture for learning hierarchical contact grasp representations and used Isaac Lab for both data collection and task-level grasp evaluation. GraspDataGen (Carlson, 2025) (see Figure 27) recently introduced a new Isaac Lab-based pipeline for modular 6 DoF grasp sampling and evaluation conditioned on object meshes. Several common industrial pinch grippers (Robotiq 2F-85, 2F-140, RG6, Franka-Panda) are supported out of the box and can be used to sample a dense set of Isaac Lab verified grasp poses for custom objects. Additionally, GraspQP (Zurbrügg et al., 2025) introduced modular Isaac Lab environments that extends grasp evaluation to multi-DoF grippers, including ShadowHand, AbilityHand,

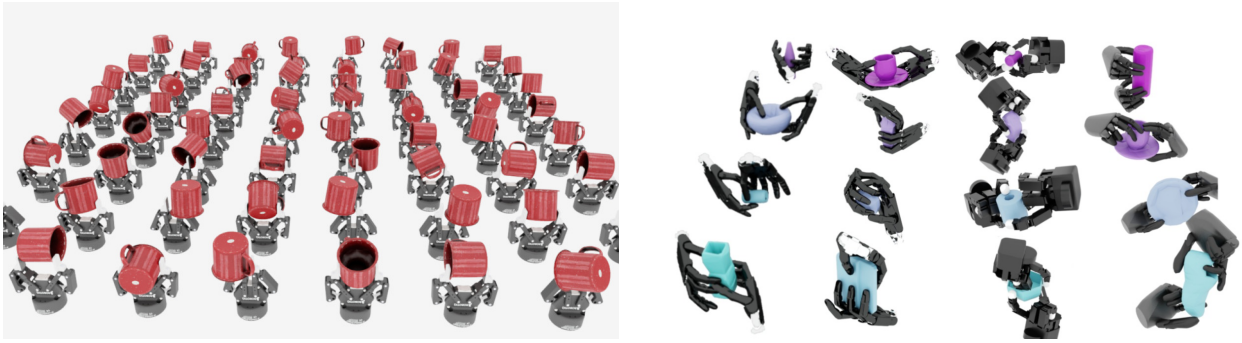
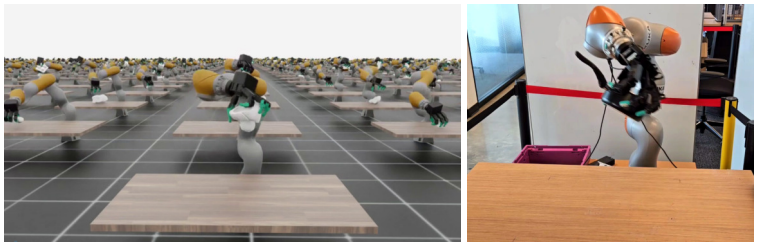


Figure 27: Left: GraspDataGen (Carlson, 2025) exhaustively evaluates 6 DoF grasps in Issac Lab given an object and gripper asset. Right: GraspQP (Zurbrügg et al., 2025) uses Isaac Lab to evaluate grasp candidates, synthesized using an analytical formulation, on multi-DoF grippers .

Allegro, and both two- and three-fingered Robotiq grippers.

6.5. Dexterous Manipulation

Dexterous manipulation with multi-fingered hands remains challenging compared to standard parallel-jaw grasping due to the high-dimensional action space and fine-grained control required for coordinated finger movements and in-hand object manipulation. While most grasping systems operate open-loop and predict gripper poses for simple contact, they often fall short in dynamic tasks or tasks with high precision requirements. Isaac Lab enables the development of advanced dexterous manipulation policies by offering high-fidelity simulation of multi-DOF hands, accurate contact modeling, and support for rich sensory inputs, including vision and proprioception. It allows for scalable policy training and deployment of learned dexterous manipulation policies on real-world robotic platforms through features such as domain randomization and tiled rendering.



(a) Simulation (b) Real

Figure 28: Left: DextrAH (Singh et al., 2025) environment training in simulation. Right: The trained policy deployed in the real world.

DextrAH-RGB (Singh et al., 2025) trained an RL policy on the KUKA arm with an Allegro hand in simulation that used privileged state information and then distilled this into a network that takes stereo RGB pairs as input. This is the first system to have shown that an end-to-end network directly operating on raw RGB streams can control arm and multi-fingered hands, all leveraging the high-quality rendering from Isaac Lab (Figure 28). More recently, Singh et al. (2025) leveraged Isaac Lab to train depth-based end-to-end policies from scratch for the DextrAH (Lum et al., 2024; Singh et al., 2025) task.

Perceptive Dexterous Humanoid Control (PDC) (Luo et al., 2025) demonstrated vision-driven whole-body control of simulated humanoids (shown in Figure 29), closing the perception–action loop through egocentric visual input. Prior approaches often rely on privileged object states from simulation, limiting the emergence of human-like behaviors such as active search. The PDC framework addresses this by using egocentric vision and proprioception only, introducing a perception-as-interface paradigm with visual cues (*e.g.* masks, 3D markers, hand indicators) for tasks like search, grasp, placement, and drawer manipulation. Leveraging Isaac Lab's large-scale GPU-accelerated simulation, policies are trained via RL with motion priors (Luo et al., 2023), scaling across procedurally generated kitchens and tabletop tasks. Results show that training directly from pixels, rather than distilling from state-based policies, led to better generalization and more natural behaviors. This highlights the power of Isaac Lab as a platform for scalable visual RL in complex loco-manipulation tasks.

Figure 29: Perceptive Dexterous Control (PDC) (Luo et al., 2025) enables a humanoid equipped with egocentric vision to search, reach, grasp, and manipulate objects in cluttered kitchen scenes. PDC uses visual perception as the sole indicator of which hand to use, which object to grasp, where to move, and which drawer to open.

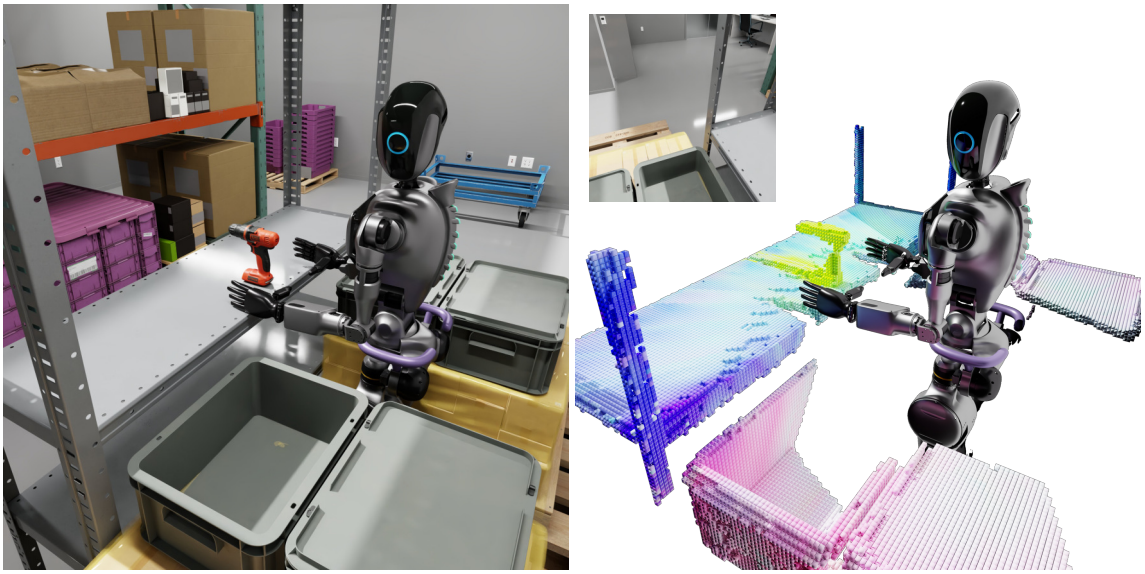


Figure 30: Spatial Memory Task (Steiner et al., 2025). A humanoid in a simulated industrial space (left) and within a metric-semantic reconstruction built by Mindmap (right) (colored by Principal Component Analysis). By using the reconstruction, Mindmap generates trajectories that depend on parts of the scene that are outside the robot's Field Of View.

Mindmap (Steiner et al., 2025), shown in Figure 30, contributes tools for extending 3D manipulation policies with spatial memory, including an imitation learning pipeline in Isaac Lab that utilizes a metric-semantic map built using *nvblox* (Millane et al., 2024). The authors demonstrate the efficacy of these tools by extending a state-of-the-art 3D diffusion policy (Ke et al., 2024), and demonstrated significantly improved success rates on challenging manipulation tasks that require spatial memory.

6.6. Healthcare

Autonomy in healthcare robotics presents unique challenges compared to other domains, such as manipulation, locomotion, and autonomous driving, due to the need for high-precision, contact-rich interaction with soft tissue and delicate instruments. Progress in this domain has been limited by the absence of domain-specific simulation tools capable of capturing the complexities of clinical environments. Simulation has become central to addressing these gaps, enabling scalable data generation, safe policy training, and sim-to-real transfer. Recent advances have focused on building high-fidelity digital twins tailored for healthcare workflows, accelerating the development of intelligent robotic systems across surgical, diagnostic, and teleoperated applications.

Figure 31: Top Row: (Left) Ultrasound simulation in Isaac for Healthcare. (Right) Digital twin of human anatomy in NVIDIA Omniverse. Bottom Row: Training fine-grained robotic maneuvers performed during surgery and the hands-on training exercises used in tabletop surgical curricula.

Isaac Lab powers the simulation backbone of Isaac for Healthcare (NVIDIA), a domain-specific developer framework designed to accelerate the development of intelligent healthcare robotic systems. It addresses key challenges such as simulating anatomical and procedural complexity, integrating clinical data, and supporting robust sim-to-real transfer. High-fidelity digital twins with contact modeling and photorealistic rendering enable scalable training, synthetic data generation, and teleoperation pipelines essential for surgical autonomy. As part of a broader stack spanning AI model training, simulation, and clinical deployment, Isaac Lab plays a central role in healthcare robotics as it supports digital prototyping, hardware-in-the-loop training, and teleoperation-based imitation learning via XR and peripheral interfaces – making it a versatile platform for developing dexterous robotic policies using GPU-accelerated reinforcement and imitation learning.

Isaac Lab's simulation infrastructure is actively enabling key Isaac for Healthcare workflows in robotic surgery, ultrasound, and telesurgery (Figure 31). In robotic surgery (Moghani et al., 2025), it enables photorealistic, physics-based simulation of surgical tasks such as needle manipulation and tool control. In ultrasound, GPU-accelerated ray tracing simulates acoustic wave propagation to produce realistic B-mode images. For telesurgery, Isaac Lab supports the development and deployment of remote systems across varying network conditions.

Yu et al. (2024) build on Isaac Lab to deliver an enhanced digital twin simulator tailored for surgical robotics, featuring contact-rich physical interactions, photorealistic rendering in NVIDIA Omniverse, and GPU-accelerated physics. It provides a benchmark for surgical tasks that represent core subtasks in surgical training. ORBIT-Surgical enables scalable reinforcement and imitation learning, teleoperation workflows, and synthetic data generation for active perception. The platform demonstrates successful sim-to-real transfer of learned policies for zero-shot deployment of needle manipulation skills on a physical surgical robot, highlighting its effectiveness in accelerating the development of autonomous surgical systems.

Additionally, Ao et al. (2025) is a scalable simulation platform for advancing autonomy in orthopedic surgery, developed on the Isaac Lab infrastructure. It features anatomically realistic, CT-derived 3D patient models

and supports ultrasound simulation using both physics-based and generative methods. SonoGym enables large-scale, parallel training and evaluation across diverse anatomical variations, providing a robust testbed for ultrasound-guided navigation and anatomical reconstruction.

6.7. Generalist Foundation Models

Foundation models for robotics aim to generalize across tasks, embodiments, and environments by leveraging large-scale, diverse data during pretraining — similar to advances in vision and language domains. GR00T N1 (NVIDIA et al., 2025) is an open foundation model for generalist humanoid robots, built as a vision-language-action model using NVIDIA's Eagle VLM and a Diffusion Transformer (DiT) to integrate visual, textual, and action data. It is pre-trained on a mixture of real robot demonstrations, Internet-scale video, and large-scale synthetic data generated using the Mimic pipeline in Isaac Lab. Synthetic data plays a key role in overcoming the scarcity of real-world data, enhancing robustness and generalization. The model can be post-trained with new demonstrations, collected or synthesized in Isaac Lab, for task specialization, and can be further improved via sim-and-real co-training (Maddukuri et al., 2025). The upgraded GR00T N1.5 improves language grounding, generalization, and real-world performance using architectural refinements and the FLARE training technique (Zheng et al., 2025), which introduces action prediction and implicit world modeling objectives.

Given a foundation model, such as GR00T N1.5, we can perform *online* post-training with RL. While this is difficult to perform in the real world, Isaac Lab's high-fidelity rendering and multi-environment setup enables sample-efficient online post-training to be done safely and efficiently in simulation. This post-training process can be scaled to a wide range of new tasks with sample-efficient RL techniques (Luo et al., 2024), residual RL, and learned reward models for dense feedback during online fine-tuning (Zhang et al., 2025). As the field advances towards real-world RL fine-tuning, this simulation-first approach within Isaac Lab is a crucial part of the pipeline for ensuring policies are robust and safe before being deployed on physical hardware.

7. Conclusion

Isaac Lab represents a significant advancement in simulation tooling for robotics, unifying high-fidelity physics, scalable rendering, and modular architecture into a single, extensible framework. Built on NVIDIA Isaac Sim and leveraging PhysX and RTX rendering, Isaac Lab delivers high-throughput simulation with support for both direct and manager-based learning workflows. Its key contributions include accurate and performant sensor simulation and actuation modeling, an extensive collection of environments, and tools for motion generation and imitation learning, enabling workflows for training foundation models, RL fine-tuning, and large-scale perception systems. The framework integrates seamlessly with popular learning libraries and adheres to industry-standard APIs, facilitating rapid experimentation and deployment.

By bridging the gap between performance and flexibility, Isaac Lab is poised to have a profound impact on both robotics research and industrial development. It provides GPU-accelerated workflows for developing and benchmarking learning algorithms and offers a scalable and photorealistic platform to simulate complex robotics systems, enabling safer, faster, and more robust robotics development cycles.

Ongoing development of Isaac Lab includes integration with the Newton physics engine, which will introduce even greater physical realism, performance and flexibility for challenging tasks. Combined with continued improvements to rendering, data generation, and scalability, Isaac Lab is positioned to become a central platform for the next generation of robotics innovation, supporting research at scale and accelerating the path from simulation to real-world deployment.

8. Future Work and Discussion

To further advance Isaac Lab, a major upcoming milestone is integration with the Newton physics engine — a GPU-accelerated, extensible, and differentiable simulator offering state-of-the-art solvers for rigid, articulated,

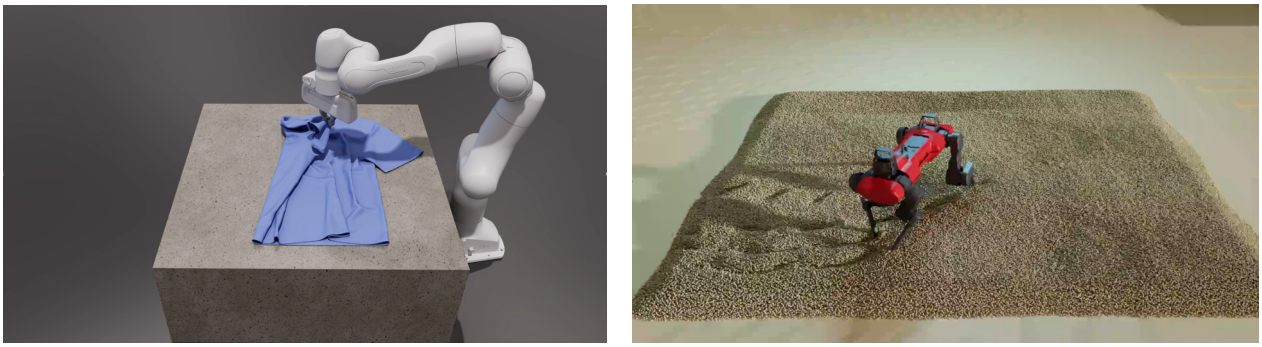


Figure 32: Newton supports multi-physics environments by coupling solvers specialized for different types of dynamics. Left: A Franka arm simulated using MuJoCo folds cloth simulated with Vertex Block Descent (VBD) solver(Chen et al., 2024) . Right: An ANYmal robot simulated with PhysX maneuvers through a non-rigid terrain simulated with Multiple Material Method (MPM) solver(Daviet and Bertails-Descoubes, 2016).

and deformable body dynamics (Figure 32). This addresses key limitations of existing engines in complex robotic scenarios. Alongside physics improvements, future releases will enhance rendering capabilities through deeper integration with NVIDIA's RTX real-time ray tracing technologies, enabling more photorealistic and physically accurate visuals for vision-based learning. Efforts will also focus on boosting performance and scalability to support large-scale perception training. Additionally, we aim to build a standardized platform for policy evaluation and benchmarking across diverse robotic domains. By incorporating a broad suite of learning-friendly environments, the platform will support rigorous, reproducible evaluation and promote widespread adoption as a unified framework for advancing generalizable robotic policies.

8.1. Newton Engine and Isaac Lab Integration

Newton is an open-source, GPU-accelerated physics simulation engine explicitly designed for roboticists and simulation researchers. Developed through a collaborative effort by NVIDIA, Google DeepMind, and Disney Research, Newton aims to advance robot learning and development by providing a robust, scalable, and extensible platform for physical AI. Built upon NVIDIA Warp (Macklin, 2022), a developer framework for accelerating simulation and spatial computing, Newton extends and generalizes Warp's existing simulation functionality, integrating MuJoCo Warp as a primary backend. It emphasizes GPU-based computation, differentiability, and user-defined extensibility, facilitating rapid iteration and scalable robotics simulation.

8.1.1. Key Characteristics and Features

Newton is distinguished by several characteristics that cater to the requirements of modern robotics research:

- **Open-Source and Community-Driven**: Newton is an open-source project with source code available on GitHub and distributed via PyPI.
- **GPU-Accelerated Performance**: Leveraging NVIDIA Warp and CUDA graphs, Newton delivers high-performance, end-to-end GPU simulations without low-level programming, eliminating CPU bottlenecks common in older physics engines.
- **Extensibility**: Newton's modular design enables easy integration of custom solvers and models, supporting realistic multiphysics simulations with diverse materials like food, cloth, soil, and cables.
- **Differentiability**: Newton's automatic differentiation of inputs, states, controls, and parameters accelerates policy training, design optimization, and system identification for efficient, gradient-based robot learning.
- **Unified API**: Newton offers a single, consistent programming interface for interacting with various physical simulations — including rigid bodies, soft bodies, granular material, and cloth — within the same framework and across multiple solvers.

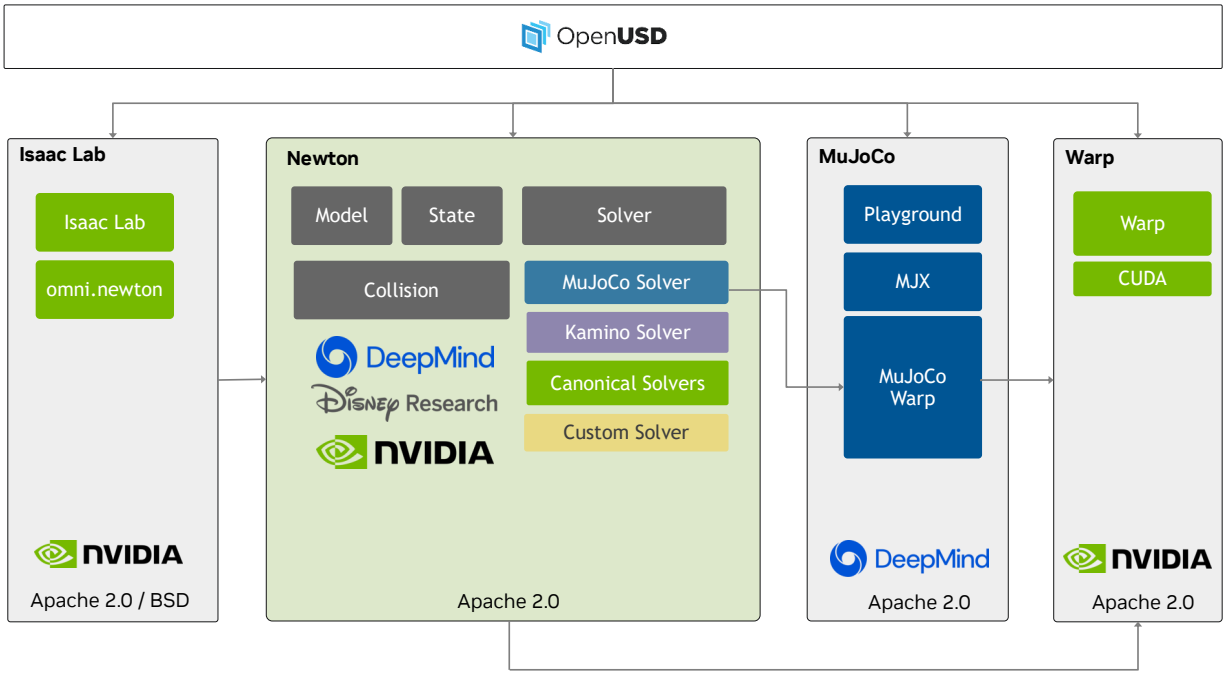


Figure 33: The Newton Physics Engine is an open-source, GPU-accelerated simulation engine built upon NVIDIA Warp, designed for roboticists and simulation researchers. Its architecture separates the **Model** (non-time-varying system definition) and **State** (time-varying physical configuration) for clarity and flexibility, while various **Solvers** advance the simulation over time by integrating physics. This design, emphasizing flat data structures, enables seamless integration with Deep Learning (DL) frameworks like PyTorch and JAX, and platforms such as Isaac Lab via the `omni.newton` extension, with key solvers like the MuJoCo Warp built on the same GPU-accelerated foundation.

- **OpenUSD Integration**: The engine utilizes the OpenUSD framework, benefiting from its flexible data model and composition engine to aggregate data describing robots and their environments.

8.1.2. Architecture and Design Principles

Newton's architecture is structured around a clear separation of concerns, and is designed for flexibility and interoperability with DL frameworks.

- **High-level architecture**: Newton organizes simulation around a few core components: `newton.ModelBuilder` for constructing models, `newton.Model` for physical structure, `newton.State` for dynamic data, and `newton.Solver` for advancing simulation. It supports importing URDF, MJCF, and USD assets via `newton.Importer` and integrates easily with platforms like Isaac Lab.
- **Separating physical model from numerical method**: Newton separates the time-invariant physical model from the time-stepping solver. This allows different solvers to be applied to the same model, optimizing for various dynamic regimes.
- **Flat data over Object Oriented Programming (OOP)**: Simulation data is represented as tensors and flat arrays, not deep class hierarchies. This design aligns with ML frameworks like PyTorch and JAX, enabling efficient vectorization, JIT compilation, zero-copy interoperability where possible and easy integration into learning workflows.
- **No hidden state**: All internal state is exposed, giving users full control over memory and computation. Solvers act like pure functions — data in, data out — with any state mutation made explicit for clarity and composability.

- **Modular, "take what you need"**: Newton is modular, from low-level geometry up to full solvers. Users can integrate only the components they need, supporting both lightweight prototypes and full production systems.
- **Flexible Selection API**: Similar to PhysX's Tensor API, Newton's Selection API allows creating specialized views of entities in the scene and enables batched, zero-copy access to all attributes in **Model**, **State**, and **Control** objects of scene subsets, such as specific robots or manipulators. It supports named attribute access, joint/link filtering, and custom ordering — making it easy to extend and integrate with ML pipelines and custom solvers.

8.1.3. Solver Implementations

Newton supports a diverse array of solver implementations including a mix of explicit and implicit methods, as well as reduced and maximal coordinate approaches, each offering unique trade-offs in terms of accuracy, memory usage, and performance.

- **Canonical Solvers**: Out-of-the-box, Newton includes classical solvers such as **XPBD** (Macklin et al., 2016), **Featherstone** (Featherstone, 1984), and **Semi-Implicit Euler**. These solvers are generally lightweight implementations of well-established methods as reference for implementers or solver developers.
- **New Solvers**: An important new development in Newton is the **MuJoCo Solver** based on the MuJoCo Warp library, which is a full re-implementation of the MuJoCo (Todorov et al., 2012) algorithms built using NVIDIA Warp. This provides significant performance gains over previous JAX-based MuJoCo XLA (MJX) implementations, particularly for complex scenes involving numerous contacts (*e.g.* dexterous manipulation, humanoid locomotion), without requiring manual contact pruning. In addition, Newton will include a dedicated maximal coordinate solver called the **Kamino Solver** from Disney Research designed to robustly handle systems with closed loops (Tsounis et al., 2025).
- **Solver Extensions**: In addition to built-in solvers, many partners are building their simulators on Newton, such as specialized IPC (Li et al., 2020) solvers for tactile manipulation (Li et al., 2025), and specialized solvers for cloth dynamics, e.g.: **Style3D Solver**.

Newton's solver abstraction supports mixed systems encompassing rigid bodies, cloth, and particle simulations. Multiple solvers can run independently to manage these diverse dynamics, with ongoing development focused on achieving two-way coupling for more intricate interactions.

8.1.4. Newton USD as a Staging Schema for USD Physics Standardization

As part of the AOUSD's initiative to advance USD as a descriptive language for Physical AI, new schemas are being developed to represent the physical properties of robotic systems in an engine-agnostic manner. This effort is aligned with Newton's solver-abstraction API, which also aims to generalize across simulation backends. Developed alongside the Newton API, the Newton USD schema serves as a staging ground where generalized simulation parameters are identified and formalized, with the express goal of proposing them for future inclusion in the USD Physics standard.

8.1.5. Newton Integration with Isaac Lab

Integration of Isaac Lab with the Newton physics engine is currently underway and accessible on Isaac Lab repository in an experimental feature branch. This early-stage integration supports a subset of robotics environments, including reinforcement learning tasks for flat-terrain locomotion, manipulation, and vision-based workflows. These implementations serve as testbeds to evaluate Newton's performance, stability, and physical accuracy through representative robotic workflows. As part of this integration effort, we have conducted both sim-to-sim comparisons against the existing PhysX backend, as well as sim-to-real validation on physical hardware platforms. These tests aim to characterize policy transferability between simulation engines and real-world deployment.

Ongoing development will focus on expanding Newton support across the full Isaac Lab feature set — including

additional sensor and actuator simulation and learning environments — with the objective of achieving feature parity with the existing PhysX backend in the coming months. This work represents a critical step toward establishing Newton as the default physics engine for high-fidelity and scalable robotics simulation within Isaac Lab. Community engagement is encouraged as we refine this integration and continue to ensure robust support for both research and industrial workflows.

8.2. Policy Evaluation and Benchmarks

8.2.1. Isaac Lab - Arena

Isaac Lab – Arena is designed to streamline and scale robotic policy evaluation. While Isaac Lab's manager-based workflow offers powerful configuration capabilities, achieving end-to-end simulation — from asset preparation to environment setup and large-scale policy evaluation — often requires significant manual effort. This leads to fragmented setups with high overhead, limited scalability, and a steep entry barrier. Arena introduces a systematic, scalable approach to evaluation, built on Isaac Lab. It provides a robust framework for setting up and executing complex experiments with minimal infrastructure overhead — serving as a launchpad to make advanced simulation-based experimentation more accessible and efficient.

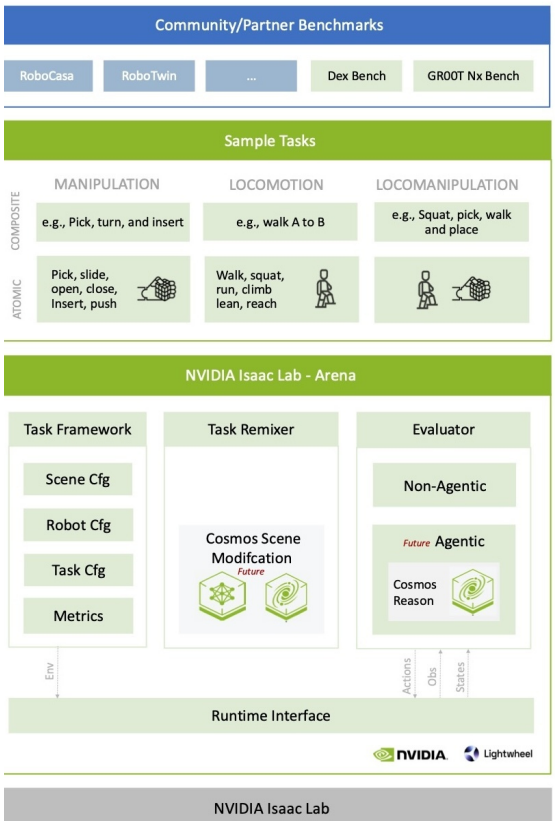


Figure 34: The architecture diagram of **Isaac Lab - Arena**, a framework and collaborative ecosystem for accessible and scalable policy evaluation in simulation.

The framework supports simplified, composable task definitions for easy customization and scene diversification, as outlined in Figure 34. It includes extensible libraries for metrics, data collection, and evaluation — starting with rule-based methods and soon expanding to include neural and agent-based approaches. Arena enables parallel, GPU-accelerated evaluations and integrates seamlessly with data generation, training and deployment frameworks to support closed-loop workflows. It also includes a growing library of sample tasks across manipulation, locomotion, and loco-manipulation.

NVIDIA is actively collaborating with policy developers, benchmark authors, and simulation partners such as Lightwheel to co-develop Arena, use it to accelerate their evaluations, and enable the contribution of their methods and benchmarks back to the community. Isaac Lab – Arena will be open-sourced on GitHub soon.

8.2.2. Assembly Benchmark

Isaac Lab provides optimized environments for evaluating and benchmarking robotic manipulation policies on contact-rich tasks. These tasks correspond to a subset of the National Institute of Standards and Technology (NIST) Assembly Task Board 1 (Kimble et al., 2020). Previous works (Noseworthy et al., 2025; Tang et al., 2023, 2024) have demonstrated that the policies trained and evaluated in these environments can be effectively transferred to real robots. A natural next step is to extend these contact-rich environments to cover the full set of NIST benchmark tasks, further broadening their applicability and impact.

8.2.3. Dexterous Manipulation Suite

Isaac Lab currently provides a basic manipulation task suite — comprising of lifting, grasping, and reorienting tasks with the KUKA Allegro hand — available under the dexsuite module. These tasks build on prior works from DextrAH (Lum et al., 2024; Singh et al., 2025) and DexPBT (Petrenko et al., 2023). Looking ahead, we plan to significantly expand this suite to include more complex and realistic scenarios across industrial, logistics, and domestic-service domains. In particular, we aim to target humanoid platforms equipped with multi-fingered hands, operating in unstructured home environments — thereby broadening the applicability of Isaac Lab to real-world, high-dexterity tasks.

A. Contributors and Acknowledgments

Core Contributors

Mayank Mittal, *NVIDIA, ETH*
Yunrong Guo, *NVIDIA*
Pascal Roth, *NVIDIA, ETH*
David Hoeller, *NVIDIA, Flexion Robotics*
James Tigue, *RAI*
Antoine Richard, *NVIDIA*
Octi Zhang, *NVIDIA*
Peter Du, *NVIDIA*
Antonio Serrano-Muñoz, *NVIDIA*
Xinjie Yao, *NVIDIA*
René Zurbrügg, *ETH*
Nikita Rudin, *NVIDIA, Flexion Robotics*

Core Contributors - Physics

Lukasz Wawrzyniak, *NVIDIA*
Milad Rakhsha, *NVIDIA*
Alain Denzler, *NVIDIA*
Eric Heiden, *NVIDIA*
Ales Borovicka, *NVIDIA*

Contributors

Ossama Ahmed, *NVIDIA*
Iretiayo Akinola, *NVIDIA*
Abrar Anwar, *NVIDIA*
Mark T. Carlson, *NVIDIA*
Ji Yuan Feng, *NVIDIA*
Animesh Garg, *NVIDIA, GaTech*
Renato Gasoto, *NVIDIA*
Lionel Gulich, *NVIDIA*
Yijie Guo, *NVIDIA*
M. Gussert, *NVIDIA*
Ankur Handa, *NVIDIA*
Alex Hansen, *RAI*
Chenran Li, *NVIDIA*
Wei Liu, *NVIDIA*
Hammad Mazhar, *NVIDIA*
Masoud Moghani, *NVIDIA*, *UofT*
Adithyavairavan Murali, *NVIDIA*
Michael Noseworthy, *MIT*
Alexander Poddubny, *NVIDIA*
Nathan Ratliff, *NVIDIA*
Clemens Schwarke, *NVIDIA*, *ETH*
Ritvik Singh, *NVIDIA*, *UC Berkeley*
James Latham Smith, *RAI*
Remo Steiner, *NVIDIA*
Bingjie Tang, *NVIDIA, USC*
Ruchik Thaker, *NVIDIA*
Matthew Trepte, *NVIDIA*
Karl Van Wyk, *NVIDIA*
Fangzhou Yu, *RAI*

Contributors - Arena

Alex Millane, *NVIDIA*
Vikram Ramasamy, *NVIDIA*
Sangeeta Subramanian, *NVIDIA*
Clemens Volk, *NVIDIA*

Contributors - Mimic

CY Chen, *NVIDIA*
Neel Jawale, *NVIDIA*
Ashwin Varghese Kuruttukulam, *NVIDIA*
Michael A. Lin, *NVIDIA*
Ajay Mandlekar, *NVIDIA*
Karsten Patzwaldt, *NVIDIA*
John Welsh, *NVIDIA*
Huihua Zhao, *NVIDIA*

Contributors - Physics

Chris Amevor, *NVIDIA*
Jan Carius, *NVIDIA*
Jumyung Chang, *NVIDIA*
Anka He Chen, *NVIDIA*
Pablo de Heras Ciechomski, *NVIDIA*
Gilles Daviet, *NVIDIA*
Mohammad Mohajerani, *NVIDIA*
Julia von Muralt, *NVIDIA*
Viktor Reutskyy, *NVIDIA*
Michael Sauter, *NVIDIA*
Simon Schirm, *NVIDIA*
Eric L. Shi, *NVIDIA*
Pierre Terdiman, *NVIDIA*
Kenny Vilella, *NVIDIA*
Tobias Widmer, *NVIDIA*
Gordon Yeoman, *NVIDIA*

Contributors - XR Teleoperation

Tiffany Chen, *NVIDIA*
Sergey Grizan, *NVIDIA*
Cathy Li, *NVIDIA*
Lotus Li, *NVIDIA*
Connor Smith, *NVIDIA*
Rafael Wiltz, *NVIDIA*

Contributors - Omniverse

Fatima Anes, *NVIDIA*
Jean-Francois Lafleche, *NVIDIA*
Nicolas Moënne-Loccoz, *NVIDIA*
Soowan Park (박수완), *NVIDIA*
Rob Stepinski, *NVIDIA*
Dirk Van Gelder, *NVIDIA*

Core Leadership

Yunrong Guo, *NVIDIA*
David Hoeller, *NVIDIA, Flexion Robotics*
Gavriel State, *NVIDIA*

Leadership

Yan Chang, *NVIDIA*
David Chu, *NVIDIA*
Farbod Farshidian, *RAI*
Ankur Handa, *NVIDIA*
Spencer Huang, *NVIDIA*
Marco Hutter, *ETH, RAI*
Yashraj Narang, *NVIDIA*
Soha Pouya, *NVIDIA*
Shiwei Sheng, *NVIDIA*
Yuke Zhu, *NVIDIA*, *UT Austin*

Leadership - Physics

Miles Macklin, *NVIDIA*
Adam Moravanszky, *NVIDIA*
Philipp Reist, *NVIDIA*

We use the following abbreviations for affiliations:

- **ETH** – Swiss Federal Institute of Technology, Zürich, Switzerland
- **RAI** – Robotics and AI Institute, United States
- **MIT** – Massachusetts Institute of Technology, Cambridge, Massachusetts, United States
- **UC Berkeley** – University of California, Berkeley, United States
- **USC** – University of Southern California, Los Angeles, United States
- **UT Austin** – The University of Texas at Austin, Austin, United States
- **UofT** - University of Toronto, Toronto, Canada
- **GaTech** - Georgia Institute of Technology, Atlanta, United States

---

The roles are defined as follows:

Core Contributor: Individuals who had a very significant and sustained impact throughout the project, across multiple subsystems, with accountability for key design and implementation decisions. Core Contributors are listed in order of merit.

Contributor: Individuals who made important contributions to the project. Contributors are listed in alphabetical order. Contributions include:

- Major code and documentation contributions to the main codebase
- Substantial engineering or production work on key underlying simulation technologies, such as the PhysX and Newton physics engines, the Omniverse RTX rendering system and related technologies, or Isaac Sim
- Authorship of research papers referenced in this white paper whose individual contributions significantly influenced the design and implementation of Isaac Lab
- Authorship of significant portions of this white paper

Core Leadership: Individuals with primary responsibility for the organizational and technical direction of the Isaac Lab effort.

Leadership: Other individuals involved in organizational and technical direction for Isaac Lab.

Acknowledgments

The development of Isaac Lab initiated from the Orbit framework (Mittal et al., 2023). We gratefully acknowledge the authors of Orbit for their foundational contributions.

Isaac Lab relies heavily on core technology from NVIDIA Omniverse and Isaac Sim, such as the Omniverse RTX Renderer and USD conversion tools for robot description formats. While certain NVIDIA staff members have contributed directly to the Isaac Lab project and are recognized above as contributors, we express our deep appreciation for everyone involved, including members of the open-source community working on connected projects such as Open USD.

We further thank the open-source community for their active engagement in the development of Isaac Lab. The project has benefited from code contributions, issue reports, documentation improvements, and valuable feedback from users worldwide. A complete and regularly updated list of contributors is available on Isaac Lab GitHub repository.

We are especially grateful to Philip Arm, Arjun Bhardwaj, Filip Bjelonic, Nicola Burger, Rafael Cathomen, Ioannis Dadiotis, Clemens Eppner, Oliver Fischer, Per Frivik, Caelen Garrett, Ayush Ghosh, Tairan He, Matthias Heyrman, Jason Jingzhou Liu, Victor Klemm, Chenhao Li, Zichong Li, Tyler Ga Wei Lum, Zhengyi Luo, Gary Lvov, Denys Makoviichuk, Viktor Makoviychuk, Takahiro Miki, AJ Miller, Kyle Morgenstein, Özhan Özen, Aleksei Petrenko, Tifanny Portela, Jean-Pierre Sleiman, Remo Steiner, Jonas Stolle, Balakumar Sundaralingam, Lorenzo Terenzi, David Tingdahl, and Fan Yang for their research contributions and constructive feedback.

Finally, we acknowledge the use of OpenAI GPT-5, which helped to refine the manuscript language. All technical content, analyses, and citations were generated and verified by the authors. We note that large language models may exhibit biases, errors, or omissions, and we take full responsibility for the accuracy and appropriateness of the manuscript.

B. Acronyms

AEC Agent Environment Cycle
AOUSD Alliance for OpenUSD
API Application Programming Interface
CPU Central Processing Unit
DLSS Deep Learning Super Sampling
DoF Degrees-of-Freedom
DR Domain Randomization
FEM Finite Element Method
IK Inverse Kinematics
IL Imitation Learning
GPU Graphics Processing Unit
LiDAR Light Detection and Ranging
MARL Multi-Agent Reinforcement Learning
MDL Material Definition Language
MJCF MuJoCo Modeling XML File
PBD Position-Based Dynamics
PBT Population-Based Training
RL Reinforcement Learning
SDF Simulation Description Format
URDF Unified Robot Description Format
USD Universal Scene Description

C. Version History

This whitepaper will evolve alongside the Isaac Lab codebase. Each version of the document corresponds to a specific release of the framework, and the following entries summarize the most significant updates. For detailed updates, please check the Release Notes on GitHub.

Version	Summary of Changes
`v2.3` (2025-09-30)	Initial release of Isaac Lab whitepaper. Covers successor relationship to Isaac Gym, modular architecture, sensor simulation, teleoperation, and data collection.

References

[1] Ananye Agarwal, Ashish Kumar, Jitendra Malik, and Deepak Pathak. Legged locomotion in challenging terrains using egocentric vision. In *Conference on Robot Learning (CoRL)*, pages 403–415. PMLR, 2023. 2

[2] Iretiayo Akinola, Jie Xu, Jan Carius, Dieter Fox, and Yashraj Narang. TacSL: A library for visuotactile sensor simulation and learning. *T-RO*, 41:2645–2661, 2025. 12, 32

[3] Arthur Allshire, Mayank Mittal, Varun Lodaya, Viktor Makoviychuk, Denys Makoviichuk, Felix Widmaier, Manuel Wüthrich, Stefan Bauer, Ankur Handa, and Animesh Garg. Transferring dexterous manipulation from gpu simulation to a remote real-world trifinger. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022. 2

[4] Yunke Ao, Masoud Moghani, Mayank Mittal, Manish Prajapat, Luohong Wu, Frederic Giraud, Fabio Carrillo, Andreas Krause, and Philipp Fürnstahl. Sonogym: High performance simulation for challenging surgical tasks with robotic ultrasound. *arXiv preprint arXiv:2507.01152*, 2025. 35

[5] Philip Arm, Mayank Mittal, Hendrik Kolvenbach, and Marco Hutter. Pedipulate: Enabling manipulation skills using a quadruped robot's leg. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 5717–5723. IEEE, 2024. 30

[6] Philip Arm, Oliver Fischer, Joseph Church, Adrian Fuhrer, Hendrik Kolvenbach, and Marco Hutter. Efficient learning-based control of a legged robot in lunar gravity. *arXiv preprint arXiv:2509.10128*, 2025. 28, 29

[7] Marco Arnold, Lukas Hildebrandt, Kaspar Janssen, Efe Ongan, Pascal Bürge, Ádám Gyula Gábriel, James Kennedy, Rishi Lolla, Quanisha Oppliger, Micha Schaaf, et al. Leva: A high-mobility logistic vehicle with legged suspension. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2025. 28, 29

[8] Qingwei Ben, Feiyu Jia, Jia Zeng, Junting Dong, Dahua Lin, and Jiangmiao Pang. Homie: Humanoid loco-manipulation with isomorphic exoskeleton cockpit. *arXiv preprint arXiv:2502.13013*, 2025. 27

[9] Joydeep Biswas, Yan Chang, Jim Fan, Pulkit Goyal, Lionel Gulich, Tairan He, Rushane Hua, Neel Jawale, H. Hawkeye King, Chenran Li, Michael Lin, Wei Liu, Zhengyi Luo, Billy Okal, Stephan Pleines, Soha Pouya, Peter Varvak, Wenli Xiao, Huihua Zhao, and Yuke Zhu. Hover: Versatile neural whole-body controller for humanoid robots. https://github.com/NVlabs/HOVER, March 2025. NVIDIA Isaac. 30

[10] Filip Bjelonic, Fabian Tischhauser, and Marco Hutter. Towards bridging the gap: Systematic sim-to-real transfer for diverse legged robots. *arXiv preprint arXiv:2509.06342*, 2025. 29

[11] Boston Dynamics. Walk, Run, Crawl, RL Fun | Boston Dynamics | Atlas. https://www.youtube.com/watch?v=I44_zbEwz_w, 2025. Accessed: 2025-09-04. 30

[12] Remi Cadene, Simon Alibert, Alexander Soare, Quentin Gallouedec, Adil Zouitine, Steven Palma, Pepijn Kooijmans, Michel Aractingi, Mustafa Shukor, Dana Aubakirova, Martino Russi, Francesco Capuano, Caroline Pascal, Jade Choghari, Jess Moss, and Thomas Wolf. Lerobot: State-of-the-art machine learning for real-world robotics in pytorch. https://github.com/huggingface/lerobot, 2024. 26

[13] Mark Carlson. GraspDataGen. https://github.com/NVlabs/GraspDataGen, 2025. Isaac Lab-based Grasp Generation. 32, 33

[14] Rafael Cathomen, Mayank Mittal, Marin Vlastelica, and Marco Hutter. Divide, discover, deploy: Factorized skill learning with symmetry and style priors. *Conference on Robot Learning (CoRL)*, 2025. 28

[15] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3d: Learning from rgb-d data in indoor environments. *arXiv preprint arXiv:1709.06158*, 2017. 17

[16] Anka He Chen, Ziheng Liu, Yin Yang, and Cem Yuksel. Vertex block descent. *arXiv preprint*, arXiv:2403.06321, 2024. doi: 10.48550/arXiv.2403.06321. URL https://arxiv.org/abs/2403.06321. 37

[17] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Conference on Robot Learning (CoRL)*, pages 66–75. PMLR, 2020. 24

[18] An-Chieh Cheng, Yandong Ji, Zhaojing Yang, Xueyan Zou, Jan Kautz, Erdem Biyik, Hongxu Yin, Sifei Liu, and Xiaolong Wang. Navila: Legged robot vision-language-action model for navigation. In *Robotics: Science and Systems (RSS)*, 2025. 31

[19] Xuxin Cheng, Jialong Li, Shiqi Yang, Ge Yang, and Xiaolong Wang. Open-television: Teleoperation with immersive active visual feedback. *arXiv preprint arXiv:2407.01512*, 2024. 16

[20] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org, 2016–2023. 2

[21] Ioannis Dadiotis, Mayank Mittal, Nikos Tsagarakis, and Marco Hutter. Dynamic object goal pushing with mobile manipulators through model-free constrained reinforcement learning. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2025. 30

[22] Gilles Daviet and Florence Bertails-Descoubes. A semi-implicit material point method for the continuum simulation of granular materials. *ACM Transactions on Graphics*, 35(4):102:1–102:13, 2016. doi: 10.1145/2897824.2925877. URL https://dl.acm.org/doi/10.1145/2897824.2925877. 37

[23] Matt Deitke, Dustin Schwenk, Jordi Salvador, Luca Weihs, Oscar Michel, Eli VanderBilt, Ludwig Schmidt, Kiana Ehsani, Aniruddha Kembhavi, and Ali Farhadi. Objaverse: A universe of annotated 3d objects. In *Computer Vision and Pattern Recognition*, 2023. 16

[24] Clemens Eppner, Adithyavairavan Murali, Caelan Garrett, Rowland O'Flaherty, Tucker Hermans, Wei Yang, and Dieter Fox. scene_synthesizer: A python library for procedural scene generation in robot manipulation. *Journal of Open Source Software*, 2024. 16

[25] Dawson-Haggerty et al. Trimesh. https://trimesh.org/. Version 3.2.0, accessed 2025-09-29. 16

[26] Roy Featherstone. Robot dynamics algorithms. *Annexe thesis digitisation project 2016 block 5*, 1984. 39

[27] Zipeng Fu, Xuxin Cheng, and Deepak Pathak. Deep whole-body control: learning a unified policy for manipulation and locomotion. In *Conference on Robot Learning (CoRL)*, pages 138–149. PMLR, 2023. 2

[28] Caelan Garrett, Ajay Mandlekar, Bowen Wen, and Dieter Fox. Skillmimicgen: Automated demonstration generation for efficient skill learning and deployment. In *Conference on Robot Learning (CoRL)*, 2024. 28

[29] Yijie Guo, Bingjie Tang, Iretiayo Akinola, Dieter Fox, Abhishek Gupta, and Yashraj Narang. Srsa: Skill retrieval and adaptation for robotic assembly tasks. *arXiv preprint arXiv:2503.04538*, 2025. 32

[30] Ankur Handa, Arthur Allshire, Viktor Makoviychuk, Aleksei Petrenko, Ritvik Singh, Jingzhou Liu, Denys Makoviichuk, Karl Van Wyk, Alexander Zhurkevich, Balakumar Sundaralingam, et al. DeXtreme: Transfer of agile in-hand manipulation from simulation to reality. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 5977–5984. IEEE, 2023. 2

[31] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 23

[32] Tairan He, Zhengyi Luo, Xialin He, Wenli Xiao, Chong Zhang, Weinan Zhang, Kris Kitani, Changliu Liu, and Guanya Shi. Omnih2o: Universal and dexterous human-to-humanoid whole-body teleoperation and learning. *Conference on Robot Learning (CoRL)*, 2024. 2

[33] Tairan He, Wenli Xiao, Toru Lin, Zhengyi Luo, Zhenjia Xu, Zhenyu Jiang, Jan Kautz, Changliu Liu, Guanya Shi, Xiaolong Wang, et al. Hover: Versatile neural whole-body controller for humanoid robots. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2025. 30

[34] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26): eaau5872, 2019. 14

[35] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26): eaau5872, 2019. 28

[36] Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. Population based training of neural networks. *CoRR*, abs/1711.09846, 2017. URL http://arxiv.org/abs/1711.09846. 24

[37] Zhenyu Jiang, Yuqi Xie, Kevin Lin, Zhenjia Xu, Weikang Wan, Ajay Mandlekar, Linxi Fan, and Yuke Zhu. Dexmimicgen: Automated data generation for bimanual dexterous manipulation via imitation learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2025. 26

[38] Tsung-Wei Ke, Nikolaos Gkanatsios, and Katerina Fragkiadaki. 3d diffuser actor: Policy diffusion with 3d scene representations. *arXiv preprint arXiv:2402.10885*, 2024. 34

[39] Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 3d gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4), July 2023. URL https://repo-sam.inria.fr/fungraph/3d-gaussian-splatting/. 8

[40] Kenneth Kimble, Karl Van Wyk, Joe Falco, Elena Messina, Yu Sun, Mizuho Shibata, Wataru Uemura, and Yasuyoshi Yokokohji. Benchmarking protocols for evaluating small parts robotic assembly systems. *IEEE robotics and automation letters*, 5(2):883–889, 2020. 40

[41] Nathan Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2149–2154. IEEE, 2004. 3

[42] Jeongseok Lee, Michael X. Grey, Sehoon Ha, Tobias Kunz, Sumit Jain, Yuting Ye, Siddhartha S. Srinivasa, Mike Stilman, and C. Karen Liu. DART: Dynamic animation and robotics toolkit. *The Journal of Open Source Software*, 3(22):500, Feb 2018. doi: 10.21105/joss.00500. URL https://doi.org/10.21105/joss.00500. 2

[43] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5(47):eabc5986, 2020. 24, 28

[44] Minchen Li, Zachary Ferguson, Teseo Schneider, Timothy Langlois, Denis Zorin, Daniele Panozzo, Chenfanfu Jiang, and Danny M. Kaufman. Incremental potential contact: Intersection- and inversion-free large deformation dynamics. *ACM Trans. Graph. (SIGGRAPH)*, 39(4), 2020. 39

[45] Yuyang Li, Wenxin Du, Chang Yu, Puhao Li, Zihang Zhao, Tengyu Liu, Chenfanfu Jiang, Yixin Zhu, and Siyuan Huang. Taccel: Scaling up vision-based tactile robotics via high-performance gpu simulation, 2025. 39

[46] Zichong Li, Filip Bjelonic, Victor Klemm, and Marco Hutter. Marladona-towards cooperative team play using multi-agent reinforcement learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 15014–15020. IEEE, 2025. 28

[47] Wei Liu, Huihua Zhao, Chenran Li, Joydeep Biswas, Soha Pouya, and Yan Chang. Compass: Cross-embodiment mobility policy via residual rl and skill synthesis. *arXiv preprint arXiv:2502.16372*, 2025. 8, 31, 32

[48] Tyler Ga Wei Lum, Albert H. Li, Preston Culbertson, Krishnan Srinivasan, Aaron Ames, Mac Schwager, and Jeannette Bohg. Get a grip: Multi-finger grasp evaluation at scale enables robust sim-to-real transfer. In *CoRL*, 2024. 2

[49] Tyler Ga Wei Lum, Martin Matak, Viktor Makoviychuk, Ankur Handa, Arthur Allshire, Tucker Hermans, Nathan D. Ratliff, and Karl Van Wyk. Dextrah-G: Pixels-to-action dexterous arm-hand grasping with geometric fabrics, 2024. URL https://arxiv.org/abs/2407.02274. 21, 24, 33, 41

[50] Jianlan Luo, Zheyuan Hu, Charles Xu, You Liang Tan, Jacob Berg, Archit Sharma, Stefan Schaal, Chelsea Finn, Abhishek Gupta, and Sergey Levine. Serl: A software suite for sample-efficient robotic reinforcement learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 16961–16969. IEEE, 2024. 36

[51] Zhengyi Luo, Jinkun Cao, Josh Merel, Alexander Winkler, Jing Huang, Kris Kitani, and Weipeng Xu. Universal humanoid motion representations for physics-based control. *arXiv preprint arXiv:2310.04582*, 2023. 33

[52] Zhengyi Luo, Chen Tessler, Toru Lin, Ye Yuan, Tairan He, Wenli Xiao, Yunrong Guo, Gal Chechik, Kris Kitani, Linxi Fan, et al. Emergent active perception and dexterity of simulated humanoids from visual reinforcement learning. *arXiv preprint arXiv:2505.12278*, 2025. 16, 24, 33, 34

[53] Miles Macklin. Warp: A high-performance python framework for gpu simulation and graphics. https://github.com/nvidia/warp, March 2022. NVIDIA GPU Technology Conference (GTC). 12, 37

[54] Miles Macklin, Matthias Müller, and Nuttapong Chentanez. Xpbd: position-based simulation of compliant constrained dynamics. In *Proceedings of the 9th International Conference on Motion in Games*, MIG '16, page 49–54, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450345927. doi: 10.1145/2994258.2994272. URL https://doi.org/10.1145/2994258.2994272. 39

[55] Abhiram Maddukuri, Zhenyu Jiang, Lawrence Yunliang Chen, Soroush Nasiriany, Yuqi Xie, Yu Fang, Wenqi Huang, Zu Wang, Zhenjia Xu, Nikita Chernyadev, Scott Reed, Ken Goldberg, Ajay Mandlekar, Linxi Fan, and Yuke Zhu. Sim-and-real co-training: A simple recipe for vision-based robotic manipulation. In *Proceedings of Robotics: Science and Systems (RSS)*, Los Angeles, CA, USA, 2025. 36

[56] Denys Makoviichuk and Viktor Makoviychuk. rl-games: A high-performance framework for reinforcement learning. https://github.com/Denys88/rl_games, May 2022. 23

[57] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac gym: High performance gpu based physics simulation for robot learning. In J. Vanschoren and S. Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks*, volume 1. Curran, 2021. 2, 6, 18

[58] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martin-Martin. What matters in learning from offline human demonstrations for robot manipulation. In *Conference on Robot Learning (CoRL)*, pages 1678–1690. PMLR, 2022. 26

[59] Ajay Mandlekar, Soroush Nasiriany, Bowen Wen, Iretiayo Akinola, Yashraj Narang, Linxi Fan, Yuke Zhu, and Dieter Fox. Mimicgen: A data generation system for scalable robot learning using human demonstrations. In *Conference on Robot Learning (CoRL)*, 2023. 26

[60] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62), 2022. 12, 28

[61] Alexander Millane, Helen Oleynikova, Emilie Wirbel, Remo Steiner, Vikram Ramasamy, David Tingdahl, and Roland Siegwart. nvblox: Gpu-accelerated incremental signed distance field mapping. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2698–2705, 2024. 34

[62] AJ Miller, Fangzhou Yu, Michael Brauckmann, and Farbod Farshidian. High-performance reinforcement learning on spot: Optimizing simulation parameters with distributional measures. *arXiv preprint arXiv:2504.17857*, 2025. 28, 29

[63] Mayank Mittal, Calvin Yu, Qinxi Yu, Jingzhou Liu, Nikita Rudin, David Hoeller, Jia Lin Yuan, Ritvik Singh, Yunrong Guo, Hammad Mazhar, et al. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters (RA-L)*, 8(6):3740–3747, 2023. 2, 44

[64] Masoud Moghani, Nigel Nelson, Mohamed Ghanem, Andres Diaz-Pinto, Kush Hari, Mahdi Azizian, Ken Goldberg, Sean Huver, and Animesh Garg. Sufia-bc: Generating high quality demonstration data for visuomotor policy learning in surgical subtasks. *arXiv preprint arXiv:2504.14857*, 2025. 35

[65] Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging ai applications. In *13th USENIX symposium on operating systems design and implementation (OSDI 18)*, pages 561–577, 2018. 23

[66] Yashraj Narang, Kier Storey, Iretiayo Akinola, Miles Macklin, Philipp Reist, Lukasz Wawrzyniak, Yunrong Guo, Adam Moravanszky, Gavriel State, Michelle Lu, et al. Factory: Fast contact for robotic assembly. *Robotics: Science and Systems (RSS)*, 2022. 2, 19, 32

[67] Newton Contributors. Newton: GPU-accelerated physics simulation for robotics and simulation research. https://github.com/newton-physics/newton. Newton, a Series of LF Projects, LLC, Apache-2.0 License, accessed 2025-09-29. 3

[68] Michael Noseworthy, Bingjie Tang, Bowen Wen, Ankur Handa, Chad Kessens, Nicholas Roy, Dieter Fox, Fabio Ramos, Yashraj Narang, and Iretiayo Akinola. Forge: Force-guided exploration for robust contact-rich manipulation under uncertainty. *IEEE Robotics and Automation Letters*, 2025. 19, 32, 40

[69] NVIDIA. Cloudxr, . URL https://developer.nvidia.com/cloudxr-sdk. 16

[70] NVIDIA. NVIDIA Isaac for Healthcare. https://github.com/isaac-for-healthcare, . Accessed 2025-09-29. 35

[71] NVIDIA, Nikita Cherniadev Johan Bjorck andFernando Castañeda, Xingye Da, Runyu Ding, Linxi "Jim" Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, Joel Jang, Zhenyu Jiang, Jan Kautz, Kaushil

Kundalia, Lawrence Lao, Zhiqi Li, Zongyu Lin, Kevin Lin, Guilin Liu, Edith Llontop, Loic Magne, Ajay Mandlekar, Avnish Narayan, Soroush Nasiriany, Scott Reed, You Liang Tan, Guanzhi Wang, Zu Wang, Jing Wang, Qi Wang, Jiannan Xiang, Yuqi Xie, Yinzhen Xu, Zhenjia Xu, Seonghyeon Ye, Zhiding Yu, Ao Zhang, Hao Zhang, Yizhou Zhao, Ruijie Zheng, and Yuke Zhu. GR00T N1: An open foundation model for generalist humanoid robots. In *ArXiv Preprint*, March 2025. 36

[72] OpenAI, Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving rubik's cube with a robot hand, 2019. URL https://arxiv.org/abs/1910.07113. 26

[73] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *CoRR*, abs/1710.06537, 2017. URL http://arxiv.org/abs/1710.06537. 25

[74] Aleksei Petrenko, Arthur Allshire, Gavriel State, Ankur Handa, and Viktor Makoviychuk. DexPBT: Scaling up dexterous manipulation for hand-arm systems with population based training, 2023. URL https://arxiv.org/abs/2305.12127. 19, 24, 25, 41

[75] Pixar Animation Studios. Universal scene description (openusd), 2016. URL https://openusd.org. Accessed: 2025-09-16. 4

[76] Tifanny Portela, Andrei Cramariuc, Mayank Mittal, and Marco Hutter. Whole-body end-effector pose tracking. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2025. 30

[77] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research (JMLR)*, 2021. 23

[78] RAI Institute. Reinforcement Learning Accelerates Humanoid Behavior Production. https://www.youtube.com/watch?v=LQizdUn5Z1k, 2025. Accessed: 2025-09-22. 30

[79] RAI Institute. Ultra Mobility Vehicle: Combining Wheeled Efficiency with Legged Agility. https://rai-inst.com/resources/blog/designing-wheeled-robotic-systems/, 2025. Accessed: 2025-09-03. 29

[80] Alexander Raistrick, Lingjie Mei, Karhan Kayan, David Yan, Yiming Zuo, Beining Han, Hongyu Wen, Meenal Parakh, Stamatis Alexandropoulos, Lahav Lipson, Zeyu Ma, and Jia Deng. Infinigen indoors: Photorealistic indoor scenes using procedural generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 21783–21794, June 2024. 16

[81] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635. JMLR Workshop and Conference Proceedings, 2011. 24

[82] Pascal Roth, Julian Nubert, Fan Yang, Mayank Mittal, and Marco Hutter. Viplanner: Visual semantic imperative learning for local navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 5243–5249. IEEE, 2024. 31

[83] Pascal Roth, Jonas Frey, Cesar Cadena, and Marco Hutter. Learned perceptive forward dynamics model for safe and platform-aware robotic navigation. In *Robotics: Science and Systems (RSS)*, 2025. 31

[84] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on Robot Learning (CoRL)*, pages 91–100. PMLR, 2022. 2, 12, 14, 19, 28

[85] Nikita Rudin, Junzhe He, Joshua Aurand, and Marco Hutter. Parkour in the wild: Learning a general and extensible agile locomotion policy using multi-expert distillation and rl fine-tuning. *arXiv preprint arXiv:2505.11164*, 2025. 28, 29

[86] Fereshteh Sadeghi and Sergey Levine. CAD2RL: Real single-image flight without a single real image. *arXiv preprint arXiv:1611.04201*, 2016. 25

[87] Clemens Schwarke, Mayank Mittal, Nikita Rudin, David Hoeller, and Marco Hutter. RSL-RL: A learning library for robotics research. *arXiv preprint arXiv:2509.10771*, 2025. 23, 29

[88] Antonio Serrano-Munoz, Dimitrios Chrysostomou, Simon Bøgh, and Nestor Arana-Arexolaleiba. skrl: Modular and flexible library for reinforcement learning. *Journal of Machine Learning Research (JMLR)*, 24(254):1–9, 2023. 23

[89] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*, 2017. URL https://arxiv.org/abs/1705.05065. 3

[90] Jinghuan Shang, Karl Schmeckpeper, Brandon B. May, Maria Vittoria Minniti, Tarik Kelestemur, David Watkins, and Laura Herlant. Theia: Distilling diverse vision foundation models for robot learning. In *Conference on Robot Learning (CoRL)*, 2024. URL https://openreview.net/forum?id=ylZHvlwUcI. 23

[91] Yitian Shi, Edgar Welte, Maximilian Gilles, and Rania Rayyes. vmf-contact: Uncertainty-aware evidential learning for probabilistic contact-grasp in noisy clutter. *arXiv preprint arXiv:2411.03591*, 2024. 32

[92] Zilin Si and Wenzhen Yuan. Taxim: An example-based simulation model for gelsight tactile sensors. *IEEE Robotics and Automation Letters*, 7(2):2361–2368, 2022. 12

[93] Ritvik Singh, Arthur Allshire, Ankur Handa, Nathan Ratliff, and Karl Van Wyk. Dextrah-RGB: Visuomotor policies to grasp anything with dexterous hands, 2025. URL https://arxiv.org/abs/2412.01791. 19, 23, 24, 26, 33, 41

[94] Ritvik Singh, Karl Van Wyk, Pieter Abbeel, Jitendra Malik, Nathan Ratliff, and Ankur Handa. End-to-end RL improves dexterous grasping policies, 2025. URL https://arxiv.org/abs/2509.16434. 21, 24, 33

[95] Jean-Pierre Sleiman, Mayank Mittal, and Marco Hutter. Guided reinforcement learning for robust multi-contact loco-manipulation. In *Conference on Robot Learning (CoRL)*, 2024. 16, 30

[96] Remo Steiner, Alex Millane, Clemens Volk, David Tingdahl, Vikram Ramasamy, Xinjie Yao, Peter Du, Soha Pouya, Shiwei Sheng, et al. mindmap: Spatial memory in deep feature maps for 3d action policies. In *CoRL workshop on RememberRL*, 2025. 34

[97] Jonas Stolle, Philip Arm, Mayank Mittal, and Marco Hutter. Perceptive pedipulation with local obstacle avoidance. In *Proceedings of the IEEE-RAS International Conference on Humanoid Robots (ICHR)*, pages 157–164, 2024. 30

[98] Julian Straub, Thomas Whelan, Lingni Ma, Yufan Chen, Erik Wijmans, Simon Green, Jakob J Engel, Raul Mur-Artal, Carl Ren, Shobhit Verma, et al. The replica dataset: A digital replica of indoor spaces. *arXiv preprint arXiv:1906.05797*, 2019. 16

[99] Balakumar Sundaralingam, Siva Kumar Sastry Hari, Adam Fishman, Caelan Garrett, Karl Van Wyk, Valts Blukis, Alexander Millane, Helen Oleynikova, Ankur Handa, Fabio Ramos, Nathan Ratliff, and Dieter Fox. Curobo: Parallelized collision-free robot motion generation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 8112–8119, 2023. doi: 10.1109/ICRA48891.2023.10160765. 15

[100] Bingjie Tang, Michael A Lin, Iretiayo Akinola, Ankur Handa, Gaurav S Sukhatme, Fabio Ramos, Dieter Fox, and Yashraj Narang. Industreal: Transferring contact-rich assembly tasks from simulation to reality. *arXiv preprint arXiv:2305.17110*, 2023. 32, 40

[101] Bingjie Tang, Iretiayo Akinola, Jie Xu, Bowen Wen, Ankur Handa, Karl Van Wyk, Dieter Fox, Gaurav S Sukhatme, Fabio Ramos, and Yashraj Narang. Automate: Specialist and generalist assembly policies over diverse geometries. *arXiv preprint arXiv:2407.08028*, 2024. 19, 32, 40

[102] Stone Tao, Fanbo Xiang, Arth Shukla, Yuzhe Qin, Xander Hinrichsen, Xiaodi Yuan, Chen Bao, Xinsong Lin, Yulin Liu, Tse-kai Chan, et al. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *Robotics: Science and Systems (RSS)*, 2025. 2

[103] J Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34:15032–15043, 2021. 20

[104] Joshua Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. *CoRR*, abs/1703.06907, 2017. URL http://arxiv.org/abs/1703.06907. 25

[105] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109. 2, 39

[106] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv e-prints*, pages arXiv–2407, 2024. 23

[107] Vassilios Tsounis, Ruben Grandia, and Moritz Bächer. On solving the dynamics of constrained rigid multi-body systems with kinematic loops, 2025. URL https://arxiv.org/abs/2504.19771. 39

[108] Aravind Elanjimattathil Vijayan, Andrei Cramariuc, Mattia Risiglione, Christian Gehring, and Marco Hutter. Multi-critic learning for whole-body end-effector twist tracking. *Conference on Robot Learning (CoRL)*, 2025. 30

[109] Ruicheng Wang, Jialiang Zhang, Jiayi Chen, Yinzhen Xu, Puhao Li, Tengyu Liu, and He Wang. Dexgraspnet: A large-scale robotic dexterous grasp dataset for general objects based on simulation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 11359–11366. IEEE, 2023. 2

[110] Yian Wang, Bingjie Tang, Chuang Gan, Dieter Fox, Kaichun Mo, Yashraj Narang, and Iretiayo Akinola. Matchmaker: Automated asset generation for robotic assembly. *arXiv preprint arXiv:2503.05887*, 2025. 32

[111] Kehan Wen, Chenhao Li, Junzhe He, and Marco Hutter. Constrained style learning from imperfect demonstrations under task optimality. *CoRL*, 2025. 28

[112] Qi Wu, Janick Martinez Esturo, Ashkan Mirzaei, Nicolas Moenne-Loccoz, and Zan Gojcic. 3dgut: Enabling distorted cameras and secondary rays in gaussian splatting. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025. 8

[113] Jie Xu, Sangwoon Kim, Tao Chen, Alberto Rodriguez Garcia, Pulkit Agrawal, Wojciech Matusik, and Shinjiro Sueda. Efficient tactile simulation with differentiability for robotic manipulation. In *CoRL*, 2022. 12

[114] Fan Yang, Per Frivik, David Hoeller, Chen Wang, Cesar Cadena, and Marco Hutter. Improving long-range navigation with spatially-enhanced recurrent memory via end-to-end reinforcement learning. *arXiv preprint arXiv:2506.05997*, 2025. 24, 31

[115] Qinxi Yu, Masoud Moghani, Karthik Dharmarajan, Vincent Schorp, William Chung-Ho Panitch, Jingzhou Liu, Kush Hari, Huang Huang, Mayank Mittal, Ken Goldberg, et al. Orbit-surgical: An open-simulation framework for learning surgical augmented dexterity. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15509–15516. IEEE, 2024. 35

[116] Kevin Zakka, Baruch Tabanpour, Qiayuan Liao, Mustafa Haiderbhai, Samuel Holt, Jing Yuan Luo, Arthur Allshire, Erik Frey, Koushil Sreenath, Lueder A Kahrs, et al. Mujoco playground. *Robotics: Science and Systems (RSS)*, 2025. 2

[117] Jiahui Zhang, Yusen Luo, Abrar Anwar, Sumedh Anand Sontakke, Joseph J Lim, Jesse Thomason, Erdem Biyik, and Jesse Zhang. Rewind: Language-guided rewards teach robot policies without new demonstrations. *Conference on Robot Learning (CoRL)*, 2025. 36

[118] Zhikai Zhang, Chao Chen, Han Xue, Jilong Wang, Sikai Liang, Yun Liu, Zongzhang Zhang, He Wang, and Li Yi. Unleashing humanoid reaching potential via real-world-ready skill space. *arXiv preprint arXiv:2505.10918*, 2025. 16

[119] Ruijie Zheng, Jing Wang, Scott Reed, Johan Bjorck, Yu Fang, Fengyuan Hu, Joel Jang, Kaushil Kundalia, Zongyu Lin, Loic Magne, et al. Flare: Robot learning with implicit world modeling. *Conference on Robot Learning (CoRL)*, 2025. 36

[120] René Zurbrügg, Andrei Cramariuc, and Marco Hutter. Graspqp: Differentiable optimization of force closure for diverse and robust dexterous grasping. *Conference on Robot Learning (CoRL)*, 2025. 32, 33