AutoMate: Specialist and Generalist Assembly Policies over Diverse Geometries

Author Names Omitted for Anonymous Review. Paper-ID 94

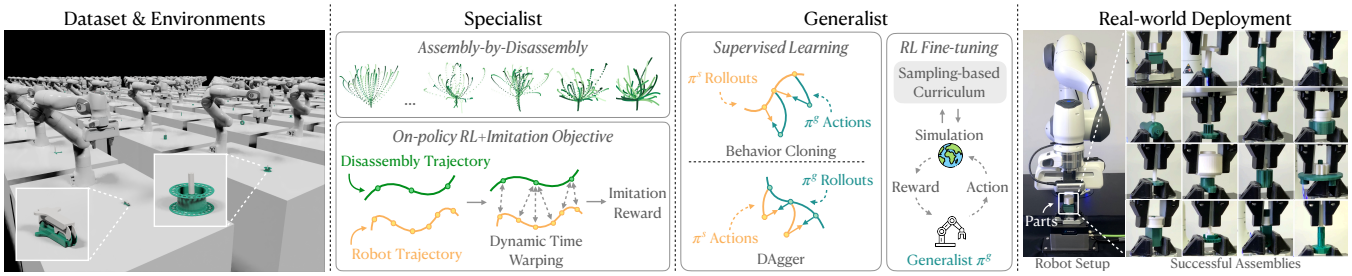

Fig. 1: We present A) a dataset of 100 interpenetration-free assemblies that can be simulated in robotics simulators and assembled in the real world, as well as simulation environments for all 100 assemblies in [48, 56]; B) specialist (i.e., part-specific) policies trained with a novel approach combining assembly-by-disassembly, RL with imitation, and dynamic time warping, which can solve 80 assemblies with ≈80%+ success rates; C) a generalist (i.e., unified) policy trained with policy distillation and RL fine-tuning, which can solve 20 assemblies with 80%+ success rates; and D) zero-shot sim-to-real transfer of the specialist and generalist policies, including end-to-end deployments. During our evaluations, we execute 5M+ simulated trials and 500 real-world trials.

***Abstract*—Robotic assembly for high-mixture settings requires adaptivity to diverse parts and poses, which is an open challenge. Meanwhile, in other areas of robotics, large models and sim-to-real have led to tremendous progress. Inspired by such work, we present** AutoMate**, a learning framework and system that consists of 4 parts: 1) a dataset of 100 assemblies compatible with simulation and the real world, along with parallelized simulation environments for policy learning, 2) a novel simulation-based approach for learning specialist (i.e., part-specific) policies and generalist (i.e., unified) assembly policies, 3) demonstrations of specialist policies that individually solve 80 assemblies with ≈80%+ success rates in simulation, as well as a generalist policy that jointly solves 20 assemblies with an 80%+ success rate, and 4) zero-shot sim-to-real transfer that achieves similar (or better) performance than simulation, including on end-to-end assembly.[1] To our knowledge,** AutoMate **provides the first simulation-based framework for learning specialist and generalist policies over a wide range of assemblies, as well as the first system demonstrating zero-shot sim-to-real transfer over such a range.**

I. Introduction

Most objects in home and industrial settings consist of multiple parts that must be assembled [89]. Human workers typically perform assembly; however, in certain industries (e.g., automotive), robotic assembly is prevalent. As industrial robots typically use stiff controllers and perform repetitive motions, robotic assembly requires highly-customized engineering of fixtures, tooling, and waypoints. Nevertheless, in high-mixture settings, *adaptive* assembly is required [36], in which robots must assemble parts with diverse geometries and poses. Adaptive assembly is non-trivial even for skilled human workers and is a major open challenge in robotics.

Meanwhile, in other areas of robotics, large models and sim-to-real methods have led to tremendous progress. Large language, vision, and visual-language models have led to vision-language-*action* models with high-level reasoning and multi-task performance [67, 6, 7, 42]. Fast simulators ([48, 83, 53]) and domain adaptation ([82, 59, 65]) have enabled robust policies for locomotion [40, 69] and manipulation [2, 27]. Nevertheless, large models and sim-to-real are extremely nascent for contact-rich tasks in industrial settings, including assembly. Prior research has often focused on training specialist (i.e., part-specific) policies for 1-5 parts (up to a max of 15-20 parts [73, 74]), and simulation-based development and transfer of generalist (i.e., unified) policies has not been explored. Furthermore, the first zero-shot sim-to-real transfer for end-to-end assembly has only recently been demonstrated [76].

In this context, we present **AutoMate**, a robotic system framework for solving diverse assembly problems with specialist and generalist policies, in simulation and with zero-shot transfer to reality (**Figure 1**). Our specific contributions are:

- **Dataset and Environments**: We provide a dataset of 100 assemblies based on [80], but interpenetration-free and with realistic clearances, enabling simulation and real-world assembly. We also provide parallelized simulation environments for all 100 assemblies, allowing others to train specialist and generalist assembly policies.
- **Learning Methods**: For training specialists, we propose a novel approach combining 3 distinct algorithms: assembly-by-disassembly, reinforcement learning (RL) with an imitation objective, and dynamic time warping. For training a generalist, we apply geometric encoding, policy distillation, and curriculum-based RL fine-tuning.
- **Specialist and Generalist Policies**: We use our dataset, environments, and methods to learn specialist policies in simulation that can individually solve 80 assemblies with ≈80%+ success rates over 500k trials. We also learn a

[1]We use *zero shot* to refer to sim-to-real without a real-world adaptation phase, and *end to end* to refer to performing assembly with perception, grasping, and insertion (rather than starting from a pre-grasped state).

generalist policy in simulation that can jointly solve 20 assemblies with an 80%+ success rate over 100k trials.
- **Sim-to-Real**: We design and demonstrate a real-world system that can deploy our specialist policies in zero-shot with 86.5% success rates over 20 assemblies and 200 trials, and our generalist policy in zero-shot with an 84.5% success rate over 20 assemblies and 200 trials. We also execute end-to-end assembly with 86.0-90.0% success rates over 5 assemblies and 100 trials.

We present the above in **Sections IV, V, VI**, and **VII**.

To our knowledge, **AutoMate** provides the first simulation-based framework for learning specialist and generalist policies over a wide range of assemblies, as well as the first system demonstrating zero-shot sim-to-real transfer over such a range. Through this work, we aim to gradually build towards the large-model paradigm for industrial robotics, while staying grounded in real-world deployment.

II. Related Works

Research in robotic assembly has recently experienced significant growth, as described in [53, 80, 76, 96]. We focus our review on 1) datasets and benchmarks for assembly of small, realistic parts, 2) our building blocks for learning specialist assembly policies (i.e., assembly-by-disassembly, RL with imitation, and dynamic time warping), 3) our core technique for learning a generalist assembly policy (i.e., policy distillation), and 4) sim-to-real transfer for assembly.

A. Assembly Datasets and Environments

There are few existing datasets and environments for assembling small, realistic parts in simulation and the real world. Simulation efforts include [91], a large-scale CAD dataset for realistic assemblies; [80], which provides a version suitable for research; [53], which provides simulation environments for peg, gear, and connector insertion, and [76], which provides similar environments, as well as a real-world benchmarking kit [77]. The most established real-world effort is [35, 36], a benchmark for tasks such as connector insertion, pulley alignment, and cable weaving, as well as a small dataset of CAD models, STL files, images, and point clouds [57]. Finally, other research efforts have provided datasets and environments for additional assembly domains (e.g., [41, 15, 28]).

Among these efforts, the state-of-the-art may be considered [80] for CAD datasets, [53, 28] for simulation environments, and [35] for real-world benchmarks. Our work draws upon the strengths of each by providing 1) a diverse CAD dataset of 100 interpenetration-free assemblies based on [80], 2) ready-to-use, parallelized simulation environments for all 100 assemblies in [48, 56], and 3) a real-world benchmarking kit corresponding to the environments. These components comprise the first unified dataset and environments for sim-to-real transfer for robotic assembly at an appreciable scale.

B. Learning Specialist Assembly Policies

There are few directly-comparable works for learning specialist policies for assembling a large number of diverse parts (here, 100). Recent efforts have focused on perception [50, 87, 21, 22] or planning [80, 81] without learning policies robust to disturbances and noise, or learning policies for a small number of assemblies (1-5), with just a few effort attempting $>$10 assemblies [73, 74, 97]. Thus, we instead review works addressing challenges that we faced when learning specialist policies: 1) generating demonstrations for robotic assembly in simulation, 2) augmenting RL with demonstrations, and 3) selecting relevant demonstrations to use during learning.

For (1), the prevailing approach is assembly-by-disassembly (i.e., generating disassembly sequences/paths and reversing them for use in assembly) [18]. State-of-the-art tree search methods for this process are developed in [80, 81]. Importantly, physical laws dictate that only sequences and paths can be reversed (rather than velocities and accelerations) [90].

For (2), there is a diverse set of effective approaches, including bootstrapping RL with behavior-cloned policies [64], adding demonstrations to the replay buffer for off-policy RL [85], augmenting the policy gradient for on-policy RL [64], learning a reward function from demonstrations [55] or human preferences [14], and explicitly including an imitation objective in the reward function for on-policy RL [58, 60].

Finally, for (3), the simplest approach is to select the demonstration closest to the initial position of the end effector. However, as we show, selecting the closest demonstration to the current position at each timestep produces more robust behavior, and selecting the closest demonstration to the *history* of positions (i.e., the end-effector path) is even more effective. Matching paths of varying lengths and discretizations is a well-known challenge; two state-of-the-art methods are dynamic time warping (DTW) [5] and signature transforms [34].

Our work combines the strengths of the preceding works by proposing a novel approach combining 3 algorithms: 1) assembly-by-disassembly, 2) RL with an imitation objective, and 3) trajectory matching via DTW. We select (2) for its simplicity and demonstrated effectiveness, and uniquely formulate our imitation objective to imitate paths rather than states or state-action pairs, and we select (3) based on subsequently-described evaluations. This combination enables effective training of specialist policies for $\approx$80% of our assemblies.

C. Learning Generalist Assembly Policies

There are few directly-comparable works for simulation-based learning of generalist policies for assembling a large number of parts. We instead contextualize our core technique for learning a generalist policy: policy distillation.

As described in **Section I**, large models have led to tremendous progress in robotics, including multi-task performance. Whereas such models often call existing skills or train from scratch, we aim for a middle ground by leveraging knowledge from specialist policies via distillation. *Distillation* refers to compressing a neural network by transferring knowledge from a larger *teacher* network to a smaller *student* network [8, 29]; *policy* distillation applies this to policy networks that map observations to actions [70]. In robotics, there are two main variations: 1) *cross-modal distillation*: transferring knowledge

from a teacher policy with privileged information (e.g., 6-DOF poses) to a student policy with realistic sensory inputs (e.g., proprioceptive data, RGB images) [9, 40, 11, 93, 1], and 2) *multi-task distillation* (a.k.a., *generalist-specialist learning*): transferring knowledge from multiple task-specific teachers to a single student [23, 32, 86]. Knowledge transfer is typically achieved via behavior cloning (BC) [62] and/or DAgger [68].

Inspired by [86], which focuses on grasping in simulation, our work leverages multi-task distillation to train our generalist policy from our specialist policies via BC, DAgger, and RL fine-tuning. In contrast, our work avoids the engineering complexity of [86], applies these techniques to robotic assembly, and deploys the learned generalist policy for the first time in the real world (as opposed to simulation only).

D. Sim-to-Real Transfer for Assembly

There have been numerous sim-to-real efforts for robotic assembly, which have used simulation to enable rapid development, safe policy learning, and scalable experimentation. [76] reviews prior studies in detail and notes that most involve large parts or clearances, use specialized fixtures or adapters, require human demonstrations, and/or require real-world policy adaptation. We briefly review more recent efforts.

[76] demonstrates zero-shot sim-to-real for end-to-end assembly on 9 tasks derived from [35]. They propose algorithms to overcome sim-to-real gap, including SAPU (penalizing interpenetration error during training) and PLAI (integrating actions to reduce steady-state error during deployment). [96] demonstrates sim-to-real with pregrasped parts and a real-world adaptation phase for 6 assemblies. They learn velocity targets and admittance gains for motion primitives in simulation and optimize gains online during deployment. [81] demonstrates sim-to-real for one 5-part assembly. They develop sequence and path-planning algorithms and execute the demonstration with precisely-fixtured parts. [95] demonstrates zero-shot sim-to-real with pregrasped parts for 6 assemblies. They train a policy in simulation to collect a dataset, which is used to train a planner and gain tuner via supervised learning; the planner and tuner are deployed in the real world.

Our work primarily applies the sim-to-real methods described and implemented in [76, 78] and requires no human demonstrations or policy adaptation. However, we allow plugs to be placed haphazardly on a platform or in the robot gripper, use a second gripper to grasp the sockets (rather than bolting them to a flat surface), optimize grasp poses (rather than manually specifying grasp heights), perform 6D pose estimation (rather than detection) during end-to-end deployments, and use identical control gains and action scales for all assemblies.

III. Problem Description

Our fundamental task is to use off-the-shelf, research-grade robot hardware to assemble a wide range of assemblies. Unlike most prior efforts, the assemblies consist of small parts with diverse geometries, the parts are initialized with appreciable 6-DOF pose randomization, no part-specific adapters or fixtures are leveraged, and no force-torque sensor is used.

Specifically, our experimental setup consists of 1) a Franka Panda robot with shore 30A finger pads mounted to a tabletop, 2) a wrist-mounted Intel RealSense D435 RGB-D camera, 3) a 3D-printed plug and socket[2] with 0.5-1.0 mm diametral clearances, and 4) a Schunk EGK40 gripper with shore 40A gripper pads mounted to the tabletop (akin to [31]). At the beginning of each experiment, the plug is haphazardly pressed into a foam block or placed in the robot gripper, and the socket is haphazardly placed in the Schunk gripper[3] (**Figure S11**).

We assume that 1) each assembly consists of 2 parts (thus, free from sequence planning [80]), 2) all parts have a size and initial position and orientation such that $\geq$1 grasp is feasible and $\geq$1 feasible grasp is sufficient to allow subsequent insertion (i.e., regrasping is not necessary), 3) a mesh file is available for each part, which is typical for industrial assembly applications, 4) the end effector can perform assembly in an approximately top-down configuration (i.e., within a 30° cone), which is also typical for many applications.

IV. Dataset and Environments

Our first contribution is a dataset of 100 assemblies compatible with both simulation and the real world, as well as parallelized simulation environments for all 100 assemblies.

A. Assembly Dataset

As described in **Section II-A**, [91] provides a large-scale CAD dataset of realistic assemblies, and [80] refines the dataset for research. However, most meshes still have nonzero interpenetration when assembled; thus, they are incompatible with simulators that enforce non-penetration constraints (e.g., [48]) and are infeasible to assemble in the real world.

We sample 100 assemblies from [80] that consist of 2 parts, are geometrically diverse, have graspable surfaces, require insertion (rather than simply alignment), and can be assembled approximately top-down. Most assemblies have 1 axisymmetric part; however, the part frequently has a symmetry-breaking feature. We perform several operations on these meshes: scaling, reorientation, translation, depenetration, chamfering (optional), and subdivision; for details, see **Appendix B**. Most critical is depenetration, where we 1) place each plug and socket in their assembled configuration, 2) compute the signed distance from each vertex on the plug to the surface of the socket [45], and 3) translate each vertex backward along its vertex normal until achieving a radial clearance of 0.5 mm.

The resulting assemblies are all interpenetration-free and have 1 mm of diametral clearance, making them simulation-compatible; furthermore, they have high triangle density, allowing simulation with fast contact methods that collide meshes against signed distance fields [46, 53] (**Figure 2**). In addition, they can all be 3D printed and assembled in the real world; due to printer overextrusion, our real-world assemblies have a tighter diametral clearance of 0.5-1.0 mm (**Figure 3**).

[2]We use *plug* to refer to a part that must be inserted, and *socket* to refer to a part that mates with the plug.

[3]Assembly tasks performed by humans typically require two hands, one for manipulation and the other for stabilization; the Schunk gripper allows us to stabilize the socket without incurring the cost of a second robot arm.

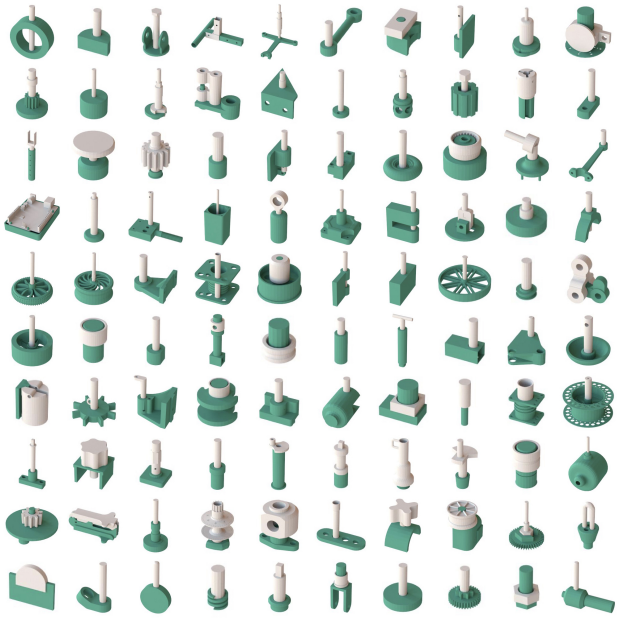

Fig. 2. **Simulation-compatible assembly dataset.** We provide a dataset of 100 assemblies derived from [80]. The assemblies are interpenetration-free, allowing them to be simulated in widely-used robotics simulators.

Fig. 3. **Real-world versions of assemblies from our dataset.** We print all 100 assemblies from our dataset in the real world and show 20 assemblies above, with unique IDs listed for later reference.

B. Assembly Environments

We provide ready-to-use, parallelized simulation environments for all 100 assemblies in the dataset in a robotics simulator [48, 56]; these environments can be used for arbitrary purposes, including training RL or imitation-learning policies (**Figure.1 A**). By default, each environment contains a Franka robot, a plug, and a socket. In the initial state, the robot and socket states are randomized and the plug is randomly initialized in the robot gripper (**Table II**); in the goal state, each plug is inserted into its corresponding socket. Initial states can be arbitrarily modified for custom applications.

We also provide utility functions that implement key algorithms used in the work (e.g., trajectory matching based on dynamic time warping). The simulation environments have been stress-tested by our own research; we aim for them to enable others to train their own specialist and generalist assembly policies and benchmark their results.

V. Learning Methods

Our second contribution is a set of methods for learning specialist and generalist policies over our assembly dataset. For specialist policies, we find that RL alone is ineffective; thus, we guide RL with imitation learning. We face 3 challenges: 1) generating demonstrations for assembly, 2) augmenting RL with demonstrations, and 3) selecting demonstrations to use during learning. To address these challenges, we propose a novel approach combining assembly-by-disassembly, RL with an imitation objective, and trajectory matching via dynamic time warping and signature transforms. We describe these building blocks in **Sections V-A, V-B, and V-C**, respectively.

For generalist policies, we face 3 additional challenges: 1) representing assembly geometry to the generalist network, 2) distilling knowledge from the specialists to the generalists, and 3) improving substandard performance. To address these challenges, we apply a combination of geometric encoding, policy distillation, and curriculum-based RL fine-tuning. We describe these components jointly in **Section V-D**. To see the results of these methods, skip to **Section VI**.

A. Specialist Learning: Assembly-by-Disassembly

When training specialists, our first challenge is to generate demonstrations for assembly. Collecting human demonstrations for assembly in simulation is challenging, requiring skilled operators and advanced teleoperation interfaces [49]. However, using motion planners is also difficult, as the kinematics of assembly are a narrow passage problem [75]. Inspired by [80, 18], we instead generate demonstration paths for disassembly, which we reverse to generate paths for assembly.

Specifically, we first perform kinematics- and geometry-based grasp sampling based on [19]; for details, see **Figure S12** and **Appendix C**. For each assembly, we initialize the plug and socket meshes in their assembled state, sample grasps along the surface of the plug, reject the samples if they violate kinematic or manipulability constraints, and repeat the process until generating 100 grasp candidates.

Next, we implement physics-based grasp evaluation. For each grasp, we randomize the pose of the assembly (**Table II**), command the robot to execute the grasp using a low-level controller, and lift the plug to a random pose (**Figure 4**). If the plug remains in the gripper after the procedure, the grasp is successful. We repeat the process for 1000 trials and compute success rates for all 100 grasps; the optimal grasp for the given assembly is the one with the highest success rate.

For each assembly, now only using the optimal grasp, we repeat the evaluation procedure. For each successful trial, we consider the trajectory of the end effector as a disassembly demonstration D_i. In general, D_i consists of states $x =$

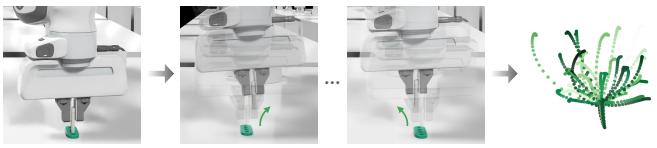

Fig. 4. **Simulation-based generation of disassembly paths.** For each assembly, we generate disassembly paths by A) executing a grasp from a grasp optimization procedure, B) using a low-level controller to lift the plug from the socket and move to a randomized pose, and C) repeating the process for additional poses, until D) collecting 100 successful disassembly paths.

$[x_i^1, x_i^2, ..., x_i^{N_i}]$, where N is the number of states; each state x_i^j can be defined as $[p_i^j; v_i^j; a_i^j]$, where p is position, v is velocity, and a is acceleration. We define a *reversed* disassembly demonstration D_i' as simply $x_i' = [x_i^{N_i}, x_i^{N_i-1}, ..., x_i^1]$, which can naively be used as an assembly demonstration. However, the corresponding velocities $v_i' = [v_i^{N_i}, v_i^{N_i-1}, ..., v_i^1]$ and accelerations $a_i' = [a_i^{N_i}, a_i^{N_i-1}, ..., a_i^1]$ are in general non-physical. Thus, for each successful trial, we record only the reversed disassembly *path* $p_i' = [p_i^{N_i}, p_i^{N_i-1}, ..., p_i^1]$. We repeat the procedure until collecting 100 successful reversed disassembly paths (**Figure 4D, Figure S13**), which we treat as assembly paths in **Sections V-B and V-C**.

B. Specialist Learning: RL with Imitation Objective

When training specialists, our second challenge is to augment RL with demonstrations. Before we describe our augmentation approach, we briefly describe our baseline RL approach, which is a reimplementation of [76]; for RL formalism and extended descriptions, see **Appendix E**.

As described in **Section:IV-B**, the environments are initialized with a robot, a plug with randomized pose in the robot gripper, and a socket with randomized pose above a tabletop (**Table II**). Ultimately, the robot must learn a policy that allows it to assemble the plug and socket while being robust to initial randomization and control/perception error.

We formulate the robotic assembly problem as a Markov decision process (MDP), where the objective is to learn a policy that maximizes the expected sum of discounted rewards (i.e., solves the assembly problem). We use proximal policy optimization (PPO) [72, 47] to learn the policy and an approximation of the value function (hyperparameters in **Table III**).

Our observation space consists of joint angles, the current end-effector pose, and the end-effector goal pose; our input to the critic also includes joint velocities, end-effector linear/angular velocities, and the current plug pose (**Table IV**) [61]). To model real-world control/perception error, we apply noise to all socket-pose observations (**Table V**). Our action space consists of incremental (Δ) pose targets for a task-space impedance controller. Finally, our reward (without imitation) consists of terms that 1) penalize distance-to-goal, 2) penalize simulation error, and 3) reward task difficulty at each timestep, as well as a term that rewards task success (**Appendix E**).

Now we describe our augmentation approach. Inspired by [58, 60], we augment RL with demonstrations by directly adding an imitation reward to our reward formulation. Specifically, we define our per-timestep reward as follows:

$$R_t = \omega^B R_t^B + \omega^I R_t^I \tag{1}$$

where R_t^B is the baseline per-timestep reward, described above and in **Appendix E**; R_t^I is an imitation-based per-timestep reward that encourages the agent to mimic demonstrations; and ω^B and ω^I are weighting hyperparameters.[4]

Following [58], we define R_t^I as the maximum per-timestep reward over all demonstrations for the given assembly (i.e., the reversed disassembly paths from **Section V-A**). Specifically,

$$R_t^I = \max_{i=1,...,M} R_t^{I_i} \tag{2}$$

where M is the number of demonstrations. Unlike [58, 60], we apply the augmentation approach to contact-rich manipulation rather than locomotion. We define $R_t^{I_i}$ in **Section V-C**.

C. Specialist Learning: Trajectory Matching via Dynamic Time Warping and Signature Transforms

When training specialists, our third and final challenge is to select demonstrations to use during learning. Specifically, for a given assembly, we must define a reward $R_t^{I_j}$ that quantifies the instantaneous value of imitating any reversed disassembly path p_i', after which we can use **Equation 2**.

Intuitively, we can define $R_t^{I_i}$ as the distance between the assembly path the robot has already traversed during an RL episode, and the reversed disassembly path p_i' under consideration. However, like most simulators, ours has a fixed Δt; thus, for a given path, the arc length between consecutive points is a function of the instantaneous velocity. In general, the path the robot has already traversed has a disparate sequence of velocities compared to any reversed path p_i', resulting in disparate spatial discretizations. We thus seek a distance metric between paths that is insensitive to speed or sampling rate.

We explore 2 powerful methods for computing such a metric, dynamic time warping (**DTW**) and **signature transforms**. DTW is a dynamic programming algorithm for quantifying the difference between time series [71]. Given two sequences $a = [a^1, a^2, ..., a^P]$ and $b = [b^1, b^2, ..., b^Q]$, DTW matches each a^i to one or more b^j and vice versa. The matching process minimizes a cost $C(a, b)$ defined as the sum of Euclidean distances between each a^i and its match(es) from b; furthermore, the process satisfies constraints that 1) a^1 must match with at least b^1 (i.e., first points aligned), 2) a^P must match with at least b^Q (i.e., last points aligned), and 3) all matches must be monotonic (i.e., if a^i matches with b^j, then a^{i+1} cannot match with b^{j-1}, nor a^{i-1} with b^{j+1}). Ultimately, DTW returns the cost $C^*(a, b)$ of the optimal matches between a and b; for intuition and pseudocode, see **Appendix F**.

When we apply DTW, at each timestep t, we first extract the path $p_e(t, N)$ of the end effector over a window of length N=10 steps, $[p_e^{t-(N-1)}, p_e^{t-(N-2)}, ..., p_e^t]$. Then, for each reversed disassembly path p_i', we find the closest subsequent

[4]We set ω^B and ω^I simply so that the baseline and imitation terms fall within the same order of magnitude.

points on p'_i from the first point $p_e^{t-(N-1)}$ and current point p_e^t on the windowed end-effector path, and we extract the segment of p'_i between those 2 closest points. We then use DTW to compute the minimum cost $C^*(p_e(t,N), p'_i)$ between the windowed end-effector path and the disassembly segment, and we set $R_t^{I_i} = 1 - \tanh C^*(p_e(t,N), p'_i)$. We repeat this procedure for each p'_i [16] and then compute R_t^I (**Equation 2**).

On the other hand, **signature transforms** represent trajectories as a collection of path integrals called a path signature [10, 44, 34, 3], which can also quantify distances between paths. In our context, given a 3-dimensional path $p(t)_{a,b} = (x_1(t), x_2(t), x_3(t))_{a,b}$, where $x_1(t)$, $x_2(t)$, and $x_3(t)$ represent x, y, and z coordinates for $t \in [a,b]$, the path signature is a tensor of all possible path integrals between the coordinates. The *first level* of the path signature is

$$S_1(x_i(t))_{a,t} = \int_a^t dx_i(t) = x_i(t) - x_i(a) \tag{3}$$

where $i = 1, 2, 3$ (i.e., 3 total integrals), and the *second level* of the path signature is

$$S_2(x_i(t), x_j(t))_{a,t} = \int_a^t S_1(x_i(t))_{a,t} dx_j(t) \tag{4}$$

where $i = 1, 2, 3$ and $j = 1, 2, 3$ (i.e., 9 total path integrals). Further levels of the path signature can be derived in similar fashion. Finally, the full path signature is

$$S(p(t))_{a,b} = (1, S_1(x_i(t)_{a,b}), S_2(x_i(t), x_j(t))_{a,b}, ...) \tag{5}$$

where all indices iterate over $1, 2, 3$.[5] The signature transform is simply the functional $T(p(t))_{a,b} : p(t)_{a,b} \rightarrow S((p(t))_{a,b}$ that takes a path as input and outputs the path signature. Path signatures inherit translation and reparameterization invariance from path integrals; as desired, these properties mitigate discretization sensitivity. For details, see **Appendix G**.

When we apply the signature transform, at each timestep t, we consider the full path $p_e(T)_{0,t}$ of the end effector from the beginning of the episode. Then, for each reversed disassembly path p'_i, we find the closest point on p'_i from the current point $p_e(t)$ on the end-effector path and extract the segment of p'_i between the start and the closest point. We then compute the path signatures $S(p_e(T))_{0,t}$ and $S(p'_i)$ of the end-effector path and disassembly path segment [33], respectively, and compute the cost $C(S(p_e(T))_{0,t}, S(p'_i))$ between signatures as

$$C\big(S(p_e(T))_{0,t}, S(p'_i)\big) = \|S(p_e(T))_{0,t} - S(p'_i)\|_2. \tag{6}$$

Finally, we set $R_t^{I_i} = 1 - \tanh C\big(S(p_e(T))_{0,t}, S(p'_i)\big)$.

We have thus described our specialist learning methods, which consist of assembly-by-disassembly, RL with an imitation objective, and trajectory matching. To see evaluations and results, skip to **Section VI**. Next, we describe our generalist learning methods, which consist of geometric encoding, policy distillation, and curriculum-based RL fine-tuning.

[5]As our data consists of discrete time series, we technically use the discrete-time form of path signatures (**Appendix G**).

D. Generalist Learning: Geometric Encoding, Policy Distillation, and Curriculum-based RL Fine-tuning

The first challenge when training a generalist is to represent assembly geometry to the generalist network. For specialist policies, the policy/value networks do not take geometry as input, as it is constant; however, for a generalist policy, the networks must take geometry as input, as assembling a wide range of parts without such knowledge would be infeasible.

Inspired by [86], we learn a latent representation of object geometry prior to learning a generalist policy. Specifically, we sample point clouds from the surfaces of all plug and socket meshes, and we train a PointNet autoencoder [63] over the point clouds to minimize reconstruction loss; for details, see **Appendix H**. We then pass the point clouds into the encoder and store the latent vectors z_i. During generalist learning, for a given assembly, we simply concatenate z_p and z_s for the corresponding plug and socket meshes as input to the policy.

The second challenge when training a generalist is to distill knowledge from the specialist policies into a generalist policy. As shown in **Section VI-B**, it is ineffective to train a generalist policy from scratch over a large number of assemblies, and we aim to reuse knowledge from already-trained specialists. Thus, we implement a simple 2-stage policy distillation procedure:

1) **Behavior Cloning (BC)** [62]: We use standard BC on the specialists. Specifically, we execute each specialist policy π_s under initial-pose randomization (**Table II**) and observation noise (**Table V**) until completing 5000 successful episodes. For each success, we record the state-action pairs as a demonstration $D_i = \{(s_i^1, a_i^1), (s_i^2, a_i^2), ..., (s_i^{N_i}, a_i^{N_i})\}$. We randomly initialize a generalist policy π_g and minimize MSE loss $\mathcal{L} = \frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{N_i}(a_i^j - \pi_g(s_i^j))^2$ between ground-truth actions in the demonstrations and actions predicted by the generalist (batch size M=128, epochs=1000). The output is an initial generalist policy π_g.
2) **DAgger** [68]: We use DAgger to reduce covariate shifts by executing the generalist policy and querying the specialists under the induced state distributions. Specifically, we execute the current π_g for 256 successful or unsuccessful episodes and record the state-action pairs. We minimize MSE loss $\mathcal{L} = \frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{N_i}(\pi_g(s_i^j) - \pi_s(s_i^j))^2$ between the actions taken by the generalist and the specialist actions queried at those same states. The output is a refined generalist policy π'_g.

The third challenge when training a generalist is to further improve performance. Although BC and DAgger can produce reasonable policies, BC is limited by dataset size and diversity, and DAgger is limited by specialist performance on states visited by the generalist. In contrast to the iterative, highly-complex approach of [86], we perform a single RL fine-tuning phase on the generalist that follows the same baseline RL-only approach we evaluate when learning specialists. As we show later (**Section VI**), the curriculum is particularly critical.

Specifically, we initialize our actor with π'_g and follow the baseline RL procedure outlined in **Appendix E**. We use a

sampling-based curriculum (SBC), where we expose the agent to the full range of initial-pose randomization at the start of the curriculum, but increase the lower bound at each stage; this curriculum outperforms naive implementations for assembly [76]. More precisely, at each curriculum stage $k = 1, ..., K$, the initial plug height $h_k^{init} \sim U[h_k^{min}, h^{max}]$, where $h_k^{min} < h_{max}$, $h_1^{init} \leq h_k^{init} \leq h_K^{init}$, and h^{max} remains constant.

VI. Specialist and Generalist Policies

Our third contribution is a demonstration that the prior learning methods enable high-performance specialist and generalist policies in simulation. We now present detailed evaluations of our specialist and generalist policies.

As a preliminary step to help with exposition, we take the latent vectors of assembly geometry from **Section V-D** and use the t-SNE algorithm [30] to reduce the dimensionality of the data to 2D (**Figure 5**). We then sample 10 assemblies that are well distributed across the resulting clusters. Although we perform simulation-based evaluations across all 100 assets, we will frequently discuss these 10 assemblies in further detail.

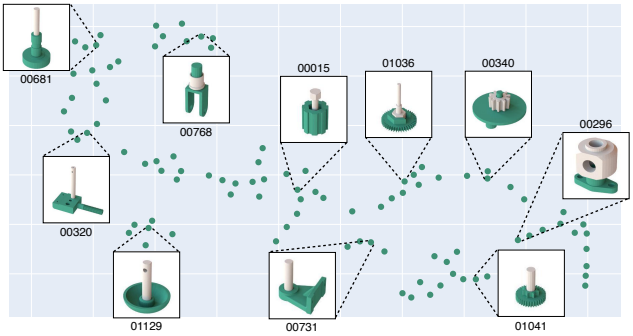

Fig. 5. **t-SNE visualization of geometric representations of 100 assemblies.** We train a PointNet-based autoencoder to learn a latent representation of assembly geometry, and we use t-SNE (with perplexity = 6) to reduce the dimensionality of the latent vectors to 2D. Here we plot the lower-dimensional representations of all 100 assemblies. For visualization, we sample 10 assets that are well distributed across clusters. We also show examples of multiple assemblies sampled from the same cluster in **Figure S15**.

A. Evaluations of Specialist Policies

We now present the results of our specialist policies, which are trained based on our combination of assembly-by-disassembly, RL with an imitation objective, and trajectory matching via dynamic time warping and signature transforms. We perform all evaluations in this section under the maximum bounds for initial-pose randomization (**Table II**) and observation noise (**Table V**) experienced during training.

For specialist policies, our first evaluation question is, **which trajectory-matching approach is most effective for AutoMate?** We evaluate the following 4 test cases:

- **IndustReal** [76]: This is a state-of-the-art RL-only approach for simulation-based robotic assembly, described in **Section V-B**. It does not use an imitation objective and thus illustrates results without trajectory matching.
- **State-based Matching**: This is a naive baseline for RL with an imitation objective. At each timestep, we calculate the distance from the end effector to every point of every disassembly path, and we compute our imitation reward based on the shortest distance.
- **Dynamic Time Warping (Ours)**: We compute our imitation reward based on the DTW distance (**Section V-C**).
- **Signature Transform (Ours)**: We compute our imitation reward based on a path-signature distance (**Section V-C**).

For each of the 4 test cases, for each of the 100 assemblies, we train a specialist policy over 5 random seeds. We select the best seed and evaluate it over 5000 trials, for a total of 2M simulated trials. **Figure 6** shows our results.

IndustReal has the lowest success rates over the 10 selected assemblies, indicating the importance of imitation. State-based matching provides a substantial improvement, but still does not result in high success rates. Dynamic time warping and signature transforms consistently have the highest success rates, with slightly better performance for the former. These results are also reflected over all 100 assemblies, with mean success rates of 38.45 $\pm$ 32.16% for IndustReal, 59.11 $\pm$ 19.65% for state-based matching, 81.50 $\pm$ 24.42% for DTW, and 77.46 $\pm$ 22.61% for signature transforms.

Our next evaluation question is, **does AutoMate perform better than naive and state-of-the-art baselines?** We evaluate the following 5 test cases:

- **Go to Goal**: This is a naive baseline for control. We use a task-space impedance controller to move the end effector directly to the goal.
- **Top Down**: This is a naive baseline for control. We use a task-space impedance controller to move the end effector directly above the goal and straight downward.
- **Follow Trajectory**: This is a naive baseline for imitation learning. We select the demonstration containing the closest point to the initial end-effector position, and we use a task-space impedance controller to move the end effector along that fixed path.
- **IndustReal**: This is the state-of-the-art RL-only approach for simulation-based robotic assembly described earlier.
- **AutoMate (Ours)**: We use a combination of assembly-by-disassembly, RL with an imitation objective, and trajectory matching with dynamic time warping.

For each of the 5 test cases, for each of the 100 assemblies, we train a specialist policy over 5 seeds. We select the best seed and evaluate it over 5000 trials, for a total of 2.5M simulated trials. **Figure 7** shows our results.

Go to Goal, Top Down, and Follow Trajectory have the lowest success rates over the 10 assemblies, as they lack robustness to controller error, observation noise, and disturbances during contact. IndustReal notably outperforms the 3 naive baselines on several assemblies, demonstrating the benefit of an RL-based approach. However, AutoMate consistently has the highest success rates. Again, the preceding results are reflected over all 100 assemblies, with mean success rates of 15.44 $\pm$ 11.56% for Go To Goal, 21.54 $\pm$ 15.49% for Top

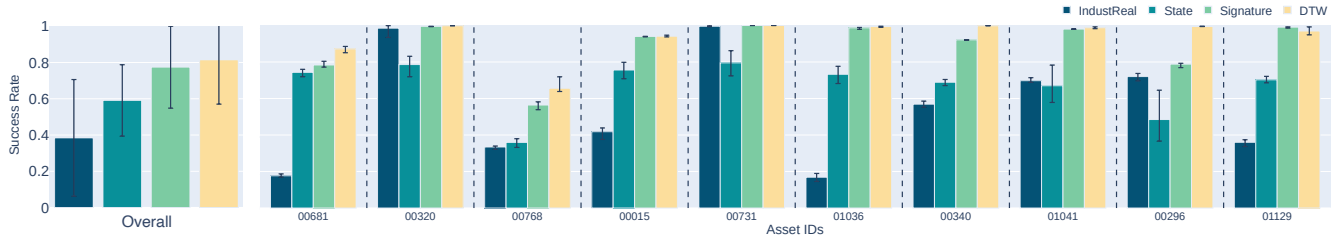

Fig. 6. **Simulation-based evaluation of trajectory-matching approaches for learning specialist policies.** For each of the 100 assemblies, we train a specialist policy with 4 different approaches for matching the current robot path with demonstrations. For each approach, we train 5 random seeds, select the best seed, and evaluate it 5 times over 1000 trials. We illustrate average results over all 100 assemblies, as well as specific results for 10 sampled assemblies (**Figure 5**). *IndustReal* is a state-of-the-art matching-free approach. *State* selects the demonstration containing the closest point to the current robot state. *Signature* selects the demonstration with the minimum signature-transform distance from the robot trajectory. *DTW* selects the demonstration with the minimum dynamic-time-warping distance from the robot trajectory. The *Signature* and *DTW* approaches significantly outperform the others.

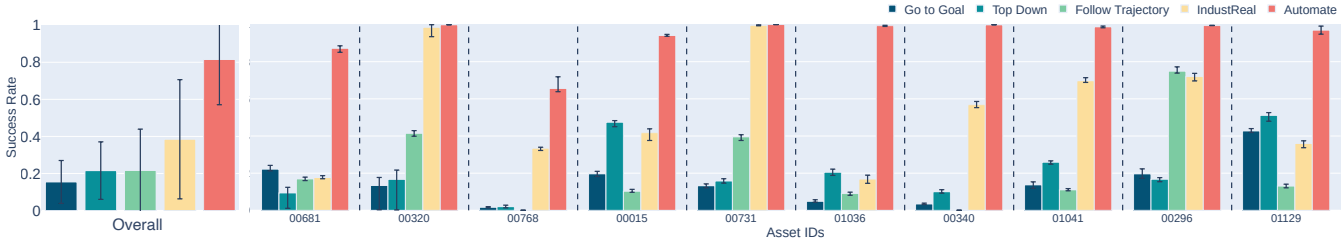

Fig. 7. **Simulation-based evaluation of control schemes for specialist policies.** For each of the 100 assemblies, we build a specialist policy with 5 different control approaches. For each approach, we train 5 random seeds, select the best seed, and evaluate it 5 times over 1000 trials. We illustrate average results over all 100 assemblies, as well as specific results for 10 sampled assemblies (**Figure 5**). *Go to Goal* uses a task-space impedance controller to move directly to the goal. *Top Down* uses the controller to move directly above the goal and then downward. *Follow Trajectory* selects the demonstration containing the point closest to the initial robot state and uses the controller to follow the demonstration path. *IndustReal* is a state-of-the-art, RL-only baseline. *AutoMate* is our proposed strategy combining assembly-by-disassembly, RL with imitation, and dynamic time warping. *Automate* significantly outperforms the others.

Down, $21.62 \pm 22.32\%$ for Follow Trajectory, $38.45 \pm 32.16\%$ for IndustReal, and $81.50 \pm 24.42\%$ for AutoMate.

Our third evaluation question is, **how does AutoMate perform across all 100 assemblies?** Specifically, rather than selecting 10 assemblies or summarizing statistics, we tabulate results over all 100 assemblies. **Figure S16** shows our results. We provide our raw data as supplementary information.

AutoMate shows consistent performance over a wide spectrum of the assemblies. Specifically, for 80 assemblies, AutoMate has $\approx$80% (78%) success rates or higher, and for 55, it has 90% success rates or higher. We conclude that AutoMate is a highly-effective strategy for training specialists over diverse geometries. The most challenging assemblies (i.e., bottom 20%) tend to have small-diameter plugs, sockets with small contact surfaces, or sockets with stair-like internal features.

B. Evaluation of Generalist Policy

We now present the results of our generalist policies, which are trained based on our combination of behavior cloning, policy distillation, and curriculum-based RL fine-tuning. As with the specialists, we perform all evaluations under the maximum bounds for initial-pose randomization (**Table II**) and observation noise (**Table V**) experienced during training.

Our primary evaluation question is, **which policy distillation approach is most effective for AutoMate?** We evaluate the following 4 test cases:

- **Behavior Cloning (BC):** We use standard BC to distill the specialist policies to a generalist policy π_g, as described in **Section V-D**.
- **BC + DAgger:** We first use standard BC and then use DAgger to produce a refined generalist policy π_g'.
- **BC + DAgger + RL fine-tuning:** We first use standard BC, then use DAgger, and finally use RL fine-tuning to produce a generalist policy π_g''.
- **BC + DAgger + RL fine-tuning with SBC (Ours):** We do the above, but also apply a sampling-based curriculum (SBC) proposed in [76], where the lower bound of initial-pose randomization increases at each stage, but the upper bound remains fixed (**Appendix E**).

For each of the 4 test cases, we train a generalist policy over 20 assemblies; these include the 10 assemblies sampled from t-SNE clusters (**Figure 5**), as well as 10 additional assemblies evenly sampled in t-SNE space. We evaluate the performance of each test case over 5000 trials per assembly, for a total of 400k trials. **Figure 8** shows our results.

As expected, BC has the lowest success rate over the 20 assemblies (28.84 $\pm$15.23%), likely due to covariate shift; however, BC + DAgger provides marginal improvement (31.06 $\pm$ 15.06%). In contrast, BC + DAgger + RL provides substantial improvement (52.85 $\pm$ 14.01%), including on assemblies where the prior techniques failed, demonstrating the value of RL-based fine-tuning. Nevertheless, BC + DAgger + RL with SBC has the highest success rate by far (80.42 $\pm$ 20.93%) and improves performance on almost every assembly, underscoring

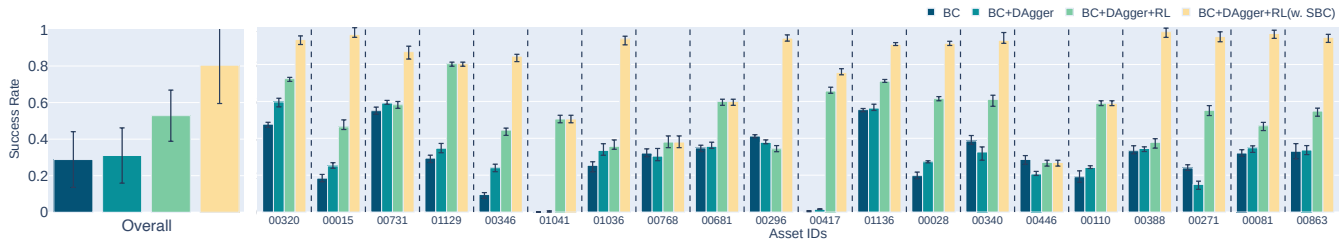

Fig. 8. **Simulation-based evaluation of training approaches for generalist policies.** We train a single generalist policy for 20 assemblies using 4 different approaches. For each approach, we train 5 random seeds, select the best seed, and evaluate it 5 times over 1000 trials. We illustrate average results over the 20 assemblies, as well as specific results for each assembly (asset ID lookup in **Figure S19**). *BC* uses behavior cloning to distill the corresponding specialist policies for the 20 assemblies into a generalist policy. *BC + DAgger* follows BC with DAgger iterations to reduce covariate shift. *BC + DAgger + RL* follows *BC + DAgger* with an RL-based fine-tuning stage. *BC + DAgger + RL (w/SBC)* uses a sampling-based curriculum (SBC) for initial-pose randomization [76], which provided a critical boost in performance; *BC + DAgger + RL (w/SBC)* significantly outperforms the others.

curriculum learning during fine-tuning. As a final sanity check, we train an RL policy with SBC from scratch over the 20 assemblies and measure a low success rate (48.43 $\pm$ 15.28%).

As a secondary evaluation question, we also ask, what is the scaling law between generalist performance and the number of specialists used in training? For details, see **Appendix J**.

VII. Sim-to-Real Transfer

Our final contribution is sim-to-real transfer of our specialist and generalist policies. We first describe our real-world system design and then present real-world evaluations and demonstrations of our specialist and generalist policies. We strongly encourage readers to view our supplementary video.

A. Real-World System Design

As first described in **Section III**, our real-world system consists of a Franka robot with a parallel-jaw gripper, a wrist-mounted RealSense D435 camera, a Schunk EGK40 parallel-jaw gripper mounted to the tabletop, and 3D-printed assemblies from our dataset (**Figure S11**). Our communications framework is closely modeled after [76]; however, our perception, grasping, and control procedures differ significantly.

For perception, we aim to accurately estimate plug and socket states while initializing them in a far less-constrained manner. We use a powerful segmentation tool [37], textureless CAD models of our parts, and a state-of-the-art pose estimator [88] to estimate the 6-DOF poses of each part from RGB-D images. **Figure S18** shows our pipeline; for details, see **Appendix K**. For grasping, we aim to avoid manually specifying grasp poses. We use our grasp sampling and evaluation procedure described (**Section C**) to generate an optimal grasp for each assembly, and we execute those grasps in the real world. Finally, we aim to avoid any assembly-specific tuning of controller gains or action scales, and we restrict ourselves to a single set of parameters for all real-world trials.

B. Real-World Policy Evaluations

We now present the results of our specialist and generalist policies in the real world. For these trials, we place the robot in lead-through, manually grasp a plug, and guide it into the socket. We then programmatically lift the plug until free from contact; apply an xy perturbation of $\pm$ 10 mm, z perturbation of 15 $\pm$ 5 mm, and yaw perturbation of $\pm$ 5°; apply x, y, and z observation noise of $\pm$2 mm each; and deploy a policy.[6]

Our first evaluation question is, **do our specialist policies transfer to the real world?** For each of 20 assemblies, we deploy the corresponding specialist policy 10 times, for a total of 200 trials. **Figure 9** (left) shows our results.

The mean success rate in the real world is 86.50 $\pm$ 16.52%, whereas the success rate in simulation is 90.65 $\pm$ 13.07%. Across assemblies, real-world success rates are within close range of simulation; in fact, for 11/20 assemblies, real-world is better. Such results likely indicate that our simulated training conditions (e.g., initial-pose randomization, observation noise) are sufficiently adverse to train robust and performant policies.

Our second evaluation question is, **do our generalist policies transfer to the real world?** For each of the same 20 assemblies, we deploy the generalist policy (trained from specialists for those assemblies) 10 times, for a total of 200 trials. **Figure 9** (right) shows our results.

The mean success rate in the real world is 84.50 $\pm$ 21.79%, whereas the success rate in simulation is 80.42 $\pm$ 20.93%. For 16/20 assemblies, real-world success rates are higher. Moreover, real-world success is within 2.0% of that of the specialists. Such results again indicate that our simulated training conditions are sufficiently adverse, and more importantly, that our distillation approach is highly effective. Qualitatively, the generalist exhibits smoother motion than the specialists with identical gains, suggesting an averaging effect during training.

C. Real-World End-to-End Evaluation

Finally, we present the results of our specialist and generalist policies as part of an end-to-end assembly workflow. For these trials, we place the plug haphazardly on a foam block and place the socket haphazardly within the Schunk gripper. We capture an RGB-D image, estimate the poses of the parts, grasp the plug, transport it to the socket, and deploy a specialist or generalist assembly policy (**Figure 10**). This workflow presents a unique challenge, as initial part poses are even less constrained, and perception and/or control error accumulates at each stage, demanding increased policy robustness.

[6]The manual grasping step is uncontrolled and likely contributes an additional 1-3 mm and 5-10 deg of perturbation.

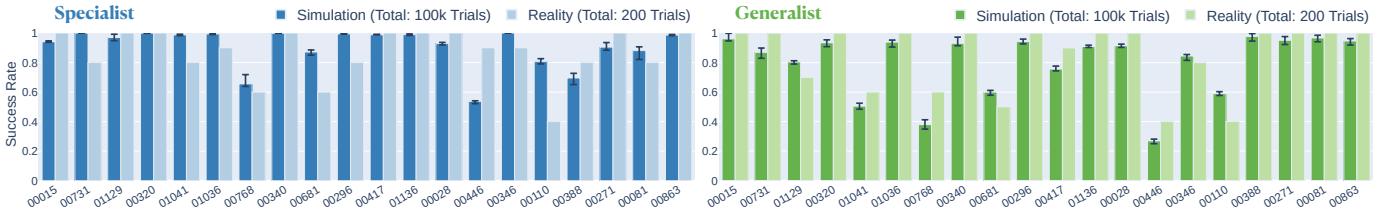

Fig. 9. **Comparison of real-world specialist-policy and generalist-policy success rates with simulated analogues.** We deploy our specialist policies over 20 assemblies and 200 trials (asset ID lookup in **Figure S19**), and we deploy our generalist policy over the same conditions. We compare the results to simulated analogues. Left: For specialists, our success rates in the real world are highly comparable to those in simulation, with a drop of only 4.15% on average. Right: For the generalist, our success rates in the real world are again highly comparable to simulation, with an *improvement* of 4.08% on average. Now comparing our real-world specialists to our real-world generalist, we note that generalist performance is within 2.0% of the specialists on average.

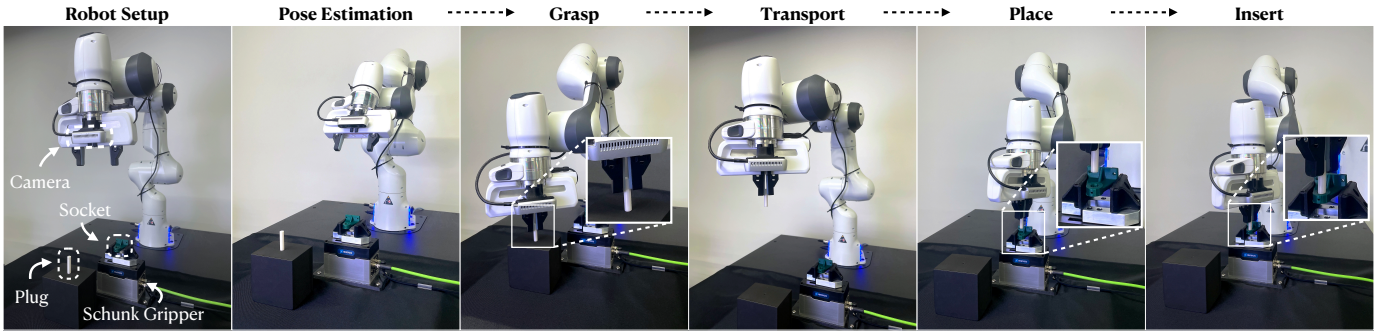

Fig. 10. **Real-world end-to-end assembly procedure.** We illustrate our procedure for performing end-to-end assembly (i.e., from perception to insertion). Setup: We press a plug haphazardly onto a foam block, and we place a socket haphazardly within a Schunk gripper on a table. Pose Estimation: We estimate the pose of the plug or socket using our perception pipeline (**Section K**). Grasp: We grasp the plug using the output of our grasp optimizer (**Section C**). Transport: We transport the plug across the workspace. Place: We place the plug above the socket. Insert: We deploy a specialist or generalist policy (**Section V**).

Our question is, **can our specialist and generalist policies help solve the end-to-end assembly task?** For 5 distinct assemblies from **Figure 3**, we deploy the corresponding specialists 10 times, for a total of 100 trials. We repeat the procedure for the generalist policy. **Table I** shows our results.

Asset ID	Specialist		Generalist	
	Policy-Only	End-to-End	Policy-Only	End-to-End
00015	10/10	10/10	10/10	10/10
00296	8/10	8/10	10/10	9/10
00320	10/10	10/10	10/10	8/10
00340	10/10	9/10	10/10	8/10
00346	9/10	8/10	8/10	8/10
Total #	47/50	45/50	48/50	43/50
Total (%)	**94.0%**	**90.0%**	**96.0%**	**86.0%**

TABLE I

REAL-WORLD EVALUATIONS FOR END-TO-END ASSEMBLY. WE PROVIDE SUCCESS RATES OF OUR SPECIALIST AND GENERALIST POLICIES AS PART OF AN END-TO-END ASSEMBLY WORKFLOW. WE ALSO COMPARE TO ISOLATED POLICY EVALUATIONS FROM **SECTION VII-B**.

For specialists, the mean success rate is 90.0%, which is within 4.0% of isolated policy evaluations from **Section VII-B**, and for generalists, it is 86.0%, which is within 10.0%. These results indicate that 6-DOF pose estimation, grasp optimization, and our proposed methods for learning specialist and generalist policies can be effectively combined to achieve end-to-end assembly. Qualitatively, higher success rates occur on assemblies with distinct visual features, as these correlate with more accurate pose estimates.

VIII. CONCLUSIONS

We present **AutoMate**, a learning framework and system for solving diverse assembly problems with specialist and generalist policies. To our knowledge, AutoMate provides the first simulation-based framework for learning specialist and generalist policies over a wide range of assemblies, as well as the first system to demonstrate zero-shot sim-to-real over such a range. We evaluate our framework and system over 5M+ simulated trials and 500 real-world trials.

Our work opens up exciting opportunities for future work. First, we currently solve 2-part assemblies, which do not require sequence planning. In future work, we will develop an accelerated sequence planner for multi-part assemblies.

Second, our trajectory-matching approaches focus on paths in $\mathbb{R}^3$. We will generalize our trajectory-matching approaches (i.e., DTW and signature transforms) to paths consisting of $SE(3)$ transforms, facilitating assembly of parts requiring significant reorientation during alignment and insertion.

Third, we train specialist policies on a wide range of assemblies and distill a generalist policy from 20 specialists. We will continue to distill generalists from more specialists using powerful model architectures and larger model capacities, while consistently evaluating our results in the real world.

Through this line of work, we aim to gradually build towards a large-model paradigm for industrial robotics, while staying grounded in real-world deployment.

References

[1] Iretiayo Akinola, Jie Xu, Jan Carius, Dieter Fox, and Yashraj Narang. TacSL: A library for visuotactile sensor simulation and learning. 2023.

[2] Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, Jonas Schneider, Nikolas Tezak, Jerry Tworek, Peter Welinder, Lilian Weng, Qiming Yuan, Wojciech Zaremba, and Lei Zhang. Solving Rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

[3] Lucas Barcelos, Tin Lai, Rafael Oliveira, Paulo Borges, and Fabio Ramos. Path signatures for diversity in probabilistic trajectory optimisation. *arXiv preprint arXiv:2308.04071*, 2023.

[4] Maria Bauza, Antonia Bronars, and Alberto Rodriguez. Tac2Pose: Tactile object pose estimation from the first touch. *The International Journal of Robotics Research*, 2023.

[5] Richard Bellman and Robert Kalaba. On adaptive control processes. *IRE Transactions on Automatic Control*, 1959.

[6] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, et al. RT-1: Robotics transformer for real-world control at scale. *arXiv preprint arXiv:2212.06817*, 2022.

[7] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, et al. RT-2: Vision-language-action models transfer web knowledge to robotic control. *arXiv preprint arXiv:23 07.15818*, 2023.

[8] Cristian Buciluǎ, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2006.

[9] Dian Chen, Brady Zhou, Vladlen Koltun, and Philipp Krähenbühl. Learning by cheating. In *Conference on Robot Learning*, 2020.

[10] Kuo-Tsai Chen. Integration of paths–A faithful representation of paths by noncommutative formal power series. *Transactions of the American Mathematical Society*, 1958.

[11] Tao Chen, Jie Xu, and Pulkit Agrawal. A system for general in-hand object re-orientation. In *Conference on Robot Learning*, 2022.

[12] Ilya Chevyrev and Andrey Kormilitzin. A primer on the signature method in machine learning. *arXiv preprint arXiv:1603.03788*, 2016.

[13] Gene Chou, Ilya Chugunov, and Felix Heide. GenSDF: Two-stage learning of generalizable signed distance functions. *Advances in Neural Information Processing Systems*, 2022.

[14] Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *Neural Information Processing Systems*, 2017.

[15] Jack Collins, Mark Robson, Jun Yamada, Mohan Sridharan, Karol Janik, and Ingmar Posner. RAMP: A benchmark for evaluating robotic assembly manipulation and planning. *arXiv preprint arXiv:2305.09644*, 2023.

[16] Marco Cuturi and Mathieu Blondel. Soft-DTW: A differentiable loss function for time-series. In *International Conference on Machine Learning*, 2017.

[17] Michael Dawson-Haggerty. Trimesh, 2019.

[18] LS Homem De Mello and Arthur C Sanderson. A correct and complete algorithm for the generation of mechanical assembly sequences. In *IEEE International Conference on Robotics and Automation*, 1989.

[19] Clemens Eppner, Arsalan Mousavian, and Dieter Fox. ACRONYM: A large-scale grasp dataset based on simulation. In *IEEE International Conference on Robotics and Automation*, 2021.

[20] Peter Foster. A brief introduction to path signatures, 2020.

[21] Bowen Fu, Sek Kun Leong, Xiaocong Lian, and Xiangyang Ji. 6D robotic assembly based on RGB-only object pose estimation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2022.

[22] Bowen Fu, Sek Kun Leong, Yan Di, Jiwen Tang, and Xiangyang Ji. LanPose: Language-instructed 6D object pose estimation for robotic assembly. *arXiv preprint arXiv:2310.13819*, 2023.

[23] Dibya Ghosh, Avi Singh, Aravind Rajeswaran, Vikash Kumar, and Sergey Levine. Divide-and-conquer reinforcement learning. *arXiv preprint arXiv:1711.09874*, 2017.

[24] Anders Grunnet-Jepsen and Dave Tong. Depth post-processing for Intel RealSense depth camera D400 series, 2023.

[25] Anders Grunnet-Jepsen, John Sweetser, Tri Khuong, Sergey Dorodnicov, Dave Tong, Ofir Mulla, Hila Eliyahu, and Evgeni Raikhel. Intel RealSense self-calibration for D400 series depth cameras, 2023.

[26] Anders Grunnet-Jepsen, John N. Sweetser, and John Woodfill. Tuning depth cameras for best performance, 2023.

[27] Ankur Handa, Arthur Allshire, Viktor Makoviychuk, Aleksei Petrenko, Ritvik Singh, Jingzhou Liu, Denys Makoviichuk, Karl Van Wyk, Alexander Zhurkevich, Balakumar Sundaralingam, Yashraj Narang, Jean-Francois Lafleche, Dieter Fox, and Gavriel State. DeXtreme: Transfer of agile in-hand manipulation from simulation to reality. In *International Conference on Robotics and Automation*, 2023.

[28] Minho Heo, Youngwoon Lee, Doohyun Lee, and Joseph J Lim. FurnitureBench: Reproducible real-world benchmark for long-horizon complex manipulation. In *Robotics: Science and Systems*, 2023.

[29] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *Neural Information*

Processing Systems, Deep Learning Workshop, 2015.

[30] Geoffrey E Hinton and Sam Roweis. Stochastic neighbor embedding. In *Neural Information Processing Systems*, 2002.

[31] Rachel Holladay, Tomás Lozano-Pérez, and Alberto Rodriguez. Robust planning for multi-stage forceful manipulation. *International Journal of Robotics Research*, 2022.

[32] Zhiwei Jia, Xuanlin Li, Zhan Ling, Shuang Liu, Yiran Wu, and Hao Su. Improving policy optimization with generalist-specialist learning. In *International Conference on Machine Learning*, 2022.

[33] Patrick Kidger and Terry Lyons. Signatory: differentiable computations of the signature and logsignature transforms, on both CPU and GPU. In *International Conference on Learning Representations*, 2021.

[34] Patrick Kidger, Patric Bonnier, Imanol Perez Arribas, Cristopher Salvi, and Terry Lyons. Deep signature transforms. *Neural Information Processing Systems*, 2019.

[35] Kenneth Kimble, Karl Van Wyk, Joe Falco, Elena Messina, Yu Sun, Mizuho Shibata, Wataru Uemura, and Yasuyoshi Yokokohji. Benchmarking protocols for evaluating small parts robotic assembly systems. *IEEE Robotics and Automation Letters*, 2020.

[36] Kenneth Kimble, Justin Albrecht, Megan Zimmerman, and Joe Falco. Performance measures to benchmark the grasping, manipulation, and assembly of deformable objects typical to manufacturing applications. *Frontiers in Robotics and AI*, 2022.

[37] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023.

[38] Yann Labbé, Justin Carpentier, Mathieu Aubry, and Josef Sivic. CosyPose: Consistent multi-view multi-object 6D pose estimation. In *European Conference on Computer Vision*, 2020.

[39] Yann Labbé, Lucas Manuelli, Arsalan Mousavian, Stephen Tyree, Stan Birchfield, Jonathan Tremblay, Justin Carpentier, Mathieu Aubry, Dieter Fox, and Josef Sivic. MegaPose: 6D pose estimation of novel objects via render & compare. In *Conference on Robot Learning*, 2022.

[40] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 2020.

[41] Youngwoon Lee, Edward S Hu, and Joseph J Lim. IKEA furniture assembly environment for long-horizon complex manipulation tasks. In *IEEE International Conference on Robotics and Automation*, 2021.

[42] Wayve Technologies Ltd. Robot car talk: Introducing Wayve's new AI model LINGO-1, 2023.

[43] Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press, 2017.

[44] Terry J Lyons. Differential equations driven by rough signals. *Revista Matemática Iberoamericana*, 1998.

[45] Miles Macklin. Warp: A high-performance Python framework for GPU simulation and graphics, 2022.

[46] Miles Macklin, Kenny Erleben, Matthias Müller, Nuttapong Chentanez, Stefan Jeschke, and Zach Corse. Local optimization for robust signed distance field collision. *ACM Computer Graphics and Interactive Techniques*, 2020.

[47] Denys Makoviichuk and Viktor Makoviychuk. rl-games: A high-performance framework for reinforcement learning, 2022.

[48] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, and Gavriel State. Isaac Gym: High performance GPU-based physics simulation for robot learning. In *Neural Information Processing Systems, Datasets and Benchmarks Track*, 2021.

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

[50] Andrew S Morgan, Bowen Wen, Junchi Liang, Abdeslam Boularias, Aaron M Dollar, and Kostas Bekris. Vision-driven compliant manipulation for reliable, high-precision assembly tasks. In *Robotics: Science and Systems*, 2021.

[51] Tongzhou Mu, Zhan Ling, Fanbo Xiang, Derek Cathera Yang, Xuanlin Li, Stone Tao, Zhiao Huang, Zhiwei Jia, and Hao Su. ManiSkill: Generalizable manipulation skill benchmark with large-scale demonstrations. In *Neural Information Processing Systems, Datasets and Benchmarks Track*, 2021.

[52] Yashraj Narang, Balakumar Sundaralingam, Miles Macklin, Arsalan Mousavian, and Dieter Fox. Sim-to-real for robotic tactile sensing via physics-based simulation and learned latent projections. In *IEEE International Conference on Robotics and Automation*, 2021.

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

[54] Yashraj Narang, Kier Storey, Iretiayo Akinola, Miles Macklin, Philipp Reist, Lukasz Wawrzyniak, Yunrong Guo, Adam Moravanszky, Gavriel State, Michelle Lu, et al. Factory documentation, 2022.

[55] Andrew Y Ng and Stuart Russell. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning*, 2000.

[56] NVIDIA. NVIDIA Isaac Sim, 2024.

[57] National Institute of Standards and Technology. NIST Manufacturing Objects and Assemblies Dataset

(MOAD), 2023.

[58] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel Van de Panne. DeepMimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions On Graphics*, 2018.

[59] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *IEEE International Conference on Robotics and Automation*, 2018.

[60] Xue Bin Peng, Erwin Coumans, Tingnan Zhang, Tsang-Wei Lee, Jie Tan, and Sergey Levine. Learning agile robotic locomotion skills by imitating animals. *Robotics: Science and Systems*, 2020.

[61] Lerrel Pinto, Marcin Andrychowicz, Peter Welinder, Wojciech Zaremba, and Pieter Abbeel. Asymmetric actor critic for image-based robot learning. *arXiv preprint arXiv:1710.06542*, 2017.

[62] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. *Neural Information Processing Systems*, 1988.

[63] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. PointNet: Deep learning on point sets for 3D classification and segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[64] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *Robotics: Science and Systems*, 2017.

[65] Fabio Ramos, Rafael Carvalhaes Possas, and Dieter Fox. BayesSim: Adaptive domain randomization via probabilistic inference for robotics simulators. In *Robotics: Science and Systems*, 2019.

[66] Anurag Ranjan, Timo Bolkart, Soubhik Sanyal, and Michael J Black. Generating 3D faces using convolutional mesh autoencoders. In *European Conference on Computer Vision*, 2018.

[67] Scott Reed, Konrad Zolna, Emilio Parisotto, Sergio Gomez Colmenarejo, Alexander Novikov, Gabriel Barth-Maron, Mai Gimenez, Yury Sulsky, Jackie Kay, Jost Tobias Springenberg, et al. A generalist agent. *arXiv preprint arXiv:2205.06175*, 2022.

[68] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*, 2011.

[69] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Conference on Robot Learning*, 2021.

[70] Andrei A Rusu, Sergio Gomez Colmenarejo, Caglar Gulcehre, Guillaume Desjardins, James Kirkpatrick, Razvan Pascanu, Volodymyr Mnih, Koray Kavukcuoglu, and Raia Hadsell. Policy distillation. *arXiv preprint arXiv:1511.06295*, 2015.

[71] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1978.

[72] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[73] Oren Spector and Dotan Di Castro. InsertionNet: A scalable solution for insertion. *IEEE Robotics and Automation Letters*, 2021.

[74] Oren Spector, Vladimir Tchuiev, and Dotan Di Castro. InsertionNet 2.0: Minimal contact multi-step insertion using multimodal multiview sensory input. *arXiv preprint arXiv:2203.01153*, 2022.

[75] Zheng Sun, David Hsu, Tingting Jiang, Hanna Kurniawati, and John H Reif. Narrow passage sampling for probabilistic roadmap planning. *IEEE Transactions on Robotics*, 2005.

[76] Bingjie Tang, Michael A Lin, Iretiayo Akinola, Ankur Handa, Gaurav S Sukhatme, Fabio Ramos, Dieter Fox, and Yashraj Narang. IndustReal: Transferring contact-rich assembly tasks from simulation to reality. In *Robotics: Science and Systems*, 2023.

[77] Bingjie Tang, Michael A Lin, Iretiayo Akinola, Ankur Handa, Gaurav S Sukhatme, Fabio Ramos, Dieter Fox, and Yashraj Narang. IndustRealKit, 2023.

[78] Bingjie Tang, Michael A Lin, Iretiayo Akinola, Ankur Handa, Gaurav S Sukhatme, Fabio Ramos, Dieter Fox, and Yashraj Narang. IndustRealLib, 2024.

[79] Bingjie Tang, Michael A Lin, Iretiayo Akinola, Ankur Handa, Gaurav S Sukhatme, Fabio Ramos, Dieter Fox, and Yashraj Narang. IndustRealSim, 2024.

[80] Yunsheng Tian, Jie Xu, Yichen Li, Jieliang Luo, Shinjiro Sueda, Hui Li, Karl DD Willis, and Wojciech Matusik. Assemble Them All: Physics-based planning for generalizable assembly by disassembly. *ACM Transactions on Graphics*, 2022.

[81] Yunsheng Tian, Karl DD Willis, Bassel Al Omari, Jieliang Luo, Pingchuan Ma, Yichen Li, Farhad Javid, Edward Gu, Joshua Jacob, Shinjiro Sueda, et al. ASAP: Automated sequence planning for complex robotic assembly with physical feasibility. *arXiv preprint arXiv:2309.16909*, 2023.

[82] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017.

[83] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012.

[84] Roger Y Tsai and Reimar K Lenz. A new technique for fully autonomous and efficient 3D robotics hand/eye calibration. *IEEE Transactions on Robotics and Automation*, 1989.

[85] Mel Vecerik, Todd Hester, Jonathan Scholz, Fumin Wang, Olivier Pietquin, Bilal Piot, Nicolas Heess, Thomas Rothörl, Thomas Lampe, and Martin Riedmiller. Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*, 2017.

[86] Weikang Wan, Haoran Geng, Yun Liu, Zikang Shan, Yaodong Yang, Li Yi, and He Wang. UniDexGrasp++: Improving dexterous grasping policy learning via geometry-aware curriculum and iterative generalist-specialist learning. In *International Conference on Computer Vision*, 2023.

[87] Bowen Wen, Wenzhao Lian, Kostas Bekris, and Stefan Schaal. You Only Demonstrate Once: Category-level manipulation from single visual demonstration. In *Robotics: Science and Systems*, 2022.

[88] Bowen Wen, Wei Yang, Jan Kautz, and Stan Birchfield. FoundationPose: Unified 6D pose estimation and tracking of novel objects. *arXiv preprint arXiv:2312.08344*, 2023.

[89] Daniel E. Whitney. *Mechanical Assemblies: Their Design, Manufacture, and Role in Product Development*. Oxford University Press, 2004.

[90] Wikipedia. T-symmetry, 2024.

[91] Karl DD Willis, Pradeep Kumar Jayaraman, Hang Chu, Yunsheng Tian, Yifei Li, Daniele Grandi, Aditya Sanghi, Linh Tran, Joseph G Lambourne, Armando Solar-Lezama, et al. JoinABLe: Learning bottom-up assembly of parametric cad joints. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022.

[92] Jie Xu, Tao Chen, Lara Zlokapa, Michael Foshey, Wojciech Matusik, Shinjiro Sueda, and Pulkit Agrawal. An end-to-end differentiable framework for contact-aware robot design. *arXiv preprint arXiv:2107.07501*, 2021.

[93] Yinzhen Xu, Weikang Wan, Jialiang Zhang, Haoran Liu, Zikang Shan, Hao Shen, Ruicheng Wang, Haoran Geng, Yijia Weng, Jiayi Chen, et al. UniDexGrasp: Universal robotic dexterous grasping via learning diverse proposal generation and goal-conditioned policy. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

[94] Chaoning Zhang, Dongshen Han, Yu Qiao, Jung Uk Kim, Sung-Ho Bae, Seungkyu Lee, and Choong Seon Hong. Faster Segment Anything: Towards lightweight SAM for mobile applications. *arXiv preprint arXiv:2306.14289*, 2023.

[95] Xiang Zhang, Masayoshi Tomizuka, and Hui Li. Bridging the sim-to-real gap with dynamic compliance tuning for industrial insertion. *arXiv preprint arXiv:2311.07499*, 2023.

[96] Xiang Zhang, Changhao Wang, Lingfeng Sun, Zheng Wu, Xinghao Zhu, and Masayoshi Tomizuka. Efficient sim-to-real transfer of contact-rich manipulation skills with online admittance residual learning. In *Conference on Robot Learning*, 2023.

[97] Tony Z Zhao, Jianlan Luo, Oleg Sushkov, Rugile Pevceviciute, Nicolas Heess, Jon Scholz, Stefan Schaal, and Sergey Levine. Offline meta-reinforcement learning for industrial insertion. In *International Conference on Robotics and Automation (ICRA)*, 2022.

[98] Xu Zhao, Wenchao Ding, Yongqi An, Yinglong Du, Tao Yu, Min Li, Ming Tang, and Jinqiao Wang. Fast segment anything. *arXiv preprint arXiv:2306.12156*, 2023.

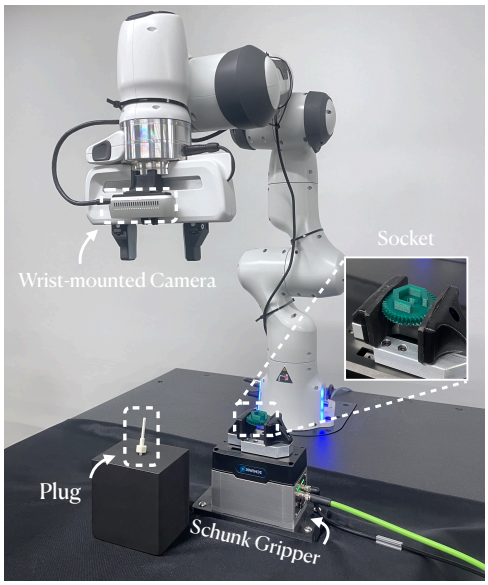

Fig. S11. **Real-world experimental setup.** A Franka Panda robot (with a wrist-mounted Intel RealSense D435 camera) and a Schunk EGK40 gripper are mounted to a tabletop. At the beginning of each experiment, a 3D-printed plug is haphazardly pressed into a foam block or placed in the robot gripper, and a 3D-printed socket is haphazardly placed in the Schunk gripper.

APPENDIX

A. Problem Description: Real-World Experimental Setup

Figure S11 shows our real-world experimental setup.

B. Methods: Mesh Preprocessing

We compile a dataset of assemblies based on Assemble Them All [80], which itself compiles a dataset based on the Fusion 360 Gallery dataset [91]. The authors of [80] preprocess the dataset from [91] to ensure that the meshes are unique, normalized, watertight, and in a fully-assembled initial state, but the majority of the meshes still exhibit a minor degree of interpenetration in this state. Fortunately, the authors of [80] use a simulator with penalty-based contact, which is robust to minor interpenetration [92]; nevertheless, most widely-used robotics simulators (e.g., [48]) enforce non-penetration constraints, which causes initially-interpenetrating parts to exhibit highly unstable dynamics. Furthermore, interpenetrating rigid parts cannot be assembled in the real world. Finally, the meshes from [80] are unitless and have a wide range of relative sizes; for any particular choice of units, many are infeasible to manipulate with widely-used research robots. Thus, we preprocess meshes from [80] such that they can be used in robotics simulators and assembled in reality.

Specifically, we preprocess each mesh as follows:

1) **Scaling**: We choose units of meters, draw an oriented bounding box, and scale the mesh such that the longest edge of the bounding box is 10 cm, allowing it to be grasped by most robotic manipulators used in research.
2) **Reorientation**: We reorient the mesh such that the primary axis of assembly (e.g., the insertion direction) is aligned with the global z-axis.
3) **Translation**: We translate the mesh such that the bottom surface of the mesh is coplanar with the global origin when the mesh is in its assembled state.
4) **Depenetration and Clearance**: If the mesh is a plug, we temporarily instantiate its corresponding socket. For each vertex on the plug, we compute its signed distance to the socket along the vertex normal using Warp [45]; if the distance is negative (corresponding to interpenetration) or less than a desired radial clearance of 0.5 mm, we translate the vertex backward along its normal until achieving the desired clearance. Occasionally, this procedure produces unexpected results, such as when the plug is very thin or the socket is hollow; in such cases, we perform manual corrections in Blender.
5) **Chamfering** (optional): We chamfer the contacting edges of the plug and socket using Blender. Chamfers are extremely common in assemblies, as they facilitate manual assembly, reduce stress concentrations, and remove burrs; for a cylindrical peg, chamfer sizes of $\frac{1}{10}$ to $\frac{1}{4}$ of the diameter are standard. As a typical plug in our dataset has a diameter $\sim$10 mm, we apply chamfers with a length of 1 mm and angle of 45 degrees. We provide chamfered and unchamfered versions of our meshes for use in simulation; in addition, we 3D print the chamfered versions for use in the real world, as our printer produces rough surfaces near curvature discontinuities.
6) **Subdivision**: We subdivide the mesh using Trimesh [17] until producing a minimum of 2000 vertices. During simulation, we use signed-distance-field (SDF)-based collisions [46, 53] as implemented in [53, 48, 56]; since this specific implementation generates a single contact per triangle, subdivision helps to ensure stable contact resolution along flat surfaces [54].

C. Methods: Grasp Optimization

The robot must first grasp the plug before assembling or disassembling the plug and socket, and the success of the assembly or disassembly process is conditioned on the quality of the grasp. Thus, for each assembly, we perform an optimization procedure to determine a grasp that may lead to a high probability of success during assembly or disassembly. This optimal grasp is then used when generating disassembly trajectories, training assembly policies, and deploying assembly policies in the real world for that assembly. Our grasp optimization procedure consists of the following steps:

1) **Grasp Sampling** (**Figure S12**): We apply a kinematics- and geometry-based grasp sampling approach based on [19]. For each assembly, we first initialize the plug and socket meshes in their assembled state. We instantiate a robot gripper mesh, randomly sample a surface normal on the plug mesh, and align the central axis of the gripper to be collinear with the surface normal. We then randomly sample a position along this normal and translate the gripper to that position. We define a *grasp sample* as the 6-DOF pose of the gripper in this state.

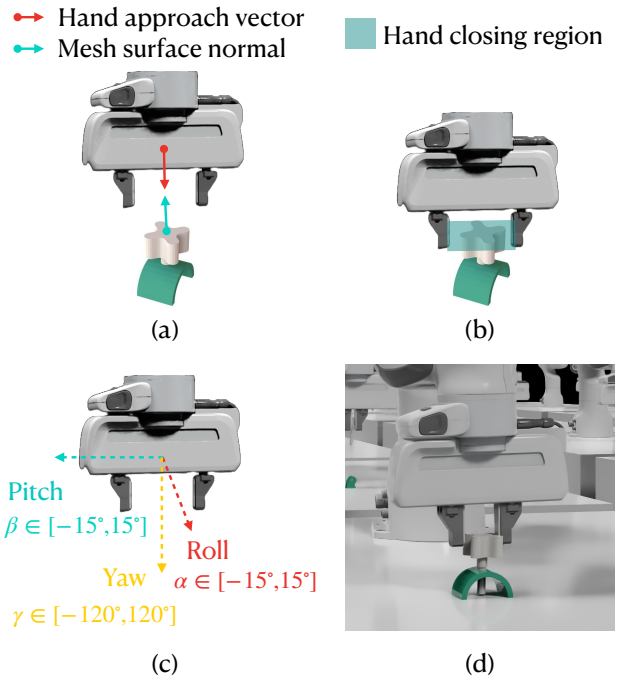

Fig. S12. **Grasp sampling and evaluation procedure.** We apply a grasp sampling approach based on [19], and we develop a a physics-based evaluation procedure. A) We sample a surface normal on the socket mesh, align the gripper axis with the normal, sample a position along the normal, and translate the gripper to that position. B) We check if the plug lies within the gripper closing region. C) We check if the Euler angles of the gripper are within specified bounds. D) We use simulation to disassemble the plug from the socket and check whether the plug remains in the gripper fingers.

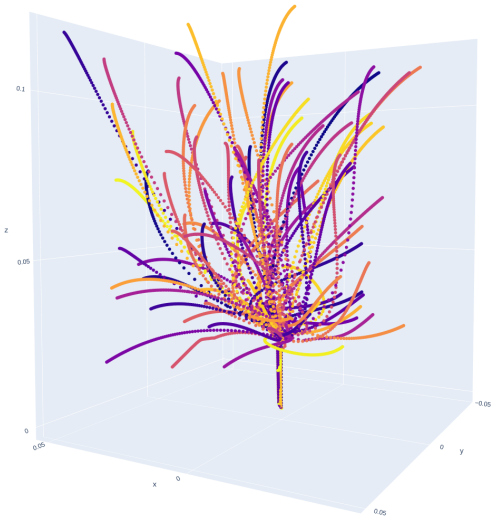

Fig. S13. **Generated disassembly paths.** We generate disassembly paths via physics simulation and reverse the paths for use in assembly. Here we visualize 100 disassembly paths for an assembly with a deep socket.

We reject the grasp sample if 1) the robot hand intersects the plug or socket meshes, 2) the plug mesh does not intersect the gripper closing region (i.e., the prismatic volume contained between the fully-opened gripper fingers), or 3) the Euler angles of the gripper are outside of specified bounds ([-15, 15] deg for roll and pitch and [-120, 120] deg for yaw). The last of these criteria is designed to ensure that the Franka robot remains in a region of its workspace with high manipulability. We repeat this process until generating 100 grasp samples.

2) **Physics-Based Evaluation**: Although the preceding grasp sampling procedure provides kinematically-feasible grasps, these samples are not guaranteed to be stable during the contact-rich interactions experienced during assembly and disassembly. Thus, we develop a subsequent physics-based evaluation phase.

   For each assembly, we first randomize the pose of the socket over a wide range (**Table II**), and we initialize the plug in its assembled state (i.e., inserted in the socket). For each of the 100 grasp samples, we execute the grasp on the plug. We use a task-space impedance controller [43] to lift the plug from the socket until the convex hull of the plug no longer intersects the convex hull of the socket, and we move the robot gripper to a pose in free space randomly sampled from specified bounds ([-0.05, 0.05] for X- and Y-position, [0, 0.05] for Z-position, and [-10, 10] deg for roll, pitch, and yaw). We check whether the grasp is successful (i.e., if the plug remains in the gripper fingers until the end of the procedure). We repeat this procedure 1000 times. Finally, we identify the grasp sample with the highest success rate and designate that sample as the optimal grasp for the given assembly. In total, we run 10 million trials (100 assemblies x 100 grasps per assembly x 1000 trials per grasp), but we distribute the evaluations over many parallel environments for efficiency.

Thus, the output of the grasp optimization procedure is a dictionary that maps each assembly in the dataset to an optimal grasp for the corresponding plug. This grasp is inherently collision-free with respect to the socket in the assembled state, robust to large variations in robot configuration and plug/socket pose, and robust to contact-rich interactions.

D. Methods: Disassembly Path Generation

Figure S13 shows a visualization of disassembly paths for a representative assembly.

E. Methods: Reinforcement Learning

We formulate the robotic assembly problem as a Markov decision process (MDP), where the agent is a simulated robot, and the environment is a simulated environment containing the parts to be assembled. We define a state space $\mathcal{S}$, observation space $\mathcal{O}$, and action space $\mathcal{A}$. Our state-transition dynamics are defined by $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, which is governed by the physical laws of rigid-body dynamics implemented in our simulator. We define a randomized initial state distribution ρ_0 (**Table II**) and reward function $R : \mathcal{S} \rightarrow \mathbb{R}$ with discount factor $\gamma \in (0, 1]$. We constrain our agents to execute actions over episodes of

length N timesteps, and we define shorter learning horizons of length T timesteps. Finally, our return G is defined as

$$G(T) = \mathbb{E}_{\pi}\left[\Sigma_{t=0}^{T-1}\gamma^{t}R(s_t)\right] \tag{7}$$

In other words, the return is the expected sum of discounted rewards over the horizon. The objective is to train a policy $\pi : \mathcal{O} \rightarrow \mathbb{P}(\mathcal{A})$ that maximizes the return.

	Parameter	Randomization Range
Socket	X-position (m)	[0.40, 0.60]
	Y-position (m)	[-0.10, 0.10]
	Z-position (m)	[0.16, 0.18]
	roll angle (deg)	[-5, 5]
	pitch angle (deg)	[-5, 5]
	yaw angle (deg)	[-5, 5]
Plug (rel. to socket)	X-position (mm)	[-10, 10]
	Y-position (mm)	[-10, 10]
	Z-position (mm)	[10, 20]
	roll angle (deg)	[-5, 5]
	pitch angle (deg)	[-5, 5]
	yaw angle (deg)	[-5, 5]
Plug (rel. to gripper)	X-position (mm)	[-1, 1]
	Y-position (mm)	[-1, 1]
	Z-position (mm)	[-1, 1]
	roll angle (deg)	[-5, 5]
	pitch angle (deg)	[-5, 5]
	yaw angle (deg)	[-5, 5]

TABLE II

RANDOMIZATION RANGES FOR INITIAL OBJECT POSES DURING TRAINING. WE LIST POSITION AND ORIENTATION RANGES FOR THE FOLLOWING RANDOMIZATION PROCEDURE: FIRST, THE SOCKET POSITION AND ORIENTATION ARE RANDOMIZED. NEXT, THE PLUG POSITION AND ORIENTATION ARE RANDOMIZED RELATIVE TO THE SOCKET POSE. THEN, THE ROBOT IS COMMANDED TO MOVE ITS GRIPPER NEAR THE PLUG POSE. FINALLY, THE PLUG POSITION AND ORIENTATION ARE RANDOMIZED AGAIN RELATIVE TO THE GRIPPER POSE. VALUES ARE ALL SAMPLED FROM UNIFORM DISTRIBUTIONS.

To train policies, we use the proximal policy optimization (PPO) algorithm [72] due to its well-established performance over a wide range of simulation and sim-to-real problems [48], as well as its ease-of-use; we mitigate the low sample efficiency of PPO by using GPU-accelerated SDF-based contact simulation [53] and a GPU-accelerated PPO implementation [47]. We use PPO to learn a stochastic policy π_θ (i.e., an actor) parameterized by a neural network with weights θ, as well as an approximation of the on-policy value function $V_\phi : \mathcal{S} \rightarrow \mathbb{R}$ (i.e., a critic) parameterized by a neural network with weights ϕ. At evaluation and deployment time, the actor is deterministic, and the critic is neglected. For network architectures and hyperparameters, see **Table III**.

Our observation space provided to the actor consists of robot-arm joint angles [$\mathbb{R}^7$], the current pose of the end effector (i.e., the pose of the robot-gripper fingertips) [SE(3)], the goal pose of the end effector [SE(3)], and the pose of the end effector relative to the current pose [SE(3)] (**Table IV**). During training, the goal pose is simply the pose of the end effector when it is grasping the plug in its optimal grasp pose (**Appendix C**) while the plug is inserted into the socket. We avoid including joint velocities, end-effector velocities, and joint torques in the observation space, as these measurements exhibit substantial noise in the real world and can impede sim-to-real transfer; we also avoid including plug pose, as measuring this quantity typically requires tactile sensing [4].

Parameter	Specialist Training	Generalist Fine-tuning
MLP network size (actor)	[256, 128, 64]	[512, 256, 128, 64]
MLP network size (critic)	[256, 128, 64]	[256, 128, 64]
LSTM network size (actor)	256	256
Horizon length (T)	32	32
Adam learning rate	1e-4	1e-4
Discount factor (γ)	0.99	0.99
GAE parameter (λ)	0.95	0.95
Entropy coefficient	0.0	0.0
Critic coefficient	2	2
Minibatch size	8192	8192
Minibatch epochs	8	8
Clipping parameter (ϵ)	0.2	0.2

TABLE III

NETWORK ARCHITECTURES AND HYPERPARAMETERS USED WITH PROXIMAL POLICY OPTIMIZATION (PPO). WE LIST OUR SPECIALIST AND GENERALIST POLICY NETWORK ARCHITECTURES FOR THE ACTOR AND CRITIC NETWORKS, AS WELL AS OUR MOST CRITICAL PPO HYPERPARAMETERS. WE USE PPO TO TRAIN SPECIALIST POLICIES FROM SCRATCH, AS WELL AS FINE-TUNE GENERALIST POLICIES AFTER DISTILLATION AS DESCRIBED IN **SECTION V-D**.

However, we adopt an asymmetric actor-critic strategy [61], where the states provided to the critic include privileged information that is not provided to the actor, as the critic is only used for training and is not deployed in the real world. Here, the states provided to the critic include joint velocities [$\mathbb{R}^7$], end-effector velocities [$\mathbb{R}^6$], and plug pose [SE(3)]. In addition, to capture real-world control error, perception error, and sensor noise, we apply uniformly-sampled noise to all observations of the position and orientation of the socket that are provided to the actor (**Table V**), but do not apply noise to the corresponding states provided to the critic.

Our action space consists of incremental pose targets [SE(3)], which represent the position and orientation difference between the current pose x_c and the target pose x_t; we choose incremental targets rather than absolute targets in order to select from a small, bounded spatial range. We pass these targets to a task-space impedance controller

$$\tau = J^T\left(k_p(x_t \ominus x_c) - k_d\dot{x}_c\right) \tag{8}$$

where $J \in \mathbb{R}^{6\times7}$ is the geometric Jacobian; $K_p \in \mathbb{R}^{6\times6}$ and $K_d \in \mathbb{R}^{6\times6}$ are diagonal matrices consisting of proportional and derivative gains, respectively; $\dot{x_c} \in \mathbb{R}^6$ is the velocity vector; and $x_t \ominus x_c$ computes the incremental pose target. We use a task-space impedance controller to generalize actions across robot configurations and avoid using inertial matrices, which have not been precisely measured for our robot manipulator. We superimpose a nullspace controller to softly constrain the robot to maintain a configuration with high manipulability, which can be compromised by elbow drift.

Finally, our baseline reward formulation (without imitation) is derived from [76] and is composed of terms that penalize distance-to-goal, penalize simulation error, reward task difficulty, and reward success. Specifically, the reward

Input	Dimensions	Actor	Critic
Arm joint angles	7	✓	✓
Fingertip pose	3 (position) + 4 (quaternion)	✓	✓
Target pose	3 (position) + 4 (quaternion)		✓
Target pose with noise	3 (position) + 4 (quaternion)	✓	
Relative target pose with noise	3 (position) + 4 (quaternion)	✓	
Arm joint velocities	7		✓
Fingertip linear velocity	3		✓
Fingertip angular velocity	3		✓
Plug pose	3 (position) + 4 (quaternion)		✓
Relative target pose	3 (position) + 4 (quaternion)		✓

TABLE IV

INPUTS TO THE ACTOR AND CRITIC FOR SPECIALIST POLICIES. WE LIST OBSERVATIONS PROVIDED TO THE ACTOR, AS WELL AS OBSERVATIONS AND STATES PROVIDED TO THE CRITIC.

Parameter	Noise Range
Socket X-position	[-2, 2] mm
Socket Y-position	[-2, 2] mm
Socket Z-position	[-2, 2] mm
Socket roll angle	[-5, 5] deg
Socket pitch angle	[-5, 5] deg
Socket yaw angle	[-5, 5] deg

TABLE V

NOISE RANGES FOR OBSERVATIONS OF SOCKET POSE DURING TRAINING. WE LIST THE RANGES FOR POSITION AND ORIENTATION NOISE APPLIED TO OBSERVATIONS OF THE SOCKET POSE, WHICH IS USED TO COMPUTE THE GOAL POSE OF THE END EFFECTOR. VALUES WERE SAMPLED FROM A UNIFORM DISTRIBUTIONS.

1) penalizes distance-to-goal through an SDF-based reward, which computes the distance between the current plug pose and the goal (i.e., assembled) plug pose through SDF queries, which are less sensitive to object symmetries than keypoint-based distance queries,
2) penalizes simulation error through a simulation-aware policy update (SAPU), which computes the maximum interpenetration distance at each timestep, weights the reward in inverse proportion to distance if it is less than a threshold, and does not update the reward otherwise,
3) rewards task difficulty through a sampling-based curriculum (SBC), which increases the lower bound (but not the upper bound) of the range of initial-pose randomization as the agent becomes more proficient at the task, and weights the return in proportion to task difficulty.

We defer precise descriptions to [76] and implementation details to [79]. Whereas [76] also rewarded success by providing a bonus at the end of every episode if a keypoint distance between the plug and its goal fell below a threshold on the final timestep, we instead reward success with a bonus at the end of every *horizon* if the *translational* distance between the plug and its goal falls below a threshold at *any* timestep.

Precisely, our return over each horizon is given as

$$G(T) = w_{SBC} \sum_{t=0}^{T-1} \left(\omega_{SAPU} (\omega_{SDF} R_{SDF} + \omega_I R_I) \right) + R_{succ} \tag{9}$$

where w_{SBC} is the weighting factor based on task difficulty, as determined by the SBC algorithm; ω_{SAPU} is the weighting factor based on simulation error, as determined by the SAPU algorithm; R_{SDF} is the distance-to-goal reward, as determined by the SDF-based reward; R_I is the imitation-based reward, as described in detail in the main text; ω_{SDF} and ω_I are hyperparameters to determine the relative importance of the distance-to-goal reward and the imitation-based reward; and R_{succ} is a success bonus applied at the end of each horizon. Parameters ω_{SDF} and ω_I are tuned simply so that R_{SDF} and R_I fall within the same order of magnitude.

F. Methods: Dynamic Time Warping

At each timestep, we aim to determine the best reversed disassembly path for the robot to mimic. More specifically, given the assembly path the robot has already traversed during the episode, we aim to select the closest disassembly path to imitate. The first method we leverage for this procedure is dynamic time warping (DTW), which is described as follows:

Consider a time sequences $a = [a^1, a^2, ..., a^P]$, which might represent the path the robot has traversed, and a time sequence $b = [b^1, b^2, ..., b^Q]$, which might represent a disassembly path; we aim to compute the distance between these paths. If we repeat this procedure for all disassembly paths, we can select the closest disassembly path as desired.

DTW matches each point a_i to one or more points b_j, and vice versa. The matching process minimizes a cost $C(a, b)$, which is defined as the sum of a manually-defined distance function (typically, Euclidean distance) between each point a_i and its match(es) from b. Moreover, the matching process satisfies the following constraints:

1) Point a_1 must match with at least point b_1 (i.e., first points are aligned)
2) Point a_P must match with at least point b_Q (i.e., last points are aligned)
3) All matches must be monotonic (i.e., if point a_i matches with point b_j, then point a_{i+1} cannot match with point b_{j-1} and point a_{i-1} cannot match with point b_{j+1})

Ultimately, DTW returns the total distance between the optimal matches of sequence a and sequence b. We leverage the fast DTW implementation from *Soft-DTW* [16].

Algorithm F provides pseudocode for a naive implementation of DTW. In this implementation, a matrix M is constructed, where each $M[i][j]$ describes the minimum cost of matching $a[i]$ with $b[j]$. The implementation loops through each $M[i][j]$ and assigns its value to the distance between $a[i]$ and $b[j]$, plus the minimum accumulated cost of all previous possible matches. Importantly, only 3 such accumulated costs need to be considered: the accumulated costs of matching $a[i-1]$ and $b[j]$, $a[i]$ and $b[j-1]$, and $a[i-1]$ and $b[j-1]$. Intuitively, these are the only accumulated costs that 1) leave no previous point unmatched, and 2) are compliant with constraint 3. The value of element $M[P][Q]$ is the final value assigned in the loop and represents the minimum accumulated cost $C^*(a,b)$ over all possible matches between a and b.

Algorithm 1 Dynamic Time Warping (DTW)

Require: Sequence a of length P and sequence b of length Q
Ensure: DTW distance between a and b

```
function DTWDISTANCE(a, b)
    Define matrix M of shape (P + 1, Q + 1)
    Initialize all elements of M to ∞
    M[0][0] ← 0
    for i ← 1 to P do
        for j ← 1 to Q do
            d ← ‖a[i] − b[j]‖_2
            M[i][j] ← d + min(M[i − 1][j], M[i][j − 1], M[i − 1][j − 1])
        end for
    end for
    return M[P][Q]
end function
```

G. Methods: Signature Transform

At each timestep, we aim to select the best reversed disassembly path for the robot to mimic. More specifically, given the assembly path the robot has already traversed during the episode, we aim to select the closest disassembly path to imitate. The second method we explore for this procedure is the signature transform. For an interactive introduction, see [20], and for a detailed overview, see [12].

Consider a continuous-time 3-dimensional path given by $X : [a,b] \to \mathbb{R}^3$. For example, we can define the path $p(t)_{a,b} = (x(t), y(t), z(t))_{a,b}$, where $x(t)$, $y(t)$, and $z(t)$ might represent the x, y, and z coordinates of the path the robot has already traversed for $t \in [a,b]$. We can also consider a second path, which might represent a disassembly path; we aim to compute the distance between these paths by simply computing the L2 norm between their path signatures, which are defined next. If we repeat this procedure for all disassembly paths, we can select the closest disassembly path as desired.

Focusing on the path $p(t)_{a,b}$, the path signature is given by the collection of all possible path integrals between $x(t)$, $y(t)$, and $z(t)$. Specifically, the *first level* of the path signature is

$$S_1(p(t))_{a,t} = \big(S_1(x(t))_{a,t}, S_1(y(t))_{a,t}, S_1(z(t))_{a,t}\big), \quad (10)$$

where

$$S_1(x(t))_{a,t} = \int_a^t dx(t) = x(t) - x(a) \quad (11)$$

$$S_1(y(t))_{a,t} = \int_a^t dy(t) = y(t) - y(a) \quad (12)$$

$$S_1(z(t))_{a,t} = \int_a^t dz(t) = z(t) - z(a) \quad (13)$$

In this case, there are 3 total path integrals, and each integral only involves a single coordinate of the path $p(t)$.

Next, the *second level* of the path signature is

$$S_2(p(t))_{a,t} = \big(S_2(x(t),x(t))_{a,t}, S_2(x(t),y(t))_{a,t}, ..., S_2(z(t),z(t))_{a,t}\big), \quad (14)$$

where

$$S_2(x(t),x(t))_{a,t} = \int_a^t S(x(t))_{a,t} dx(t) \quad (15)$$

$$S_2(x(t),y(t))_{a,t} = \int_a^t S(x(t))_{a,t} dy(t) \quad (16)$$

$$... = ...$$

$$S_2(z(t),z(t))_{a,t} = \int_a^t S(z(t))_{a,t} dz(t) \quad (17)$$

In this case, there are 9 total path integrals, and each individual integral involves 2 coordinates of the path $p(t)$. (Note that when $a = 0$, **Equation 16** can be interpreted as the area under the curve when $x(t)$ is plotted against $y(t)$.) Further levels of the continuous-time path signature can be derived in similar fashion, where the ith level consists of 3^i path integrals.

The full continuous-time path signature $S(p(t))_{a,b}$ for $t \in [a,b]$ consists of the ordered set of all integrals. Specifically,

$$S(p(t))_{a,b} = (1, S_1(p(t))_{a,b}, S_2(p(t))_{a,b}, ...) \quad (18)$$

where the first element is equal to 1 by convention. Conveniently, because path integrals are translation invariant (i.e., unaffected if the integrand is shifted by a constant) and reparameterization invariant (i.e., unaffected if the integrand traces its path slower or faster in time), path signatures inherit these properties. Thus, they are an extremely convenient representation for time-series data that may have translational offsets and/or disparate discretization or sampling schemes. Finally, the signature transform is simply the functional $T(p(t))_{a,b} : p(t)_{a,b} \to S((p(t))_{a,b}$ that takes a path as input and produces the path signature as output for $t \in [a,b]$.

As our data is not continuous, but discrete, we use the discrete-time form of the path signature. The first level $S_1(p[N])_{A,N}$, where A and N are the first and current timestep

indices, respectively, can be expressed as

$$S_1(x[N])_{A,N} = \sum_{i=A}^{N-1}(x[i+1]-x[i]) = x[N]-x[A] \tag{19}$$

$$S_1(y[N])_{A,N} = \sum_{i=A}^{N-1}(y[i+1]-y[i]) = y[N]-y[A] \tag{20}$$

$$S_1(z[N])_{A,N} = \sum_{i=A}^{N-1}(z[i+1]-z[i]) = z[N]-z[A] \tag{21}$$

The second level $S_2(p[N])_{A,N}$ can be expressed as

$$S_2(x[N],x[N])_{A,N} = \sum_{i=A}^{N-1}(x[i+1]-x[A])(x[i+1]-x[i]) \tag{22}$$

$$S_2(x[N],y[N])_{A,N} = \sum_{i=A}^{N-1}(x[i+1]-x[A])(y[i+1]-y[i]) \tag{23}$$

$$\ldots = \ldots$$

$$S_2(z[N],z[N])_{A,N} = \sum_{i=A}^{N-1}(z[i+1]-z[A])(z[i+1]-z[i]) \tag{24}$$

As in continuous time, further levels of the discrete-time path signature can be derived in similar fashion, where the ith level consists of 3^i path summations.

Finally, the full discrete-time path signature $S(p[N])_{A,B}$ for $N \in [A,B]$, where B is the last timestep index, consists of the ordered set of all summations. Specifically,

$$S(p[N])_{A,B} = (1, S_1(p[N])_{A,B}, S_2(p[N])_{A,B}, \ldots) \tag{25}$$

We leverage the fast, GPU-based signature transform implementation from Signatory [33].

H. Methods: Point-Cloud Autoencoder

Our specialist policies do not take part geometry as an observation, as geometry is constant for each policy and would not benefit policy learning. However, our generalist policy *does* take part geometry as an observation; unlike comparatively-imprecise tasks such as part reorientation [11], assembling a wide range of parts without knowledge of geometry would be exceedingly difficult. At the same time, the meshes for our parts typically consist of 1000-5000 vertices and edges. We may consider A) taking mesh data directly as input to the policy or B) pretraining a network to extract a latent representation of mesh data and passing the latent vector to the policy. We choose option B, as directly consuming mesh data would require an exceptionally-large observation space, and learning a latent representation and an assembly policy simultaneously would be computationally challenging.

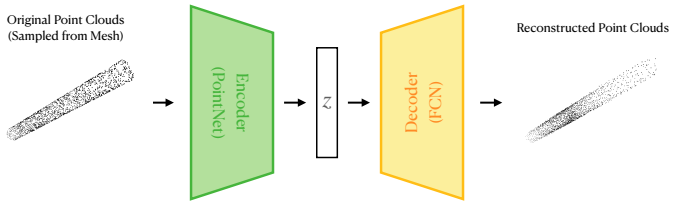

Fig. S14. **Schematic of the point-cloud autoencoder.** We pass a point cloud as input to a PointNet encoder based on [51] to produce a latent vector z, which is in turn passed to a fully-convolutional decoder based on [86]. The autoencoder is trained to minimize reconstruction loss.

Specifically, we train an autoencoder on a large set of meshes M. Each mesh $m_i \in M$ consists of (V_i, E_i), where V are the vertices and E are the (undirected) edges. At each iteration, we sample a batch of meshes $B \subset M$; for each $m_i \in B$, we sample a point cloud P_i online, with each point $p_i^j \in P_i$ lying on the surface of m_i. The point cloud P_i is passed to a PointNet encoder [63] based on the implementation from [51] to produce a latent vector z_i. Vector z_i is passed to a fully-convolutional decoder based on the implementation from [86] to produce a reconstructed point cloud Q_i (**Figure H**). The network is trained to minimize reconstruction loss, defined here as the chamfer distance between P_i and Q_i:

$$L_{\text{CD}} = \frac{1}{|P_i|}\sum_{p\in P_i}\min_{q\in Q_i}\|p-q\|_2^2 + \frac{1}{|Q_i|}\sum_{q\in Q_i}\min_{p\in P_i}\|p-q\|_2^2 .$$

In our final training procedure, $|M|$ = 1000 meshes from [80], N = 2000, and $|z_i|$ = 32.

We briefly note two aspects of our training procedure that improve reconstruction accuracy: 1) We normalize the mean and variance of the vertices V_i for each mesh m_i prior to training, such that the network is not biased by a non-uniform distribution of mesh sizes, and 2) We increase the depth of the encoder relative to the decoder, such that the encoder can learn a more abstract latent representation, whereas the decoder is discouraged from overfitting to the input data.

Future work may focus on training an autoencoder with explicit or implicit surface information, which we hypothesize can improve the success rate of the generalist policy. Possible methods include 1) augmenting each point p_i^j with the local surface normal, 2) using a graph neural net (e.g., a graph convolutional network) that takes both points and edges as input [66, 52], or 3) learning low-dimensional signed-distance-field (SDF) representations of the objects [13].

I. Results: Specialist Policies

Figure S16 shows the results of our final training approach for specialist policies over all 100 assemblies.

J. Results: Generalist Policies

As a supplementary evaluation question, we ask, **what is the scaling law between generalist performance and the number of specialists used in training?**

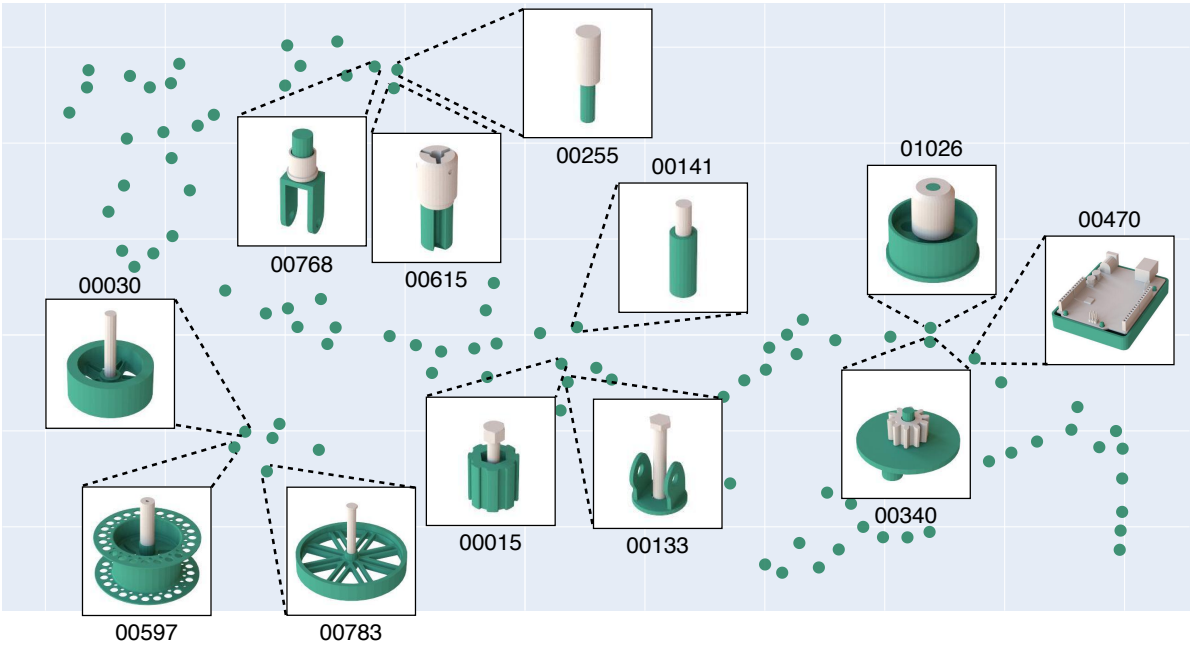

Fig. S15. **t-SNE visualization of geometric representations of 100 assemblies.** As in **Figure 5**, we plot the t-SNE representations of all 100 assemblies. Here, we show assemblies sampled from the same or nearby clusters; samples that are close in the lower-dimensional space have similar visual properties.

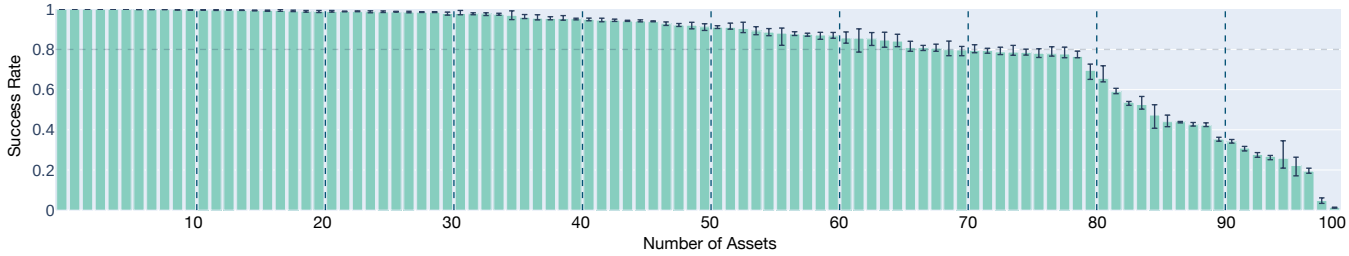

Fig. S16. **Simulation-based evaluation of final training approach for specialist policies.** For each of the 100 assemblies, we train a specialist policy with the final AutoMate learning approach. For this approach, we train 5 random seeds, select the best seed, and evaluate it 5 times over 1000 trials. AutoMate maintains consistent performance across the majority of the assemblies and achieves the critical milestone of solving approximately 80% of the assemblies with 80% success rates or higher under substantial initial-pose randomization (**Table II**) and observation noise (**Table V**)

To answer this question, we consider batches of {10, 20, ..., 80} assemblies, where each batch is evenly sampled in t-SNE space (**Figure S15**). For each batch, we train a generalist policy with *RL + DAgger + RL (w/SBC)*, from all specialists corresponding to that batch of assemblies. We evaluate each generalist policy over all the assemblies in its corresponding batch over 5000 trials, for a total of 1.8M trials. **Figure S17** shows our results. We observe high success rates for a generalist trained from 10 and 20 specialists (≈80%), a steep drop for a generalist trained on 30 or 40 specialists (≈55% and ≈30%), and consistent, low success rates for a generalist trained on more specialists (≈20%). In future work, we aspire to formulate methods that can preserve generalist performance as the number of assemblies increases.

K. Methods: Perception

1) Camera Calibration and Tuning: We observe the environment with a single wrist-mounted Intel RealSense D435 RGB-D camera. It is a common sentiment among robotics researchers that off-the-shelf cameras may not be sufficient for high-precision tasks; moreover, it is common practice among the robot learning community to compensate for the weaknesses of such cameras primarily via data-driven strategies (e.g., increased data collection, data augmentation, etc.). However, we have found that diligent camera calibration and tuning can greatly improve the RGB image quality, point cloud quality, and performance in downstream perception modules (e.g., pose estimators), to a level sufficient for high-precision assembly of parts far smaller than the robot manipulator.

Specifically, we take the following steps:

- **Extrinsics calibration**: We calibrate our *absolute* extrinsics (i.e., the pose of our RGB camera in the robot frame)

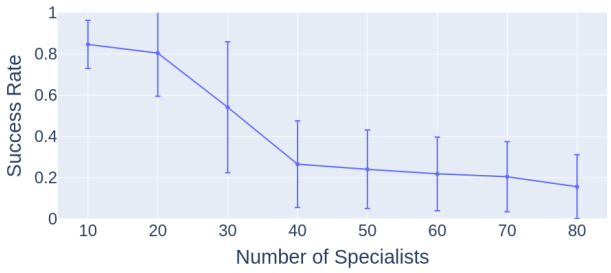

Fig. S17. **Simulation-based evaluation of scaling law for training generalist policies.** We consider batches of {10, 20, ..., 80} assemblies, where each batch is evenly sampled in t-SNE space (**Figure S15**). For each batch, we train a generalist policy with the final AutoMate learning approach, from all specialists corresponding to that batch of assemblies. We evaluate each generalist policy over all the assemblies in its corresponding batch over 5000 trials. We observe high success rates for a generalist trained over 10 and 20 assemblies (≈80%), a steep drop at 30 and 40 assemblies (≈55% and ≈30%), and consistent, low success rates for more assemblies (≈20%).

using the procedure described in [76], with a lightly-modified implementation of the corresponding code in [78]. The procedure consists of moving the end effector to randomized target poses, capturing an image of an AprilTag in each pose, computing the pose of the AprilTag in the camera frame from each image, and using the Tsai-Lenz algorithm [84] to compute the extrinsics matrix. We do not calibrate our *relative* intrinsics (i.e., the pose of our depth camera with respect to our RGB camera) and rely on the RealSense-provided matrix.

- **Intrinsics calibration**: We calibrate our intrinsics matrix using the procedure described in the RealSense whitepaper for on-chip self-calibration [25]. The procedure consists of capturing an image of a textured target and running the manufacturer-provided calibration function.
- **Camera settings**: We tune our camera settings using the suggestions provided in the RealSense whitepapers on tuning depth cameras [26] and depth image post-processing [24]. The most impactful settings were
  - RGB and depth camera exposure: We tune the exposure to maximize the quality of the color and depth map images on our assemblies, rather than using the default autoexposure. Furthermore, we tune the RGB and depth camera exposures simultaneously (i.e., set them to the same value), rather than separately.
  - Laser power: We increase power from its default value to increase the density of the depth image.
  - Spatial hole-filling: We apply a hole-filling filter in post-processing to repair holes in the depth image.

Finally, to optimize the performance of our downstream pose estimator (described next), we maximized RGB camera resolution (in order to increase the number of pixels on small surfaces), captured images from angled (rather than overhead) view, and avoided direct lighting of the assemblies.

2) Pose Estimation: In simulation, we train RL policies from 6D poses of the parts rather than from RGB images or point clouds, which would substantially increase compute requirements. On the other hand, in the real world, we observe the environment using a single Intel RealSense D435 RGB-D camera mounted on the wrist of the robot. Thus, in order to deploy our simulation-trained policies in the real world, we may consider A) moving simulation towards reality (i.e., distilling the simulation-trained policies to use RGB-image and/or point-cloud inputs) or B) moving reality towards simulation (i.e., extracting 6D poses from real-world RGB images and/or point clouds). We choose option B for several reasons: 1) We can avoid a cross-modal distillation process, 2) Simulated and real-world RGB images have a substantial sim-to-real gap, 3) The quality of real-world point clouds is poor on our small, mildly-reflective parts, and 4) The accuracy of pose estimators has improved dramatically in recent years [38, 39, 88].

We assume that each part of each assembly has a known CAD model (specifically, an OBJ file with no associated texture), which is typical in industrial assembly settings. We use a pose estimation pipeline that takes as input 1) an RGB-D image of a part in the real world, 2) the camera intrinsics matrix, 3) the camera extrinsics matrix, and 4) a CAD model of the part, and then predicts the 6D pose of the part. This pose can subsequently be passed as input to our RL policies.

Our pose estimation pipeline consists of the following steps:

1) **Image capture**: The RealSense camera is used to capture a 1280 x 720 RGB image and 1280 x 720 depth image of a part with a known CAD model.
2) **Part selection**: The RGB image is shown to the user. The user can left-click on the part to provide a positive annotation (i.e., a pixel that lies on the part of interest).
3) **Segmentation**: The RGB image, pixel location(s), and annotation(s) are passed to [37], which produces a high-accuracy segmentation mask for the part.
4) **Refinement (optional)**: If the mask does not span the part, the user can provide another positive annotation; conversely, if the mask includes background features, the user can right-click to provide a negative annotation (i.e., a pixel that does not correspond to the part). Segmentation is then executed with the additional annotations.
5) **Model-based estimation**: The RGB image, depth image, camera intrinsics, segmentation mask, and textureless CAD model are fed to [88], which regresses to the 6D object pose in the camera frame.
6) **Frame transformation**: The 6D object pose in the camera frame is combined with the camera extrinsics to compute the 6D object pose in the robot frame.

In total, the steps above take ≈20 seconds to execute.

Future work may focus on replacing the click-based interface for selecting parts with a language interface, using faster implementations of the segmentation model (e.g., [98, 94]), automatically retrieving the appropriate CAD model from a database, and optimizing [88] for faster performance.

As first described in **Section III**, our real-world system consists of a Franka robot with a standard parallel-jaw gripper, a wrist-mounted RealSense D435 camera, a Schunk EGK40 parallel-jaw gripper mounted to the tabletop, and 3D-printed

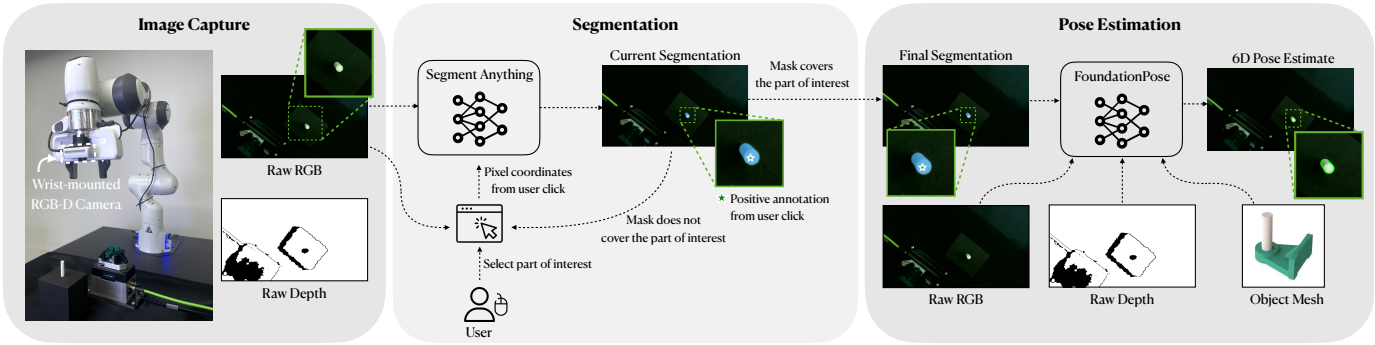

Fig. S18. **Real-world perception pipeline.** Left: At the beginning of our pipeline, we use an Intel RealSense D435 RGB-D camera mounted on the wrist of the robot to capture an RGB image and depth image. Middle: We show the RGB image to the user, who clicks on the plug or socket of interest. We then pass the RGB image and pixel coordinates through a powerful segmentation tool [37] and compute a segmentation mask for the plug or socket. Right: We pass the RGB image, depth image, segmentation mask, and CAD model for the plug or socket into a state-of-the-art pose estimator [88] to estimate the 6-DOF pose of the part in the camera frame. We later use robot kinematics and camera extrinsics to convert the pose to the robot frame.

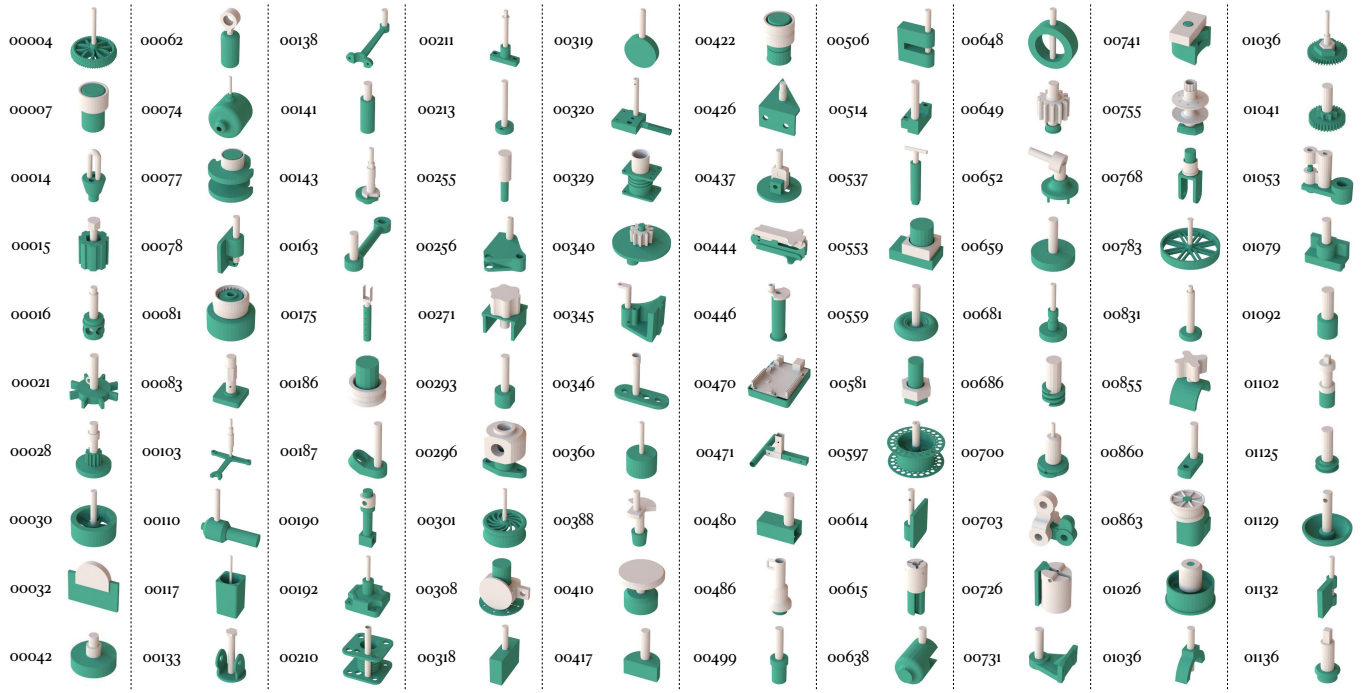

Fig. S19. **Assembly lookup chart.** For each assembly investigated in this work, we provide its unique assembly ID and a rendering. The assemblies are the same as those visualized in **Figure 2**. The asset IDs are referenced in figures throughout this paper.

assemblies from our dataset (**Figure S11**). Our communications framework is closely modeled after [76]; however, our perception, grasping, and control procedures differ significantly.

For perception, we aim to estimate plug and socket states while initializing them in a far less-constrained manner. We use a powerful segmentation tool [37], textureless CAD models of our parts, and a state-of-the-art pose estimator [88] to estimate the 6-DOF poses of each part from RGB-D images. **Figure S18** shows our pipeline; for details, see **Appendix K**.