When Senses Come Online: Order Matters for Robots Too

Isaac Rudnick

Vassar College

April 24, 2026

Senior thesis submitted in partial fulfillment of the requirements

for the major in Cognitive Science

First Reader: Ken Livingston

Second Reader: Josh de Leeuw

Special thanks to Jesse Hayward for his help deploying my code on HPC infrastructure

Overview

While traditional reinforcement learning techniques provide all sensors throughout training, mammals and birds develop their senses in a fixed order, which Turkewitz and Kenny (1982) hypothesized benefits perceptual organization. This thesis tests whether that principle applies to a simulated 4 DoF arm trained through three task phases, comparing three sensor curricula: a classic all-sensors approach, developmentally-ordered sensor introduction, and random dropout. In early phases, the developmentally-ordered approach trains substantially faster, and surpasses random-sensor and matches all-sensor baselines on the most complex task, suggesting that it is a compelling approach for developing a generalizable model without adjusting the algorithm or model architecture.

Table of Contents

Introduction

Robots have been used industrially for over 60 years (Kawasaki Heavy Industries, Ltd., n.d.). Many large-scale assembly processes are fully automated, such as in China's dark factories (Shaikhutdinova, 2025), named for having no lights, since their robots function entirely without lighting and no humans are present. Robots have slowly crept into home and workplace settings (consider 3D printers, for example, which extrude molten plastic at blazing speeds and yet are generally reliable and safe enough for children to use them with moderate training). Traditional robot control relies on deterministic preprogramming: a 3D printer, for example, moves to specified locations and no more. Later generations of robots have added sensors to flag discrepancies for human operators to correct, such as manufacturing failure detection (Jiang, 2019).

However, these robots are a far cry from the vision of a robot butler capable of performing varied household tasks. The next frontier is robots that handle discrepancies themselves, automatically locating parts in cluttered environments **(**Qian et al., 2024**)**, reorienting objects for manipulation (Wada et al., 2022), and learning what a successful outcome looks like with only a few examples (Black et al., 2024). Unlike their declaratively-controlled predecessors, these robots learn much more like humans do, in part because of their use of networks of virtual "neurons." Neural networks (NNs) were inspired by the impressive capability of naturally-occurring NNs in animal brains (McCulloch & Pitts, 1943). Computer and robotic scientists have increasingly looked for inspiration in cognitive science's theories about agents, their embodiment, and their environments in their quest to develop self-correcting, smart robots. Using NNs as a control and learning framework for robotic actuators has been a developing subject for almost forty years (e.g., Miyamoto et al., 1988). Using such a network is often

referred to as a "black box," though, since it is effectively impossible to determine what the patterns of activation in a neural network represent in a way we can understand. This is much the same reason why the brain remains so mysterious to us in contrast to equally important but less computationally complex organs like the heart.

Classical approaches to cognition relied upon an agent constantly updating a large, symbolic knowledge base of the world. While neural networks do not necessarily hold such symbolic information, there are other, simpler approaches to cognition that are even simpler. The embodiment of a robot is deeply connected to not just what it can do in the environment, but also how it does so. The finite constraints of a physical body can be restrictive, but they also allow the environment to work *for* the agent. An agent can, for example, drop things *above* a target position, throw them, balance them, and more. A great example of this is "passive walkers," structures that walk by themselves when placed on a slight slope (Pfeifer & Bongard, 2006). While these are not really agents, it demonstrates that in some cases there is no need for sensors, processing, or even actuators. Thus, such items should only be added if the task requires them, not as a baseline assumption.

Thus, sensors should not be added needlessly. In the real world, sensors can wear and break, provide noisy readings, or drift over time. Even in simulated environments, an excessive amount of inputs can result in longer training times for NN systems. Some sensors also need a lot more adjustment if bridging the sim-to-real gap. A camera, for example, is very hard to simulate. Real-world angle discrepancies, camera settings, and colors can all render a model trained in simulation useless. Other sensors, however, are much more robust. Internal proprioceptive sensors are generally much more reliable and, since they output far less information, they are easier to tune for.

However, just because sensors are noisy doesn't mean you should provide fewer. In fact, to offset this, it is often beneficial to add multiple sensors that provide overlapping information about the world (Kaelbling et al., 1998; Ernst & Banks, 2002). The positioning of these sensors on the robot can greatly affect how it reacts to even the simplest of stimuli (Braitenberg, 1984). Our two eyes, for example, provide information about depth only when they work in conjunction. A prey animal's eyes provide inferior depth perception but a superior field of view. Developing a robot that can accomplish tasks is about not just the software or the design, but also the relationship between the two.

Finally, embodied agents operate continuously. While many early neural networks were used for image recognition, sensorimotor relationships often can't wait: a robot can't afford to pause and think when balancing (O'Regan & Noë, 2001). Thus, we need a system which is capable of quickly intaking information and responding with appropriate actions (without the higher-level reasoning assumed by proponents of classical cognition).

How Robots Learn

While humans often communicate with implied subtext and vague instructions, even those robots that learn must usually be given relatively clear feedback signals or have their digital "brains" (called their architecture) specifically designed for a specific type of learning. Some tasks have a clearly correct output. For example, when training a neural network to classify images as being of cats or dogs, you might have a set of images that you have already labeled. Training an image-classifying network with these pre-labeled datasets is like giving the system a test that you have the answer key to. Instead of "go here, do this" instructions, you're instead giving feedback in the form of a desired output.

However, sometimes the desired output is more than just an answer. For robots performing a task, we often care a lot more about *what* the robot accomplishes than *how* it does so. The internal processes behind the completion of a task may not resemble how we might think the task is done. Goodale et al. (1991) demonstrated that even in humans, our ability to consciously perceive object properties is separate from our visuomotor systems that modify our actions based on those properties. Diedrichsen and Kornysheva (2015) discussed evidence that motor skills are acquired hierarchically. With practice, intermediate representations turn small movements into chunks of actions. These can then be combined into reusable actions across tasks that are still adaptable. Once you learn to pick something up and place it elsewhere, for example, you can repeat that even for objects with a slightly different position, shape, or destination.

A common solution for task training in robots is to give intermediate feedback, a more holistic approach that encourages not an outcome that is either right or wrong, but rather progress towards a goal. Think of instructing an inexperienced robot to learn to bake cookies. This task would be very hard for the robot to learn if you only told them at the end “no, these cookies were bad” but much easier if you give them a score based on how well they did overall (akin to “partial credit” on homework). You may care about some things (what ingredients are mixed together) but not others (how the robot navigates the kitchen). You might also realize that the robot is doing something that is technically in line with your described partial-credit system but not what you hoped for, such as spilling lots of ingredients if there is no penalty for doing so. Thus, this system takes careful tuning so that you account for all desirable (positively-rewarded) and undesirable (negatively-rewarded) behaviors while still leaving space for the robot to find its own way of accomplishing the task. This kind of feedback is instrumental because it allows for

your robot to find the best ways of doing things without needing to follow an exact preprogrammed set of movements.

Let us now imagine we are trying to teach a robot how to make cookies in a *simulated environment* (or many parallel simulated environments). If you want the robot to learn quickly, you can't rely on a human to assign the reward manually; you'd be overwhelmed and your (physical) constraints would be the limiting factor of the training process. Instead, you must define a clear system that the computer uses to reward the robot. Imagine that we start with just rewarding the outcome: 100 points (an arbitrary number without context) for completely baking the cookies. With enough training time and exploration, the robot would eventually stumble into a policy, or mapping from observations to actions, that was capable of baking the cookies. However, this would take an inordinate number of trials to result in success; imagine if you were told that you'd be stuck in a kitchen until *you* figured out *and executed* a specific, unknown recipe. Even the best computer simulations are only so fast. Just as evolutionary selection compounds small advantages over generations, we can define intermediate rewards that guide the policy toward the target behavior. However, the wrong path can lead to mistakes; imagine adding to our reward system +5 points for picking up any tools or ingredients, a choice designed to incentivize their use. Great! The robot will quickly move to boxes or whisks, and pick them up, and put them down, and pick them up, and put them down, and continue forever. This is a "local maximum," since the robot has an opportunity cost (Montevirgen, 2026) in doing something else. Balancing the reward function is tricky, and requires iteration for a system which encourages exploration while still guiding the policy.

Beyond simple reward functions that help with intermediate behaviors, another way to make the task easier for the robot to learn is to decompose it so that smaller tasks can be learned

separately. Eppe et al. (2019) trained robots with two systems, a planner and a worker. The high-level planner provided simple requirements to the worker, such as moving objects to specific positions. This meant the planner didn't need to figure out how to actually manipulate objects and the worker didn't need to plan out multi-step actions, just execute simple ones. Just as a practiced typist does not need to consciously direct every finger while typing, but needs only to think about *what* they wish to type, the planner doesn't need to worry about the low-level action details (see Saveriano et al., 2023, for a broader review of task decomposition methods). While this approach has a higher success rate than classic deep reinforcement learning (RL) policies, it is slower to train. The "curriculum effect" that Eppe et al. built upon is still useful, though, even if a manager task is not present. This was demonstrated by Bengio et al. (2009)'s foundational paper on curriculum learning, which structures tasks. A structured approach allows robots to gain some of the advantages of human compositionality (ability to complete complex tasks) without the high-level ability to understand their actions nor the ability to generalize. This is achieved by training the robot on increasingly complex tasks. This allows it to work its way up to the most complex task, but does not necessarily impart the same level of compositional task ability that Eppe et al. did. However, it is much faster to train without generalizability, and robots need not specialize in every task; sometimes it is worth focusing on just one.

One way to develop a robot's ability to interact with the world for a specific task is an actor-critic system. Essentially, there are two networks trained in parallel: the actor, which proposes an action, and the critic, which guesses what the reward from that action will be. When the agent actually takes the action, the reward from the environment updates both the actor and the critic. It's advantageous to have both; the actor is the policy that gets deployed at inference time, and the critic learns what positions are strong and what might be achievable from them.

This is analogous to the intermediate feedback we gave to the agent learning to make cookies, but the agent itself is learning what kinds of positions are strong. Think of the critic like a chess player's intuition and the actor as the actual decision-making process that chooses a move (Schulman et al., 2017).

Despite some advances in few-shot learning (learning from a handful of examples at most), robots learn very slowly with deep RL (Botvinick et al., 2019). The advantage that robots have that we do not (yet) is the ability to learn in a simulated world. Simulations allow robots to learn at lightning speed, since time can be essentially sped up inside a simulation. Furthermore, multiple simulations can be run in parallel. Simulations are cheaper than real life, too. Simulated robots are free and can be reset from a stuck or broken state automatically. Simulated environments and agents also allow for access to "ground truth" information; you can know exactly where something is, whereas in the real world you often must estimate from sensors. This allows for better training of the robot, since it can receive feedback from precise information (Ibarz et al., 2021). Simulations are also useful because they allow for iteration of the robot's physical design and environment before physical manufacturing needs to begin. They further allow for programmers to iterate on the requirements of the robot by testing how well different types of robots perform, what sensors are most important, and what they can modify about the robot's tools or interactive objects (for example, adding AprilTags).

Whether you take a classic computational approach by programming a simulated robot, or leverage a self-learning system, a critical problem emerges when you deploy the code or policy on a robot embodied in our physical world: a disconnect between simulated and real-world output. This is due to myriad factors, including minor compounding differences in the joints and actuators of the robot, differences in the quality and accuracy of sensors, differences

between the simulated and real environment, and more. It is incredibly difficult to make the simulation as close to real life as possible, as smaller and smaller improvements yield diminishing results. This problem, known as the sim-to-real gap, has many solutions. One approach is domain randomization, which attempts to work around real-world details. It varies the simulation's setup across trials enough that the real world fits in among these varied examples. Another approach is meta-learning, in which the robot is trained not on tasks themselves, but the ability to quickly pick up new tasks. This means that the final training in the real world is minimal. Similarly, you can use imitation learning, in which the simulated robot learns from examples and attempts to mimic their actions at a higher level (Zhao et al., 2020), though this still often requires a well-trained controller for the motor actions themselves and therefore builds on techniques like domain randomization and meta-learning.

Liu et al. (2017) demonstrated that randomly disabling sensors during training resulted in reduced policy sensitivity to particular sensors or subsets thereof. This is similar to the classic neuron dropout technique (Hinton et al., 2012), which disables random neurons throughout training and prevents overreliance on specific neurons (Hinton et al., 2012; Srivastava et al., 2014), but this dropout functions only at the input layer and on blocks of neurons, not individual neurons. Their robot was a simulated racecar with three modalities of sensors: physical state (position, velocity, orientation to centerline, wheel speeds, engine speed, GPS), a laser-range finder (providing values across multiple timesteps), and a front-view RGB camera (which provided almost 90% of the total state). They tested how well the policy transferred to a new racetrack across three conditions: all sensors, noisy sensors, and random subsets of sensors disabled. For the third condition, only some subset of sensors was available in each episode, but the agent was able to train with all sensors across its iterations. Their results showed that when

the agent was not able to rely upon any given sensor, it ended up developing a robust system that used all of them. In animal studies, the loss of one sense results in reorganization of the brain such that cortices traditionally used for one modality (e.g., sight) can become used for other senses if the first sense is lost (Bavelier & Neville, 2002).

How Biological Agents Learn

Despite RL being one of the most useful ways of implementing skill learning and the primary driver of an agent's behavior (Belousov et al., 2021), it is not the only approach. Friston (2010) argues that *natural* agents focus on avoiding surprise, rather than maximizing a reward signal. "Surprise" here is not emotional but representational: the mismatch between one's internal model of the world and actual experience, as when a seemingly heavy object turns out to be light, or someone is in a room you thought was empty. Friston calls this the "free energy principle" since it relates to an organism's ability to stave off thermodynamic entropy by minimizing free energy. The crux of his argument is that organisms don't learn what is rewarding, but rather they expect reward and act to fulfill those expectations, as hunger drives eating. The action brings the organism to the state that it expects.

Yet the gap between natural cognition and the RL approaches discussed above may be narrower than this contrast suggests. Schultz et al. (1997) demonstrated that dopamine neurons fire in a pattern matching temporal difference algorithms, a common RL approach that mirrors the role of the critic in an actor-critic framework. Temporal difference algorithms track the gap between expected and actual outcomes, sharing notable parallels with Friston's (2010) free energy principle. The two differ in scope: temporal difference algorithms learn only to predict rewards, while Friston's model describes the brain's broader ability to minimize surprise across

sensory inputs, including by acting on the environment to bring it in line with expectations. Temporal difference algorithms, at best, equip an actor model to better maximize reward, not to minimize surprise. Despite these differences, both frameworks treat prediction error as the core driver of learning, both compare actual to expected outcomes, and both apply to virtual and embodied agents alike.

Friston's account, however, presupposes that the organism carries an internal model rich enough to generate those expectations in the first place. Essentially, we need a generative model of what will happen in the world. In order to have that, he argues, we need internal stand-ins (representations) for things in the world around us. Then, our cognitive processes can operate over those representations to create predictions. That assumption is not obvious. Gibson (1979/2015) claims that our interpretation of objects is based upon what we can do with them. He calls what the environment offers the agent “affordances” because they afford something: a chair affords sitting, a door handle affords pulling, and a baseball affords throwing. Affordances are about the relation of an agent to its environment, not merely a property of the object. The agent doesn’t infer the properties of something because of an internal representation, but rather perceives those properties directly.

Brooks (1991) raises a related concern. He argues that when we analyze direct interaction with the world, no need for representation is evident. Parallel systems interact not with each other, but with the environment, which then provides feedback (allowing for live updates in response). Even if we do hold representations in our minds, that is not a requirement for intelligent systems. As Brooks puts it, "It turns out to be better to use the world as its own model" (1991, p. 139). These arguments make different cases against representation: Gibson claims that perception doesn't require it, while Brooks claims that intelligent action doesn't either.

To make robots reasoning agents rather than simple pattern recognition systems, it helps to examine how humans solve tasks. Lake et al. (2016) argue that human-informed agents must build causal models of the world that demonstrate understanding, possess intuitive theories of physics (often called "folk physics," the instincts about what will physically happen to objects during simple interactions), and learn to learn, developing their own capacity to acquire generalizable knowledge rather than relying on designer-scaffolded training. These requirements are achievable. Lake and Baroni (2023) demonstrated that modern neural networks can perform tasks once thought unique to humans, such as systematic generalization: the ability to combine familiar elements in entirely new ways. Language is the canonical example, since a speaker can produce a meaningful sentence they have never encountered before. The same principle, combining primitives into novel complex actions, extends beyond language into motor behavior.

AI does well on specialized tasks, such as cancer classification from medical imaging, where it outperforms humans (Alves et al., 2026). Humans, however, retain the advantage as task complexity grows, particularly for tasks without clear, labeled training data (i.e., deep RL tasks). Even so, the gap has been narrowing for well over a decade: deep RL agents were already matching or exceeding human performance across dozens of Atari games, learning directly from raw pixel input with no task-specific programming (Mnih et al., 2015).

Ernst and Banks (2002) demonstrated that when humans receive conflicting information from multiple senses, the brain weighs each sense more the less variance there is. Essentially, senses that are less reliable are correspondingly relied upon less. This is called maximum-likelihood estimation. Neural network structure allows the network to learn to ignore certain inputs by weighting them very low. However, it takes longer for the network to "realize" that they are noise than if it simply did not have those connections. Angelaki (2008)

demonstrated that the quality of a sense is not assumed fixed, but rather that humans dynamically reweight sensory channels when one degrades. This type of degradation can happen either because the sensor is weakened (e.g., cataracts) or its information source is weakened (e.g., dim lighting).

Research Question

Before adaptation can occur, sensory systems must first develop, and they do not do so all at once. Evidence suggests that in both birds and mammals, sensory systems do not all come online at once, but rather in a specific developmental sequence: tactile (touch), vestibular (balance and spatial orientation), chemical (taste and smell), auditory, and finally visual. Notably, this sequence holds not just for altricial animals (those born helpless, like human infants or kittens), but also for precocial animals (those born relatively capable and independent) (Gottlieb, 1971; Lickliter, 2005).

While Liu et al. (2017) demonstrated the benefits of random sensor dropout, their approach treated all sensors equivalently from a curriculum perspective. Gottlieb's (1971) findings suggest that natural agents' senses come online in an invariant order, with earlier senses providing a foundation for later, complex senses to be integrated into. Turkewitz and Kenny (1982) further argued that limiting senses in early development does not simply constrain the organism, but rather benefits the organization of perceptual systems by preventing overwhelming amounts of information from being provided to the organism before it can use them effectively.

Many animals, for example, are born with closed eyes, taking between days and months to open them (Van Cruchten et al., 2017; Peng et al., 2001). Griswold and Van Hooser (2025) argued that this happens specifically because the neural pathways are not yet developed enough

to make use of this information, and that waiting to provide visual senses is advantageous, whereas overwhelming a newly-developing system causes irregular, detrimental development. This raises a question: if a robot's senses were introduced not all at once, nor randomly on and off, but in a sequence informed by natural sensory development, would the robot learn faster, perform better, or develop more robust systems than a counterpart trained with random sensor dropout? What about one trained with all sensors available from the very start?

The real world is continuous at every scale that matters here, not discrete. Physics comes baked in, whereas simulations only approximate it. Still, learning in the real world is slow, as already discussed. The purpose of this simulation is *not* to model the real world faithfully, but to examine what might be feasible if we place a near-real agent in a close-enough approximation of it. My goal is to demonstrate the feasibility of a cognitive-science-inspired approach to robotic arm training, not to create a physical working product. The gap between the real and simulated world is increasingly shrinking, though. In March 2026, researchers built a neural controller whose architecture is a linear approximation of the complete wiring diagram (connectome) of a fruit fly's brain (with over 100,000 neurons and their synaptic connections). They used that simulated topology to control a simulated fly body. Compared to standard neural network controllers, this connectome-derived linear model trained faster and achieved lower error across every locomotion task tested (Jin et al., 2026).

This paper extends Liu et al. by replacing random sensor dropout with biologically-ordered sensor introduction throughout several curriculum phases, starting with easy tasks and progressing to more complex ones. If Turkewitz's and Kenny's (1982) theory extends to simulated agents, a developmentally-ordered policy which gains sensors as it progresses

should learn more quickly than a policy which is provided with all sensors all the time, assuming that the naturally-occurring developmental model is a result of learning optimization.

Methods

This section is very long and technical. You may wish to skip ahead to the Results and Discussion section unless you are particularly inclined to parse through lots of formulas and other jargon. If you do choose to read this section, note that all formulas are accompanied by an explanation in layman's terms.

Simulation Setup

Using digital copies of physical bodies keeps the sim-to-real gap within striking distance. Thus, I am using a real-world robot arm in the simulation, meaning that the policies I train could feasibly be ported over to a real arm, though it is almost guaranteed that fine-tuning training would be needed. I wanted to find a general-purpose robot arm that has many degrees of freedom (think of a degree of freedom as somewhat like a joint), open-source support (Susnjara & Smalley, n.d.), provides three-dimensional files for simulation, and uses standard controllers. The Lynxmotion 4 DoF arm has all of these and its three-dimensional files specify not just the geometry of the object, but also provides a "Universal Robot Descriptor File" (URDF), which contains data about not just the physical construction of the robot, but also the capabilities of the motors, grippers, linkages, joints and more.

The URDF file extracted from the original Lynxmotion source was modified to better reflect real-world behaviour. Joint damping was increased from the URDF default to $\zeta = 0.5$, and per-joint maximum velocity was capped at $\omega_{max} = 1.5\ rad/s$, well below the

manufacturer's no-load spec listed in the URDF of $2\pi\ rad/s$. The four arm joint ranges and the parallel-jaw gripper joint ranges are listed in Table 1; the gripper fingers are mimic-coupled, meaning both fingers' positions are controlled by a single action output value (just as in the real robot, where gears connect the movement of the two fingers) rather than controlled independently. Cartesian targets and position observations are normalized to a workspace box spanning $[-\ 0.15, -\ 0.15, 0.10]m$ to $[0.15, 0.15, 0.35]m$. Finger pads and cube use friction coefficients $\mu_{lat} = 2.0$, $\mu_{spin} = 0.3$, $\mu_{roll} = 0.05$, and collisions between non-finger arm links and the cube are disabled. This prevents the arm from learning to use non-gripper parts of its assembly to move the cube, which would generalize worse to the real world since it relies upon physics and geometry constants very specific to the simulation. Observed final behavior for all models showed no instances where the model took advantage of this collision disablement.

Table 1

Joint ranges (rad).

Joint	Range (rad)
q_1	$[-\ \pi, +\ \pi]$
q_2	$[-\ 2.182, +\ 2.182]$
q_3	$[-\ 1.920, +\ 1.571]$
q_4	$[-\ 2.094, +\ 2.094]$
Gripper q_5	$[0, \pi/2]$
Mimic q_6	$[-\ \pi/2, 0]$

There are many physics simulation softwares available. In order to eventually open-source this code, I wanted to write it in a broadly-accessible programming language, so I decided to filter my search to only include Python-integrated environments or libraries. While proponents of low-level languages often cite their superior performance compared to high-level, interpreted languages like Python, many Python libraries are written in C and provide a simplistic, high-level interface for programming that is heavily optimized behind the curtain. Thus, working in a high-level language does not necessarily come with performance drawbacks as long as one commits to using well-supported and maintained libraries that are specifically designed to run fast on advanced hardware. Isaac Sim, MuJoCo (Multi-Joint dynamics with Contact), Brax, and Gazebo were promising choices but are designed for more rigorous environment definitions and have a greater initial setup time. These systems have a steeper learning curve and are designed for industry development more than simulated trials or research. PyBullet, however, is designed for quick iteration and provides a dedicated quickstart guide (Coumans & Bai, 2021). Despite being focused on sim-to-real transfer, its default environment is simple to set up and does not require rigorous definition. PyBullet has a dedicated rendering engine, integrates with other Python libraries easily, and can be used easily with OpenAI Gym or its successor Gymnasium. Gymnasium provides tools for reinforcement learning that abstract what is often confusingly called "the environment" which in this case refers not to the world, but the way in which the reinforcement learning is done. Much like PyBullet, Gymnasium exposes a fully-featured Python interface and allows for easy abstraction of the reinforcement learning process (Brockman et al., 2016; Towers et al., 2024).

The physics simulation is stepped at $f_{sim} = 240\ Hz$, with $k = 10$ sub-steps per agent action, yielding a control rate of 24 Hz. This means every virtual second, the agent acts 24 times.

Each time the agent acts, the physics engine does 10 physics steps, each one lasting only $1/240$ of a second. The agent therefore observes the world and emits a new action 24 times per simulated second, while the underlying physics integrates 10 times more finely between actions for stability. This was chosen so that the collisions between the robot and the cube would be calculated more accurately (after initial testing showed that a lower sub-step count resulted in occasional object-on-object clipping).

Reinforcement learning is a very complex field with wide-reaching applications. The specific type of RL used in this project is deep RL. The example of teaching the robot to make cookies by giving it feedback through rewards is deep RL. There are many deep RL algorithms, but a very common one in robotics control is proximal policy optimization (PPO) (Schulman et al., 2017). PPO learns from rewards provided by the environment, balancing exploration and exploitation (Sutton & Barto, 2018) with an actor-critic system. There are myriad hyperparameters that affect how the policy learns, and iterative selection of these parameters is often necessary to find a good set for the specific task being optimized for.

The policy is given sensor inputs and then outputs an action vector $a \in [-1, 1]^5$ for every control step. This output contains the four normalized arm-joint position targets and one normalized gripper target. Each component is linearly re-scaled to the corresponding joint's physical range as described above, so the target position for joint i is given by:

$$q_i^{tgt} = \frac{1}{2}(q_i^{hi} + q_i^{lo}) + a_i \cdot \frac{1}{2}(q_i^{hi} - q_i^{lo})$$

All final simulations were run on Vassar's Hopper Cluster on the hardware-identical EMC-node3 and node6 partitions. These nodes feature 2 AMD EPYC 7601 32-Core Processors operating at 2.10 GHz (128 logical CPUs across 2 sockets, with 2 threads per core) and 503 GB

of RAM. No dedicated GPUs are present on either node. Graphics cards are best used for accelerating video data such as in the arm's RGBD camera inputs, but are not as important as high CPU power for physics simulation. The parallel processing capability of the CPUs on the EMC nodes allows up to 128 virtual environments, though only 96 were used for easy parity during development with another server which had only 48 CPUs. All final simulations were run on the EMC-node3 and node6 partitions, however.

Experimental Design

Curriculum Phases

I ultimately wanted a robot arm that was capable of finding a cube placed randomly within its reach, picking it up using its gripper, and placing it somewhere else (also randomly selected). Given the complexity of this behavior, I needed to build upon the curriculum learning techniques presented previously as well as meticulous reward function scaffolding. I began by developing the reward functions and environment (both the concrete 3D environment and the abstract learning "environment") for an arm given all sensors throughout the entire process.

I trained the arm through three different phases, starting with a simple task and working up towards the complex pick-and-place task based on the discoveries of Bengio et al. (2009) and Eppe et al. (2019). First, I trained the arm to move to a specific point in space by placing an intangible sphere there that could be seen by the camera. Visualizing the trained policy showed it moving in a sweeping, scanning, circular motion until the object was visible. Then, it moved slowly towards the object keeping it in view. This first phase trains the arm to search in its environment.

The way that the arm policy outputs moves is frame by frame. Given a current set of inputs, it outputs some discrete action (move each joint x amount) and then receives new updates on the state of its sensors after that action. This allows it to receive feedback about its actions and the value they provide. However, it ended up oscillating around the point instead of iteratively homing in on it. This is likely because the output actions are computed based on the sum of many values in the neural network, and it is very hard to tune all the levels to sum to the same value as the current positions when the arm is exactly on the goal. Instead, the policy can learn to get very good at moving very close, but it slightly overshoots. Thus, a sub-phase was created with the same task but with a penalty for excessive movement. Penalizing such movement in the first phase would create a local maximum from which the policy might never escape, since moving would result in lower reward unless it just so happened to move right to the point it was supposed to. However, by starting with a policy that already tries to get to the point, we can evolve new behavior from that starting point. Closeness to the goal object is not enough: the arm must be still. Having only the former is a challenge for humans with Parkinson's-caused tremors, for example (Parkinson's Foundation, n.d).

Next, instead of rewarding the arm for moving to an immaterial point, it had to learn to raise a cube placed on a table. This phase was very difficult to determine an effective reward function for, because the arm often attempted to find a specific action that found a single fixed action that maximized expected reward across episodes rather than adapting to the cube's actual position. Specifically, it spent a lot of time placing its gripper as close to the base of the arm as possible. This makes sense when you consider that across many iterations, the center of the robot is the closest point on average to the cube, since it spawns around the base. However, the average of many good actions is not necessarily a good one. Consider reaching for the space between two

objects because you cannot choose which one you'd rather have; that average action is worse than either of the first two. Problems like this (task variance adaptation) were solved by increasing training timesteps. Other problems required the addition of new reward terms, as described later.

Once that phase was structured, I created the final phase, in which an agent that can pick up the cube is rewarded for putting it down on another table instead of simply moving it into the air. This rewards the arm for the same starting actions (finding the cube, reaching for it, grasping it, and moving it from the table) as the previous reward function, but this time the policy must learn that it should next look for the table on which I want it to end, not simply raise it in the air. Each of these phases was carried out and refined iteratively, since I needed to change the reward function and environment to properly scaffold the arm's learning.

Sensor Curricula

There are three conditions under which the arm learns: "*all*," where the arm is given all sensors from the very start; "*random*," where each episode provides the arm some subset of sensors; and "*developed*," where, as training progresses, the arm is slowly provided more sensors based on the developmental ordering described above.

Table 2

Sensor names, dimensions, types, and the phase in which each is first introduced under the developed curriculum.

Sensor Name	Sensor Description	# Values	Assigned "Type"	Phase Introduced
ProprioceptiveSensor	Joint positions and velocities, normalized	12	Tactile/ Proprioceptive	Phase 1 (Reach)

Sensor Name	Sensor Description	# Values	Assigned "Type"	Phase Introduced
EEPositionSensor	End-effector XYZ, normalized to workspace bounds	3	Spatial / Vestibular	Phase 1 (Reach)
TargetPositionSensor	Target XYZ, normalized to workspace bounds	3	Spatial / Vestibular	Phase 1 (Reach)
DistanceSensor	Scalar distance from end-effector to target	1	Spatial / Vestibular	Phase 1 (Reach)
TouchSensor	Contact flag and force per gripper finger	4	Tactile/ Proprioceptive	Phase 2 (Grasp)
UltrasonicSensor	Single forward ray from end-effector	1	Proximity / Auditory	Phase 2 (Grasp)
ObjectPositionSensor	Object XYZ, normalized to workspace bounds; zeros when no object present	3	Visual (proto)	Phase 2 (Grasp)
RGBDSensor	Combined RGB and depth image	H×W×4 (64×64×4)	Visual	Phase 3 (Pick-and-Place)

When a sensor is disabled under any condition, its values are replaced with a non-sentinel value of 0 across all its observation dimensions. This means that the arm is still receiving a value from the sensor's pathways, but it's just an input of 0. In the case where sensors are randomly disabled (with a 15% chance for each sensor), this will make it harder for the policy to learn, since there is no difference between the input for "your arm is at position zero" and "the sensor for your arm's position is disabled," but other techniques to disable the sensor would require very inconsistent architectures between curricula.

Therefore, to help the random-sensor policy distinguish between sensors being off and on, we provide it one final "sensor," with six values. Each value corresponds to the status of another sensor indicating whether or not it is disabled. That way, the policy has a way to distinguish between a flat value of 0 and a signal of 0. This signal was only provided to that policy, not the full-sensor policy and developmental-ordering policy since they have very little

change between episodes in sensor availability and thus are overwhelmingly unlikely to extract anything valuable from such an input.

To avoid overwhelming the developed-sensors policy with a wave of new inputs at the start of the phase, the weight of newly-unlocked sensors is set to zero. The policy can then slowly begin to weight them more heavily as it progresses through the task.

Network Architecture

The network has two core components of note. The first is the feature extractor. This uses the sensors to gain information for the task (this is different from the sensors themselves; it is trained to extract useful information from them). The second is the policy head. This is how the arm determines what to do with the information gained from the sensors. The policy head is reset between phases, while the feature extractor weights are kept.

The feature extractor has a vector branch and an image branch (as described by Noh et al., 2025). The vector branch is a single Linear ($27 \rightarrow 64$) layer with ReLU activation, processing the values produced by the non-image sensors. In layman's terms, it takes the 27 non-image scalar inputs and learns to extract 64 values of significance from them by learning relevant combinations over time. The image branch is a NatureCNN (Mnih et al., 2015) operating on the $64 \times 64 \times 4$ RGB-D image and projecting to a 32-dim vector. The two branches are concatenated into a 96-dim feature vector which is inputted into both heads. The policy and value heads are each a 2-layer head of width 256 with $tanh$ activations, followed by a final linear layer; the policy head outputs the mean of a diagonal Gaussian over the 5-dim action space, with state-independent log-standard-deviation as a learnable parameter. Essentially, the

policy head learns to take the 64 features from the vector sensors and the 32 features from the image and then turns those into an action output for the arm.

Stable-Baselines3's default CombinedExtractor allots 256 features to the CNN branch and only as many vector features as vector inputs, which would result in ~90% of the policy's input coming from raw pixels. To avoid this, I provided more vector-branch features, fewer image-branch features, and I froze the CNN's weights for Phases 1A and 1B, turning it into a fixed random projection that contributes a constant signal but no gradients. This means that the camera features exist, but they aren't updated during the first phase. This is supported by Chen et al. (2021) and Seo et al. (2021), which demonstrate that early CNN freezing is not detrimental and can be beneficial for model learning. The CNN is unfrozen at the start of Phase 2 (Phase 3 for *developed*) once a working policy exists, at which point it can begin to learn more useful visual features.

Reward Functions

The reward functions for the various tasks, arm environment and settings, and training hyperparameters were all set after tedious manual iteration to ensure the full-sensor policy would succeed. This was done by running through training and visually observing the output behavior as well as the reward function terms (i.e., what each component of the reward function was contributing). For example, if the final pick-and-place policy learned to pick up the cube but not to move it to the table, then the reward for closeness to the target was added or increased. If the grasp policy learned to approach the cube but not lift it, then the reward corresponding to the height of the cube was added or increased. The hyperparameters most adjusted were the entropy coefficient, which drives exploration breadth (Lixandru, 2024), and the target KL, which serves

as an early-stopping threshold that halts policy updates when the divergence between successive policies grows too large (Dossa et al., 2021).

Constructing the alternative curricula required some code modifications, but my goal in doing so was to alter as little as possible about the environment, hyperparameters, and reward functions that were refined during the development of the all-sensors baseline. As a result, the alternative curricula are evaluated against a deliberately stacked baseline, meaning any comparable or superior performance from them is a meaningful result.Because random struggled to learn and I wanted to make it a viable competitor for developed, I adjusted its entropy coefficient and target KL because initial trials showed it failing to learn.

All reward functions have an action-magnitude and action-smoothness penalty,

$$R_{action}(a, a_{prev}) = -0.005\,\|a\|^2 - 0.05\,\|a - a_{prev}\|^2,$$

as well as the symbol $d = \|x_{grasp} - x_{tgt}\|$, which denotes the distance between the middle of the two fingers and the target.

For the first task, the arm is given a reward for closeness to the target that scales exponentially when very close. We also penalize it for large actions and reversals; this prevents wobbling and prioritizes smooth, direct action to the goal. However, to avoid the arm learning to move as little as possible to avoid those movement penalties, they are not as strong as they are in the second phase of the reach task, which has a larger velocity penalty. This is only introduced when the arm has already succeeded in learning to move where it needs to.

The Phase 1A (Reach) reward is

$$R_{1A} = -\frac{d}{d_{max}} + max\left(0, 1 - \frac{d}{\delta_{bonus}}\right) + 5 \cdot 1[d < \delta_{succ}] + R_{action},$$

with $d_{max} = 0.5\ m$, $\delta_{bonus} = 0.10\ m$, and $\delta_{succ} = 0.05\ m$. The first term is a continuous distance penalty that becomes less negative as the arm approaches the target. The second term is a proximity bonus that activates only inside $10\ cm$ and grows linearly to $+\ 1$ as it closes in, providing the steeper-near-goal gradient referenced above. The third term is a flat $+\ 5$ at the moment the arm comes within $5\ cm$ of the target. Episodes are 200 steps long; success is declared whenever $d < \delta_{succ}$, but no early termination occurs. This way, the arm can continue to receive a reward for staying on target. If there was early termination, the arm would likely learn to get close enough for a positive reward but not achieve success (since success would end the episode and prevent further reward).

Phase 1B (Reach-Hold) adds a strong penalty on the gripper's movement Δx_{EE}:

$$R_{1B} = R_{1A}^{(base)} - w_v \|\Delta x_{EE}\|, \qquad w_v = 5.0.$$

Standing perfectly still earns no penalty from this term, so once the arm is at the target it is now positively incentivised to remain motionless rather than oscillate.

In the grasp task, there is a time penalty to discourage dawdling, an approach reward to encourage moving towards the cube, a reward for closing the gripper when close to the cube, a reward for both sides of the gripper contacting the cube, a lift reward to ensure the arm is actually holding the cube and not just touching it (since if it was only touching it, it would not be able to lift it), and an inherent penalty for dropping the cube below the table, which results in ending the episode. That is bad for the policy as it prevents further reward progress, but is better

than applying a consistent penalty for dropping the cube, which leads to the policy being afraid to even go near the cube (since its initial, uncoordinated actions often result in dropping the cube and being penalized for the rest of the episode if it is not terminated).

$\ell = z_{cube} - z_{cube}^{(0)}$ denotes the cube's lift (relative to its starting height), θ_g the gripper joint angle (where 0 is closed and $\pi/2$ is fully open), and c_L, c_R the per-finger contact counts (since there are multiple contact points). A *valid grasp* indicator,

$$G = 1\left[c_L > 0 \ \wedge \ c_R > 0 \ \wedge \ \theta_g < \theta_{close}\right], \qquad \theta_{close} = 1.0\, rad,$$

is considered true if and only if both fingers are touching the cube *and* the gripper is at least partly closed. This prevents the policy from harvesting contact reward by parking wide-open fingers around the cube, which it might otherwise do (since early touching of the cube is more likely to result in dropping the cube, which is harmful and would result in the policy learning to avoid touching the cube at all). The full Phase 2 reward is:

$$R_2 = -c_t - w_d d_{cube} +$$

$$w_g (1 - d_{cube}/\rho)_+ \cdot (1 - \theta_g/(\pi/2))_+ (proximity - gated\ grip\ shaping)$$

$$+ \ r_g G(grasp\ reward) + w_\ell \cdot clip(\ell, 0, 0.10) \cdot G \cdot 1[\ell > 0.005](lift) + R_{action}$$

with $c_t = 0.3$, $w_d = 2.0$, $w_g = 1.0$, $\rho = 0.05\ m$, $r_g = 5.0$, $w_\ell = 500$. If the cube falls more than 2 cm below its spawn Z, the reward switches to $-c_t - w_d \|x_{grasp} - x_{spawn}\|$ (so the agent cannot farm reward by chasing the cube along the floor) and the episode terminates. Success requires $\ell > 0.05\ m$ with $G = 1$ for 10 consecutive control steps. (Essentially, success means

getting the cube high enough for long enough to be sure it's being held, not just thrown up in the air)

The Pick-and-Place task has the most complex reward function. Consistent iteration has resulted in a reward function that has made many attempts to discourage the arm from finding local optimal options that are not successful. For example, the arm initially found that it could bend over backwards and hold the cube near the destination to farm the proximity bonus. Despite this, it was not able to bend far enough backwards to place the cube down. The only way to succeed and put the cube on the destination table without throwing it involves the arm pivoting about its base to face the destination table. Thus, I introduced a reward for pivoting the base in the direction of the table, but only when the cube is being held. The reward function includes similar rewards to the other functions as well as a transport progress, joint alignment, goldilocks-height, no-fling, proper placement, and no-drop reward (with some implemented as penalties for the inverse). Notably, this kind of fine-tuning further worsens the generalizability of the trained policy to other environments (including the real world).

$D = \|x_{cube} - x_{dest}\|$ is the cube-to-destination distance, with $D_{max} = 0.5\ m$ and a lift threshold $L_{min} = 0.02\ m$ below which transport-related terms do not activate (so the agent lifts the cube enough that it is less likely to catch on the table). The full Phase 3 reward is the sum of the contributions in Table 3, with each row activating only under the listed condition.

Table 3

Phase 3 (Pick-and-Place) reward terms. All distances in metres.

Term	Form	Active when
Time penalty	$-c_t = -0.3$	always
Approach	$-w_d \cdot d_{cube}, w_d = 2.0$	always
Grip shaping	$w_g(1 - d_{cube}/\rho)_+(1 - \theta_g/(\pi/2))_+$	always
Grasp	$r_g \cdot G = 5.0 \cdot G$	always
Lift (tapered)	$w_\ell \cdot clip(\ell, 0, 0.10) \cdot min(1, D/D_{taper})$, $w_\ell = 350, D_{taper} = 0.15$	$G = 1,\ \ell > 0.005$
One-time lift bonus	$+100$, fired once per episode	$G = 1,\ \ell > 0.05$, not yet earned
Transport (potential)	$w_T \cdot (D_{t-1} - D_t), w_T = 100$	$G = 1,\ \ell > L_{min}$
Stepped proximity	$+\{15, 30, 30\}$ if $D < \{0.15, 0.10, 0.07\}$ respectively (tightest band only)	$G = 1,\ \ell > L_{min}$
J1 alignment	$w_{J1}(1 - \lvert\Delta q_1\rvert/\pi)^2, w_{J1} = 50$	$G = 1,\ \ell > L_{min}$
J4 tilt penalty	$-w_{J4} \cdot max(0, q_4 - q_4^*), w_{J4} = 15$, $q_4^* = 0.785$	$G = 1,\ \ell > L_{min}$
Overheight penalty	$-500 \cdot (\ell - 0.12)$	$G = 1,\ \ell > 0.12$
Hold-distance cost	$-w_h \cdot (D/D_{max}) \cdot min(1, \ell/0.10), w_h = 120$	$G = 1,\ \ell > 0.005$
Release shaping	$w_r \cdot (1 - D/\delta_r) \cdot (\theta_g/(\pi/2)), w_r = 30$, $\delta_r = 0.10$	$G = 1,\ D < \delta_r,\ \ell <$
Fling penalty	$-200 \cdot max(0, z_{cube} - (z_0 + 0.05))$	$G = 0$ (cube ungrasped & airborne)
Placement bonus	$+1500$	$D < 0.05,\ G = 0,$
Action smoothness	R_{action}	always

$q_1^* = \pi/2 - atan2(x_{dest}, y_{dest})$ is the base angle that points the arm directly at the destination,

$\Delta q_1 = q_1 - q_1^*$ is the angular error wrapped to $[-\pi, \pi]$, and the squared term peaks at $+50$

/step when the arm is perfectly aligned. The "fling penalty" and the velocity-gated placement bonus together kill the reward signal of throwing the cube at the destination: a flung cube costs reward every step it is airborne ungrasped, and lands too fast to satisfy the placement bonus's $v_{cube} < 0.3\ m/s$ gate. Success requires $D < 0.05\ m$ with the cube ungrasped and nearly stationary for 5 consecutive control steps; episodes are 300 steps long at maximum, though in testing the average episode length was always shorter (< 150).

PPO Hyperparameters and Transfer Chain

All phases use 96 parallel SubprocVecEnv workers, 30 evaluation environments, $n_{steps} = 512$, a batch size of 256, $\gamma = 0.99$, $\lambda_{GAE} = 0.95$, clip range $\epsilon = 0.2$, max-grad-norm of 0.5, and were run on node3 or node6. The phase-specific hyperparameters that did require iteration are listed in Table 4.

Table 4

Phase-specific PPO hyperparameters. The Reach-Hold sub-phase fine-tunes from the Reach checkpoint and inherits its sensor curriculum.

Hyperparameter	Reach (1A)	Reach-Hold (1B)	Grasp (2)	Pick-and-Place (3)
Initial LR (η_0)	3×10^{-4}	1×10^{-4}	5×10^{-5}	1×10^{-4}
LR schedule	constant	constant	linear warm-up, $\eta_0/20 \rightarrow \eta_0$ over first 10%	linear warm-up, $\eta_0/20 \rightarrow \eta_0$ over first 10%
Entropy coefficient c_{ent}	0 (default)	0 (default)	0.005 (*all / developed*), 0.015 (*random*)	[0.03 (*all / developed*), .02 (*random*)] $\rightarrow$ 0 (linear, $7.5M \rightarrow 12.5M$ steps)

Hyperparameter	Reach (1A)	Reach-Hold (1B)	Grasp (2)	Pick-and-Place (3)
Value coefficient c_{vf}	1.0	0.5	0.5	0.5
n_{epochs}	5	5	3	5
Target KL	0.05	0.15	0.15	0.15 (all/developed), .05 (random)
Default timesteps	5×10^5	1×10^6	5×10^6	1×10^7
VecNormalize	N/A	N/A	yes (obs + rew, clip 10)	yes (seeded from Grasp obs stats; reward stats reset)
CNN frozen?	yes	yes	unfrozen on transfer	unfrozen on transfer
MLP-head reset?	N/A	N/A	both heads reset	value head only

Each transfer between policies $\pi_{reach} \rightarrow \pi_{reach-hold} \rightarrow \pi_{grasp} \rightarrow \pi_{pick-place}$ keeps the feature extractor's weights and resets the value head, since the value head learns task-specific value functions which would not apply to the new phase. At the Reach-Hold → Grasp transition, the policy MLP is also reset and the policy's $log(\sigma)$ is reinitialized to 0; without this, the converged Reach-Hold policy enters Grasp with near-zero exploration variance (because the Reach-Hold velocity penalty drives $log(\sigma)$ very negative) and explores so little that it never discovers contact. Under the *developed* curriculum, an additional zero-init step is applied at the start of each phase for newly activated sensor inputs.

The Pick-and-Place entropy coefficient is decayed linearly from 0.03 to 0 over training steps $[7.5M, 12.5M]$ to encourage policy self-fine-tuning near the end of training. SB3's otherwise pushes $log(\sigma)$ up. This lets the policy gradient equilibrate σ on its own.

Scene Layout

Each episode spawns a source table at a fixed radius $r = 0.20\ m$ from the base of the arm at a uniformly-sampled angle $\alpha_1 \in [0, 2\pi)$. The cube is spawned on the source table with $\pm\ 2\ cm$ XY jitter at $z = 0.10\ m$. A destination table is additionally spawned at angle $\alpha_2 = \alpha_1 + \beta$, with the offset $\beta \sim U(60°, 120°)$. The source-table angle range was constrained to $[0, \pi/2]$ in Grasp and $[0, 2\pi/3]$ in Pick-and-Place.

In the Grasp phase, 25% of episodes have a *warm-start* in which the arm is positioned directly into a predefined reaching pose ($q_2, q_3, q_4 = (-0.5, -0.7, 1.9)$), the gripper is started fully open, and the cube is force-spawned such that closing the gripper makes immediate contact. This allows the policy to learn that this is a higher-reward state and facilitates it learning to get from the initial random spawn to that state more reliably.

Results

Note that quantitative comparisons will be rare in this section. The methods of this thesis (iteration until success) run contrary to the assumptions of most well-known, reliable statistical analyses. Running such analyses would also require far more tests to run, since small initial changes can lead to disparities in later phases, and so a proper analysis would require rerunning the entire multi-phase pipeline across many seeds per condition, which exceeds the compute budget available for this project.

This analysis compares the *developed* curriculum against two baselines. The *all* condition serves as a conventional training benchmark, with every sensor available from the first episode. The *random* condition represents an alternative form of input restriction via random dropout, a

technique known to improve policy robustness, though not necessarily task success within a fixed training budget. Together, these baselines isolate the effect of developmentally-ordered sensor introduction from both unrestricted input and unstructured restriction.

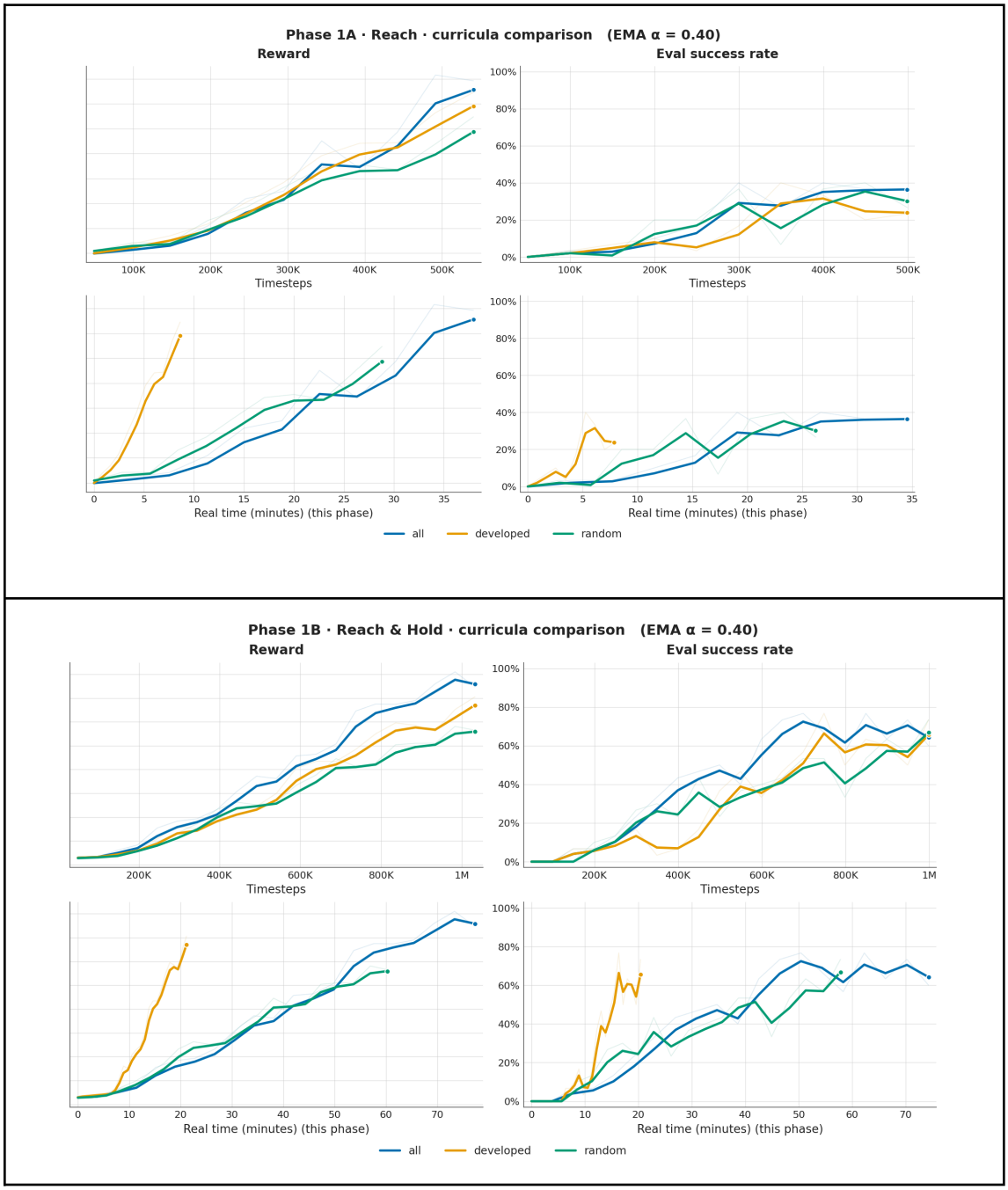
While comparing final success rates for each policy is helpful, we can gain much insight by looking at the success rate over time. Note though, that the policy itself is striving for a high reward, not necessarily success. Success is just a likely result of a well-crafted reward function. Thus, to evaluate the policy's learning, we can gain more by looking at the reward gained over time. It is important to consider two *different kinds* of time, however.

The simulated world operates in discrete steps: it takes a snapshot, queries the policy for an action, applies the resulting joint commands, and repeats. This is far more computationally tractable than solving continuous differential physics, while keeping step sizes small enough that motion appears smooth. Crucially, simulated time is defined by step count, not wall-clock duration. Like watching a video at 2x speed, the speedup is entirely in the observer's frame. The runner didn't run faster. However, wall-clock ("real") time still matters in practice. While the all-sensors condition in this work trains in roughly one day, more sophisticated extensions, such as domain randomization or more complex tasks, could require substantially longer. Tracking both reward and success across both time axes therefore provides meaningful (and different) insights.

While the final goal of the training pipeline was the pick-and-place task, it can also be helpful to examine the earlier tasks' results, as this is where *developed* and *all* differ more in their inputs. It will also allow us to compare *developed* to *random*, a more traditional method that restricts sensor access differently. Figure 1 demonstrates the success over time in phase 1B for all three policies. All outcome metric graphs will have a smoothing effect (Hyndman &

Athanasopoulos, 2021) applied due to noise as a result of evaluation episode sample constraints. Notably, all three curricula have similar final success rates, but *developed* trains in approximately one-fifth (phase 1A) and one-fourth (phase 1B) the time that *all* takes.

Figure 1 (Phase 1A and 1B)



Note that the reward axis for all figures will have no units, since they are entirely arbitrary

Figure 2 demonstrates the performance of all three curricula for phase 2. In phase 2, all three runs had close final successes but *developed* fell slightly below *random* and *all*. Despite this, it trained in less than one-fourth the time that *all* took. If the grasp phase was the final task, then it would be worth running *developed* for the same total time as *all* to see if it would catch up in success rate. However, the reason the earlier phases were introduced was predominantly to prepare the models for the final and most complex task, not to compare them on their performance in the earlier phases.

Figure 2

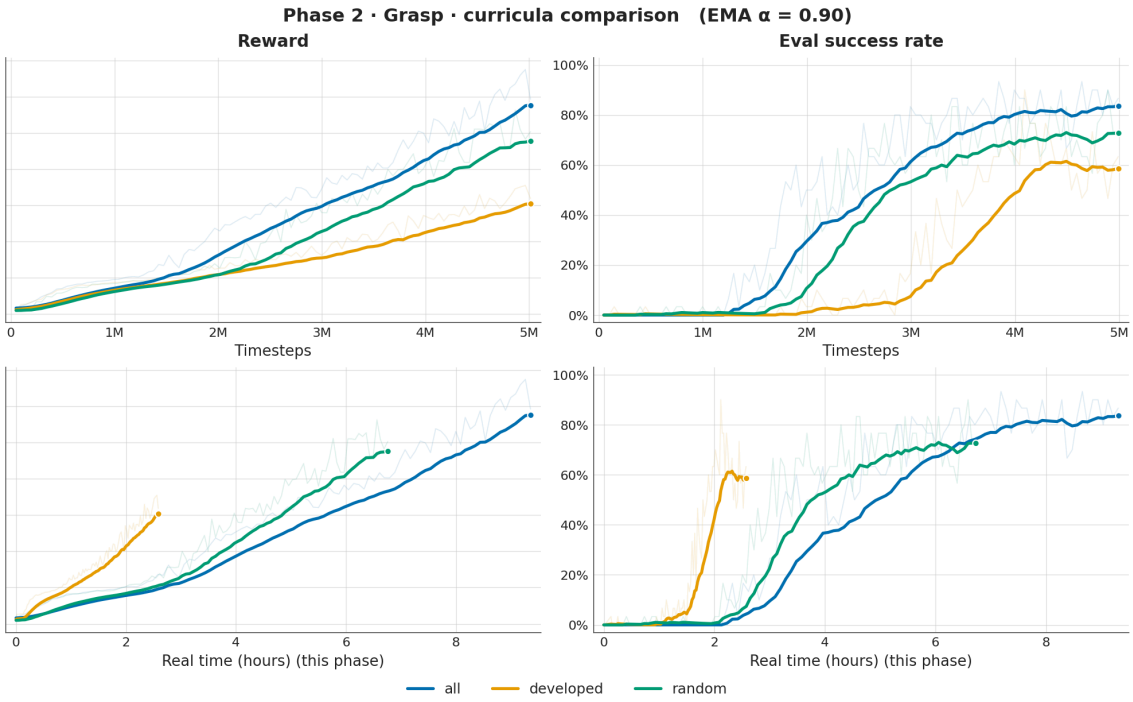
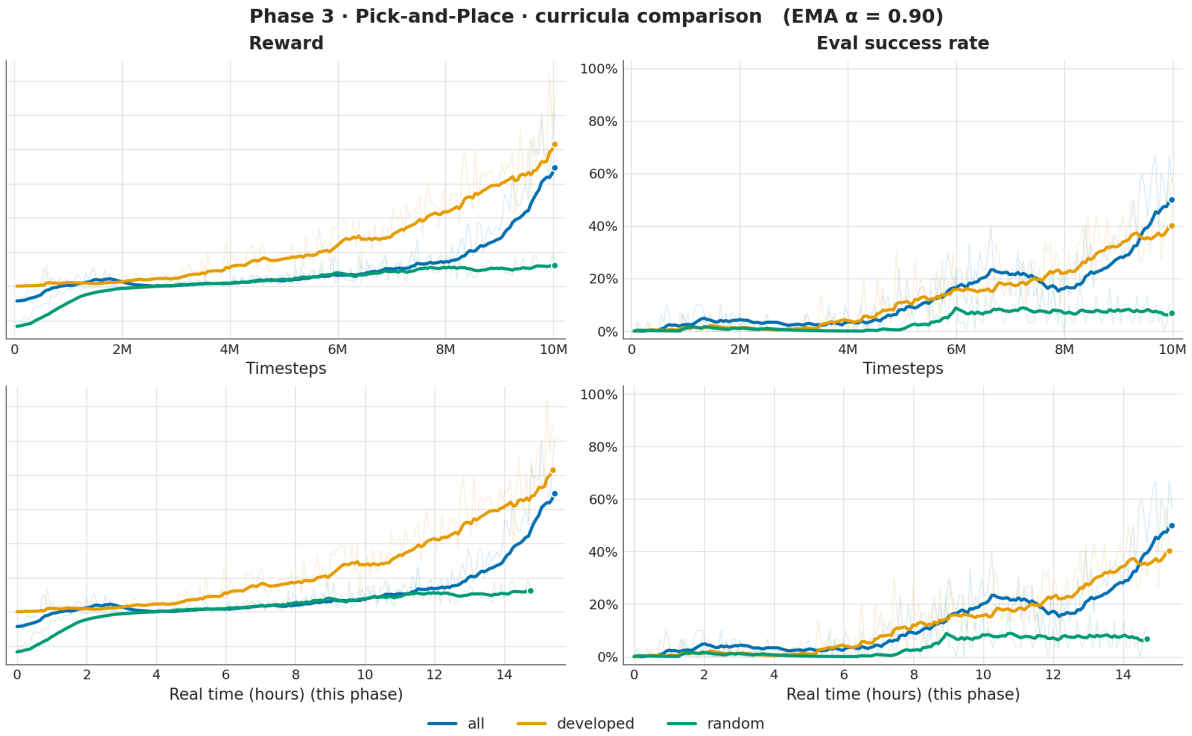


Figure 3 shows the result of phase 3's training, in which both *developed* and *all* have all sensors unlocked. The end result shows that *all* and *developed* achieve a similar final success

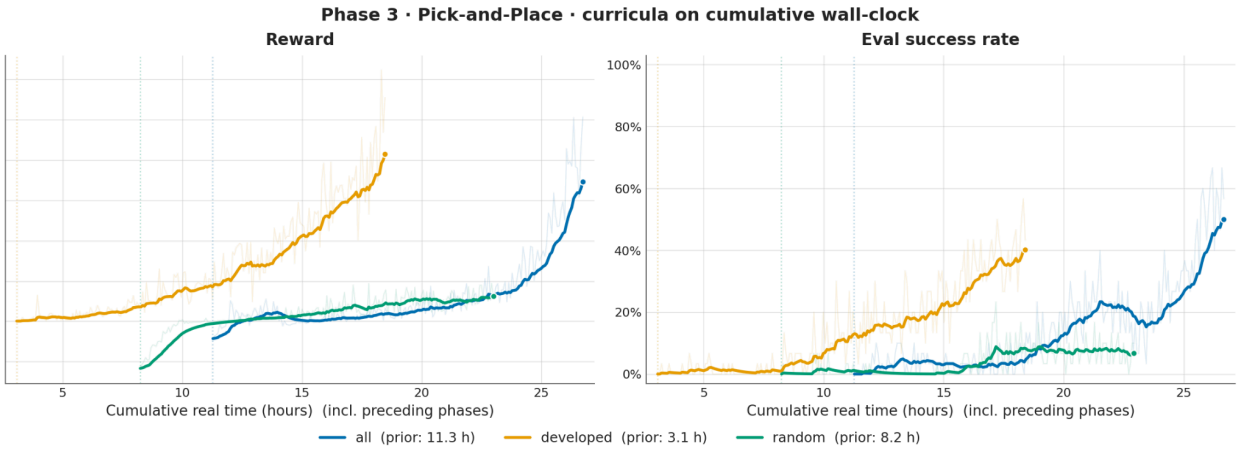
rate, with *random* falling far below both. The learning curve is similar for *all* and *developed*, unlike *random*'s much slower learning curve. Notably, the addition of new sensors for *developed* in phase 3 was not visibly detrimental, which had an initial reward above *all*. This is likely due to the zero-weighting of new sensors at the start of each phase for *developed*. In real animals, the senses are enabled at least somewhat (closed eyes still provide some information, for example), so zero-weighting might not be an ideal reflection of this. From a training perspective though, it allows for much better transfer and puts the onus on the policy to begin to integrate the sensor into its action calculations, as opposed to forcibly ramping up an external weight for the sensor input value.

Figure 3



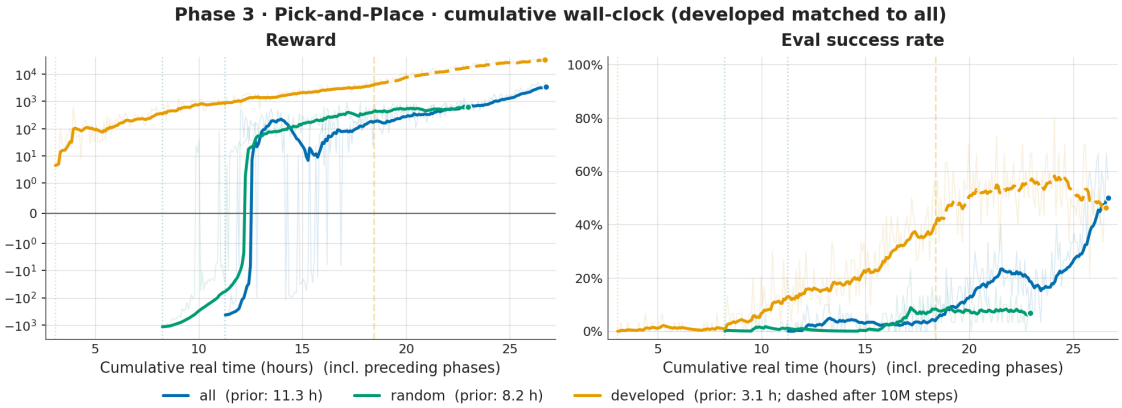
Given the training efficiency of *developed* in earlier phases, we can reexamine phase 3 by offsetting each condition's Phase-3 start time by its cumulative prior training time. Figure 4 shows that *developed* reaches its successes much sooner because of the headstart yielded by beginning training with fewer sensors. For tasks requiring days, weeks, or months of compute, the advantage of smaller input compounds significantly.

Figure 4



Furthermore, extending *developed*'s wall-clock time until it matches *all* shows near-equal success rates, demonstrating that the training time advantages in the early phases do not affect final training success for this task under equal time constraint conditions.

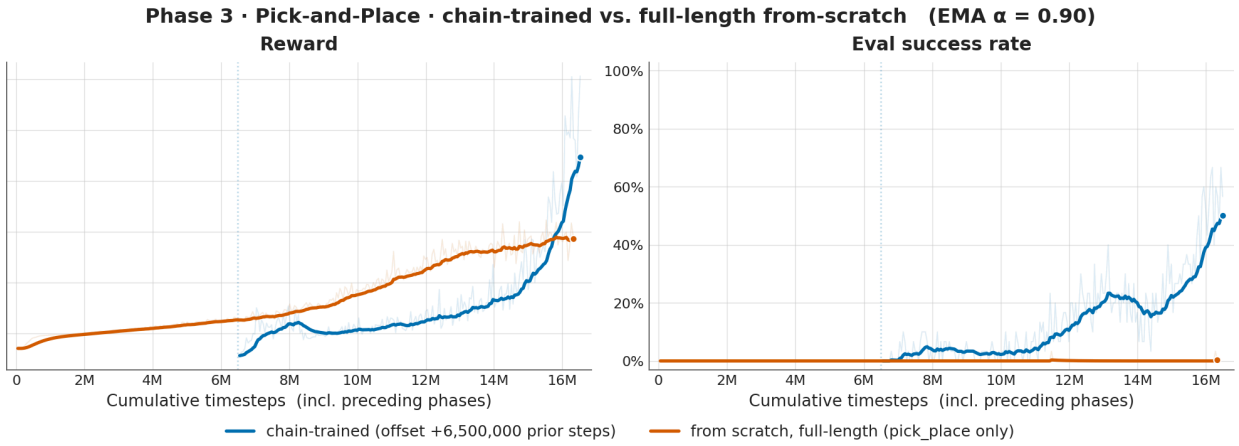
Figure 5



Note the symlog scaling on the y-axis of Reward. Developed *achieves high enough success that it would otherwise flatten the appearance of* all *and* random*'s reward*

A further benefit of phase-based training is the ability to build a well-trained base model using a core sensor set, then branch into multiple task-specific variants with additional, complex sensors (such as cameras). This mirrors training Phases 1 and 2 extensively under the *developed* curriculum before fine-tuning on downstream tasks. Figure 5 demonstrates that *developed* is able to ultimately reach the same reward as *all* when total training time is matched. Even without the intent to train earlier models, Figure 6 confirms the necessity of phase-based pretraining: a pick-and-place model initialized from scratch with no prior training achieves no successes and negligible reward gain, even if trained for as many total timesteps as all phases combined.

Figure 6



Discussion

Turkewitz and Kenny (1982) predicted that the limitation of early sensors in *developed* would help perceptual organization. The reward curve in phases 2 and 3 suggest not better organization of *developed* compared to *all*, but they do suggest faster successful organization. In phase 1, *developed* was not only faster than *random* and *all*, but had nearly-equivalent success to both. This might suggest that fine-tuning sensor introduction timing and selection could result in equivalent or superior performance from *developed* as from *all*. A limitation of this study is that the environment, hyperparameters, and adjustments made to get the simulation code running were all based on iteratively improving the success of the *all* curriculum. When that curriculum struggled, changes were made to the reward function and to the training hyperparameters. It is possible that if similar iterative changes were made to *developed*, it would have achieved a similar success rate in not just phase 2, but phase 3 as well.

Liu et al.'s (2017) paper on random sensor dropout also included extra sensors; their approach had more information coming from multiple sensors. Despite this, the *random* condition in this paper provided extra information: the model was told which sensors were on

and which were off. Thus, the model in this paper had both advantages and disadvantages compared to Liu et al.'s paper. While some of this simulation's sensors overlapped in information, many did not. Therefore, *random* similarly might outperform *all* if more sensors were included, such as multiple "ultrasonic" sensors at various angles, more information-dense proprioceptive sensors, and sensors that provided information across multiple timesteps, all of which were present in Liu et al.'s original example.

Gottlieb's (1971) claim that early sensors provide a foundation for later sensory development seems to extend to this agent, task, and environment. Turkewitz's and Kenny's (1982) claim that this can benefit the organism by facilitating faster organization is partially supported by these results; while the real time organization was faster, the virtual timesteps needed were nearly identical. One of the largest gaps between deep RL and natural brains is that the latter takes far fewer examples to learn (Khajehnejad et al., 2024; Botvinick et al., 2019). Thus, the real-time advantage of *developed* might be the more biologically analogous result. Natural agents do not experience consecutive episodes to be optimized over, but continuous and different tasks in which sensors must provide meaningful information quickly enough to support survival. Wall-clock time therefore captures how quickly an effective policy can arise in a metric that we care about, and the results demonstrate that it is able to achieve the same ultimate success as if all sensors were on from the start.

It is possible that *developed* would have done better in phase 2 if vision was introduced at that point. Griswold and Van Hooser (2025) claimed that the reason that visual systems are the last to develop is because they would provide information that the neural pathways were not yet ready for. The decision to introduce the senses as batches exactly between tasks was somewhat arbitrary. A different strategy for sensory development timing may have resulted in superior

performance. It is also possible that the advantages that Griswold and Van Hooser theorize come with later vision introduction would be more notable with more sensors, *developed* may have performed even better.

Bengio et al.'s (2009) work on training through multiple tasks was essential to the development of all three final policies in this paper. Training a model from scratch on the final task resulted in slower reward growth and the policy didn't achieve success at all. Thus, it's possible that adding in even more tasks, functioning as smaller steps closer to the final task, would have been beneficial.

Future Work

A notable failure of the third phase's reward function was that success gain over time was not tied strongly enough to reward gain; the model should have learned to be more successful as it improved its reward, but exploratory analyses of running the model for longer actually show eventual decreases in success (despite increases in reward). Visual inspection of the model's behavior shows that it learns a combination of behaviors that return slightly more reward than success alone, and thus the success rate comparison between these models fails to account for differences between true model capabilities and different behaviors (for example, it is possible that the models are exploring different fruitful branches in behavior, such as diverging evolutionary processes). At later timesteps, the reward function could have slowly phased out non-success rewards. This would have pushed the model (once it already learned to succeed in some cases) to optimize only for success itself, meaning it would learn the optimal extent to which different component terms contributed to that. This would also help offset the reward function tuning that this study relied upon.

Given that human brain cells learn with fewer examples than AI does (Khajehnejad et al., 2024), biologically-inspired training structures offer a plausible route toward narrowing the sample-efficiency gap between natural and artificial learners (Lake et al., 2016; Botvinick et al., 2019). The developmental sensor-introduction curriculum presented here provides concrete evidence along that route: imposing an evolutionarily-ordered input schedule, without altering the underlying algorithm or architecture, yields substantial real-time gains over both random dropout and all-sensor baselines. Phase-based training can compound the advantages that ordered sensor introduction achieves, since a model pretrained on a core sensor set can be extended with task-specific sensors rather than retrained from scratch. Figure 6 demonstrates that training from scratch is less effective (at least for this task) than phase-based training.

Further work could include analyzing a different sensor-introduction order. Unlike *developmental's* ordering from animals, this could, for example, introduce sensors in order from fewest to most dimensions as in the below table:

Sensor Name	Sensor Description	# Values	Assigned "Type"	Phase Introduced
DistanceSensor	Scalar distance from end-effector to target	1	Proximity	Phase 1A (Reach)
UltrasonicSensor	Single forward ray from end-effector	1	Proximity	Phase 1A (Reach)
EEPositionSensor	End-effector XYZ, normalized to workspace bounds	3	Spatial	Phase 1B (Reach)
TargetPositionSensor	Target XYZ, normalized to workspace bounds	3	Spatial	Phase 1B (Reach)
ObjectPositionSensor	Object XYZ, normalized to workspace bounds; zeros when no object present	3	Spatial	Phase 1B (Reach)
TouchSensor	Contact flag and force per gripper finger	4	Tactile/ Proprioceptive	Phase 2 (Grasp)

ProprioceptiveSensor	Joint positions and velocities, normalized	12	Tactile/ Proprioceptive	Phase 2 (Grasp)
RGBDSensor	Combined RGB and depth image	H×W×4 (64×64×4)	Visual	Phase 3 (Pick-and-Place)

The types of analyses here are also somewhat restricted to a very narrow task and environment. A larger analysis could find existing simulated deep-RL task-learning projects (such as the one in Liu et al., 2017) and adapt their work to ensure there was an equivalent condition to *all*, *random*, *developed*, and an alternatively-ordered sensor-introduction as described above. This would extend the potential generalizability of this approach and also compare the sensor introduction approach observed in nature against other approaches.

Another potential future analysis is to analyze the robustness of each curricula's models to sensor loss. Real-world agents must grapple with noisy sensors that can fail. Therefore, probing the adaptability of the model under such conditions can test how well each of these curricula adapts over time as Angelaki (2008) noted that humans do. Finally, a transfer-learning exploration could determine whether *developed* is just faster than *all* or if it is also more adaptable to new phase-3 tasks.

References

Alves, N., et al. (2026). Artificial intelligence and radiologists in pancreatic cancer detection using standard of care CT scans (PANORAMA): an international, paired, non-inferiority, confirmatory, observational study. The Lancet Oncology, 27(1), 116–124. https://doi.org/10.1016/S1470-2045(25)00567-4

Angelaki, D. E., & Cullen, K. E. (2008). Vestibular System: The Many Facets of a Multimodal Sense. Annual Review of Neuroscience, 31(1), 125–150. https://doi.org/10.1146/annurev.neuro.31.060407.125555

Bavelier, D., & Neville, H. J. (2002). Cross-modal plasticity: where and how? Nature Reviews Neuroscience, 3(6), 443–452. https://doi.org/10.1038/nrn848

Belousov, B., Abdulsamad, H., Klink, P., Parisi, S., & Peters, J. (Eds.). (2021). Reinforcement Learning Algorithms: Analysis and Applications. Springer International Publishing.

Bengio, Y., Louradour, J., Collobert, R., & Weston, J. (2009). Curriculum learning. In Proceedings of the 26th Annual International Conference on Machine Learning (pp. 41–48). ACM. ICML '09: The 26th Annual International Conference on Machine Learning held in conjunction with the 2007 International Conference on Inductive Logic Programming. https://doi.org/10.1145/1553374.1553380

Black, K., Brown, N., Driess, D., Esmail, A., Equi, M., Finn, C., … Zhilinsky, U. (2024). π_0: A vision-language-action flow model for general robot control. https://doi.org/10.48550/arXiv.2410.24164

Botvinick, M., Ritter, S., Wang, J. X., Kurth-Nelson, Z., Blundell, C., & Hassabis, D. (2019). Reinforcement Learning, Fast and Slow. Trends in Cognitive Sciences, 23(5), 408–422. https://doi.org/10.1016/j.tics.2019.02.006

Braitenberg, V. (1984). Vehicles: Experiments in synthetic psychology. MIT Press.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). OpenAI Gym (Version 1). arXiv. https://doi.org/10.48550/ARXIV.1606.01540

Brooks, R. A. (1991). Intelligence without representation. Artificial Intelligence, 47(1–3), 139–159. https://doi.org/10.1016/0004-3702(91)90053-m

Chen, L., Lee, K., Srinivas, A., & Abbeel, P. (2021). Improving computational efficiency in visual reinforcement learning via stored embeddings. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, & J. Wortman Vaughan (Eds.), Advances in Neural Information Processing Systems (Vol. 34, pp. 26779–26791). Curran Associates. https://proceedings.neurips.cc/paper/2021/hash/e140dbab44e01e699491a59c9978b924-Abstract.html

Coumans, E., & Bai, Y. (2021). PyBullet quickstart guide. GitHub. https://raw.githubusercontent.com/bulletphysics/bullet3/master/docs/pybullet_quickstartguide.pdf

Diedrichsen, J., & Kornysheva, K. (2015). Motor skill learning between selection and execution. Trends in Cognitive Sciences, 19(4), 227–233. https://doi.org/10.1016/j.tics.2015.02.003

Dossa, R. F. J., Huang, S., Ontañón, S., & Matsubara, T. (2021). An empirical investigation of early stopping optimizations in proximal policy optimization. IEEE Access, 9, 117981–117992. https://doi.org/10.1109/ACCESS.2021.3106662

Eppe, M., Nguyen, P. D. H., & Wermter, S. (2019). From Semantics to Execution: Integrating Action Planning With Reinforcement Learning for Robotic Causal Problem-Solving. Frontiers in Robotics and AI, 6. https://doi.org/10.3389/frobt.2019.00123

Ernst, M. O., & Banks, M. S. (2002). Humans integrate visual and haptic information in a statistically optimal fashion. Nature, 415(6870), 429–433. https://doi.org/10.1038/415429a

Friston, K. (2010). The free-energy principle: a unified brain theory? Nature Reviews Neuroscience, 11(2), 127–138. https://doi.org/10.1038/nrn2787

Gibson, J. J. (1979/2015). The Ecological Approach to Visual Perception. Psychology Press.

Goodale, M. A., Milner, A. D., Jakobson, L. S., & Carey, D. P. (1991). A neurological dissociation between perceiving objects and grasping them. Nature, 349(6305), 154–156. https://doi.org/10.1038/349154a0

Gottlieb, G. (1971). Early Development of Species-Specific Auditory Perception in Birds. In Studies on the Development of Behavior and the Nervous System (pp. 237–280). Elsevier. https://doi.org/10.1016/b978-0-12-609303-2.50016-9

Griswold, S. V., & Van Hooser, S. D. (2025). Premature vision drives aberrant development of response properties in primary visual cortex. eLife Sciences Publications, Ltd. https://doi.org/10.7554/elife.106513.2

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint https://doi.org/10.48550/arXiv.1207.0580

Hyndman, R. J., & Athanasopoulos, G. (2021). Exponential smoothing. In Forecasting: Principles and practice (3rd ed.). OTexts. https://otexts.com/fpp3/expsmooth.html

Ibarz, J., Tan, J., Finn, C., Kalakrishnan, M., Pastor, P., & Levine, S. (2021). How to train your robot with deep reinforcement learning: lessons we have learned. The International

Journal of Robotics Research, 40(4–5), 698–721. https://doi.org/10.1177/0278364920987859

Jiang, K. (2019). TheSpaghettiDetective [Computer software]. GitHub. https://github.com/kennethjiang/TheSpaghettiDetective

Jin, Z., Zhu, Y., Zhang, C., & Sui, Y. (2026). Whole-Brain Connectomic Graph Model Enables Whole-Body Locomotion Control in Fruit Fly (Version 2). arXiv. https://doi.org/10.48550/ARXIV.2602.17997

Kaelbling, L. P., Littman, M. L., & Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. Artificial Intelligence, 101(1–2), 99–134. https://doi.org/10.1016/S0004-3702(98)00023-X

Kawasaki Heavy Industries, Ltd. (n.d.). Kawasaki Robotics 50th anniversary. https://kawasakirobotics.com/jp-sp/50th-anniversary/en/

Khajehnejad, M., Habibollahi, F., Paul, A., Razi, A., & Kagan, B. J. (2024). Biological Neurons Compete with Deep Reinforcement Learning in Sample Efficiency in a Simulated Gameworld (Version 1). arXiv. https://doi.org/10.48550/ARXIV.2405.16946

Lake, B. M., & Baroni, M. (2023). Human-like systematic generalization through a meta-learning neural network. Nature, 623(7985), 115–121. https://doi.org/10.1038/s41586-023-06668-3

Lake, B. M., Ullman, T. D., Tenenbaum, J. B., & Gershman, S. J. (2016). Building Machines That Learn and Think Like People (Version 3). arXiv. https://doi.org/10.48550/ARXIV.1604.00289

Lickliter, R. (2005). Prenatal Sensory Ecology and Experience: Implications for Perceptual and Behavioral Development in Precocial Birds. In Advances in the Study of Behavior (pp. 235–274). Elsevier. https://doi.org/10.1016/s0065-3454(05)35006-6

Liu, G.-H., Siravuru, A., Prabhakar, S., Veloso, M., & Kantor, G. (2017). Learning End-to-end Multimodal Sensor Policies for Autonomous Navigation (Version 2). arXiv. https://doi.org/10.48550/ARXIV.1705.10422

Lixandru, A. (2024). Proximal policy optimization with adaptive exploration (arXiv:2405.04664). arXiv. https://arxiv.org/abs/2405.04664

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4), 115–133.

Miyamoto, H., Kawato, M., Setoyama, T., & Suzuki, R. (1988). Feedback-error-learning neural network for trajectory control of a robotic manipulator. Neural Networks, 1(3), 251–265. https://doi.org/10.1016/0893-6080(88)90030-5

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529–533. https://doi.org/10.1038/nature14236

Montevirgen, K. (2026, April 30). opportunity cost | Marginal Cost, Scarcity, & Trade-Offs. Encyclopedia Britannica. https://www.britannica.com/money/opportunity-cost

Noh, S., Lee, W., & Myung, H. (2025). Sample-efficient and occlusion-robust reinforcement learning for robotic manipulation via multimodal fusion dualization and representation

normalization. Neural Networks, 185, 107202. https://doi.org/10.1016/j.neunet.2025.107202

O'Regan, J. K., & Noë, A. (2001). A sensorimotor account of vision and visual consciousness. Behavioral and Brain Sciences, 24(5), 939–973. https://doi.org/10.1017/S0140525X01000115

Parkinson's Foundation. (n.d.). Tremor. https://www.parkinson.org/understanding-parkinsons/symptoms/movement-symptoms/tremor

Peng, J., Jiang, Z., Liu, W., Huang, S., Zhang, J., & Wang, W. (2001). Growth and development of giant panda (Ailuropoda melanoleuca) cubs at Beijing Zoo. Journal of Zoology, 254(2), 261–266. https://doi.org/10.1017/s0952836901000772

Pfeifer, R., & Bongard, J. (2006). How the body shapes the way we think: A new view of intelligence. MIT Press.

Qian, Y., Zhu, X., Biza, O., Jiang, S., Zhao, L., Huang, H., Qi, Y., & Platt, R. (2024). ThinkGrasp: A Vision-Language System for Strategic Part Grasping in Clutter (Version 1). arXiv. https://doi.org/10.48550/ARXIV.2407.11298

Saveriano, M., Abu-Dakka, F. J., Kramberger, A., & Peternel, L. (2023). Dynamic movement primitives in robotics: A tutorial survey. The International Journal of Robotics Research, 42(13), 1133–1184. https://doi.org/10.1177/02783649231201196

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal Policy Optimization Algorithms (Version 2). arXiv. https://doi.org/10.48550/ARXIV.1707.06347

Schultz, W., Dayan, P., & Montague, P. R. (1997). A Neural Substrate of Prediction and Reward. Science, 275(5306), 1593–1599. https://doi.org/10.1126/science.275.5306.1593

Seo, Y., Chen, L., Shin, J., Lee, H., Abbeel, P., & Lee, K. (2021). State entropy maximization with random encoders for efficient exploration. In M. Meila & T. Zhang (Eds.), Proceedings of the 38th International Conference on Machine Learning (Vol. 139, pp. 9443–9454). PMLR. https://proceedings.mlr.press/v139/seo21a.html

Shaikhutdinova, K. (2025, July 18). Inside China's dark factories where robots run the show [Video]. The Wall Street Journal. https://www.wsj.com/video/series/in-depth-features/inside-chinas-dark-factories-where-robots-run-the-show/0BAB0212-DE97-4843-BE77-82DF366B53EA

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(56), 1929–1958. https://dl.acm.org/doi/10.5555/2627435.2670313

Susnjara, S., & Smalley, I. (n.d.). What is open source software? IBM. Retrieved May 4, 2026, from https://www.ibm.com/think/topics/open-source

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning, Second edition: An introduction. Bradford Book.

Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., Perez-Vicente, R., Pierré, A., Schulhoff, S., Tai, J. J., Tan, H., & Younis, O. G. (2024). Gymnasium: A Standard Interface for Reinforcement Learning Environments (Version 4). arXiv. https://doi.org/10.48550/ARXIV.2407.17032

Turkewitz, G., & Kenny, P. A. (1982). Limitations on input as a basis for neural organization and perceptual development: A preliminary theoretical statement. Developmental Psychobiology, 15(4), 357–368. https://doi.org/10.1002/dev.420150408

Van Cruchten, S., Vrolyk, V., Perron Lepage, M., Baudon, M., Voute, H., Schoofs, S., Haruna, J., Benoit‐Biancamano, M., Ruot, B., & Allegaert, K. (2017). Pre‐ and Postnatal Development of the Eye: A Species Comparison. Birth Defects Research, 109(19), 1540–1567. https://doi.org/10.1002/bdr2.1100

Wada, K., James, S., & Davison, A. J. (2022). ReorientBot: Learning Object Reorientation for Specific-Posed Placement (Version 1). arXiv. https://doi.org/10.48550/ARXIV.2202.11092

Zhao, W., Queralta, J. P., & Westerlund, T. (2020). Sim-to-Real Transfer in Deep Reinforcement Learning for Robotics: a Survey. arXiv. https://doi.org/10.48550/ARXIV.2009.13303