# Memory, Benchmark & Robots: A Benchmark for Solving Complex Tasks with Reinforcement Learning

**Anonymous Author(s)**
Affiliation
Address
email

**Abstract:** Memory is crucial for enabling agents to tackle complex tasks with temporal and spatial dependencies. While many reinforcement learning (RL) algorithms incorporate memory, the field lacks a universal benchmark to assess an agent's memory capabilities across diverse scenarios. This gap is particularly evident in tabletop robotic manipulation, where memory is essential for solving tasks with partial observability and ensuring robust performance, yet no standardized benchmarks exist. To address this, we introduce **MIKASA** (**M**emory-**I**ntensive **Sk**ills **A**ssessment **S**uite for **A**gents), a comprehensive benchmark for memory RL, with three key contributions: (1) we propose a comprehensive classification framework for memory-intensive RL tasks, (2) we collect **MIKASA-Base** – a unified benchmark that enables systematic evaluation of memory-enhanced agents across diverse scenarios, and (3) we develop **MIKASA-Robo** – a novel benchmark of 32 carefully designed memory-intensive tasks that assess memory capabilities in tabletop robotic manipulation. Our work introduces a unified framework to advance memory RL research, enabling more robust systems for real-world use.

**Keywords:** Memory, Benchmark, Robots

## 1 Introduction

Many real-world problems involve partial observability [1], where agents lack full access to the environment's state. Such tasks often require sequential decision-making [2], long-term information retention [3, 4], and handling delayed rewards. A common solution is to equip agents with memory to exploit historical context [5, 6]. While NLP has well-established memory benchmarks [7, 8], evaluation in reinforcement learning (RL) remains fragmented. Existing suites like POPGym [9], DMLab-30 [10], and MemoryGym [11] address only specific domains of memory use.

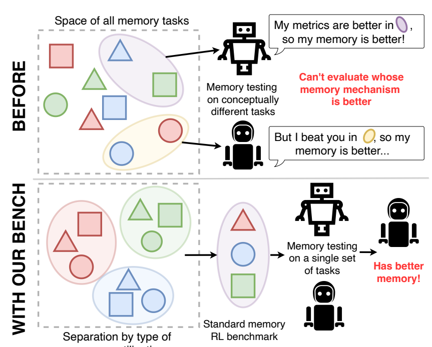


Figure 1: Systematic classification of problems with memory in RL reveals distinct memory utilization patterns and enables objective evaluation of memory mechanisms across different agents.

Unlike classical RL, where benchmarks like Atari [12] and MuJoCo [13] serve as universal standards, memory-based agents are usually tested on custom environments tied to their proposals Table 2. This fragmentation masks key differences in performance across memory tasks—for example, an agent may retain object attributes well but struggle with sequential recall. Such narrow evaluations hide task-specific strengths and weaknesses, highlighting the need for a unified benchmark covering diverse memory demands.

The challenge of evaluating memory is especially evident in robotics. While some tasks naturally involve partial observability, e.g., navigation [14, 15], many studies simulate it by adding noise or masking MDP states [16, 17, 5, 18]. Yet, these simplifications fail to capture real-world complex-

Table 1: **MIKASA-Robo**: A benchmark comprising **32 memory-intensive robotic manipulation tasks** across 12 categories. Each task varies in difficulty and configuration modes. The table specifies episode timeout (T), the necessary information that the agent must memorize in order to succeed (Oracle Info), and task instructions (Prompt) for each environment. See Appendix H for details.

| Memory Task | Mode | Brief description of the task | T | Oracle Info | Prompt | Memory |
|---|---|---|---|---|---|---|
| **ShellGame** | Touch<br>Push<br>Pick | Memorize the position of the ball after some time being covered by the cups and then interact with the cup the ball is under | 90 | `cup_with_ball_number` | — | Object |
| **Intercept** | Slow<br>Medium<br>Fast | Memorize the positions of the rolling ball, estimate its velocity through those positions, and then aim the ball at the target | 90 | `initial_velocity` | — | Spatial |
| **InterceptGrab** | Slow<br>Medium<br>Fast | Memorize the positions of the rolling ball, estimate its velocity through those positions, and then catch the ball with the gripper and lift it up | 90 | `initial_velocity` | — | Spatial |
| **RotateLenient** | Pos<br>PosNeg | Memorize the initial position of the peg and rotate it by a given angle | 90 | `y_angle_diff` | `target_angle` | Spatial |
| **RotateStrict** | Pos<br>PosNeg | Memorize the initial position of the peg and rotate it to a given angle without shifting its center | 90 | `y_angle_diff` | `target_angle` | Spatial |
| **TakeItBack-v0** | — | Memorize the initial position of the cube, move it to the target region, and then return it to its initial position | 180 | `xyz_initial` | — | Spatial |
| **RememberColor** | 3 \ 5 \ 9 | Memorize the color of the cube and choose among other colors | 60 | `true_color_indices` | — | Object |
| **RememberShape** | 3 \ 5 \ 9 | Memorize the shape of the cube and choose among other shapes | 60 | `true_shape_indices` | — | Object |
| **RememberShape-AndColor** | 3×2\3×3\<br>5×3 | Memorize the shape and color of the cube and choose among other shapes and colors | 60 | `true_shapes_info`<br>`true_colors_info` | — | Object |
| **BunchOfColors** | 3 \ 5 \ 7 | Remember the colors of the set of cubes shown simultaneously in the bunch and touch them in any order | 120 | `true_color_indices` | — | Capacity |
| **SeqOfColors** | 3 \ 5 \ 7 | Remember the colors of the set of cubes shown sequentially and then select them in any order | 120 | `true_color_indices` | — | Capacity |
| **ChainOfColors** | 3 \ 5 \ 7 | Remember the colors of the set of cubes shown sequentially and then select them in the same order | 120 | `true_color_indices` | — | Sequential |
| **Total: 32 tabletop robotic manipulation memory-intensive tasks in 12 groups** | | | | | | |

ity [17], where robots must recall past object states, manipulate occluded items, or execute multi-step procedures requiring memory. Examples include a service robot remembering a plate hidden under a towel, or a home robot wiping a microwave door multiple times; without memory, the plate would be missed and the door wiped endlessly.

In this paper, we aim to address these challenges with the following four contributions:

1. **Memory Tasks Classification.** We propose a simple yet comprehensive framework that organizes memory-intensive tasks into four key categories. This structure enables systematic evaluation without added complexity (Figure 1), offering a clear guide for selecting environments that reflect core memory challenges in RL and robotics (Section 4).
2. **Memory-RL Benchmark.** We introduce **MIKASA-Base**, a Gymnasium-based [19] framework for evaluating memory-enhanced RL agents (Section 5).
3. **Robotic Manipulation Tasks.** We introduce **MIKASA-Robo**, a suite of 32 robotic tasks targeting specific memory-dependent skills in realistic settings (Section 6), and evaluate them using popular Online RL baselines (Subsection 6.2) and Visual-Language-Action (VLA) models (Subsection 6.4).
4. **Robotic Manipulation Datasets.** We release datasets for all 32 MIKASA-Robo memory-intensive tasks to support Offline RL research (see Appendix B), and conduct extensive evaluations using a range of Offline RL baselines (Subsection 6.3).

## 2 Related Works

Several RL benchmarks assess agents' memory abilities. DMLab-30 [10] provides 3D navigation and puzzle tasks focused on long-horizon exploration, while PsychLab [20] adds working-memory probes. MiniGrid and MiniWorld [21] emphasize partial observability in lightweight 2D/3D settings, and MiniHack [22], built on NetHack [23], offers roguelike scenarios requiring both short- and long-term memory. BabyAI [24] combines natural language with grid tasks demanding multi-step recall. POPGym [9] standardizes memory evaluation with diverse puzzles and decision tasks. BSuite [25] tests exploration, credit assignment, and scalability under controlled setups. Memory Gym [11] introduces partially observable 2D grids, including endless variants for long-term evaluation, while Memory Maze [26] presents 3D mazes requiring efficient memory use.

While existing benchmarks shed light on memory mechanisms, they largely focus on abstract puzzles or navigation and lack coverage of the full spectrum of memory use. Tasks also differ


Figure 2: Illustration of demonstrative memory-intensive tasks execution from the proposed MIKASA-Robo benchmark. The `ShellGameTouch-v0` task requires the agent to memorize the ball's location under mugs and touch the correct one. In `RememberColor9-v0`, the agent must memorize a cube's color and later select the matching one. In `RotateLenientPos-v0`, the agent must rotate a peg while keeping track of its previous rotations.

fundamentally across suites, hindering direct comparison of memory-enhanced agents. In robotics, memory demands are more challenging: manipulation involves complex interactions and multi-step procedures requiring both spatial and temporal recall. Current benchmarks, though diagnostic, fail to capture these domain-specific difficulties, as the physical control and object interaction in manipulation introduce complexities beyond traditional evaluation frameworks.

Prior work classifies memory-intensive environments along different axes. Ni et al. [44] distinguish memory vs. credit assignment by temporal horizon, while Yue et al. [45] propose memory dependency pairs linking past events to current decisions. Cherepanov et al. [46] define memory as long- vs. short-term (context length) and declarative vs. procedural (environmental vs. episodic), and Leibo et al. [20] adapt cognitive psychology tasks for RL. Though insightful, these taxonomies neglect physical aspects of robotics, where interaction and memory are tightly coupled, motivating a spatio-temporal framework for real-world tasks.

Concurrent with our work, Fang et al. [47] introduced MemoryBench, a manipulation benchmark with three tasks targeting only spatial memory. Built on RLBench [48], it lacks efficient training parallelization.

Table 2: Key memory-intensive environments from the reviewed studies for evaluating agent memory. The Atari [12] environment with frame stacking is included to illustrate that many memory-enhanced agents are tested solely in MDP. Benchmark first introduced in the same work. Benchmark is open-sourced.

| | DRQN [27] | DTQN [28] | HCAM [3] | AMAGO [29] | GTrXL [4] | R2I [30] | RATE [31] | R2A [32] | Modified S5 [33] | Neural Map [34] | GBMR [35] | EMDQN [36] | MRA [37] | FMRQN [38] | ADRQN [39] | DCEM [40] | R2D2 [41] | ERLAM [42] | AdaMemento [43] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Atari w/o FrameStack | ✓ | | | | | | | | | | | | | | ✓ | | | | ✓ |
| Atari with FrameStack | | | | | | ✓ | ✓ | ✓ | | | ✓ | ✓ | | | | | ✓ | ✓ | |
| gym-gridverse | | ✓ | | | | | | | | | | | | | | | | | |
| car flag | | ✓ | | | | | | | | | | | | | | | | | |
| memory card | | ✓ | | | | | | | | | | | | | | | | | |
| Hallway | | ✓ | | | | | | | | | | | | | | | | | |
| HeavenHell | | ✓ | | | | | | | | | | | | | | | | | |
| Ballet | | | ✓ | | | | | | | | | | | | | | | | |
| Object Permanence | | | ✓ | | | | | | | | | | | | | | | | |
| DMLab-30 | | | ✓ | | ✓ | | | | | | | | | | | | ✓ | | |
| POPGym | | | | ✓ | | ✓ | | | ✓ | | | | | | | ✓ | | | |
| Passive T-Maze | | | | ✓ | | | ✓ | | | | | | | | | | | | |
| ViZDoom-Two-Colors | | | | | | | ✓ | | | | | | | | | | | | |
| Numpad | | | | | ✓ | | | | | | | | | | | | | | |
| Memory Maze | | | | | | ✓ | ✓ | | | | | | | | | | | | |
| Memory Maze (apples) | | | | | ✓ | | | | | | | | | | | | | | |
| Minigrid-Memory | | | | | | | ✓ | | | | | | | | | | | | |
| BSuite | | | | | | ✓ | | | ✓ | | | | | | | | | | |
| Goal-Search | | | | | | | | | | ✓ | | | | | | | | | |
| Doom Maze | | | | | | | | | | ✓ | | | | | | | | | |
| PsychLab | | | | | | | | | | | | | ✓ | | | | | | |
| Spot the Difference | | | | | | | | | | | | | ✓ | | | | | | |
| Goal Navigation | | | | | | | | | | | | | ✓ | | | | | | |
| Transitive Inference | | | | | | | | | | | | | ✓ | | | | | | |
| I-Maze | | | | | | | | | | | | | | ✓ | | | | | |
| Pattern Matching | | | | | | | | | | | | | | ✓ | | | | | |
| Random Maze | | | | | | | | | | | | | | ✓ | | | | | |
| Unity Fast-Mapping Task | | | | | | | | | | | | | | | | ✓ | | | |
| Action Associative Retrieval | | | | | | | ✓ | | | | | | | | | | | | |
| BabyAI | | | | | | | | ✓ | | | | | | | | | | | |

# 3 Background

## 3.1 Partially Observable Markov Decision Process

Partially Observable Markov Decision Process (POMDP) [49] extend MDP to account for partial observability, where an agent observes only noisy or incomplete information about the true environments state. POMDP defined by a tuple $(S, A, T, R, \Omega, O, \gamma)$, where: $S$ is the set of states representing the complete environment configuration; $A$ is the action space; $T(s'|s, a) : S \times A \times S \to [0, 1]$ is the transition function defining the probability of reaching state $s'$ from state $s$ after taking action $a$; $R(s, a) : S \times A \to \mathbb{R}$ is the reward function specifying the immediate reward for taking action $a$ in state $s$; $\Omega$ is the observation space containing all possible observations; $O(o|s, a) : S \times A \times \Omega \to [0, 1]$ is the observation function defining the probability of observing $o$ after taking action $a$ and reaching state $s$; $\gamma \in [0, 1)$ is the discount factor determining the importance of future rewards. The objective is to find a policy $\pi$ that maximizes the expected discounted cumulative reward: $\mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right]$, where $a_t \sim \pi(\cdot|o_{1:t})$ depends on the history of observations rather than the true state. Relying on partial observations makes POMDPs harder to solve than MDPs.

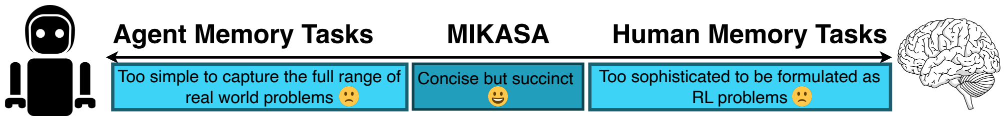


Figure 3: MIKASA bridges the gap between human-like memory complexity and RL agents requirements. While agents tasks don't require the full spectrum of human memory capabilities, they can't be reduced to simple spatio-temporal dependencies. MIKASA provides a balanced framework that captures essential memory aspects for agents tasks while maintaining practical simplicity.

### 3.2 Memory-intensive environments

Memory-intensive environment is an environment where agents must leverage past experiences to make decisions, often in problems with long-term dependencies or delayed rewards. More formally, following Cherepanov et al. [46], a memory-intensive task is a POMDP where there exists a correlation horizon $\xi > 1$, representing the minimum number of timesteps between an event critical for decision-making and when that information must be recalled. Popular memory-intensive environments in RL are listed in Table 2. One way to solving memory-intensive environments is to augment agents with memory mechanisms (see Appendix E).

## 4 Classification of memory-intensive tasks

The evaluation of memory capabilities in RL faces two major challenges. First, as shown in Table 2, research studies use different sets of environments with minimal overlap, making it difficult to compare memory-enhanced agents across studies. Second, even within individual studies, benchmarks may focus on testing similar memory aspects (e.g., remembering object locations) while neglecting others (e.g., reconstructing sequential events), leading to incomplete evaluation of agents' memory.

Different architectures may exhibit varying performance across memory tasks. For instance, an architecture optimized for long-term object property recall might struggle with sequential memory tasks, yet these limitations often remain undetected due to the narrow focus of existing evaluation approaches. To address these challenges, we propose a systematic approach to memory evaluation in RL. Drawing from established research in developmental psychology and cognitive science, where similar memory challenges have been extensively studied in humans, we develop a categorization framework consisting of four distinct memory task classes, detailed in Subsection 4.2.

### 4.1 Memory: From Cognitive Science to RL

In developmental psychology and cognitive science, memory is classified into categories based on cognitive processes. Key concepts include object permanence [50], which involves remembering the existence of objects out of sight, and categorical perception [51], where objects are grouped based on attributes like color or shape. Working memory [52] and memory span [53] refer to the ability to hold and manipulate information over time, while causal reasoning [54] and transitive inference [55] involve understanding cause-and-effect relationships and deducing hidden relationships, respectively.

The RL field has attempted to utilize these concepts in the design of specific memory-intensive environments [37, 3], but these have been limited at the task design level. Of particular interest, however, is how existing memory-intensive tasks can be categorized using these concepts to develop a benchmark on which to test the greatest number of memory capabilities of memory-enhanced agents, and it is this problem that we address in this paper. Thus, we aim to provide a balanced framework that covers important aspects of memory for real-world applications while maintaining practical simplicity (see Figure 3).

### 4.2 Taxonomy of Memory Tasks

We introduce a comprehensive task classification framework for evaluating memory mechanisms in RL. Our framework categorizes memory-intensive tasks into four fundamental types, each targeting distinct aspects of memory capabilities:

1. **Object Memory.** Tasks that evaluate an agent's ability to maintain object-related information over time, particularly when objects become temporarily unobservable. These tasks align with the cognitive concept of object permanence, requiring agents to track object properties when occluded, maintain object state representations, and recognize encountered objects. Example: a robot remembers which fruit it put in the fridge.
2. **Spatial Memory.** Tasks focused on environmental awareness and navigation, where agents must remember object locations, maintain mental maps of environment layouts, and navigate based on previously observed spatial information. Example: the robot remembers the position of a mug it moved while cleaning and returns it to its place.
3. **Sequential Memory.** Tasks that test an agent's ability to process and utilize temporally ordered information, similar to human serial recall and working memory. These tasks require remembering action sequences, maintaining order-dependent information, and using past decisions to inform future actions. Example: a robot memorizes the order of the ingredients it has added to a soup.
4. **Memory Capacity.** Tasks that challenge an agent's ability to manage multiple pieces of information simultaneously, analogous to human memory span. These tasks evaluate information retention limits and multi-task information processing. Example: a robot is able to memorize the positions of several different objects while cleaning a table.

This classification framework enables systematic evaluation of memory-enhanced RL agents across diverse scenarios. By providing a structured approach to memory task categorization, we establish a foundation for comprehensive benchmarking that spans the wide spectrum of memory requirements. In the following section, we present a carefully curated set of tasks based on this classification, forming the basis of our proposed MIKASA benchmark.

## 5 MIKASA-Base

**Motivation and Overview.** Despite the importance of memory in decision-making, the RL community lacks standardized tools for benchmarking memory capabilities. Existing studies typically introduce bespoke environments tailored to their proposed algorithms, leading to fragmentation and limited comparability across works (see Table 2). Moreover, many popular memory benchmarks focus narrowly on specific memory types, overlooking the diversity of memory demands found in real-world applications. To address this gap, we introduce **MIKASA-Base**, a unified benchmark that consolidates widely used open-source memory-intensive environments under a common Gym-like API. Our goal is to streamline reproducibility, support fair comparisons, and promote systematic evaluation of memory in RL.

Table 3: Analysis of established robotics frameworks with manipulation tasks, comparing their support for memory-intensive tasks. † – excluding Franka Kitchen. $*$ – concurrent work with three memory tasks with only one type of memory.

| Robotics Framework with Manipulation Tasks | Memory Tasks | | |
|---|---|---|---|
| | Manipulation | Atomic | Low-level actions |
| MIKASA-Robo (**Ours**) | ✓ | ✓ | ✓ |
| MemoryBench* [47] | ✓ | ✓ | ✓ |
| ManiSkill3 [56] | ✗ | ✗ | ✗ |
| ManiSkill-HAB [57] | ✗ | ✗ | ✗ |
| FetchBench [58] | ✗ | ✗ | ✗ |
| RoboCasa [59] | ✗ | ✗ | ✗ |
| Gymnasium-Robotics† [60] | ✗ | ✗ | ✗ |
| BEHAVIOR-1K [61] | ✓ | ✗ | ✗ |
| ARNOLD [62] | ✗ | ✗ | ✗ |
| iGibson 2.0 [63] | ✓ | ✗ | ✗ |
| VIMA [64] | ✓ | ✓ | ✗ |
| Isaac Sim [65] | ✗ | ✗ | ✗ |
| panda-gym [66] | ✗ | ✗ | ✗ |
| Habitat 2.0 [67] | ✗ | ✗ | ✗ |
| Meta-World [68] | ✗ | ✗ | ✗ |
| CausalWorld [69] | ✗ | ✗ | ✗ |
| RLBench [48] | ✗ | ✗ | ✗ |
| robosuite [70] | ✗ | ✗ | ✗ |
| dm_control [71] | ✗ | ✗ | ✗ |
| Franka Kitchen [72] | ✗ | ✗ | ✗ |
| SURREAL [73] | ✗ | ✗ | ✗ |
| AI2-THOR [74] | ✗ | ✗ | ✗ |

**Benchmark Design Principles.** MIKASA-Base is designed around core principles that support rigorous and interpretable evaluation of memory in RL. To disentangle memory from unrelated challenges, we organize tasks into two tiers. The first tier consists of **diagnostic** vector-based environments that isolate specific memory mechanisms. The second tier includes **complex** image-based tasks that incorporate realistic perception challenges, thus more closely resembling real-world settings. This hierarchical structure enables researchers to validate memory capabilities incrementally – from atomic reasoning to high-dimensional sensory input.

**Task Classification and Selection.** Building on our taxonomy from Subsection 4.2, we systematically reviewed open-source memory benchmarks and categorized their tasks into four distinct types of memory usage. We selected a diverse yet representative subset of environments to cover this taxonomy – ranging from object permanence to sequential planning. All selected tasks are unified under a single, consistent API. Descriptions are provided in Appendix I, and an overview of MIKASA-Base tasks appears in Table 7. This consolidation supports architectural ablations, direct comparison of methods, and simplified evaluation pipelines. Implementation details can be found in Appendix C.

MIKASA-Base provides the first systematic and unified benchmark for evaluating memory in RL. It mitigates fragmentation by standardizing task access and evaluation, and its structured progression enables precise attribution of memory-related agent failures. By covering a broad spectrum of memory challenges within a common framework, MIKASA-Base offers a foundation for robust, reproducible research in memory-centric RL.

# 6 MIKASA-Robo

Robotic manipulation frameworks show gaps in addressing memory-intensive tasks. While navigation widely studies partial observability, manipulation is mostly tested under full observability with little emphasis on memory (see Table 3). Existing efforts like BEHAVIOR-1k [61] and iGibson 2.0 [63] feature highly complex tasks that obscure specific memory mechanisms, while VIMA [64] uses high-level action abstractions, limiting temporal memory evaluation. To fill these gaps, we present **MIKASA-Robo**, a benchmark for diverse memory skills in manipulation via fine-grained, isolated tasks. Concurrent with our work, Fang et al. [47] proposed **MemoryBench**, a benchmark focused on spatial memory with three robotic tasks. In contrast, MIKASA-Robo spans four memory categories and 32 tasks, enabling broader, more systematic evaluation of memory mechanisms in RL agents.

**MIKASA-Robo** is a benchmark designed for memory-intensive robotic tabletop manipulation tasks, simulating real-world challenges commonly encountered by robots. These tasks include locating occluded objects, recalling previous configurations, and executing complex sequences of actions over extended time horizons. By incorporating meaningful partial observability, this framework offers a systematic approach to test an agent's memory mechanisms.

Building upon the robust foundation of ManiSkill3 framework [56], our benchmark leverages its efficient parallel GPU-based training capabilities to create and evaluate these tasks.

## 6.1 MIKASA-Robo Manifestation

Our tasks build on the four memory types in our classification framework (Subsection 4.2). We designed **32 tasks in 12 categories of robotic tabletop manipulation**, each probing object, spatial, sequential memory, or memory capacity. Tasks vary in complexity to systematically test different mechanisms—for example, tracking occluded objects for object permanence or reproducing strict action sequences for sequential recall. Table 1 summarizes tasks by memory type, with details in Appendix H.

To illustrate the concept of our memory-intensive framework, we present `ShellGameTouch-v0`, `RememberColor-v0`, and `RotateLenientPos-v0` tasks in Figure 2. In the `ShellGameTouch-v0` task, the agent observes a red ball placed in one of three positions over the first 5 steps ($t \in [0, 4]$). At $t = 5$, the ball and the three positions are covered by mugs. The agent must then determine the location of the ball by interacting with the correct mug. In the simplest mode (`Touch`), the agent only needs to touch the correct mug, whereas in other modes, it must either push or lift the mug. In the `RememberColor-v0` task, the agent observes a cube of a specific color for 5 steps ($t \in [0, 4]$). After the cube disappears for 5 steps, 3, 5, or 9 (depending on task mode) cubes of different colors appear at $t = 10$. The agent's task is to identify and select the same cube it initially saw. In the `RotateLenientPos-v0` task, the agent must rotate a randomly oriented peg by a specified clockwise angle.

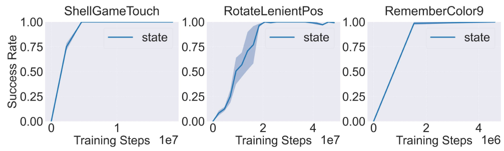


Figure 4: Performance of PPO-MLP trained in `state` mode, i.e., in MDP mode without the need for memory. These results suggest that the proposed tasks are inherently solvable with a success rate of 100%.

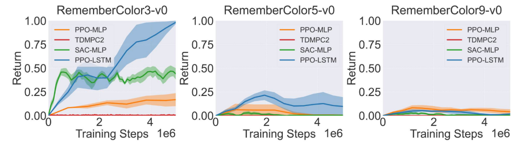


Figure 5: Online RL baselines with MLP and LSTM backbones trained in `RGB+joints` mode on the `RememberColor-v0` environment with dense rewards. Both architectures fail to solve medium and high complexity tasks.

The MIKASA-Robo benchmark offers multiple training modes: `state` (complete vector information including oracle data and Tool Center Point (TCP) pose), `RGB` (top-view and gripper-camera images with TCP position), `joints` (joint states and TCP pose), `oracle` (task-specific environment data for debugging), and `prompt` (static task instructions). While any mode combination is possible, **`RGB+joints` serves as the standard memory testing configuration**, with `state` mode reserved for MDP-based tasks.

The MIKASA-Robo benchmark implements two types of reward functions: dense and sparse. The dense reward provides continuous feedback based on the agent's progress towards the goal, while the sparse reward only signals task completion. While dense rewards facilitate faster learning in our experiments, sparse rewards better reflect real-world scenarios where intermediate feedback is often unavailable, making them crucial for evaluating practical applicability of memory-enhanced agents.

### 6.2 Online RL baselines

For the experimental evaluation, we chose on-policy Proximal Policy Optimization (PPO, [75]) with two underlying architectures: Multilayer Perceptron (MLP) and Long Short-Term Memory (LSTM, [76]), as well as popular in robotics off-policy Soft Actor-Critic (SAC, [77]) and model-based Temporal Difference Learning for Model Predictive Control (TD-MPC2, [78]).

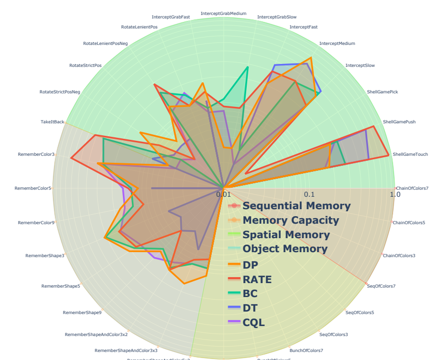


Figure 6: Results of Offline RL baselines with memory (RATE, DT) and without memory (BC-MLP, CQL-MLP, DP) on all 32 MIKASA-Robo tasks. Training was performed in `RGB` mode with sparse rewards (success condition).

We use an MLP as a memory-less baseline and an LSTM as a standard memory mechanism effective in POMDPs [6]. This allows direct comparison of memory-less vs. memory-enabled agents and validates our benchmark's ability to assess memory. We focus on these fundamental models for benchmark validation rather than broad algorithm comparison. To confirm all environments are solvable with 100% success rate (SR), we trained a PPO-MLP agent in `state` mode with full system access. Selected results appear in Figure 4, full results in Appendix F.

Training under the `RGB+joints` mode with dense rewards reveals the memory-intensive nature of our tasks. Using the `RememberColor-v0` task as an example, PPO-LSTM demonstrates superior performance compared to PPO-MLP when distinguishing between three colors (see Figure 5). However, both agents' success rates drop dramatically to near-zero as the task complexity increases to five or nine colors. Moreover, under sparse reward conditions, both architectures fail to solve even the three-color variant (see Appendix F, Figure 10). Additionally, our findings indicate that, while SAC and TD-MPC2 exhibit higher sample efficiency compared to PPO-MLP, when faced with more complex challenges, the lack of an explicit memory mechanism becomes a critical shortcoming, resulting in low performance, which also emphasizes the inappropriateness of algorithms common in the robotics community for memory-intensive tasks. These results validate our benchmark's effectiveness in evaluating agents' memory, showing clear performance degradation as memory demands increase.

Table 4: Performance of VLA models on selected memory-intensive tasks from the MIKASA-Robo benchmark. Reported values denote average success rates over 100 evaluation episodes (mean $\pm$ sem). Tasks include spatial reasoning (`ShellGameTouch`, `InterceptMedium`) and color-based memory retrieval (`RememberColor3/5/9`).

| Model | ShellGameTouch | InterceptMedium | RememberColor3 | RememberColor5 | RememberColor9 |
|---|---|---|---|---|---|
| **Octo-small** | $0.46 \pm 0.05$ | $0.39 \pm 0.04$ | $0.45 \pm 0.06$ | $0.17 \pm 0.03$ | $0.11 \pm 0.03$ |
| **OpenVLA** ($K$=4) | $0.12 \pm 0.05$ | $0.06 \pm 0.02$ | $0.21 \pm 0.00$ | $0.09 \pm 0.02$ | $0.08 \pm 0.02$ |
| **OpenVLA** ($K$=8) | $0.47 \pm 0.05$ | $0.14 \pm 0.03$ | $0.59 \pm 0.04$ | $0.16 \pm 0.03$ | $0.06 \pm 0.02$ |

### 6.3 Offline RL baselines

Since dense rewards are rarely available in the real world, we focus on sparse rewards represented as a binary success flag. Online learning models are difficult to train in this setting, so we evaluate five Offline RL methods: Decision Transformer (DT) [2] and Recurrent Action Transformer with Memory (RATE) [31] using Transformer backbones, Standard Behavioral Cloning (BC) and Conservative Q-Learning (CQL) [79] with MLP backbones, and Diffusion Policy (DP) [80], a recent method for robotic manipulation using diffusion models for direct action prediction.

Experimental results with Offline RL models trained on two RGB views and sparse rewards are shown in Figure 6. None of the models—including sequence-based ones—solved most MIKASA-Robo tasks, highlighting the benchmark's difficulty. Training used 1000 successful trajectories per task (see Appendix B). In particular, no model succeeded on tasks demanding high Memory Capacity or Sequential Memory, underscoring their complexity. Detailed Offline RL results are provided in Appendix Table 6.

### 6.4 VLA baselines

To assess Visual-Language-Action (VLA) models on memory-intensive robotic tasks, we evaluate two baselines: Octo [81] and OpenVLA [82]. While neither claims advanced memory mechanisms, these experiments reveal current memory capabilities in VLA agents.

Octo is a transformer with diffusion heads trained on 25 Open X-Embodiment datasets [83]; we fine-tuned only the readout heads with context length 10 and action chunk size `K=4`. OpenVLA employs a Prismatic-7B backbone [84], fine-tuned with LoRA adapters, chunking, and $L_1$ loss [85]; we tested `K=4` and `K=8`. Both models were trained on 250 expert trajectories per task with $128 \times 128$ RGB base/wrist views and end-effector control (see Appendix D).

Experimental results (Table 4) show clear trends. Octo (context size 10) surpasses random on simple tasks, hinting at limited memory, but fails to scale with complexity. OpenVLA behaves differently: with $K = 8$, it beats random on tasks like `RememberColor3` and `ShellGameTouch`, but collapses on harder tasks; with $K = 4$, it fails entirely. Larger chunks partly bypass memory by generating trajectories from early cues, but this breaks down with smaller chunks, exposing the lack of true memory.

The sharp performance drop on harder tasks underscores the need for dedicated memory architectures and validates MIKASA-Robo's multi-difficulty design to prevent such "shortcuts." Our Octo and OpenVLA results highlight a core gap in VLA models: without effective long-term memory, performance remains brittle, reinforcing the value of MIKASA-Robo.

## 7 Conclusion

We present **MIKASA**, a unified benchmark suite for evaluating memory in RL. Our work addresses key gaps in the field by introducing: (1) a taxonomy of memory types – object, spatial, sequential, and capacity; (2) **MIKASA-Base**, a standardized collection of open-source memory tasks; (3) **MIKASA-Robo**, a suite of 32 robotic manipulation tasks targeting diverse memory demands; and (4) accompanying offline datasets to support reproducible evaluation. Experiments with online, offline, and VLA agents reveal that current methods struggle with many tasks, highlighting the need for better memory architectures. MIKASA aims to guide and accelerate progress in memory-intensive RL for real-world applications.

## References


[1] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134, 1998. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(98)00023-X. URL https://www.sciencedirect.com/science/article/pii/S000437029800023X.

[2] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[3] A. Lampinen, S. Chan, A. Banino, and F. Hill. Towards mental time travel: a hierarchical memory for reinforcement learning agents. *Advances in Neural Information Processing Systems*, 34:28182–28195, 2021.

[4] E. Parisotto, F. Song, J. Rae, R. Pascanu, C. Gulcehre, S. Jayakumar, M. Jaderberg, R. L. Kaufman, A. Clark, S. Noury, et al. Stabilizing transformers for reinforcement learning. In *International conference on machine learning*, pages 7487–7498. PMLR, 2020.

[5] L. Meng, R. Gorbet, and D. Kulić. Memory-based deep reinforcement learning for pomdps. In *2021 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 5619–5626. IEEE, 2021.

[6] T. Ni, B. Eysenbach, and R. Salakhutdinov. Recurrent model-free rl can be a strong baseline for many pomdps. *arXiv preprint arXiv:2110.05038*, 2021.

[7] C. An, S. Gong, M. Zhong, X. Zhao, M. Li, J. Zhang, L. Kong, and X. Qiu. L-eval: Instituting standardized evaluation for long context language models. *arXiv preprint arXiv:2307.11088*, 2023.

[8] Y. Bai, X. Lv, J. Zhang, H. Lyu, J. Tang, Z. Huang, Z. Du, X. Liu, A. Zeng, L. Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. *arXiv preprint arXiv:2308.14508*, 2023.

[9] S. Morad, R. Kortvelesy, M. Bettini, S. Liwicki, and A. Prorok. POPGym: Benchmarking partially observable reinforcement learning. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=chDrutUTs0K.

[10] C.-C. Hung, T. Lillicrap, J. Abramson, Y. Wu, M. Mirza, F. Carnevale, A. Ahuja, and G. Wayne. Optimizing agent behavior over long time scales by transporting value, 2018. URL https://arxiv.org/abs/1810.06721.

[11] M. Pleines, M. Pallasch, F. Zimmer, and M. Preuss. Memory gym: Partially observable challenges to memory-based agents in endless episodes. *arXiv preprint arXiv:2309.17207*, 2023.

[12] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47: 253–279, 2013.

[13] E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi:10.1109/IROS.2012.6386109.

[14] B. Ai, W. Gao, D. Hsu, et al. Deep visual navigation under partial observability. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 9439–9446. IEEE, 2022.

[15] K. Yadav, J. Krantz, R. Ramrakhya, S. K. Ramakrishnan, J. Yang, A. Wang, J. Turner, A. Gokaslan, V.-P. Berges, R. Mootaghi, O. Maksymets, A. X. Chang, M. Savva, A. Clegg, D. S. Chaplot, and D. Batra. Habitat challenge 2023. https://aihabitat.org/challenge/2023/, 2023.

[16] H. Kurniawati. Partially observable markov decision processes and robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 5(1):253–277, 2022.

[17] M. Lauri, D. Hsu, and J. Pajarinen. Partially observable markov decision processes in robotics: A survey. *IEEE Transactions on Robotics*, 39(1):21–40, Feb. 2023. ISSN 1941-0468. doi:10.1109/tro.2022.3200138. URL http://dx.doi.org/10.1109/TRO.2022.3200138.

[18] M. T. J. Spaan. Partially observable Markov decision processes. In M. Wiering and M. van Otterlo, editors, *Reinforcement Learning: State of the Art*, pages 387–414. Springer Verlag, 2012.

[19] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

[20] J. Z. Leibo, C. de Masson d'Autume, D. Zoran, D. Amos, C. Beattie, K. Anderson, A. G. Castañeda, M. Sanchez, S. Green, A. Gruslys, S. Legg, D. Hassabis, and M. M. Botvinick. Psychlab: A psychology laboratory for deep reinforcement learning agents, 2018. URL https://arxiv.org/abs/1801.08116.

[21] M. Chevalier-Boisvert, B. Dai, M. Towers, R. de Lazcano, L. Willems, S. Lahlou, S. Pal, P. S. Castro, and J. Terry. Minigrid & miniworld: Modular & customizable reinforcement learning environments for goal-oriented tasks, 2023. URL https://arxiv.org/abs/2306.13831.

[22] M. Samvelyan, R. Kirk, V. Kurin, J. Parker-Holder, M. Jiang, E. Hambro, F. Petroni, H. Küttler, E. Grefenstette, and T. Rocktäschel. Minihack the planet: A sandbox for open-ended reinforcement learning research, 2021. URL https://arxiv.org/abs/2109.13202.

[23] H. Küttler, N. Nardelli, A. H. Miller, R. Raileanu, M. Selvatici, E. Grefenstette, and T. Rocktäschel. The nethack learning environment, 2020. URL https://arxiv.org/abs/2006.13760.

[24] M. Chevalier-Boisvert, D. Bahdanau, S. Lahlou, L. Willems, C. Saharia, T. H. Nguyen, and Y. Bengio. Babyai: A platform to study the sample efficiency of grounded language learning, 2019. URL https://arxiv.org/abs/1810.08272.

[25] I. Osband, Y. Doron, M. Hessel, J. Aslanides, E. Sezener, A. Saraiva, K. McKinney, T. Lattimore, C. Szepesvári, S. Singh, B. Van Roy, R. Sutton, D. Silver, and H. van Hasselt. Behaviour suite for reinforcement learning. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rygf-kSYwH.

[26] J. Pasukonis, T. Lillicrap, and D. Hafner. Evaluating long-term memory in 3d mazes, 2022. URL https://arxiv.org/abs/2210.13383.

[27] M. Hausknecht and P. Stone. Deep recurrent q-learning for partially observable mdps, 2015.

[28] K. Esslinger, R. Platt, and C. Amato. Deep transformer q-networks for partially observable reinforcement learning. *arXiv preprint arXiv:2206.01078*, 2022.

[29] J. Grigsby, L. Fan, and Y. Zhu. Amago: Scalable in-context reinforcement learning for adaptive agents, 2024. URL https://arxiv.org/abs/2310.09971.

[30] M. R. Samsami, A. Zholus, J. Rajendran, and S. Chandar. Mastering memory tasks with world models, 2024. URL https://arxiv.org/abs/2403.04253.

[31] E. Cherepanov, A. Staroverov, D. Yudin, A. K. Kovalev, and A. I. Panov. Recurrent action transformer with memory. *arXiv preprint arXiv:2306.09459*, 2024. URL https://arxiv.org/abs/2306.09459.

[32] A. Goyal, A. Friesen, A. Banino, T. Weber, N. R. Ke, A. P. Badia, A. Guez, M. Mirza, P. C. Humphreys, K. Konyushova, et al. Retrieval-augmented reinforcement learning. In *International Conference on Machine Learning*, pages 7740–7765. PMLR, 2022.

[33] C. Lu, Y. Schroecker, A. Gu, E. Parisotto, J. Foerster, S. Singh, and F. Behbahani. Structured state space models for in-context reinforcement learning, 2023. URL https://arxiv.org/abs/2303.03982.

[34] E. Parisotto and R. Salakhutdinov. Neural map: Structured memory for deep reinforcement learning, 2017. URL https://arxiv.org/abs/1702.08360.

[35] Y. Kang, E. Zhao, Y. Zang, L. Li, K. Li, P. Tao, and J. Xing. Sample efficient reinforcement learning using graph-based memory reconstruction. *IEEE Transactions on Artificial Intelligence*, 5(2):751–762, 2024. doi:10.1109/TAI.2023.3268612.

[36] Z. Lin, T. Zhao, G. Yang, and L. Zhang. Episodic memory deep q-networks, 2018. URL https://arxiv.org/abs/1805.07603.

[37] M. Fortunato, M. Tan, R. Faulkner, S. Hansen, A. P. Badia, G. Buttimore, C. Deck, J. Z. Leibo, and C. Blundell. Generalization of reinforcement learners with working and episodic memory, 2020. URL https://arxiv.org/abs/1910.13406.

[38] J. Oh, V. Chockalingam, S. Singh, and H. Lee. Control of memory, active perception, and action in minecraft, 2016. URL https://arxiv.org/abs/1605.09128.

[39] P. Zhu, X. Li, P. Poupart, and G. Miao. On improving deep reinforcement learning for pomdps, 2018. URL https://arxiv.org/abs/1704.07978.

[40] F. Hill, O. Tieleman, T. von Glehn, N. Wong, H. Merzic, and S. Clark. Grounded language learning fast and slow, 2020. URL https://arxiv.org/abs/2009.01719.

[41] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney. Recurrent experience replay in distributed reinforcement learning. In *International Conference on Learning Representations*, 2018. URL https://api.semanticscholar.org/CorpusID:59345798.

[42] G. Zhu, Z. Lin, G. Yang, and C. Zhang. Episodic reinforcement learning with associative memory. In *International Conference on Learning Representations*, 2020. URL https://api.semanticscholar.org/CorpusID:212799813.

[43] R. Yan, Y. Gan, Y. Wu, J. Xing, L. Liangn, Y. Zhu, and Y. Cai. Adamemento: Adaptive memory-assisted policy optimization for reinforcement learning, 2024. URL https://arxiv.org/abs/2410.04498.

[44] T. Ni, M. Ma, B. Eysenbach, and P.-L. Bacon. When do transformers shine in rl? decoupling memory from credit assignment, 2023. URL https://arxiv.org/abs/2307.03864.

[45] W. Yue, B. Liu, and P. Stone. Learning memory mechanisms for decision making through demonstrations. *arXiv preprint arXiv:2411.07954*, 2024.

[46] E. Cherepanov, N. Kachaev, A. Zholus, A. K. Kovalev, and A. I. Panov. Unraveling the complexity of memory in rl agents: an approach for classification and evaluation, 2024. URL https://arxiv.org/abs/2412.06531.

[47] H. Fang, M. Grotz, W. Pumacay, Y. R. Wang, D. Fox, R. Krishna, and J. Duan. Sam2act: Integrating visual foundation model with a memory architecture for robotic manipulation. *arXiv preprint arXiv:2501.18564*, 2025.

[48] S. James, Z. Ma, D. R. Arrojo, and A. J. Davison. Rlbench: The robot learning benchmark & learning environment. *IEEE Robotics and Automation Letters*, 5(2):3019–3026, 2020.

[49] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[50] J. Piaget. The origins of intelligence in children. *International University*, 1952.

[51] A. M. Liberman, K. S. Harris, H. S. Hoffman, and B. C. Griffith. The discrimination of speech sounds within and across phoneme boundaries. *Journal of experimental psychology*, 54(5): 358, 1957.

[52] A. Baddeley. Working memory. *Science*, 255(5044):556–559, 1992.

[53] M. Daneman and P. A. Carpenter. Individual differences in working memory and reading. *Journal of verbal learning and verbal behavior*, 19(4):450–466, 1980.

[54] D. Kuhn. The development of causal reasoning. *Wiley Interdisciplinary Reviews: Cognitive Science*, 3(3):327–335, 2012.

[55] S. Heckers, M. Zalesak, A. P. Weiss, T. Ditman, and D. Titone. Hippocampal activation during transitive inference in humans. *Hippocampus*, 14(2):153–162, 2004.

[56] S. Tao, F. Xiang, A. Shukla, Y. Qin, X. Hinrichsen, X. Yuan, C. Bao, X. Lin, Y. Liu, T.-k. Chan, et al. Maniskill3: Gpu parallelized robotics simulation and rendering for generalizable embodied ai. *arXiv preprint arXiv:2410.00425*, 2024.

[57] A. Shukla, S. Tao, and H. Su. Maniskill-hab: A benchmark for low-level manipulation in home rearrangement tasks. *arXiv preprint arXiv:2412.13211*, 2024.

[58] B. Han, M. Parakh, D. Geng, J. A. Defay, G. Luyang, and J. Deng. Fetchbench: A simulation benchmark for robot fetching. *arXiv preprint arXiv:2406.11793*, 2024.

[59] S. Nasiriany, A. Maddukuri, L. Zhang, A. Parikh, A. Lo, A. Joshi, A. Mandlekar, and Y. Zhu. Robocasa: Large-scale simulation of everyday tasks for generalist robots. *arXiv preprint arXiv:2406.02523*, 2024.

[60] R. de Lazcano, K. Andreas, J. J. Tai, S. R. Lee, and J. Terry. Gymnasium robotics, 2024. URL http://github.com/Farama-Foundation/Gymnasium-Robotics.

[61] C. Li, R. Zhang, J. Wong, C. Gokmen, S. Srivastava, R. Martín-Martín, C. Wang, G. Levine, W. Ai, B. Martinez, et al. Behavior-1k: A human-centered, embodied ai benchmark with 1,000 everyday activities and realistic simulation. *arXiv preprint arXiv:2403.09227*, 2024.

[62] R. Gong, J. Huang, Y. Zhao, H. Geng, X. Gao, Q. Wu, W. Ai, Z. Zhou, D. Terzopoulos, S.-C. Zhu, et al. Arnold: A benchmark for language-grounded task learning with continuous states in realistic 3d scenes. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 20483–20495, 2023.

[63] C. Li, F. Xia, R. Martín-Martín, M. Lingelbach, S. Srivastava, B. Shen, K. E. Vainio, C. Gokmen, G. Dharan, T. Jain, A. Kurenkov, K. Liu, H. Gweon, J. Wu, L. Fei-Fei, and S. Savarese. igibson 2.0: Object-centric simulation for robot learning of everyday household tasks. In A. Faust, D. Hsu, and G. Neumann, editors, *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pages 455–465. PMLR, 08–11 Nov 2022. URL https://proceedings.mlr.press/v164/li22b.html.

[64] Y. Jiang, A. Gupta, Z. Zhang, G. Wang, Y. Dou, Y. Chen, L. Fei-Fei, A. Anandkumar, Y. Zhu, and L. Fan. Vima: General robot manipulation with multimodal prompts. *arXiv preprint arXiv:2210.03094*, 2(3):6, 2022.

[65] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[66] Q. Gallouédec, N. Cazin, E. Dellandréa, and L. Chen. panda-gym: Open-Source Goal-Conditioned Environments for Robotic Learning. *4th Robot Learning Workshop: Self-Supervised and Lifelong Learning at NeurIPS*, 2021.

[67] A. Szot, A. Clegg, E. Undersander, E. Wijmans, Y. Zhao, J. Turner, N. Maestre, M. Mukadam, D. S. Chaplot, O. Maksymets, et al. Habitat 2.0: Training home assistants to rearrange their habitat. *Advances in neural information processing systems*, 34:251–266, 2021.

[68] T. Yu, D. Quillen, Z. He, R. Julian, K. Hausman, C. Finn, and S. Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on robot learning*, pages 1094–1100. PMLR, 2020.

[69] O. Ahmed, F. Träuble, A. Goyal, A. Neitz, Y. Bengio, B. Schölkopf, M. Wüthrich, and S. Bauer. Causalworld: A robotic manipulation benchmark for causal structure and transfer learning. *arXiv preprint arXiv:2010.04296*, 2020.

[70] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, K. Lin, S. Nasiriany, and Y. Zhu. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.

[71] S. Tunyasuvunakool, A. Muldal, Y. Doron, S. Liu, S. Bohez, J. Merel, T. Erez, T. Lillicrap, N. Heess, and Y. Tassa. dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020.

[72] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*, 2019.

[73] L. Fan, Y. Zhu, J. Zhu, Z. Liu, O. Zeng, A. Gupta, J. Creus-Costa, S. Savarese, and L. Fei-Fei. Surreal: Open-source reinforcement learning framework and robot manipulation benchmark. In A. Billard, A. Dragan, J. Peters, and J. Morimoto, editors, *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pages 767–782. PMLR, 29–31 Oct 2018. URL https://proceedings.mlr.press/v87/fan18a.html.

[74] E. Kolve, R. Mottaghi, W. Han, E. VanderBilt, L. Weihs, A. Herrasti, M. Deitke, K. Ehsani, D. Gordon, Y. Zhu, et al. Ai2-thor: An interactive 3d environment for visual ai. *arXiv preprint arXiv:1712.05474*, 2017.

[75] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[76] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, nov 1997. ISSN 0899-7667. doi:10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

[77] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. Pmlr, 2018.

[78] N. Hansen, H. Su, and X. Wang. Td-mpc2: Scalable, robust world models for continuous control. *arXiv preprint arXiv:2310.16828*, 2023.

[79] A. Kumar, A. Zhou, G. Tucker, and S. Levine. Conservative q-learning for offline reinforcement learning. *Advances in neural information processing systems*, 33:1179–1191, 2020.

[80] C. Chi, Z. Xu, S. Feng, E. Cousineau, Y. Du, B. Burchfiel, R. Tedrake, and S. Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, page 02783649241273668, 2023.

[81] O. M. Team, D. Ghosh, H. Walke, K. Pertsch, K. Black, O. Mees, S. Dasari, J. Hejna, T. Kreiman, C. Xu, J. Luo, Y. L. Tan, L. Y. Chen, P. Sanketi, Q. Vuong, T. Xiao, D. Sadigh, C. Finn, and S. Levine. Octo: An open-source generalist robot policy, 2024. URL https://arxiv.org/abs/2405.12213.

[82] M. J. Kim, K. Pertsch, S. Karamcheti, T. Xiao, A. Balakrishna, S. Nair, R. Rafailov, E. Foster, G. Lam, P. Sanketi, Q. Vuong, T. Kollar, B. Burchfiel, R. Tedrake, D. Sadigh, S. Levine, P. Liang, and C. Finn. Openvla: An open-source vision-language-action model, 2024. URL https://arxiv.org/abs/2406.09246.

[83] E. Collaboration, A. O'Neill, A. Rehman, A. Gupta, A. Maddukuri, A. Gupta, A. Padalkar, A. Lee, A. Pooley, A. Gupta, A. Mandlekar, A. Jain, A. Tung, A. Bewley, A. Herzog, A. Irpan, A. Khazatsky, A. Rai, A. Gupta, A. Wang, A. Kolobov, A. Singh, A. Garg, A. Kembhavi, A. Xie, A. Brohan, A. Raffin, A. Sharma, A. Yavary, A. Jain, A. Balakrishna, A. Wahid, B. Burgess-Limerick, B. Kim, B. Schölkopf, B. Wulfe, B. Ichter, C. Lu, C. Xu, C. Le, C. Finn, C. Wang, C. Xu, C. Chi, C. Huang, C. Chan, C. Agia, C. Pan, C. Fu, C. Devin, D. Xu, D. Morton, D. Driess, D. Chen, D. Pathak, D. Shah, D. Büchler, D. Jayaraman, D. Kalashnikov, D. Sadigh, E. Johns, E. Foster, F. Liu, F. Ceola, F. Xia, F. Zhao, F. V. Frujeri, F. Stulp, G. Zhou, G. S. Sukhatme, G. Salhotra, G. Yan, G. Feng, G. Schiavi, G. Berseth, G. Kahn, G. Yang, G. Wang, H. Su, H.-S. Fang, H. Shi, H. Bao, H. B. Amor, H. I. Christensen, H. Furuta, H. Bharadhwaj, H. Walke, H. Fang, H. Ha, I. Mordatch, I. Radosavovic, I. Leal, J. Liang, J. Abou-Chakra, J. Kim, J. Drake, J. Peters, J. Schneider, J. Hsu, J. Vakil, J. Bohg, J. Bingham, J. Wu, J. Gao, J. Hu, J. Wu, J. Wu, J. Sun, J. Luo, J. Gu, J. Tan, J. Oh, J. Wu, J. Lu, J. Yang, J. Malik, J. Silvério, J. Hejna, J. Booher, J. Tompson, J. Yang, J. Salvador, J. J. Lim, J. Han, K. Wang, K. Rao, K. Pertsch, K. Hausman, K. Go, K. Gopalakrishnan, K. Goldberg, K. Byrne, K. Oslund, K. Kawaharazuka, K. Black, K. Lin, K. Zhang, K. Ehsani, K. Lekkala, K. Ellis, K. Rana, K. Srinivasan, K. Fang, K. P. Singh, K.-H. Zeng, K. Hatch, K. Hsu, L. Itti, L. Y. Chen, L. Pinto, L. Fei-Fei, L. Tan, L. J. Fan, L. Ott, L. Lee, L. Weihs, M. Chen, M. Lepert, M. Memmel, M. Tomizuka, M. Itkina, M. G. Castro, M. Spero, M. Du, M. Ahn, M. C. Yip, M. Zhang, M. Ding, M. Heo, M. K. Srirama, M. Sharma, M. J. Kim, M. Z. Irshad, N. Kanazawa, N. Hansen, N. Heess, N. J. Joshi, N. Suenderhauf, N. Liu, N. D. Palo, N. M. M. Shafiullah, O. Mees, O. Kroemer, O. Bastani, P. R. Sanketi, P. T. Miller, P. Yin, P. Wohlhart, P. Xu, P. D. Fagan, P. Mitrano, P. Sermanet, P. Abbeel, P. Sundaresan, Q. Chen, Q. Vuong, R. Rafailov, R. Tian, R. Doshi, R. Martín-Martín, R. Baijal, R. Scalise, R. Hendrix, R. Lin, R. Qian, R. Zhang, R. Mendonca, R. Shah, R. Hoque, R. Julian, S. Bustamante, S. Kirmani, S. Levine, S. Lin, S. Moore, S. Bahl, S. Dass, S. Sonawani, S. Tulsiani, S. Song, S. Xu, S. Haldar, S. Karamcheti, S. Adebola, S. Guist, S. Nasiriany, S. Schaal, S. Welker, S. Tian, S. Ramamoorthy, S. Dasari, S. Belkhale, S. Park, S. Nair, S. Mirchandani, T. Osa, T. Gupta, T. Harada, T. Matsushima, T. Xiao, T. Kollar, T. Yu, T. Ding, T. Davchev, T. Z. Zhao, T. Armstrong, T. Darrell, T. Chung, V. Jain, V. Kumar, V. Vanhoucke, V. Guizilini, W. Zhan, W. Zhou, W. Burgard, X. Chen, X. Chen, X. Wang, X. Zhu, X. Geng, X. Liu, X. Liangwei, X. Li, Y. Pang, Y. Lu, Y. J. Ma, Y. Kim, Y. Chebotar, Y. Zhou, Y. Zhu, Y. Wu, Y. Xu, Y. Wang, Y. Bisk, Y. Dou, Y. Cho, Y. Lee, Y. Cui, Y. Cao, Y.-H. Wu, Y. Tang, Y. Zhu, Y. Zhang, Y. Jiang, Y. Li, Y. Li, Y. Iwasawa, Y. Matsuo, Z. Ma, Z. Xu, Z. J. Cui, Z. Zhang, Z. Fu, and Z. Lin. Open x-embodiment: Robotic learning datasets and rt-x models, 2025. URL https://arxiv.org/abs/2310.08864.

[84] S. Karamcheti, S. Nair, A. Balakrishna, P. Liang, T. Kollar, and D. Sadigh. Prismatic vlms: Investigating the design space of visually-conditioned language models, 2024. URL https://arxiv.org/abs/2402.07865.

[85] M. J. Kim, C. Finn, and P. Liang. Fine-tuning vision-language-action models: Optimizing speed and success, 2025. URL https://arxiv.org/abs/2502.19645.

[86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986. URL https://api.semanticscholar.org/CorpusID:205001834.

[87] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[88] D. Wierstra, A. Förster, J. Peters, and J. Schmidhuber. Recurrent policy gradients. *Logic Journal of the IGPL*, 18:620–634, 10 2010. doi:10.1093/jigpal/jzp049.

[89] A. Gu, K. Goel, and C. Ré. Efficiently modeling long sequences with structured state spaces. *arXiv preprint arXiv:2111.00396*, 2021.

[90] J. T. H. Smith, A. Warrington, and S. W. Linderman. Simplified state space layers for sequence modeling, 2023. URL https://arxiv.org/abs/2208.04933.

[91] A. Gu and T. Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[92] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson. Learning latent dynamics for planning from pixels. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2555–2565. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/hafner19a.html.

[93] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[94] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun. Graph neural networks: A review of methods and applications. *AI open*, 1:57–81, 2020.

[95] D. Zhu, L. E. Li, and M. Elhoseiny. Value memory graph: A graph-structured world model for offline reinforcement learning, 2023. URL https://arxiv.org/abs/2206.04384.

[96] J. Humplik, A. Galashov, L. Hasenclever, P. A. Ortega, Y. W. Teh, and N. Heess. Meta reinforcement learning as task inference, 2019. URL https://arxiv.org/abs/1905.06424.

[97] A. Sorokin, N. Buzun, L. Pugachev, and M. Burtsev. Explain my surprise: Learning efficient long-term memory by predicting uncertain outcomes. *Advances in Neural Information Processing Systems*, 35:36875–36888, 2022.

[98] A. G. Barto, R. S. Sutton, and C. W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983. doi:10.1109/TSMC.1983.6313077.

[99] K. Doya. Temporal difference learning in continuous time and space. In *Neural Information Processing Systems*, 1995. URL https://api.semanticscholar.org/CorpusID:1170136.

[100] A. Slivkins. Introduction to multi-armed bandits, 2024. URL https://arxiv.org/abs/1904.07272.

# Table of Contents

# A MIKASA-Robo Implementation Details

An example of running the environment from the MIKASA-Robo benchmark is shown in Code 1. For ease of debugging, we also added various wrappers (found in `mikasa_robo_suite/utils/wrappers/`) that display useful information about the episode on the video (Code 2). Thus, `RenderStepInfoWrapper()` displays the current step in the environment; `DebugRewardWrapper()` displays information about the full reward at the current step in the environment; `DebugRewardWrapper()` displays information about each component that generates the reward function at the current step. In addition, we also added task-specific wrappers for each environment. For example, `RememberColorInfoWrapper()` displays the target color of the cube in the `RememberColor-v0` task, and `ShellGameRenderCupInfoWrapper()` displays which mug the ball is actually under in the `ShellGame-v0` task.

Code 1: Getting started with MIKASA-Robo using the `RememberColor9-v0` environment.

```
import mikasa_robo_suite
from mikasa_robo_suite.utils.wrappers import
    ↪ StateOnlyTensorToDictWrapper
from tqdm.notebook import tqdm
import torch
import gymnasium as gym

# Create the environment via gym.make()
# obs_mode="rgb" for modes "RGB", "RGB+joint", "RGB+oracle" etc.
# obs_mode="state" for mode "state"
episode_timeout = 90
env = gym.make("RememberColor9-v0", num_envs=512 obs_mode="rgb",
    ↪ render_mode="all")
env = StateOnlyTensorToDictWrapper(env) # * always use this wrapper!

obs, _ = env.reset(seed=42)
print(obs.keys())
for i in tqdm(range(episode_timeout)):
    action = torch.from_numpy(env.action_space.sample())
    obs, reward, terminated, truncated, info = env.step(action)

env.close()
```

Code 2: MIKASA-Robo wrappers system.

```
import mikasa_robo_suite, torch
from mikasa_robo_suite.dataset_collectors.get_mikasa_robo_datasets
    ↪ import env_info
import gymnasium as gym
from mani_skill.utils.wrappers import RecordEpisode
from IPython.display import Video

env = gym.make("RememberColor9-v0", num_envs=512, obs_mode="rgb",
    ↪ render_mode="all")
state_wrappers_list, episode_timeout = env_info("RememberColor9-v0")
for wrapper_class, wrapper_kwargs in state_wrappers_list:
    env = wrapper_class(env, **wrapper_kwargs)
env = RecordEpisode(env, f"./videos", max_steps_per_video=
    ↪ episode_timeout)

obs, _ = env.reset(seed=42)
for i in range(episode_timeout):
    action = torch.from_numpy(env.action_space.sample())
    obs, reward, terminated, truncated, info = env.step(action)

Video(f"./videos/0.mp4", embed=True, width=640)
env.close()
```

## B MIKASA-Robo Datasets for Offline RL

To train Offline RL baselines on camera images (in "RGB" mode) with sparse rewards (success condition), we collected datasets for each of the 32 MIKASA-Robo tasks. Datasets were collected using a PPO-MLP agent trained to SR=100% in "state" mode (i.e., with full information about the task being solved) with sparse rewards (success condition). Thus, each dataset is represented by 1000 successful trajectories, where each trajectory consists of:

1. "rgb" (shape: $(T, 128, 128, 6)$) - two RGB images (view from above and from the gripper)
2. "joints" (shape: $(T, 25)$) - Tool Center Point (TCP) position and rotation, and joint positions and velocities
3. "action" (shape: $(T, 8)$) - action (8-dimensional vector)
4. "reward" (shape: $(T, )$) - (dense) reward for each step
5. "success" (shape: $(T,)$) - (sparse) success flag for each step
6. "done" (shape: $(T,)$) - done flag for each step

These datasets are available for download from the project website. We have also published the weights of the PPO-MLP agent used to collect the dataset, as well as scripts for collecting datasets of any size, to our repository.

## C MIKASA-Base Implementation Details

An example of running an environment from the MIKASA-Base benchmark is shown in Code 3. MIKASA-Base supports the standard Gymnasium API and is fully compatible with all its wrappers. This allows users to leverage various functionalities, including parallelization using `AsyncVectorEnv`. MIKASA-Base provides a predefined set of environments with different levels of difficulty. However, users can customize the environment parameters by passing specific arguments (see Code 3).

Code 3: Example code for running `MemoryLength-v0` environment.

```python
import mikasa_base
import gymnasium as gym

# use pre-defined env
# env_id = "MemoryLengthEasy-v0"
# env_kwargs = None

# create env using custom parameters
env_id = "MemoryLength-v0"
env_kwargs = {"memory_length": 10, "num_bits": 1}
seed = 123

env = gym.make(env_id, env_kwargs)

obs, _ = env.reset(seed=seed)

for i in range(11):
    action = env.action_space.sample()
    next_obs, reward, terminations, truncations, infos = env.step(
        ↪ action)
env.close()
```

## D MIKASA-Robo setup for VLA baselines

For experiments involving Vision-Language-Action (VLA) models, we focused on a representative subset of spatial and object memory tasks from MIKASA-Robo. For each task, we generated a

Table 5: Tasks configurations for fine-tuning VLA models. The table lists the task ID, number of evaluation steps (T), and the associated language instruction

| Task | T | Language instruction |
|---|---|---|
| `RememberColor3/5/9-v0` | 60 | Remember the color of the cube and then pick the matching one |
| `ShellGameTouch-v0` | 90 | Memorize the position of the cup covering the ball, then pick that cup |
| `InterceptMedium-v0` | 90 | Track the ball's movement, estimate its velocity, then aim the ball at the target |

dataset of 250 episodes using an oracle PPO policy with full access to the environment state. At every timestep, the policy recorded two synchronized RGB frames (one from the static "base" camera and one from the robot's wrist camera) along with the corresponding end-effector control actions ( `pd_ee_delta_pose` controller from [56]). Each task was also paired with a concise language instruction (see Table 5).

All VLA baselines were trained for 50000 iterations and evaluated independently on each task. Complete training/evaluation scripts, language instruction templates, and detailed model hyperparameter settings are provided in the accompanying supplementary code.

# E Memory Mechanisms in RL

In RL, memory mechanisms are techniques or models used to enable agents to retain and recall information from past interactions with the environment.

There are several approaches to incorporating memory into RL, including recurrent neural networks (RNNs) [86, 76, 87] which uses hidden states to store information from previous steps [88, 27], state-space models (SSMs) [89, 90, 91] which uses system state to store historical information [92, 30], transformers [93] which uses attention mechanism to capture sequential dependencies inside the context window [4, 3, 44], graph neural networks (GNNs) [94] which uses graphs to store information [95, 35] etc. Popular agents with memory mechanisms are summarized in Table 2.

# F Classic baselines performance on the MIKASA-Robo benchmark

In this section, we present a comprehensive evaluation of PPO-MLP and PPO-LSTM baselines on our MIKASA-Robo benchmark. Our experiments with PPO-MLP in `state` mode using dense rewards demonstrate perfect performance across all tasks, consistently achieving 100% success rate, as shown in Figure 7 and Figure 8. This remarkable performance serves as a crucial validation of our benchmark design: when an agent has access to complete state information and receives dense rewards, it can master these tasks completely. Therefore, any performance degradation in `RGB+joints` mode observed with other algorithms or training configurations must stem from the algorithmic limitations or learning challenges rather than any inherent flaws in the task design. This empirical evidence confirms that our environments are well-calibrated and properly designed, establishing a solid foundation for evaluating memory-enhanced algorithms. All results are presented as mean $\pm$ standard error of the mean (SEM), where the mean is computed across three independent training runs, and each trained agent is evaluated on 16 different random seeds to ensure robust performance assessment.

The performance evaluation of PPO-MLP and PPO-LSTM with dense rewards in the `RGB+joints` mode is presented in Figure 9. This mode specifically tests the agents' memory capabilities, as it requires remembering and utilizing historical information to solve the tasks. Our results demonstrate a clear distinction between memory-less and memory-enhanced architectures, while also revealing the limitations of conventional memory mechanisms.

Consider the `RememberColor-v0` environment as an illustrative example. In its simplest configuration with three cubes, the memory-less PPO-MLP achieves only 25% success rate. In contrast, PPO-LSTM, leveraging its memory mechanism, achieves perfect performance with 100% success rate.

However, as task complexity increases to five or nine cubes, even the LSTM's memory capabilities prove insufficient, with performance degrading significantly.

These results validate two key aspects of our benchmark: first, its effectiveness in distinguishing between memory-less and memory-enhanced architectures, and second, its ability to challenge even sophisticated memory mechanisms as task complexity increases. This demonstrates that MIKASA-Robo provides a competitive yet meaningful evaluation framework for developing and testing advanced memory-enhanced agents.

Our evaluation of PPO-MLP and PPO-LSTM baselines under sparse reward conditions in `RGB+joints` mode reveals the true challenge of our benchmark tasks. As shown in Figure 10, both architectures – even the memory-enhanced LSTM – consistently fail to achieve any meaningful success rate across nearly all considered environments. This striking result underscores the extreme difficulty of memory-intensive manipulation tasks when only terminal rewards are available, highlighting the substantial gap between current algorithms and the level of memory capabilities required for real-world robotic applications.

# G Experiments Reproducing and Compute Resources

All baselines were trained and evaluated under a reproducible standardized setup. Training for every algorithm was performed on a single NVIDIA A100 GPU. For evaluation, each task was run for 100 independent episodes, with environment and agent random seeds ranging from 1 to 100. We report performance metrics as the mean success rate $\pm$ the standard error of the mean (SEM) over these 100 trials.

# H MIKASA-Robo Detailed Tasks Description

In this section, we provide comprehensive descriptions of the 32 memory-intensive tasks that comprise the MIKASA-Robo benchmark. Each task is designed to evaluate specific aspects of memory capabilities in robotic manipulation, ranging from object tracking and spatial memory to sequential decision-making. For each task, we detail its objective, memory requirements, observation space, reward structure, and success criteria. Additionally, we explain how task complexity increases across different variants and discuss the specific memory challenges they present. The following subsections describe each task category and its variants in detail.

Each of the proposed environment supports multiple observation modes:

- **State**: Full state information including ball position
- **RGB+joints**: Two camera views (top-down and gripper) plus robot joint states
- **RGB**: Only visual information from two cameras

In the case of `RotateLenient-v0` and `RotateStrict-v0`, the prompt information available at each step is additionally added to each observation.

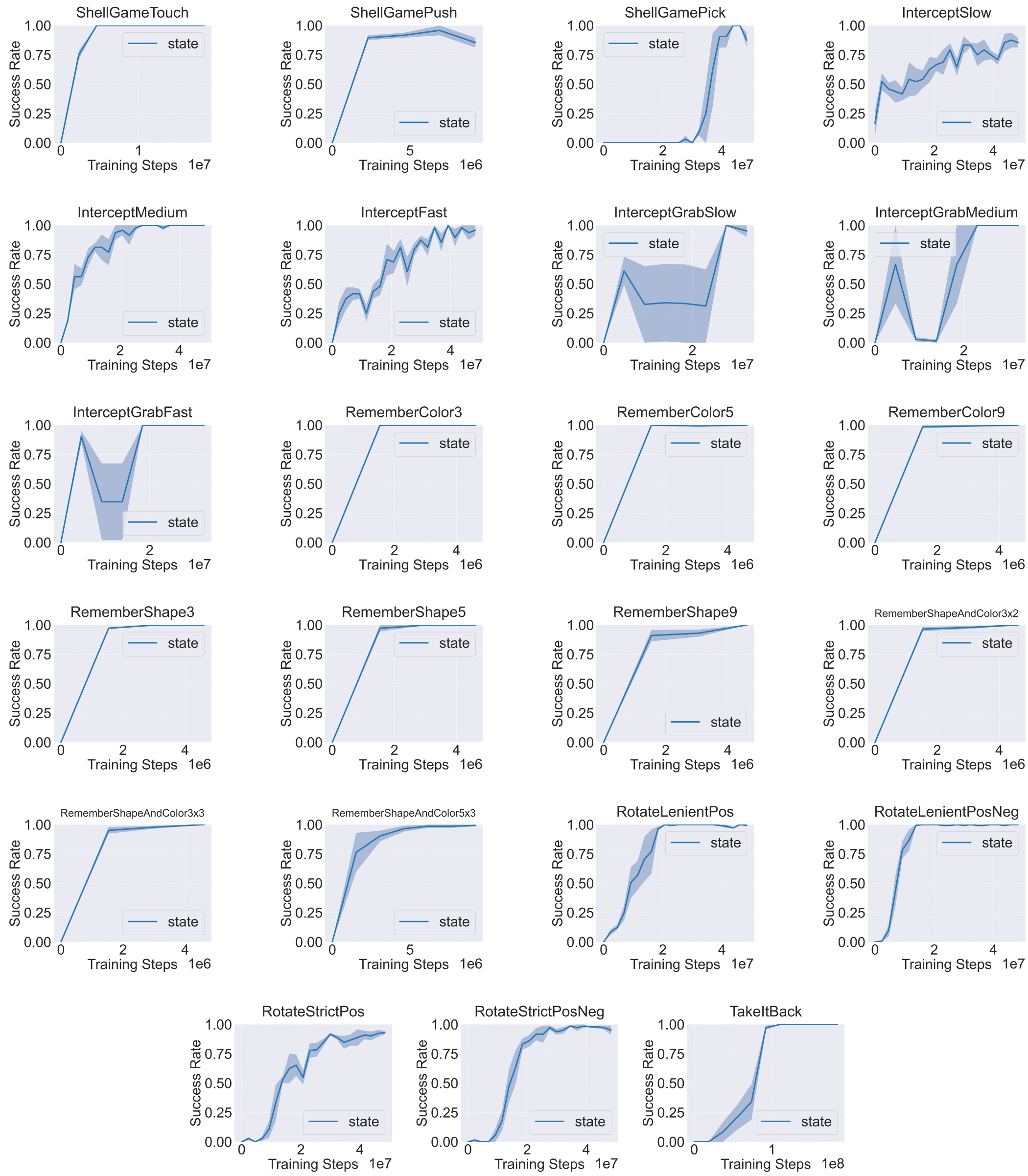


Figure 7: Demonstration of PPO-MLP performance on MIKASA-Robo benchmark when trained with oracle-level `state` information. In this learning mode, MDP problem formulation is considered, i.e. memory is not required for successful problem solving. At the same time, the obtained results show that it is possible to solve these problems and obtain 100% Success Rate.

854

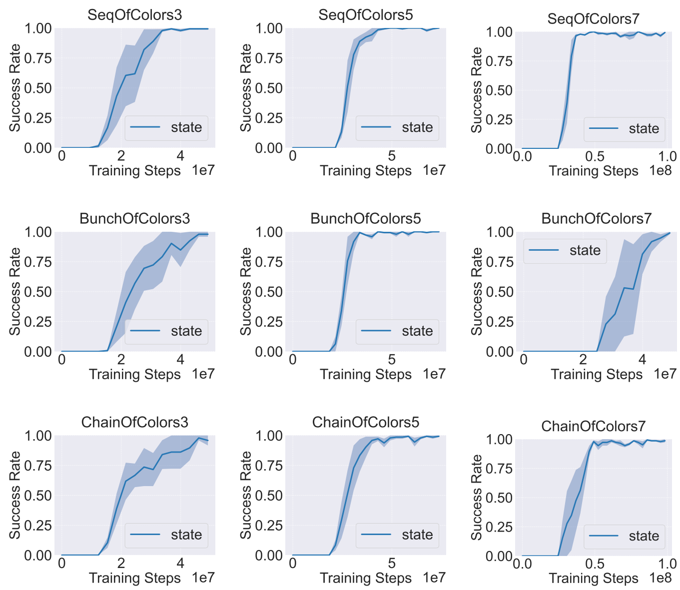


Figure 8: Demonstration of PPO-MLP performance on MIKASA-Robo benchmark when trained with oracle-level `state` information. Results are shown for memory capacity (`SeqOfColors[3,5,7]-v0`, `BunchOfColors[3,5,7]-v0`) and sequential memory (`ChainOfColors[3,5,7]-v0`).

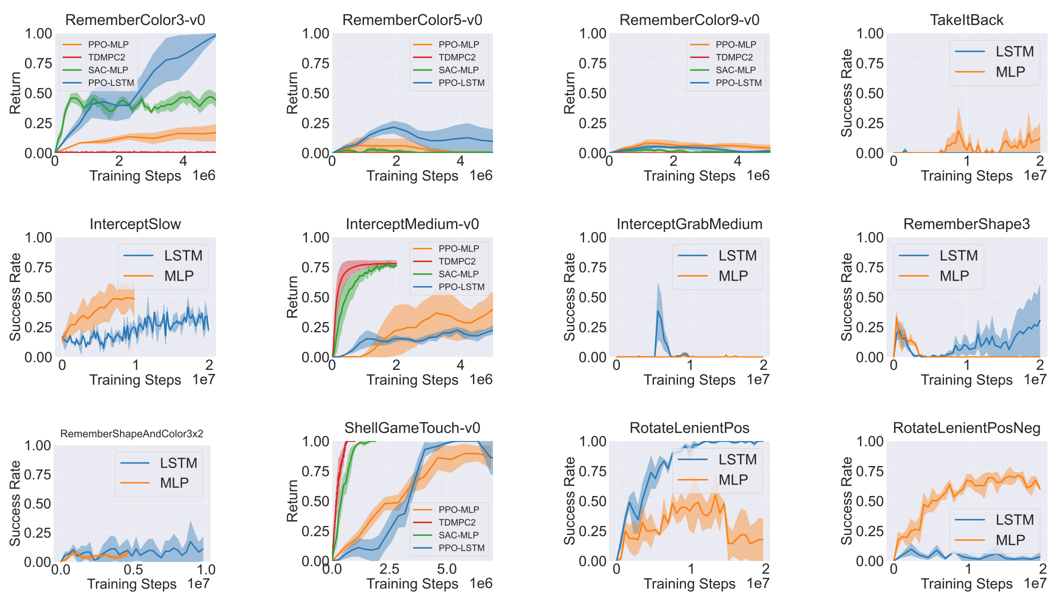


Figure 9: Performance evaluation of PPO-MLP and PPO-LSTM on the MIKASA-Robo benchmark using the “RGB+joints” training mode with dense reward function, where the agent only receives images from the camera (from above and from the gripper) and information about the state of the joints (position and velocity). The results demonstrate that numerous tasks pose significant challenges even for PPO-LSTM agents with memory, establishing these environments as effective benchmarks for evaluating advanced memory-enhanced architectures.

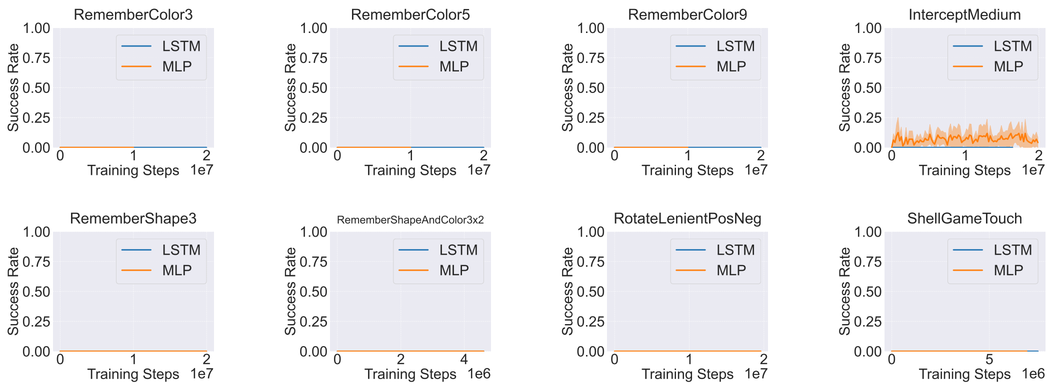


Figure 10: Performance evaluation of PPO-MLP and PPO-LSTM on the MIKASA-Robo benchmark using the “RGB+joints” with sparse reward function training mode, where the agent only receives images from the camera (from above and from the gripper) and information about the state of the joints (position and velocity). This training mode with sparse reward function causes even more difficulty for the agent to learn, making this mode even more challenging for memory-enhanced agents.

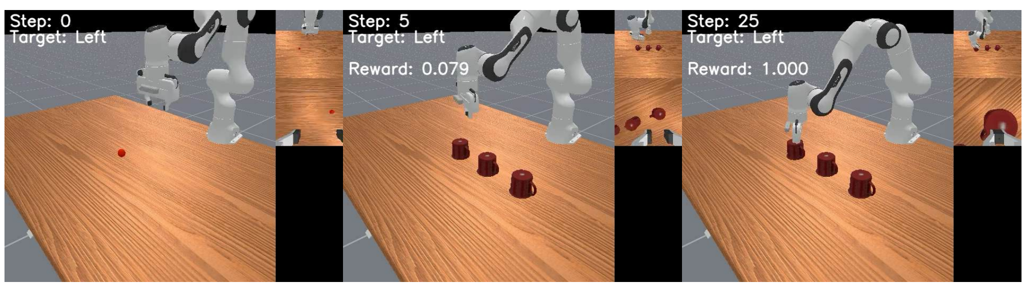


Figure 11: `ShellGameTouch-v0`: The robot observes a ball in front of it. next, this ball is covered by a mug and then the robot has to touch the mug with the ball underneath.

### H.1 ShellGame-v0

The `ShellGame-v0` task (Figure 11) is inspired by a simplified version of the classic shell game, which tests a person’s ability to remember object locations when they become occluded. This task evaluates an agent’s capacity for object permanence and spatial memory, crucial skills for real-world robotic manipulation where objects frequently become temporarily hidden from view.

**Environment Description** The environment consists of three identical mugs placed on a table and a red ball. The task proceeds in three phases:

1. **Observation Phase** (steps 0-4): The ball is placed at one of three positions, and the agent can observe its location.
2. **Occlusion Phase** (step 5): The ball and positions are covered by three identical mugs.
3. **Action Phase** (steps 6+): The agent must interact with the mug covering the ball’s location. The type of target interaction depends on the selected mode: `Touch`, `Push` and `Pick`.

**Task Modes** The task includes three variants of increasing difficulty:

- `Touch`: The agent only needs to touch the correct mug
- `Push`: The agent must push the correct mug to a designated area
- `Pick`: The agent must pick and lift the correct mug above a specified height

Table 6: **Results for Offline RL baselines**. The table shows comparison of transformer-based baselines (RATE, DT), behavior cloning (BC), classic Offline RL baselines (CQL), and Diffusion Policy (DP) on all 32 tasks from the MIKASA-Robo benchmark. Results are presented as mean ± sem across the three runs, where each run is averaged over 100 episodes and sem is the standard error of the mean. Training was performed using only RGB observations (two cameras: top view and gripper view) and using sparse rewards (success once condition). The results show that even models with memory (RATE, DT) are not able to solve most of the benchmark problems, which makes it challenging and promising for further validation of the algorithm.

| # | Environment | RATE | DT | BC | CQL | DP |
|---|---|---|---|---|---|---|
| 1 | ShellGameTouch-v0 | 0.92±0.01 | 0.53±0.07 | 0.28±0.01 | 0.16±0.04 | 0.18±0.02 |
| 2 | ShellGamePush-v0 | 0.78±0.06 | 0.62±0.14 | 0.27±0.01 | 0.25±0.01 | 0.22±0.03 |
| 3 | ShellGamePick-v0 | 0.02±0.01 | 0.00±0.00 | 0.01±0.01 | 0.00±0.00 | 0.01±0.00 |
| 4 | InterceptSlow-v0 | 0.23±0.02 | 0.40±0.02 | 0.37±0.06 | 0.25±0.01 | 0.33±0.05 |
| 5 | InterceptMedium-v0 | 0.32±0.02 | 0.56±0.01 | 0.31±0.14 | 0.03±0.01 | 0.68±0.02 |
| 6 | InterceptFast-v0 | 0.30±0.04 | 0.36±0.04 | 0.03±0.02 | 0.02±0.02 | 0.21±0.05 |
| 7 | InterceptGrabSlow-v0 | 0.09±0.03 | 0.00±0.00 | 0.28±0.18 | 0.03±0.00 | 0.03±0.01 |
| 8 | InterceptGrabMedium-v0 | 0.09±0.03 | 0.00±0.00 | 0.11±0.02 | 0.08±0.04 | 0.03±0.01 |
| 9 | InterceptGrabFast-v0 | 0.14±0.03 | 0.11±0.03 | 0.09±0.02 | 0.08±0.03 | 0.18±0.02 |
| 10 | RotateLenientPos-v0 | 0.11±0.04 | 0.01±0.01 | 0.15±0.03 | 0.16±0.02 | 0.11±0.02 |
| 11 | RotateLenientPosNeg-v0 | 0.29±0.03 | 0.05±0.02 | 0.22±0.01 | 0.12±0.02 | 0.14±0.05 |
| 12 | RotateStrictPos-v0 | 0.03±0.02 | 0.05±0.04 | 0.01±0.00 | 0.03±0.01 | 0.06±0.02 |
| 13 | RotateStrictPosNeg-v0 | 0.08±0.01 | 0.05±0.03 | 0.04±0.02 | 0.04±0.02 | 0.15±0.01 |
| 14 | TakeItBack-v0 | 0.42±0.24 | 0.08±0.04 | 0.33±0.10 | 0.04±0.01 | 0.05±0.02 |
| 15 | RememberColor3-v0 | 0.65±0.04 | 0.01±0.01 | 0.27±0.03 | 0.29±0.01 | 0.32±0.01 |
| 16 | RememberColor5-v0 | 0.13±0.03 | 0.07±0.05 | 0.12±0.01 | 0.15±0.02 | 0.10±0.02 |
| 17 | RememberColor9-v0 | 0.09±0.02 | 0.01±0.01 | 0.12±0.02 | 0.15±0.01 | 0.17±0.01 |
| 18 | RememberShape3-v0 | 0.21±0.04 | 0.05±0.04 | 0.31±0.04 | 0.20±0.10 | 0.32±0.05 |
| 19 | RememberShape5-v0 | 0.17±0.04 | 0.04±0.04 | 0.18±0.01 | 0.15±0.00 | 0.21±0.04 |
| 20 | RememberShape9-v0 | 0.05±0.00 | 0.05±0.02 | 0.10±0.02 | 0.14±0.01 | 0.11±0.02 |
| 21 | RememberShapeAndColor3x2-v0 | 0.14±0.02 | 0.04±0.02 | 0.13±0.02 | 0.11±0.05 | 0.14±0.02 |
| 22 | RememberShapeAndColor3x3-v0 | 0.08±0.03 | 0.06±0.06 | 0.09±0.02 | 0.09±0.02 | 0.16±0.01 |
| 23 | RememberShapeAndColor5x3-v0 | 0.07±0.02 | 0.01±0.01 | 0.09±0.01 | 0.09±0.02 | 0.11±0.03 |
| 24 | BunchOfColors3-v0 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 |
| 25 | BunchOfColors5-v0 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 |
| 26 | BunchOfColors7-v0 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 |
| 27 | SeqOfColors3-v0 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 |
| 28 | SeqOfColors5-v0 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 |
| 29 | SeqOfColors7-v0 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 |
| 30 | ChainOfColors3-v0 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 |
| 31 | ChainOfColors5-v0 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 |
| 32 | ChainOfColors7-v0 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 | 0.00±0.00 |

**Success Criteria** Success is determined by:

- `Touch`: Contact between the gripper and the correct mug
- `Push`: Moving forward the correct mug to the target zone
- `Pick`: Elevating the correct mug above 0.1m

**Reward Structure** The environment provides both sparse and dense reward variants:

- **Sparse**: Binary reward (1.0 for success, 0.0 otherwise)
- **Dense**: Continuous reward based on:
  - Distance between gripper and target mug
  - Robot's motion smoothness (static reward based on joint velocities)
  - Task completion status (additional reward when the task is solved)

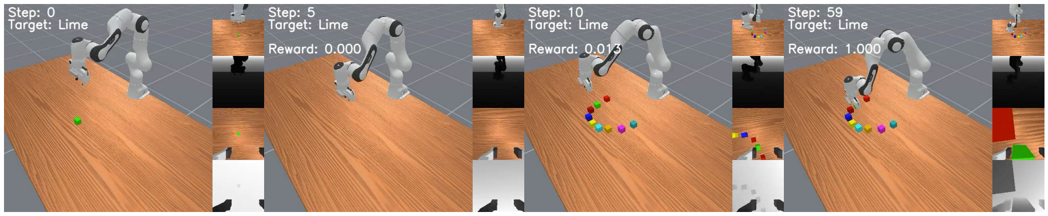


Figure 12: `RememberColor9-v0`: The robot observes a colored cube in front of it, then this cube disappears and an empty table is shown. Then 9 cubes appear on the table, and the agent must touch a cube of the same color as the one it observed at the beginning of the episode.

## H.2 RememberColor-v0

The `RememberColor-v0` task (Figure 12) tests an agent's ability to remember and identify objects based on their visual properties. This capability is essential for real-world robotics applications where agents must recall and match object characteristics across time intervals.

**Environment Description** The environment presents a sequence of colored cubes on a table. The task proceeds in three phases:

1. **Observation Phase** (steps 0-4): A cube of a specific color is displayed, and the agent must memorize its color.
2. **Delay Phase** (steps 5-9): The cube disappears, leaving an empty table.
3. **Selection Phase** (steps 10+): Multiple cubes of different colors appear (3, 5, or 9 depending on difficulty), and the agent must identify and interact with the cube matching the original color.

**Task Modes** The task includes three complexity levels:

- `3` (easy): Choose from 3 different colors (red, lime, blue)
- `5` (Medium): Choose from 5 different colors (red, lime, blue, yellow, magenta)
- `9` (Hard): Choose from 9 different colors (red, lime, blue, yellow, magenta, cyan, maroon, olive, teal)

**Success Criteria** Success is determined by:

- Correctly identifying and touching the cube that matches the color shown in the observation phase
- Maintaining contact with the correct cube for at least 0.1 seconds

**Reward Structure** The environment provides both sparse and dense reward variants:

- **Sparse**: Binary reward (1.0 for success, 0.0 otherwise)
- **Dense**: Continuous reward based on:
  - Distance between gripper and target cube
  - Robot's motion smoothness (static reward based on joint velocities)
  - Additional reward for robot being static while touching the correct cube
  - Task completion status (additional reward when the task is solved)

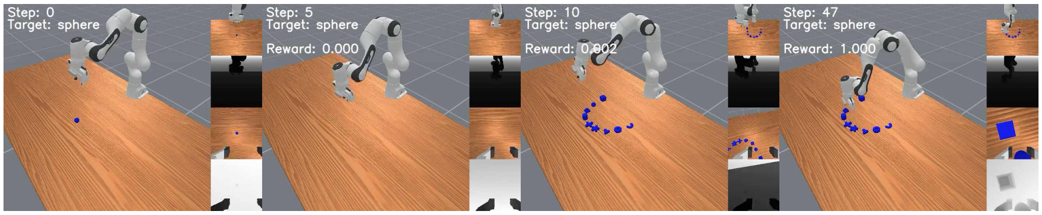


Figure 13: `RememberShape9-v0`: The robot observes an object with specific shape in front of it, then the object disappears and an empty table appears. Then 9 objects of different shapes appear on the table, and the agent must touch an object of the same shape as the one it observed at the beginning of the episode.

### H.3 RememberShape-v0

The `RememberShape-v0` task (Figure 13) evaluates an agent's ability to remember and identify objects based on their geometric properties. This capability is crucial for robotic applications where shape recognition and recall are essential for successful manipulation.

**Environment Description** The environment presents a sequence of geometric shapes on a table. The task proceeds in three phases:

1. **Observation Phase** (steps 0-4): A shape (cube, sphere, cylinder, etc.) is displayed, and the agent must memorize its geometry.
2. **Delay Phase** (steps 5-9): The shape disappears, leaving an empty table.
3. **Selection Phase** (steps 10+): Multiple shapes appear (3, 5, or 9 depending on difficulty), and the agent must identify and interact with the shape matching the original geometry.

**Task Modes** The task includes three complexity levels:

- `3` (Easy): Choose from 3 different shapes (cube, sphere, cylinder)
- `5` (Medium): Choose from 5 different shapes (cube, sphere, cylinder cross, torus)
- `9` (Hard): Choose from 9 different shapes (cube, sphere, cylinder cross, torus, star, pyramid, t-shape, crescent)

**Success Criteria** Success is determined by:

- Correctly identifying and touching the object with the same shape shown in the observation phase
- Maintaining contact with the correct shape for at least 0.1 seconds

**Reward Structure** The environment provides both sparse and dense reward variants:

- **Sparse**: Binary reward (1.0 for success, 0.0 otherwise)
- **Dense**: Continuous reward based on:
  - Distance between gripper and target object
  - Robot's motion smoothness (static reward based on joint velocities)
  - Additional reward for maintaining static position when touching correct object
  - Task completion status (additional reward when the task is solved)

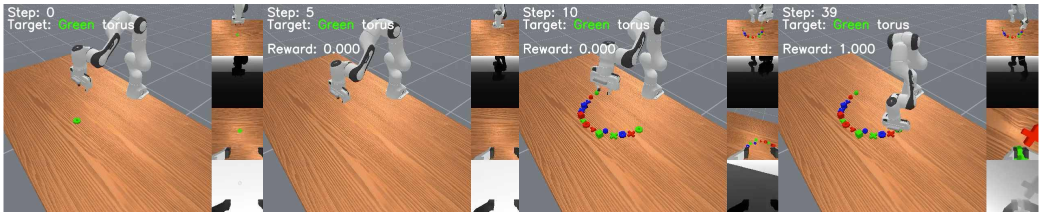


Figure 14: `RememberShapeAndColor5x3-v0`: An object of a certain shape and color appears in front of the agent. Then the object disappears and the agent sees an empty table. Then objects of 5 different shapes and 3 different colors appear on the table and the agent has to touch what it observed at the beginning of the episode.

### H.4 RememberShapeAndColor-v0

The `RememberShapeAndColor-v0` task (Figure 14) evaluates an agent's ability to remember and identify objects based on multiple visual properties simultaneously. This task combines shape and color recognition, testing the agent's capacity to maintain and match multiple object features across time intervals.

**Environment Description** The environment presents a sequence of colored geometric shapes on a table. The task proceeds in three phases:

1. **Observation Phase** (steps 0-4): An object with specific shape and color is displayed, and the agent must memorize both properties.
2. **Delay Phase** (steps 5-9): The object disappears, leaving an empty table.
3. **Selection Phase** (steps 10+): Multiple objects with different combinations of shapes and colors appear, and the agent must identify and interact with the object matching both the original shape and color.

**Task Modes** The task includes three complexity levels based on the number of shape-color combinations:

- `3x2` (Easy): Choose from 6 objects (3 shapes × 2 colors); shapes: cube, sphere, t-shape; colors: red, green
- `3x3` (Medium): Choose from 9 objects (3 shapes × 3 colors); shapes: cube, sphere, t-shape; colors: red, green, blue
- `5x3` (Hard): Choose from 15 objects (5 shapes × 3 colors); shapes: cube, sphere, t-shape, cross, torus; colors: red, green, blue

**Success Criteria** Success is determined by:

- Correctly identifying and touching the object that matches both the shape and color shown in the observation phase
- Maintaining contact with the correct object for at least 0.1 seconds

**Reward Structure** The environment provides both sparse and dense reward variants:

- **Sparse**: Binary reward (1.0 for success, 0.0 otherwise)
- **Dense**: Continuous reward based on:
  - Distance between gripper and target object
  - Robot's motion smoothness (static reward based on joint velocities)
  - Additional reward for maintaining static position while touching correct object
  - Task completion status (additional reward when the task is solved)

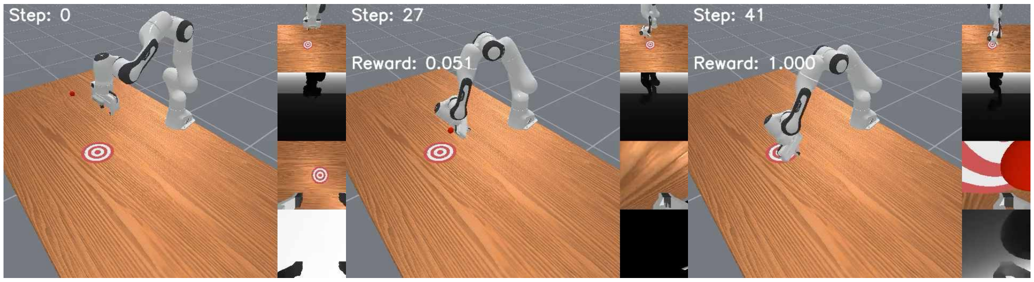

Figure 15: `InterceptMedium-v0`: A ball rolls on the table in front of the agent with a random initial velocity, and the agent's task is to intercept this ball and direct it at the target zone.

### H.5 Intercept-v0

The `Intercept-v0` task (Figure 16) evaluates an agent's ability to predict and intercept a moving object based on its initial trajectory. This task tests the agent's capacity for motion prediction and spatial-temporal reasoning, which are essential skills for dynamic manipulation tasks in robotics.

**Environment Description** The environment consists of a red ball moving across a table and a target zone. The task requires the agent to:

1. Observe the ball's initial position and velocity
2. Predict the ball's trajectory
3. Guide the ball to reach a designated target zone

**Task Modes** The task includes three variants with increasing ball velocities:

- `Slow`: Ball velocity range of 0.25-0.5 m/s
- `Medium`: Ball velocity range of 0.5-0.75 m/s
- `Fast`: Ball velocity range of 0.75-1.0 m/s

**Success Criteria** Success is determined by:

- Guiding the ball to enter the target zone
- The ball must come to rest within the target area (radius 0.1m)

**Reward Structure** The environment provides both sparse and dense reward variants:

- **Sparse**: Binary reward (1.0 for success, 0.0 otherwise)
- **Dense**: Continuous reward based on:
  - Distance between gripper and ball
  - Distance between ball and target zone
  - Robot's motion smoothness (static reward based on joint velocities)
  - Task completion status (additional reward when the task is solved)

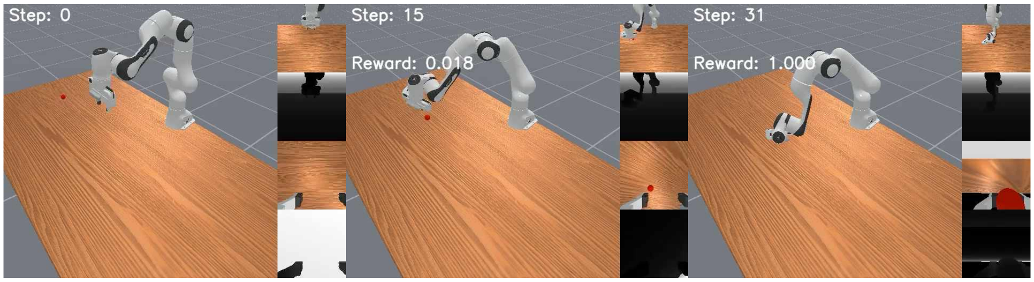


Figure 16: `InterceptGrabMedium-v0`: A ball rolls on the table in front of the agent with a random initial velocity, and the agent's task is to intercept this ball with a gripper and lift it up.

### H.6 InterceptGrab-v0

The `InterceptGrab-v0` task (Figure 16) extends the `Intercept-v0` task by requiring the agent to not only predict the trajectory of a moving object but also grasp it while in motion. This task evaluates the agent's ability to combine motion prediction with precise manipulation timing, simulating real-world scenarios where robots must catch or intercept moving objects.

**Environment Description** The environment consists of a red ball moving across a table. The task requires the agent to:

1. Observe the ball's initial position and velocity
2. Predict the ball's trajectory
3. Position the gripper to intercept the ball's path
4. Time the grasping action correctly to catch the ball
5. Maintain a stable grasp while bringing the ball to rest

**Task Modes** The task includes three variants with increasing ball velocities:

- `Slow`: Ball velocity range of 0.25-0.5 m/s
- `Medium`: Ball velocity range of 0.5-0.75 m/s
- `Fast`: Ball velocity range of 0.75-1.0 m/s

**Success Criteria** Success is determined by:

- Successfully grasping the moving ball
- Maintaining a stable grasp until the ball comes to rest
- The robot must be static with the ball firmly grasped

**Reward Structure** The environment provides both sparse and dense reward variants:

- **Sparse**: Binary reward (1.0 for success, 0.0 otherwise)
- **Dense**: Continuous reward based on:
  - Distance between gripper and ball
  - Grasping reward
  - Robot's motion smoothness (static reward based on joint velocities)
  - Task completion status (additional reward when the task is solved)

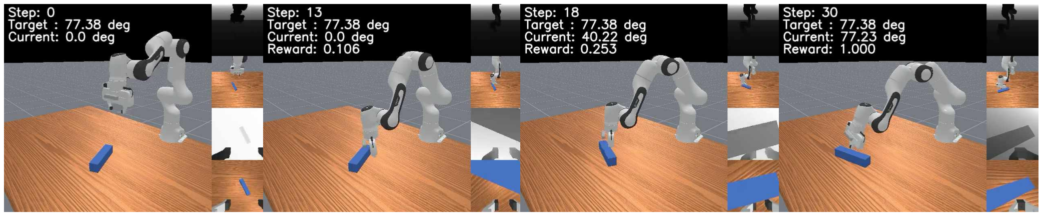


Figure 17: `RotateLenientPos-v0`: A randomly oriented peg is placed in front of the agent. The agent's task is to rotate this peg by a certain angle (the center of the peg can be shifted).

### H.7 RotateLenient-v0

The `RotateLenient-v0` task (Figure 17) evaluates an agent's ability to remember and execute specific rotational movements. This task tests the agent's capacity to maintain and reproduce angular information, which is crucial for manipulation tasks requiring precise orientation control. This task tests the agent's ability to hold information in memory about how far peg has already rotated at the current step relative to its initial position.

**Environment Description** The environment consists of a blue-colored peg on a table that must be rotated by a specified angle. The task proceeds in one phase, but the static prompt information about the target angle is available to the agent at each timestep:

1. **Action Phase**: The agent must rotate the peg to match the target angle

**Task Modes** The task includes two variants with different rotation requirements:

- `Pos`: Rotate by a positive angle between 0 and $\pi/2$
- `PosNeg`: Rotate by either positive or negative angle between $-\pi/4$ and $\pi/4$

**Success Criteria** Success is determined by:

- Rotating the peg to within the angle threshold (±0.1 radians) of the target angle
- Maintaining the final orientation in a stable position
- The robot must be static with the peg at the correct orientation

**Reward Structure** The environment provides both sparse and dense reward variants:

- **Sparse**: Binary reward (1.0 for success, 0.0 otherwise)
- **Dense**: Continuous reward based on:
  - Distance between gripper and peg
  - Angular distance to target rotation
  - Stability of the peg's orientation
  - Robot's motion smoothness (static reward based on joint velocities)
  - Task completion status (additional reward when the task is solved)

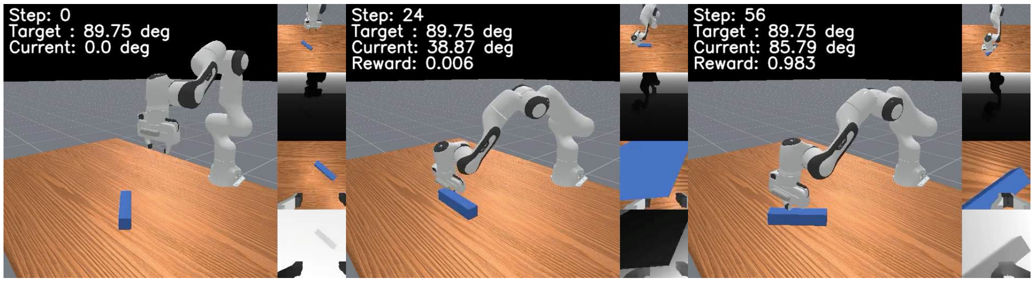


Figure 18: `RotateStrictPos-v0`: A randomly oriented peg is placed in front of the agent. The agent's task is to rotate this peg by a certain angle (it is not allowed to move the center of the peg)

### H.8 RotateStrict-v0

The `RotateStrict-v0` task (Figure 18) extends the `RotateLenient-v0` task with more stringent requirements for precise rotational control.

**Environment Description** The environment consists of a blue-colored peg on a table that must be rotated by a specified angle while maintaining its position. The task proceeds in one phase, but the static prompt information about the target angle is available to the agent at each timestep:

1. **Action Phase**: The agent must rotate the peg to match the target angle while keeping it centered

**Task Modes** The task includes two variants with different rotation requirements:

- `Pos`: Rotate by a positive angle between 0 and $\pi/2$
- `PosNeg`: Rotate by either positive or negative angle between $-\pi/4$ and $\pi/4$

**Success Criteria** Success is determined by:

- Rotating the peg to within the angle threshold (±0.1 radians) of the target angle
- Maintaining the peg's position within 5cm of its initial XY coordinates
- The robot must be static with the peg at the correct orientation
- No significant deviation in other rotation axes

**Reward Structure** The environment provides both sparse and dense reward variants:

- **Sparse**: Binary reward (1.0 for success, 0.0 otherwise)
- **Dense**: Continuous reward based on:
  - Distance between gripper and peg
  - Angular distance to target rotation
  - Position deviation from initial location
  - Stability of the peg's orientation
  - Robot's motion smoothness (static reward based on joint velocities)
  - Task completion status (additional reward when the task is solved)

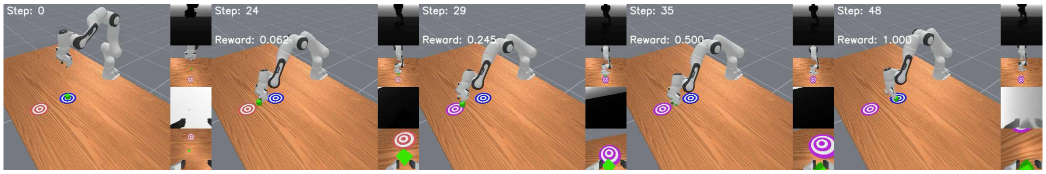


Figure 19: `TakeItBack-v0`: The agent observes a green cube in front of him. The agent's task is to move the green cube to the red target, and as soon as it lights up violet, return the cube to its original position (the agent does not observes the original position of the cube).

### H.9 TakeItBack-v0

The `TakeItBack-v0` task (Figure 19) assesses the agent's ability to perform sequential tasks and memorize the starting position. This task tests the agent's capacity for sequential memory and spatial reasoning, requiring it to maintain information about past locations and achievements while executing a multi-step plan.

**Environment Description** The environment consists of a green cube and two target regions (initial and goal) on a table. The task proceeds in two phases:

1. **First Phase**: The agent must move the cube from its initial position to a goal region
2. **Second Phase**: After reaching the goal, goal region change it's color from red to magenta, and the agent must return the cube to its original position (marked by the initial region and invisible for the agent)

**Success Criteria** Success is determined by:

- First reaching the goal region with the cube
- Then returning the cube to the initial region
- Both goals must be achieved in sequence

**Reward Structure** The environment provides both sparse and dense reward variants:

- **Sparse**: Binary reward (1.0 for success, 0.0 otherwise)
- **Dense**: Continuous reward based on:
  - Distance between gripper and cube
  - Distance to current target region
  - Progress through the task sequence
  - Stability of cube manipulation
  - Robot's motion smoothness (static reward based on joint velocities)
  - Task completion status (additional reward when the task is solved)

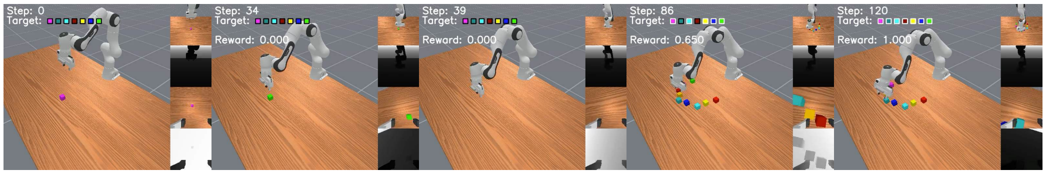


Figure 20: `SeqOfColors7-v0`: In front of the agent, 7 cubes of different colors appear sequentially. After the last cube is shown, the agent observes an empty table. Then 9 cubes of different colors appear on the table and the agent has to touch the cubes that were shown at the beginning of the episode in any order.

### H.10 SeqOfColors-v0

The `SeqOfColors-v0` task (Figure 20) evaluates an agent's ability to remember and reproduce an unordered sequence of colors. This task tests memory capacity capabilities essential for robotic tasks that require following specific patterns or sequences.

**Environment Description** The environment presents a sequence of colored cubes that must be reproduced in any order. The task proceeds in two phases:

1. **Observation Phase** (steps 0-($5N - 1$)): A sequence of N colored cubes is shown one at a time, with each cube visible for 5 steps.
2. **Delay Phase** (steps ($5N$)-($5N + 4$)): All cubes disappear
3. **Selection Phase** (steps ($5N + 5$)+): A larger set of cubes appears, and the agent must identify and touch all previously shown cubes in any order

**Task Modes** The task includes three complexity levels:

- `3` (Easy): Remember 3 colors demonstrated sequentially
- `5` (Medium): Remember 5 colors demonstrated sequentially
- `7` (Hard): Remember 7 colors demonstrated sequentially

**Success Criteria** Success is determined by:

- Correctly identifying and touching all cubes from the observation phase
- Order of selection doesn't matter
- Each cube must be touched for at least 0.1 seconds
- The demonstrated set must be touched without any mistakes

**Reward Structure** The environment provides both sparse and dense reward variants:

- **Sparse**: Binary reward (1.0 for success, 0.0 otherwise)
- **Dense**: Continuous reward based on:
  - Distance between gripper and next target cube
  - Number of correctly identified cubes
  - Static reward for stable contact
  - Robot's motion smoothness (static reward based on joint velocities)
  - Task completion status (additional reward when the task is solved)

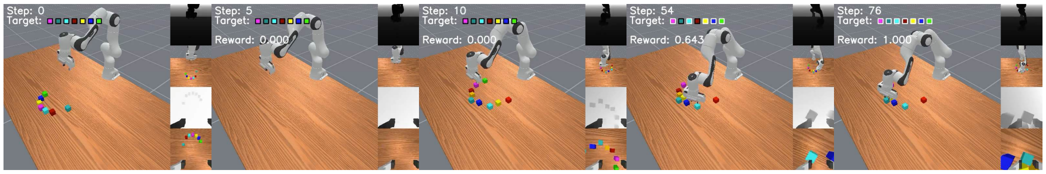


Figure 21: `BunchOfColors7-v0`: 7 cubes of different colors appear simultaneously in front of the agent. After the agent observes an empty table. Then, 9 cubes of different colors appear on the table and the agent has to touch the cubes that were shown at the beginning of the episode in any order.

### H.11 BunchOfColors-v0

The `BunchOfColors-v0` task (Figure 21) tests an agent's memory capacity by requiring it to remember multiple objects simultaneously. This capability is crucial for tasks requiring parallel processing of multiple items.

**Environment Description** The environment presents multiple colored cubes simultaneously. The task proceeds in three phases:

1. **Observation Phase** (steps 0-4): Multiple colored cubes are displayed simultaneously
2. **Delay Phase** (steps 5-9): All cubes disappear
3. **Selection Phase** (steps 10+): A larger set of cubes appears, and the agent must identify and touch all previously shown cubes in any order

**Task Modes** The task includes three complexity levels:

- `3` (Easy): Remember 3 colors demonstrated simultaneously
- `5` (Medium): Remember 5 colors demonstrated simultaneously
- `7` (Hard): Remember 7 colors demonstrated simultaneously

**Success Criteria** Success is determined by:

- Correctly identifying and touching all cubes from the observation phase
- Order of selection doesn't matter
- Each cube must be touched for at least 0.1 seconds
- The demonstrated set must be touched without any mistakes

**Reward Structure** The environment provides both sparse and dense reward variants:

- **Sparse**: Binary reward (1.0 for success, 0.0 otherwise)
- **Dense**: Continuous reward based on:
  - Distance between gripper and next target cube
  - Static reward for stable contact
  - Number of correctly touched cubes
  - Robot's motion smoothness (static reward based on joint velocities)
  - Task completion status (additional reward when the task is solved)

Figure 22: `ChainOfColors7-v0`: In front of the agent, 7 cubes of different colors appear sequentially. After the last cube is shown, the agent sees an empty table. Then 9 cubes of different colors appear on the table and the agent must unmistakably touch the cubes that were shown at the beginning of the episode, in the same strict order.

### H.12 ChainOfColors-v0

The `ChainOfColors-v0` task (Figure 22) evaluates the agent's ability to store and retrieve ordered information. This task simulates scenarios where the agent must track changing relationships between objects over time.

**Environment Description** The environment presents am ordered sequence (chain) of colored cubes that must be followed. The task proceeds in multiple phases:

1. **Observation Phase** (steps 0-($5N - 1$)): A sequence of N colored cubes is shown one at a time, with each cube visible for 5 steps.
2. **Delay Phase** (steps ($5N$)-($5N + 4$)): All cubes disappear
3. **Selection Phase** (steps ($5N + 5$)+): A larger set of cubes appears, and the agent must identify and touch all previously shown cubes in the exact order as demonstrated

**Task Modes** The task includes three complexity levels:

- `3` (Easy): Remember 3 colors demonstrated sequentially
- `5` (Medium): Remember 5 colors demonstrated sequentially
- `7` (Hard): Remember 7 colors demonstrated sequentially

**Success Criteria** Success is determined by:

- Correctly identifying and touching all cubes from the observation phase in the exact order
- Each cube must be touched for at least 0.1 seconds
- The demonstrated set must be touched without any mistakes

**Reward Structure** The environment provides both sparse and dense reward variants:

- **Sparse**: Binary reward (1.0 for success, 0.0 otherwise)
- **Dense**: Continuous reward based on:
  - Distance between gripper and next target cube
  - Static reward for stable contact
  - Number of correctly touched cubes
  - Robot's motion smoothness (static reward based on joint velocities)
  - Task completion status (additional reward when the task is solved)

Table 7: Classification of environments from the MIKASA-Base benchmark according to the suggested memory-intensive tasks classification from the Subsection 4.2.

| Environment | Memory Task | Brief description of the task | Observation Space | Action Space |
|---|---|---|---|---|
| Memory Cards | Capacity | Memorize the positions of revealed cards and correctly match pairs while minimizing incorrect guesses. | vector | discrete |
| Numpad | Sequential | Memorize the sequence of movements and navigate the rolling ball on a 3×3 grid by following the correct order while avoiding mistakes. | image, vector | discrete, continuous |
| BSuite Memory Length | Object | Memorize the initial context signal and recall it after a given number of steps to take the correct action. | vector | discrete |
| Minigrid-Memory | Object | Memorize the object in the starting room and use this information to select the correct path at the junction. | image | discrete |
| Ballet | Sequential, Object | Memorize the sequence of movements performed by each uniquely colored and shaped dancer, then identify and approach the dancer who executed the given pattern. | image | discrete |
| Passive Visual Match | Object | Memorize the target color displayed on the wall during the initial phase. After a brief distractor phase, identify and select the target color among the distractors by stepping on the corresponding ground pad. | image | discrete |
| Passive-T-Maze | Object | Memorize the goal's location upon initial observation, navigate through the maze with limited sensory input, and select the correct path at the junction. | vector | discrete |
| ViZDoom-two-colors | Object | Memorize the color of the briefly appearing pillar (green or red) and collect items of the same color to survive in the acid-filled room. | image | discrete |
| Memory Maze | Spatial | Memorize the locations of objects and the maze structure using visual clues, then navigate efficiently to find objects of a specific color and score points. | image | discrete |
| MemoryGym Mortar Mayhem | Capacity, Sequential | Memorize a sequence of movement commands and execute them in the correct order. | image | discrete |
| MemoryGym Mystery Path | Capacity, Spatial | Memorize the invisible path and navigate it without stepping off. | image | discrete |
| POPGym Repeat First | Object | Memorize the initial value presented at the first step and recall it correctly after receiving a sequence of random values. | vector | discrete |
| POPGym Repeat Previous | Sequential, Object | Memorize the value observed at each step and recall the value from $k$ steps earlier when required. | vector | discrete |
| POPGym Autoencode | Sequential | Memorize the sequence of cards presented at the beginning and reproduce them in the same order when required. | vector | discrete |
| POPGym Count Recall | Object, Capacity | Memorize unique values encountered and count how many times a specific value has appeared. | vector | discrete |
| POPGym vectorless Cartpole | Sequential | Memorize velocity data over time and integrate it to infer the position of the pole for balance control. | vector | continuous |
| POPGym vectorless Pendulum | Sequential | Memorize angular velocity over time and integrate it to infer the pendulum's position for successful swing-up control. | vector | continuous |
| POPGym Multiarmed Bandit | Object, Capacity | Memorize the reward probabilities of different slot machines by exploring them and identify the one with the highest expected reward. | vector | discrete |
| POPGym Concentration | Capacity | Memorize the positions of revealed cards and match them with previously seen cards to find all matching pairs. | vector | discrete |
| POPGym Battleship | Spatial | Memorize the coordinates of previous shots and their HIT or MISS feedback to build an internal representation of the board, avoid repeat shots, and strategically target ships for maximum rewards. | vector | discrete |
| POPGym Mine Sweeper | Spatial | Memorize revealed grid information and use numerical clues to infer safe tiles while avoiding mines. | vector | discrete |
| POPGym Labyrinth Explore | Spatial | Memorize previously visited cells and navigate the maze efficiently to discover new, unexplored areas and maximize rewards. | vector | discrete |
| POPGym Labyrinth Escape | Spatial | Memorize the maze layout while exploring and navigate efficiently to find the exit and receive a reward. | vector | discrete |
| POPGym Higher Lower | Object, Sequential | Memorize previously revealed card ranks and predict whether the next card will be higher or lower, updating the reference card after each prediction to maximize rewards. | vector | discrete |

# I MIKASA-Base Benchmark Tasks Description

This section provides a detailed description of all environments included in the MIKASA-Base benchmark Section 5. Understanding the characteristics and challenges of these environments is crucial for evaluating RL algorithms. Each environment presents unique tasks, offering diverse scenarios to test the memory abilities of RL agents.

## I.1 Memory Cards

The Memory Cards environment [28] is a memory game environment with 5 randomly shuffled pairs of hidden cards. At each step, the agent sees one revealed card and must find its matching pair. A correct guess removes both cards; otherwise, the card is hidden again, and a new one is revealed. The game continues until all pairs are removed.

## I.2 Numpad

The Numpad environment [96] consists of an $N \times N$ grid of tiles. The agent controls a ball that rolls between tiles. At the beginning of an episode, a random sequence of $n$ neighboring tiles (excluding diagonals) is selected, and the agent must follow this sequence in the correct order. The environment is structured so that pressing the correct tile lights it up, while pressing an incorrect tile resets progress. A reward of +1 is given for the first press of each correct tile after a reset. The episode ends after a fixed number of steps. To succeed, the agent must memorize the sequence and navigate it correctly without mistakes. The ability to “jump” over tiles is not available.

### I.3 BSuite MemoryLength

The MemoryLength environment [25] represents a sequence of observations, where at each step, the observation takes a value of either +1 or -1. The environment is structured so that a reward is given only at the final step if the agent correctly predicts the $i$-th value from the initial observation vector $obs$. The index of this $i$-th value is specified at the last step observation vector in $obs[1]$. To succeed, the agent must remember the sequence of observations and use this information to make an accurate prediction at the final step.

### I.4 Minigrid-Memory

Minigrid-Memory [21] is a two-dimensional grid-based environment that features a T-shaped maze with a small room at the beginning of the corridor, containing an object. The agent starts at a random position within the corridor. Its task is to reach the room, observe and memorize the object, then proceed to the junction at the maze’s end and turn towards the direction where an identical object is located. The reward function is defined as $R_t = 1 - 0.9 \times \frac{t}{T}$ for a successful attempt; otherwise, the agent receives zero reward. The episode terminates when the agent makes a choice at the junction or exceeds a time limit of steps.

### I.5 Ballet

In the Ballet environment [3] tasks take place in an $11 \times 11$ tiled room, consisting of a $9 \times 9$ central area surrounded by a one-tile-wide wall. Each tile is upsampled to 9 pixels, resulting in a $99 \times 99$ pixel input image. The agent is initially placed at the center of the room, while dancers are randomly positioned in one of 8 possible locations around it. Each dancer has a distinct shape and color, selected from 15 possible shapes and 19 colors, ensuring uniqueness. These visual features serve only for identification and do not influence behavior. The agent itself is always represented as a white square. The agent receives egocentric visual observations, meaning its view is centered on its own position, which has been shown to enhance generalization.

### I.6 Passive T-Maze

The Passive T-Maze environment [44] consists of a corridor leading to a junction that connects two possible goal states. The agent starts at a designated position and can move in four directions: left, right, up, or down. At the beginning of each episode, one of the two goal states is randomly assigned as the correct destination. The agent observes this goal location before starting its movement. The agent stays in place if it attempts to move into a wall. To succeed, the agent must navigate to the correct goal based on its initial observation. The optimal strategy involves moving through the corridor towards the junction and then selecting the correct path.

### I.7 ViZDoom-Two-Colors

The ViZDoom-Two-Colors [97] is an environment where an agent is placed in a room with constantly depleting health. The room contains red and green objects, one of which restores health (+1 reward), while the other reduces it (-1 reward). The beneficial color is randomly assigned at the beginning of each episode and indicated by a column. The environment is structured so that the agent must memorize the column’s color to collect the correct items. Initially, the column remains visible, but in a harder variant, it disappears after 45 steps, increasing the memory requirement. To succeed, the agent must maximize survival by collecting beneficial objects while avoiding harmful ones.

### I.8 Memory Maze

The Memory Maze environment [26] is a procedurally generated 3D maze. Each episode, the agent spawns in a new maze with multiple colored objects placed in fixed locations. The agent receives a first-person view and a prompt indicating the color of the target object. It must navigate the maze, memorize object positions, and return to them efficiently. The agent receives a reward of 1 for reaching the correct object and no reward for incorrect objects.

### I.9 MemoryGym Mortar Mayhem

Mortar Mayhem [11] is a grid-based environment where the agent must memorize and execute a sequence of commands in the correct order. The environment consists of a finite grid, where the agent initially observes a series of movement instructions and then attempts to reproduce them accurately. Commands include movements to adjacent tiles or remaining in place. The challenge lies in the agent's ability to recall and execute a growing sequence of instructions, with failure resulting in episode termination. A reward of +0.1 is given for each correctly executed command

### I.10 MemoryGym Mystery Path

Mystery Path [11] presents an invisible pathway that the agent must traverse without deviating. If the agent steps off the path, it is returned to the starting position, forcing it to remember the correct trajectory. The path is procedurally generated, meaning each episode introduces a new configuration. Success in this environment requires the agent to accurately recall previous steps and missteps to avoid repeating errors. The agent is rewarded +0.1 for progressing onto a previously unvisited tile

### I.11 POPGym environments

The following environments are included from the POPGym benchmark [9], which is designed to evaluate RL agents in partially observable settings. POPGym provides a diverse collection of lightweight vectorized environments with varying difficulty levels.

#### I.11.1 POPGym Autoencode

The environment consists of a deck of cards that is shuffled and sequentially shown to the agent during the watch phase. While observing the cards, a watch indicator is active, but it disappears when the last card is revealed. Afterward, the agent must reproduce the sequence of cards in the correct order. The environment is structured to evaluate the agent's ability to encode a sequence of observations into an internal representation and later reconstruct the sequence one observation at a time.

#### I.11.2 POPGym Concentration

The environment represents a classic memory game where a shuffled deck of cards is placed face-down. The agent sequentially flips two cards and earns a reward if the revealed cards form a matching pair. The environment is designed in such a way that the agent must remember previously revealed cards to maximize its success rate.

#### I.11.3 POPGym Repeat First

The environment presents the agent with an initial value from a set of four possible values, along with an indicator signaling that this is the first value. In subsequent steps, the agent continues to receive random values from the same set but without the initial indicator. The structure requires the agent to retain the first received value in memory and recall it accurately to receive a reward.

#### I.11.4 POPGym Repeat Previous

The environment consists of a sequence of observations, where each observation can take one of four possible values at each timestep. The agent is tasked with recalling and outputting the value that appeared a specified number of steps in the past.

#### I.11.5 POPGym Stateless Cartpole

This is a modified version of the traditional Cartpole environment [98] where angular and linear position information is removed from observations. Instead, the agent only receives velocity-based data and must infer positional states by integrating this information over time to successfully balance the pole.

#### I.11.6 POPGym Stateless Pendulum

In this variation of the swing-up pendulum environment [99], angular position data is omitted from the agent's observations. The agent must infer the pendulum's position by processing velocity information and use this estimate to determine appropriate control actions.

#### I.11.7 POPGym Noisy Stateless Cartpole

This environment builds upon Stateless Cartpole by introducing Gaussian noise into the observations. The agent must still infer positional states from velocity information while filtering out the added noise to maintain control of the pole.

#### I.11.8 POPGym Noisy Stateless Pendulum

This variation extends the Stateless Pendulum environment by incorporating Gaussian noise into the observations. The agent must manage this uncertainty while using velocity data to estimate the pendulum's position and swing it up effectively.

#### I.11.9 POPGym Multiarmed Bandit

The Multiarmed Bandit environment is an episodic formulation of the multiarmed bandit problem [100], where a set of bandits is randomly initialized at the start of each episode. Unlike conventional multiarmed bandit tasks, where reward probabilities remain fixed across episodes, this structure resets them every time. The agent must dynamically adjust its exploration and exploitation strategies to maximize long-term rewards.

#### I.11.10 POPGym Higher Lower

Inspired by the higher-lower card game, this environment presents the agent with a sequence of cards. At each step, the agent must predict whether the next card will have a higher or lower rank than the current one. Upon making a guess, the next card is revealed and becomes the new reference. The agent can enhance its performance by employing card counting strategies to estimate the probability of future values.

#### I.11.11 POPGym Count Recall

At each timestep, the agent is presented with two values: a next value and a query value. The agent must determine and output how many times the query value has appeared so far. To succeed, the agent must maintain an accurate count of past occurrences and retrieve the correct number upon request.

#### I.11.12 POPGym Battleship

A partially observable variation of the game Battleship, where the agent does not have access to the full board. Instead, it receives feedback on its previous shot, indicating whether it was a HIT or

MISS, along with the shot's location. The agent earns rewards for hitting ships, receives no reward for missing, and incurs a penalty for targeting the same location more than once. The environment challenges the agent to construct an internal representation of the board and update its strategy based on past observations.

#### I.11.13 POPGym Mine Sweeper

A partially observable version of the computer game Mine Sweeper, where the agent lacks direct visibility of the board. Observations include the coordinates of the most recently clicked tile and the number of adjacent mines. Clicking on a mined tile results in a negative reward and ends the game. To succeed, the agent must track previous selections and deduce mine locations based on the numerical clues, ensuring it avoids mines while uncovering safe tiles.

#### I.11.14 POPGym Labyrinth Explore

The environment consists of a procedurally generated 2D maze in which the agent earns rewards for reaching new, unexplored tiles. Observations are limited to adjacent tiles, requiring the agent to infer the larger maze layout through exploration. A small penalty per timestep incentivizes efficient navigation and discovery strategies.

#### I.11.15 POPGym Labyrinth Escape

This variation of Labyrinth Explore challenges the agent to find an exit rather than merely exploring the maze. The agent retains the same restricted observation space, seeing only nearby tiles. Rewards are only given upon successfully reaching the exit, making it a sparse reward environment where the agent must navigate strategically to achieve its goal.

# J Limitations

While our benchmark provides a comprehensive evaluation framework, some limitations remain. In particular, the performance of Octo and OpenVLA may not reflect their full potential, as we performed limited fine-tuning due to computational constraints. Future work could explore more extensive adaptation of large VLA models within MIKASA to better assess their memory capabilities. Additionally, while MIKASA covers a broad range of memory challenges, further extensions could incorporate tasks with longer temporal dependencies or meta-RL.