

AlphaNIST: Emergent Assembly Skills from Binary Task Rewards

Octi Zhang¹, Iretiayo Akinola¹, Rosario Scalise², Mateo Guaman Castro², Patrick Yin^{1,2}
 Tobias Widmer¹, Miles Macklin¹, Chris Amey¹, Dylan Turpin¹, Dirk Van Gelder¹
 Jan Carius¹, Ankur Handa¹, Gavriel State¹, Yunrong Guo¹, and Yashraj Narang¹

¹NVIDIA ²University of Washington



Fig. 1. **Emergent multi-component assembly with a single policy.** Trained with a binary task reward, AlphaNIST learns to select, grasp, reorient, and assemble diverse components, while autonomously sequencing multiple assemblies within a single episode. Shown are snapshots from a ten-component assembly sequence.

Abstract—The NIST Assembly Task Board #1 remains a challenging benchmark for learning-based methods in robotic manipulation. Over short horizons, it demands precise, contact-rich control of small, tight-fitting parts. Over long horizons, it requires recovery and coordination across extended action sequences. Demonstrations, task decomposition, and dense progress rewards have enabled research advances by providing guidance on how to assemble. But how much of the strategy must be specified? The progression from AlphaGo to AlphaGo Zero showed that RL at scale can discover superhuman strategies without prescribing how to win. AlphaNIST asks whether a sparse binary task reward and task diversity admitted by this task-agnostic reward can similarly let a robot discover how to assemble. In simulation, we train a *single* policy from scratch across all 20 assembly tasks on the NIST Assembly Task Board #1, reaching 98.2% overall training success without demonstrations, progress shaping, scripted primitives, or per-task tuning. The framework also trains policies to complete ten assembly pairs in a single episode without a prescribed order, with task success rewarded *only* when all ten pairs are simultaneously complete. The learned policies discover dexterous and planning-like behaviors: choosing grasps for later assembly, using surrounding geometry to reorient parts, and preparing one part, assembling another, then returning to finish the first without an explicit planner. For videos, see our website: <https://alpha-nist.github.io/>

I. INTRODUCTION

Robotic assembly is a longstanding challenge, and the NIST Assembly Task Boards are a persistent benchmark [1]; to the best of our knowledge, no learning-based approaches have solved all tasks on any of the boards. Assembly Task Board #1 (ATB1) consists of 20 *assembly pair* tasks (a part and its fixed mating geometry on the board) that are relevant for manufacturing, including peg insertion, gear meshing, electrical connector insertion, and nut threading. Solving this task board is exceptionally challenging for three

key reasons: (1) The sequence for solving the tasks might be unspecified, and not all sequences may be efficient or valid (e.g., solving task N may obstruct the robot from accessing the parts in task N + 1). (2) Each task is long-horizon, encompassing grasping, transport, and contact-rich assembly [2]. Parts may begin in a wide range of poses and require multiple reorientations before grasping. During transport, parts may need further reorientation in-hand or through extrinsic contact. During assembly, parts may slip, become misaligned, or jam during insertion or threading [2], requiring robust recovery and reexecution. (3) Solving the tasks requires precise control during complex contact interactions. The parts themselves are small (as small as 4 mm in diameter), have high geometric detail (e.g., helical threads), and have tight clearances between mating surfaces (as low as zero). Tasks such as nut threading involve learning helical motion during frictional contact between multiple mating surfaces, during which a nut can tilt, cross-thread, or jam instead of advancing along the bolt [2].

Learning-free approaches typically rely on planning and strong human priors: assembly sequence planning [3]; customized fixtures, adapters, and tooling [1], [4]; engineered motion and force primitives [5]; and tuned admittance, impedance, or force-motion controllers [4], [5].

Learning-based approaches, on the other hand, can relax some of these constraints, but have shown progress only within a limited scope. Most efforts have assumed a fixed assembly sequence, heavily constrained the initial poses of parts (e.g., beginning in an already-grasped state) [6], assumed access to accurate pose estimates, learned specialist rather than multi-task policies [2], addressed only a subset of tasks on the board [6], and/or achieved only low-to-

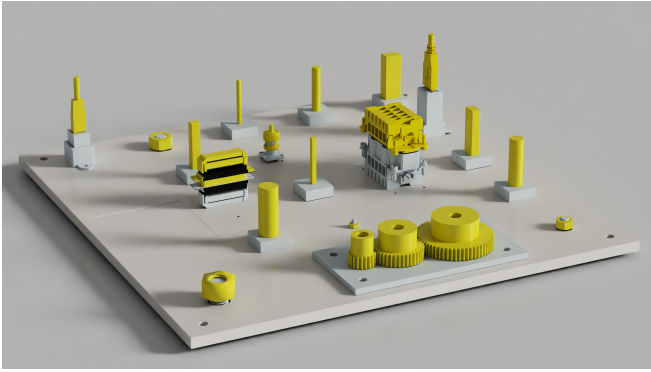


Fig. 2. **The complete NIST Assembly Task Board #1 in simulation.** Yellow indicates the 20 movable parts in their fully-assembled states.

moderate success rates by industrial standards. Success-Guided Sampling (SGS) [7], for example, solves tasks one at a time, but requires a separate policy per task and is considerably less compute-efficient.

We draw inspiration from the landmark AlphaGo series in reinforcement learning, which surpassed the strongest human Go players. The progression from AlphaGo to AlphaGo Zero showed that learning from a terminal task objective could encourage superhuman decision-making over long horizons without human demonstrations [8], [9]. In this paper, we ask whether a similar approach is possible for robotics: can sparse binary task rewards, combined with appropriate training strategies, enable a robot to solve sequences of long-horizon, contact-rich manipulation tasks, perhaps leveraging dexterous behaviors that no human could feasibly specify or demonstrate? To make learning from sparse task rewards more tractable, we take a seemingly counterintuitive step: train jointly across heterogeneous assembly pairs differing in geometries, tolerances, and assembly requirements (a nut and its bolt, a gear and its gearshaft, a peg and its hole) from a broad coverage of initial states, **converting MDP diversity into a resource rather than a burden.**

Guided by this insight, we develop AlphaNIST, a learning framework that trains a *single policy* under sparse binary task rewards across all 20 tasks on the NIST Assembly Task Board 1 (Fig. 2). The policy handles an exceptional range of initial part poses, learns from point-cloud inputs without first training specialist policies, and achieves a 98.2% overall success rate. Furthermore, the same framework can train policies to complete 10 assembly pairs in sequence, with a task reward issued only when all pairs are assembled with *no prescribed order*. In other words, the policy exhibits planning-like behavior.

Our contributions are the following:

- 1) **A highly-effective multi-task RL framework** consisting of a novel combination of the following components: A) generation of a cache of assembly-relevant reset states [10] over heterogeneous assembly pairs, B) success-guided sampling within the cache [7], C) a unified sparse binary task reward, D) multi-GPU training at scale. These components are supported by careful ablations. All tasks share one training configuration, without per-task tuning or demonstrations.

- 2) **The first multi-task RL policy capable of solving all 20 tasks on the NIST Assembly Task Board 1.** The policy takes point clouds as input; initial part poses are sampled from a wide range, including adverse conditions (e.g., upside-down); and each task can be solved with near-perfect success rates. Positive transfer is observed, where multi-task policies can solve the 20 tasks with higher success rate than individual specialists, including three tasks that the specialists struggle to solve.
- 3) **The first assembly policy capable of solving up to 10 diverse assembly pairs in sequence,** without an explicit hierarchy or prescribed task order. Thus, this work demonstrates that sequencing behavior can emerge within a single low-level assembly policy.

We observe a repertoire of unique emergent behaviors including dexterous skills such as pushing and sliding (to singulate parts), in-hand pivoting, multiple sequential regrasps, extrinsic contact (to reorient parts prior to insertion), and single-gripper nut threading. The single policy also exhibits long-horizon planning-like behavior such as bringing parts into reach, moving obstructing parts aside, and leaving a task partially-finished and then returning to it later. None of these skills and behaviors were specified. We encourage readers to watch our supplementary videos ¹ for additional examples of these emergent behaviors. We plan to release our code following the review and revision period.

II. RELATED WORK

A. Scalable Learning for Contact-Rich Assembly

Parallel physics simulation supports experience collection at scale for on-policy methods such as PPO [11], [12], with applications ranging from dexterous in-hand manipulation [13], [14], [15], [16] to industrial assembly. GPU-accelerated physics engines such as PhysX [17] and Newton [18] support large-scale parallel simulation for robot learning. For industrial assembly, Factory [2] demonstrated real-time or faster simulation of 1,000 simultaneous nut-and-bolt interactions in Isaac Gym, later incorporated into Isaac Lab. AlphaNIST uses this simulation throughput to collect experience across heterogeneous assembly pairs.

OmniReset [10] addressed state coverage through diverse, manipulation-centric resets, demonstrating long-horizon on-policy RL with fixed algorithm hyperparameters and no curricula or human demonstrations. Success-Guided Sampling (SGS) [7] addressed experience allocation by concentrating simulation near the policy’s changing capability frontier, scaling beyond one million parallel environments using sparse rewards. In its reported assembly experiments, SGS trains a separate policy for each single task. AlphaNIST retains SGS’s sampling rule but trains one multi-task policy across all 20 ATB1 tasks under a common binary task reward. Our experiments show that this shared policy learns assemblies that remain unsolved in single-task training (Fig. 5).

¹<https://alpha-nist.github.io/>

B. Multi-Task Reinforcement Learning

Multi-task RL aims to improve learning by sharing experience across tasks. However, standard joint-training baselines have achieved lower success rates than separately trained policies on multi-task manipulation benchmarks [19]. One approach to obtaining a single multi-task policy is to distill multiple separately trained specialists into a shared policy [20]. Other approaches address the difficulty of joint training directly. PopArt [21] addresses differences in reward scales with a separate value output and normalization statistics for each task, while PCGrad [19] removes conflicting components of task gradients before combining them. However, Kurin et al. [22] show that shared-network baselines with standard regularization and stabilization can match specialized multi-task optimizers.

AlphaNIST takes a simple approach to multi-task learning by exploiting the shared structure of assembly: threading, gear meshing, and insertion all receive the same binary task reward when assembly error falls below a common threshold. Under this shared objective, we train one geometry-conditioned policy and one value function with standard PPO at scale, without task-specific value heads, gradient projection, or specialist pretraining.

C. Learning-Based Robot Assembly

Prior learning-based assembly systems make exploration tractable by shortening the task horizon or restricting initialization to staged, pre-grasped, or local-insertion problems [23], [24]. Factory [2] trains separate pick, place, and screw subpolicies, while AutoMate [20] combines assembly-by-disassembly, imitation learning, and curriculum-based RL to obtain specialist and generalist assembly policies. Refinery [25] improves pretrained assembly specialists through Bayesian-optimization-guided sampling of initial conditions during fine-tuning and Gaussian-mixture-model-based initialization at deployment. The refined policies can then be chained to complete multi-part assemblies. For longer sequences, ARCH [26] uses a high-level imitation-learned policy to select from a library of RL and model-based primitives. These methods structure learning through initialization, demonstrations, or prescribed subskills. OmniReset [10] uses a generic dense objective with fixed weights, including gripper-object and object-goal distance terms. Zhang et al. [7] present the first end-to-end RL solution to individual tasks from the NIST ATB1, but retain task-specific auxiliary grasp- and gripper-related shaping. Such terms can aid exploration and discourage reward exploitation, but their relative influence can change with object size, geometry, and assembly motion. Even with fixed coefficients, a new task family requires checking that the shaped reward remains aligned with successful assembly.

AlphaGo Zero provides a broader motivation: terminal win/loss and search-generated policy targets support strategy discovery without human game data or intermediate reward shaping [9]. AlphaNIST asks whether binary task success can similarly support strategy discovery in continuous, contact-

rich control, with one shared policy learning heterogeneous assemblies end to end.

III. PROBLEM FORMULATION

A. Task Definition and MDP

We formulate all 20 ATB1 assembly tasks [1] as a multi-task reinforcement-learning problem with one shared policy.

Let $e \in \mathcal{E}$ denote an assembly task, with $|\mathcal{E}| = 20$. Each task corresponds to one *assembly pair*: a movable *part*, its *mating geometry*, and a fully-assembled configuration.

We model the problem as a finite-horizon MDP,

$$\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, r, \gamma, T, \rho), \quad (1)$$

where $p(s_{t+1} | s_t, a_t)$ is the transition kernel and ρ is the joint distribution over task identities and initial states. At the beginning of each episode, $(e, s_0) \sim \rho$. A shared policy $\pi_\theta(a_t | o_t)$ acts on partial observations o_t of the simulator state s_t and maximizes

$$J(\theta) = \mathbb{E}_{\rho, p, \pi_\theta} \left[\sum_{t=0}^{T-1} \gamma^t r_t \right]. \quad (2)$$

Task variation is represented through surface point clouds in o_t , rather than an explicit discrete task identifier.

IV. METHOD

A. AlphaNIST Training Framework

AlphaNIST trains one policy across all 20 assembly tasks with standard PPO and a binary task reward for completed assembly. Its training distribution combines diverse cached reset states from OmniReset [10] with beta-distributed success-guided sampling (SGS) [7], while a common binary task reward defines successful assembly across all tasks. The three components play distinct roles:

- Reset bank:** Which states are available?
- Beta sampling:** Which states are sampled?
- Binary task reward:** What counts as success?

B. Reset Coverage and Experience Allocation

We adapt OmniReset [10] to heterogeneous assembly by caching the assembly identity together with each reset state:

$$\mathcal{D}_{\text{reset}} = \{(e_i, \bar{s}_i)\}_{i=1}^M. \quad (3)$$

Here, M is the total number of entries across all 20 assemblies. The assembly identity e_i selects the part and mating-geometry assets and their inertial properties, while $\bar{s}_i$ specifies the robot, board, and movable-part state. Sampling an entry therefore selects both the assembly and its initial configuration. The cached bank spans ungrasped workspace, pickup, transport, near-mating, and partially assembled configurations (Fig. 4), enabling GPU-parallel initialization. These reset families define regions of the training distribution, not a prescribed sequence; the policy receives no stage identifier.

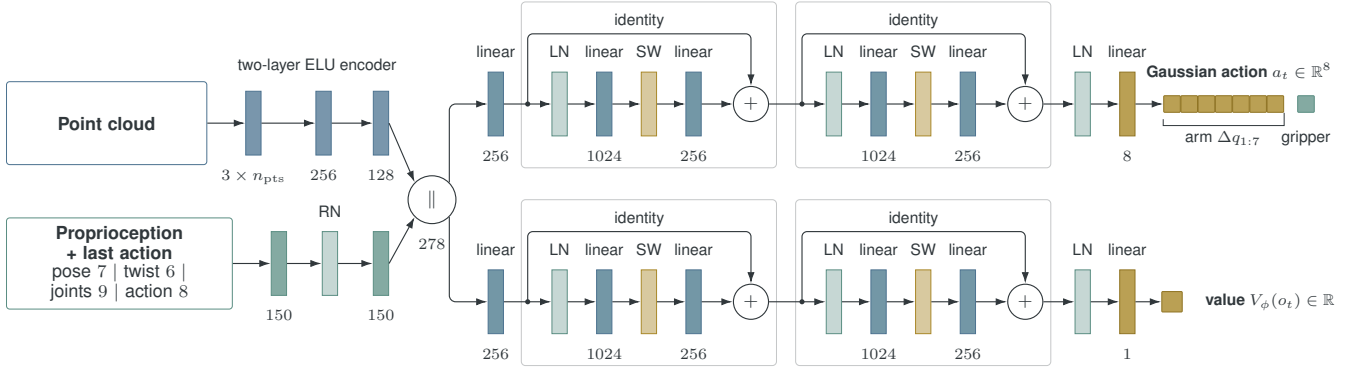


Fig. 3. **Policy architecture.** Actor and critic have the same architecture with separate parameters. Inputs are drawn once; each network contains two consecutive residual blocks. RN: running normalization; LN: layer normalization; SW: Swish activation; n_{pts} : number of input points.

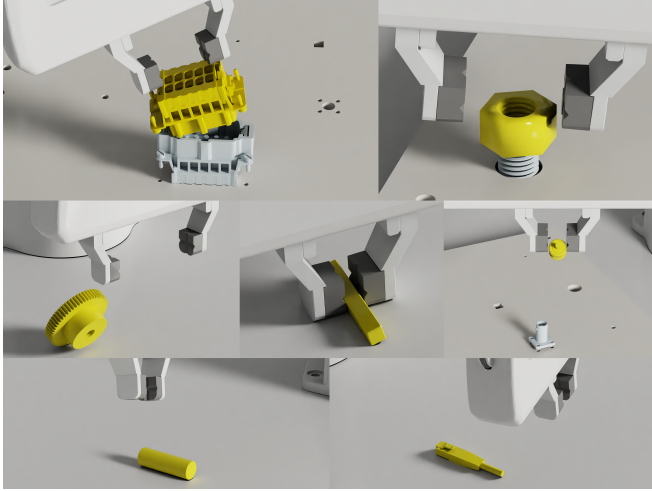


Fig. 4. **Diverse assembly reset states.** Examples of resets across tasks, ranging from random object poses to in-hand grasps and partial assemblies.

Let $\hat{p}_i$ be the empirical success rate over the most recent L episodes initialized from entry $(e_i, \bar{s}_i)$. Under beta sampling, entry i is selected with probability q_i :

$$q_i \propto \varepsilon_q + \sqrt{\hat{p}_i(1 - \hat{p}_i)}, \quad i = 1, \dots, M, \quad (4)$$

normalized over the entire bank. Sampling $i \sim \text{Cat}(\mathbf{q})$ selects an assembly–reset pair with $\rho(e_i, \bar{s}_i) = q_i$. The kernel favors intermediate success rates, while $\varepsilon_q > 0$ keeps every entry sampleable. Each episode’s outcome updates $\hat{p}_i$ for its sampled entry. The uniform ablation sets $q_i = 1/M$; the bank, policy, reward, and optimizer remain unchanged. Algorithm 1 summarizes training.

C. Binary Full-Assembly Objective

Across geometries, the assembly objective is the same: complete the assembly. For each instance, an assembly–relative coordinate $\Phi_e(\alpha)$, $\alpha \in [0, 1]$, parameterizes the part in the mating frame, with $\Phi_e(1)$ denoting its fully assembled pose. Let $d_{\text{asm}}^e(s)$ be the distance between the part’s current relative pose and $\Phi_e(1)$. For a single-pair episode, the binary task reward is

$$r_t^{\text{succ}} = \begin{cases} 1, & d_{\text{asm}}^e(s_{t+1}) < \epsilon, \\ 0, & \text{otherwise.} \end{cases} \quad (5)$$

where $\epsilon = 0.001$ m. The success reward is issued once, and the episode terminates. Intermediate progress receives no task reward. We supplement the binary task reward with a control cost to discourage large action magnitudes, rapid joint motion, and excessive torque. This regularization biases learning toward smoother, lower-effort behaviors that are more compatible with safe deployment on physical robots, without providing information about assembly progress.

$$r_t = \underbrace{r_t^{\text{succ}}}_{\text{binary task reward}} - \underbrace{c(\|a_t\|_2^2 + \|\dot{q}_t\|_2^2 + \|\tau_t\|_2^2)}_{\text{control cost}}, \quad c \geq 0. \quad (6)$$

Here, a_t , $\dot{q}_t$, and τ_t are the action, robot joint velocity, and applied joint torque. The control penalties are evaluated at every step, with a single coefficient c fixed across tasks. They do not measure assembly progress; the binary task reward specifies successful completion. Episodes end on assembly success, at horizon T , or when an arm joint exceeds its hardware velocity or torque limit. In multi-pair episodes, the success reward remains zero until every active pair is assembled, with no prescribed pair order or intermediate completion reward.

D. Shared Policy and Robot Control

We implement AlphaNIST in Isaac Lab [11] with GPU-parallel environments. Each PPO batch combines rollouts from assembly–reset pairs sampled using Eq. (4). The policy, reward, controller, and training configuration are shared across all 20 tasks. The policy observes the robot state and surface point clouds of the movable part, mating geometry, and robot, without a discrete task identifier. We use separate actor and critic SimBa networks [27], [28], summarized in Fig. 3. Each combines a point-cloud embedding with the proprioception and last action shown in the figure.

We simulate a 7-DoF Franka Emika Panda with identified inertial and drivetrain parameters and a parallel-jaw gripper with one driven and one mechanically coupled passive finger. At 25 Hz, the policy outputs a 7D relative joint position target ($q_t^{\text{target}} = q_t + \Delta q_t$) and a 1D gripper command. An implicit impedance controller uses the manufacturer’s *libfranka* stiffness and damping profile with hardware-matched armature and joint friction. Joint-speed limits are 2.175 rad/s for joints 1–4 and 2.61 rad/s for joints 5–7; torque limits are

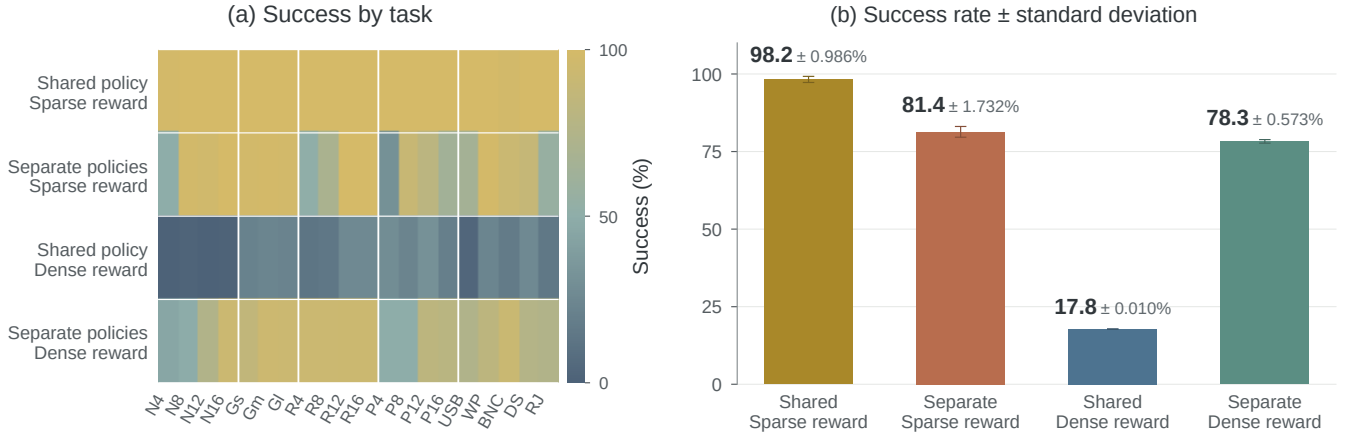


Fig. 5. **Training across heterogeneous assembly pairs substantially improves learning from a sparse binary task reward.** At 64K environments across 25k iterations: (a) compares 20-task success rates for one shared policy versus 20 separately trained policies. The shared policy (first row) achieved above 98% success rate across all tasks, including three precision assemblies that corresponding specialists rarely solve. Dense shaping does not exhibit the same benefit. Panel (b) shows overall success, note that the separate-policy success rate is average across all 20 per-task policies; *Asset notation*: N, G, R, and P denote nuts, gears, rods, and pegs, respectively; numeric suffixes indicate nominal diameters in millimeters, while s, m, and l indicate gear size. "Sparse reward" denotes Eq. (6); "dense reward" additionally includes the shaping terms in Eq. (7).

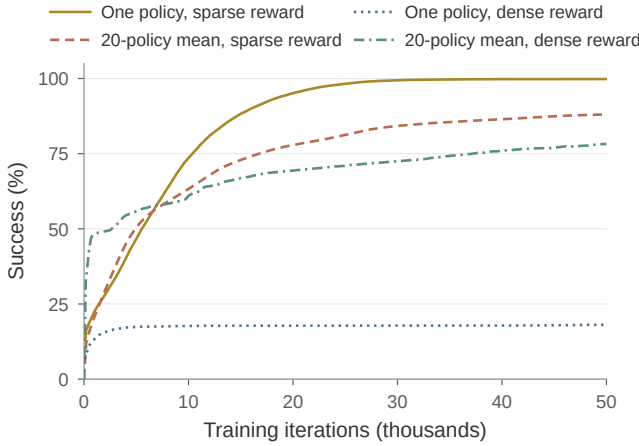


Fig. 6. **Joint sparse training converges faster than separate policies.** Curves compare one shared policy trained on all 20 tasks with the mean of 20 policies trained separately, one per task.

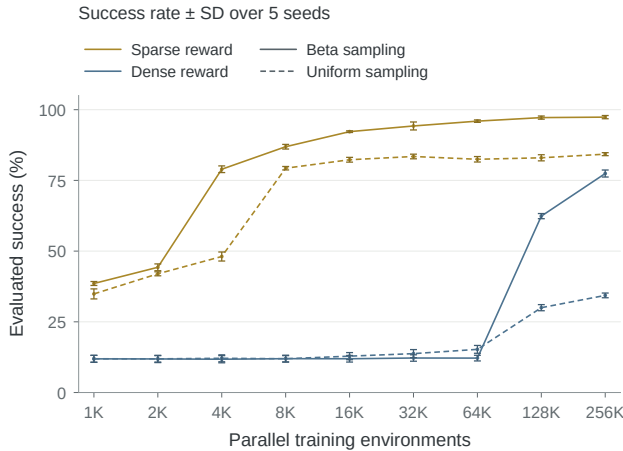


Fig. 7. **Sparse task rewards benefit strongly from parallel experience.** Mean success $\pm$ standard deviation over 5 evaluation seeds for 25k-iteration checkpoints, with 32 trials per task per seed across 20 assemblies. Color denotes reward; solid and dashed lines denote beta and uniform training-reset sampling.

Algorithm 1 AlphaNIST Heterogeneous Assembly Learning

Require: $\mathcal{D}_{\text{reset}} = \{(e_i, \bar{s}_i)\}_{i=1}^M$; history length L ; batch size N ;
shared π_θ, V_ϕ
1: initialize each rolling success rate $\hat{p}_i \leftarrow 0$
2: **for** each PPO iteration **do**
3: $\mathcal{B} \leftarrow \emptyset$
4: **while** $|\mathcal{B}| < N$ **do**
5: **for** all completed episodes n **do**
6: $y_n \leftarrow \mathbb{K}[\text{full assembly}]$ for entry i_n
7: $\hat{p}_{i_n} \leftarrow$ mean of its latest L outcomes
8: **end for**
9: update reset probabilities $\mathbf{q}$ using Eq. (4)
10: **for** all environments n requiring reset **do**
11: $i_n \sim \text{Cat}(\mathbf{q}); (e_n, s_{n,0}) \leftarrow (e_{i_n}, \bar{s}_{i_n})$
12: **end for**
13: $a_{n,t} \sim \pi_\theta(\cdot | o_{n,t}), \forall n$
14: step all environments and append transitions to $\mathcal{B}$
15: **end while**
16: update π_θ and V_ϕ using standard PPO on $\mathcal{B}$
17: **end for**

87Nm and 12Nm, respectively. These constraints keep actions within a hardware-realistic envelope, preventing the policy from exploiting unrealistic speeds or torques. Similar hardware-aligned simulation has enabled transfer of assembly policies to the same Franka Panda platform [20], [29].

E. Distributed Training and Scale

We scale synchronous PPO across 1–256K environments, with 4K per GPU in multi-GPU runs. Each GPU samples its local reset bank using rolling success estimates. Actor and critic gradients are averaged across GPUs before each optimizer step. Each iteration collects 32 steps per environment and optimizes for 5 epochs with 4 minibatches each. Network architecture and PPO settings are fixed across scales.

V. EXPERIMENTS

We organize the experiments around three questions: **(Q1) Heterogeneity:** Under a binary task reward, how does joint training compare with training each task separately? **(Q2) Scaling:** How do parallel experience and reset-state sampling

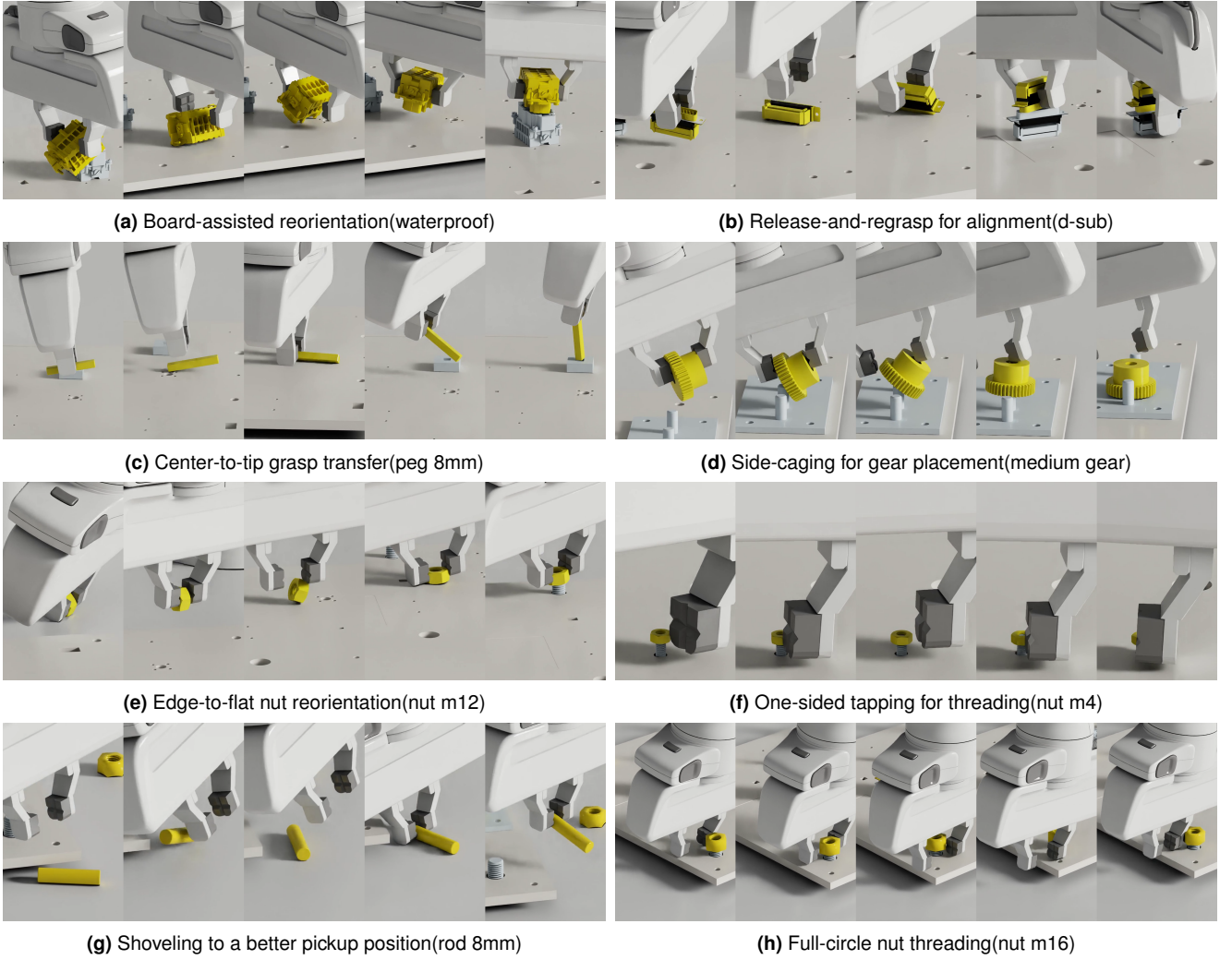


Fig. 8. **One shared policy discovers diverse contact strategies from a sparse binary task reward.** Each panel proceeds from left to right. (a) Waterproof connector: board contact helps reorient the part before insertion. (b) D-sub connector: the part is set down and regrasped for alignment with its socket. (c) 8 mm peg: the part is dropped from a center grasp and regrasped near one end, leaving its remaining length free for insertion. (d) Medium gear: the part pivots from a side grasp into a horizontal pose over its mounting post. (e) M12 nut: the nut is rolled from its edge onto a flat face before placement on the bolt. (f) M4 nut: repeated one-sided taps advance the nut along the thread. (g) 8 mm rod: the fingertip shovels the rod to a position where it is easier to lift. (h) M16 nut: the gripper moves around the nut, using fingertip contact to turn it along the bolt.

affect learning from a binary task reward? **(Q3) Reward formulation:** How does a binary task reward compare with dense assembly-progress shaping as heterogeneous training scales? We additionally explore multi-pair assembly and inspect the learned policy for behaviors that arise without direct supervision.

For heterogeneity, we compare one policy trained across all 20 ATB1 tasks with one separately trained policy per task, using matched per-run interaction budgets and otherwise identical settings. We use 64K environments per run to keep the comparison affordable; the shared policy reaches 98.2% training success at this scale.

For parallel scaling, GPUs, environments, and rollout batch size increase together while the task distribution, reward, policy, and environment remain fixed. The sampling ablation compares against uniform sampling over the same reset bank instead of beta sampling. For the scaling evaluation, all checkpoints use stochastic actions and a common frozen mixed-reset bank, sampled uniformly within

each task. Success requires full assembly within 15 seconds without a simultaneous safety failure. The reward ablation adds two dense assembly-progress terms to Eq. (6):

$$r_t^{\text{dense}} = r_t + c_2 r_t^{\text{reach}} + c_3 r_t^{\text{align}}, \quad c_2, c_3 \geq 0, \quad (7)$$

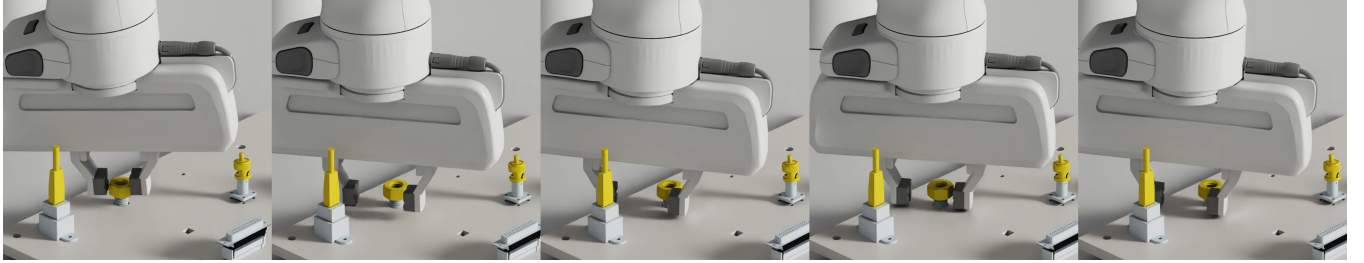
where r_t^{reach} rewards end-effector proximity to the part centroid and r_t^{align} rewards alignment with the target assembly orientation. These are common shaping terms in manipulation and assembly RL [2], [10], [20]. We set $c_2 = c_3 = 0.01$, matching the coefficients used in OmniReset [10]. Across all experiments, success rates cover all reset configurations.

A. Assembly-Pair Heterogeneity Enables Learning

From randomly initialized states, separately-trained policies have $\leq 0.3\%$ success on M4 nut threading, as well as 4 mm peg insertion and rod insertion. The main obstacle is that the small parts are difficult to grasp reliably from the table. In contrast, the shared policy produces successful grasp-and-assembly rollouts on these same tasks without task-



(a) Bring USB-A closer, assemble the 8 mm rod, then return



(b) Making small turns threading M12 to avoid RJ45 collision

Fig. 9. **Emergent coordination in multi-pair assembly.** (a) The policy brings the USB-A connector from a distant corner to a nearby working position, leaves it there while assembling the 8 mm rod, then returns to insert the connector. (b) During M12 nut threading, the end effector makes small turns to avoid touching the neighboring RJ45 connector. Each sequence proceeds from left to right.

TABLE I

SUCCESS RATE FOR MULTI-PAIR ASSEMBLY EACH POLICY IS TRAINED AND EVALUATED ON THE SAME NUMBER OF ACTIVE ASSEMBLY PAIRS, FROM 1–10. ALPHANIST SOLVES THESE TASKS WITHOUT A PRESCRIBED ORDER OR EXPLICIT TASK DECOMPOSITION. FOR EXAMPLE, WHEN THERE ARE 10 ACTIVE PAIRS, THE POLICY RECEIVES A SUCCESS REWARD ONLY WHEN ALL 10 ARE IN THEIR ASSIGNED BOARD SLOTS.

# Active Pairs	1	2	3	4	5	6	7	8	9	10
Success Rate (%)	99	97	95	93	92	90	89	87	86	84

specific changes. This result suggests that shared experience can help with assemblies that are difficult to learn separately. This benefit depends strongly on the reward formulation. At their final checkpoints, the 20 separately trained dense-reward policies reach a mean success rate of 78.3%, ranging from 45.0% on M4 threading to 93.6% on the medium gear. Joint dense training reaches 17.8% overall success at 25K iterations, with low completion across tasks (Fig. 5). One possible explanation is that dense shaping terms favor different partial solutions across geometries, whereas the binary task reward gives a unified full-assembly objective for every task.

Joint training reaches higher success at lower total cost. At 64K parallel environments across 25K iterations, the **shared policy trained with the sparse binary task reward reaches 98.2% training success** in 410 GPU-hours (Fig. 6), while the 20 separately trained policies average 81.4% and consume 16.1K total GPU-hours, $39.3\times$ more device-hours.

B. Sparse Binary Task Rewards Benefit from Scale

Performance with the sparse binary task reward improves rapidly with increased parallel experience (Fig. 7). With beta sampling, evaluation success rises from 38.6% at 1K environments to 78.9% at 4K and 86.9% at 8K. With dense reward shaping, success remains at 11.8–12.0% over 1K–4K environments. Beta sampling provides an additional gain

over uniform reset sampling. At 256K environments, success with the sparse binary task reward reaches **97.4% with beta sampling** versus 84.3% with uniform sampling; the corresponding results with dense reward shaping are 77.4% and 34.3%. While the advantage of the sparse task objective is already visible at modest scale, beta sampling provides further gains as parallel experience increases.

C. Emergent Strategies over Short Horizons

The policy exhibits a broad range of dexterous strategies across assemblies (Fig. 8), including grasp adaptation, extrinsic reorientation, and contact-driven assembly. It releases and reacquires parts when the initial grasp is unsuitable for insertion, uses surrounding geometry to change object orientation, and discovers intermittent tapping and pivoting motions. These behaviors emerge within a single policy, without demonstrations or rewards specifying the individual maneuvers. For a peg, a successful pickup may not produce a grasp suitable for insertion; the policy drops a center-held peg and regrips it near one end, exposing enough shaft length for complete insertion. For a nut, repeated taps on one side engage and advance the thread, using intermittent contact to make assembly progress.

D. Emergent Strategies over Long Horizons

Beyond the main single-assembly experiments, we train policies with multiple assembly pairs per episode. The bi-

nary task reward is issued only when every active pair is assembled; episodes allow 15 seconds per pair and terminate upon success. Table I shows the success rates for episodes with up to ten active pairs. This multi-pair setting reveals coordination over substantially longer horizons (Fig. 9). The policy need not finish assembling a pair before turning to another, and its approach to one assembly changes based on neighboring parts. These planning-like behaviors—taking actions whose utility is realized later and adapting one assembly to the state of another—emerge without an explicit planner, policy switching, or prescribed sequence.

VI. LIMITATIONS

First, all results are in simulation. Hardware-matched velocity and torque limits constrain the learned motions, but do not establish robustness to perception error, latency, or contact-model mismatch. Transfer and repeatability will be evaluated on the physical board using sensor-derived observations and randomized contact and latency models. Second, while applying the 10-pair policy to successive groups of parts could offer a route to completing the full board, extending our multi-pair framework to 20 parts remains an exciting research goal. Achieving this at scale, under a single terminal task reward, will likely benefit from more efficient simulation and learning methods.

VII. CONCLUSION

AlphaNIST shows that heterogeneous contact-rich assembly can be learned with a *single* policy using a binary task objective, diverse reset coverage, and large-scale reinforcement learning. In simulation, one policy learns all 20 ATB1 tasks without demonstrations, prescribed stages, or per-task tuning, while outperforming separately trained specialists on the most difficult small-part assemblies. Across all tested scales, the binary task objective outperforms the evaluated dense reward shaping. The learned policy discovers grasp adaptation, extrinsic reorientation, tapping, threading, and longer-horizon coordination without direct supervision. In multi-pair assembly, it can prepare one part, complete another, and return to finish the first, without an intermediate assembly reward or a planner prescribing the sequence.

More broadly, these findings challenge the assumption that task complexity and diversity must be reduced to make robot learning tractable. Instead, they show that diverse experience under a shared objective can itself become a source of reusable dexterity and coordination. We view this work as a critical step toward general-purpose assembly policies that discover how to solve complex physical tasks, rather than requiring those strategies to be specified in advance.

REFERENCES

- [1] K. Kimble *et al.*, “Benchmarking protocols for evaluating small parts robotic assembly systems,” *IEEE Robot. Automat. Lett.*, vol. 5, no. 2, pp. 883–889, 2020.
- [2] Y. Narang *et al.*, “Factory: Fast contact for robotic assembly,” in *Proc. Robot.: Sci. Syst.*, 2022.
- [3] T. Lozano-Pérez and R. H. Wilson, “Assembly sequencing for arbitrary motions,” in *Proc. IEEE Int. Conf. Robot. Automat.*, 1993, pp. 527–532.
- [4] W. Lian, T. Kelch, D. Holz, A. Norton, and S. Schaal, “Benchmarking off-the-shelf solutions to robotic assembly tasks,” in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 791–798.
- [5] A. S. Morgan, Q. Bateux, M. Hao, and A. M. Dollar, “Towards generalized robot assembly through compliance-enabled contact formations,” in *Proc. IEEE Int. Conf. Robot. Automat.*, 2023, pp. 8010–8016.
- [6] J. Luo *et al.*, “Robust multi-modal policies for industrial assembly via reinforcement learning and demonstrations: A large-scale study,” in *Proc. Robot.: Sci. Syst.*, 2021.
- [7] O. Zhang, M. Guaman Castro, P. Yin, A. Gupta, R. Scalise, and B. Boots, “A balanced data diet: Addressing the bottleneck in mega-scale RL for robot control,” 2026. [Online]. Available: <https://sgs-rl.github.io/>
- [8] D. Silver *et al.*, “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, pp. 484–489, 2016.
- [9] —, “Mastering the game of Go without human knowledge,” *Nature*, vol. 550, pp. 354–359, 2017.
- [10] P. Yin *et al.*, “Emergent dexterity via diverse resets and large-scale reinforcement learning,” in *Proc. Int. Conf. Learn. Representations*, 2026. [Online]. Available: <https://arxiv.org/abs/2603.15789>
- [11] M. Mittal *et al.*, “Isaac Lab: A GPU-accelerated simulation framework for multi-modal robot learning,” 2025, arXiv:2511.04831.
- [12] K. Zakka, Q. Liao, B. Yi, L. L. Lay, K. Sreenath, and P. Abbeel, “mjlab: A lightweight framework for GPU-accelerated robot learning,” 2026, arXiv:2601.22074.
- [13] M. Andrychowicz *et al.*, “Learning dexterous in-hand manipulation,” *Int. J. Robot. Res.*, vol. 39, no. 1, pp. 3–20, 2020.
- [14] H. Qi *et al.*, “General in-hand object rotation with vision and touch,” in *Proc. Conf. Robot Learn.*, 2023, pp. 2549–2564.
- [15] R. Singh, A. Allshire, A. Handa, N. Ratliff, and K. Van Wyk, “DextrAH-RGB: Visuomotor policies to grasp anything with dexterous hands,” 2024, arXiv:2412.01791.
- [16] A. Handa *et al.*, “Dextreme: Transfer of agile in-hand manipulation from simulation to reality,” in *Proc. IEEE Int. Conf. Robot. Automat. (ICRA)*, 2023, pp. 5977–5984.
- [17] NVIDIA, “GPU simulation,” PhysX SDK 5.4.1 documentation, accessed: Sep. 14, 2026. [Online]. Available: <https://nvidia-omniverse.github.io/PhysX/physx/5.4.1/docs/GPURigidBodyBodies.html>
- [18] The Newton Contributors, “Newton: GPU-accelerated physics simulation for robotics and simulation research,” 2025. [Online]. Available: <https://github.com/newton-physics/newton>
- [19] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn, “Gradient surgery for multi-task learning,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 5824–5836.
- [20] B. Tang *et al.*, “AutoMate: Specialist and generalist assembly policies over diverse geometries,” in *Proc. Robot.: Sci. Syst.*, 2024.
- [21] M. Hessel, H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. van Hasselt, “Multi-task deep reinforcement learning with PopArt,” in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, no. 1, 2019, pp. 3796–3803.
- [22] V. Kurin, A. De Palma, I. Kostikov, S. Whiteson, and P. K. Mudigonda, “In defense of the unitary scalarization for deep multi-task learning,” in *Proc. Adv. Neural Inf. Process. Syst.*, 2022, pp. 12 169–12 183.
- [23] G. Schoettler *et al.*, “Deep reinforcement learning for industrial insertion tasks with visual inputs and natural rewards,” in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 5548–5555.
- [24] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel, “Reverse curriculum generation for reinforcement learning,” in *Proc. Conf. Robot Learn.*, 2017, pp. 482–495.
- [25] B. Tang *et al.*, “Refinery: Active fine-tuning and deployment-time optimization for contact-rich policies,” 2025, arXiv:2510.11019.
- [26] J. Sun *et al.*, “ARCH: Hierarchical hybrid learning for long-horizon contact-rich robotic assembly,” in *Proc. Conf. Robot Learn.*, 2025. [Online]. Available: <https://arxiv.org/abs/2409.16451>
- [27] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
- [28] H. Lee *et al.*, “SimBa: Simplicity bias for scaling up parameters in deep reinforcement learning,” in *Proc. Int. Conf. Learn. Representations*, 2025. [Online]. Available: https://proceedings.iclr.cc/paper_files/paper/2025/hash/741aabb8b41a2987867acc9939ad50383-Abstract-Conference.html
- [29] B. Tang *et al.*, “IndustReal: Transferring contact-rich assembly tasks from simulation to reality,” in *Proc. Robot.: Sci. Syst.*, 2023.