Appendix A Qualitative Results

Fig. 1 and Fig. 2 show qualitative results of our system assembling a variety of multi-part objects using different robot arms, both in simulation and in the real world. Once the hardware configuration is provided, our planner works seamlessly with a wide range of robot arms and end-effectors without requiring additional tuning.

The assembly process begins with picking up parts from the fixture, then transferring them to the assembly area for either holding or insertion. The system determines when to switch roles between the two arms, for example by handing over a part or changing the holding grasp, in order to maintain stability and accuracy. After each insertion, the robot returns to retrieve the next part, and this process continues until the assembly is complete. Finally, both arms return to their initial positions.

All plans are generated automatically, including grasp selection, arm coordination, motion generation, and fixture design. The only manual input required is the initial setup of the workcell and placement of the hardware.

For real-world execution, we demonstrate robust sim-to-real transfer across long-horizon assembly tasks, with consistent step-by-step correspondence between simulation and physical execution. The system maintains geometric and temporal alignment across all stages of the assembly, including grasping, insertion, and part switching. This highlights the reliability of our simulation-informed planning and policy execution.

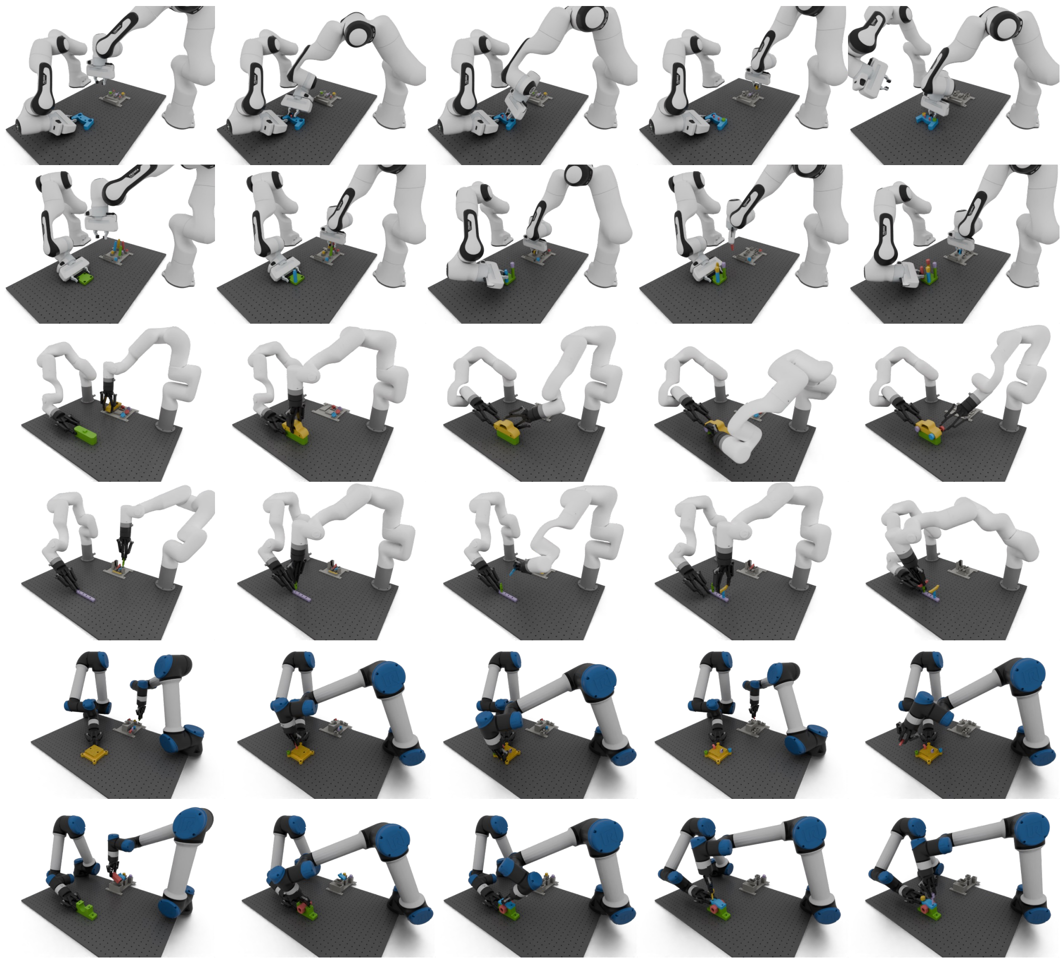


Figure 1: Step-by-step rendered assembly executions on different assemblies with different robots.

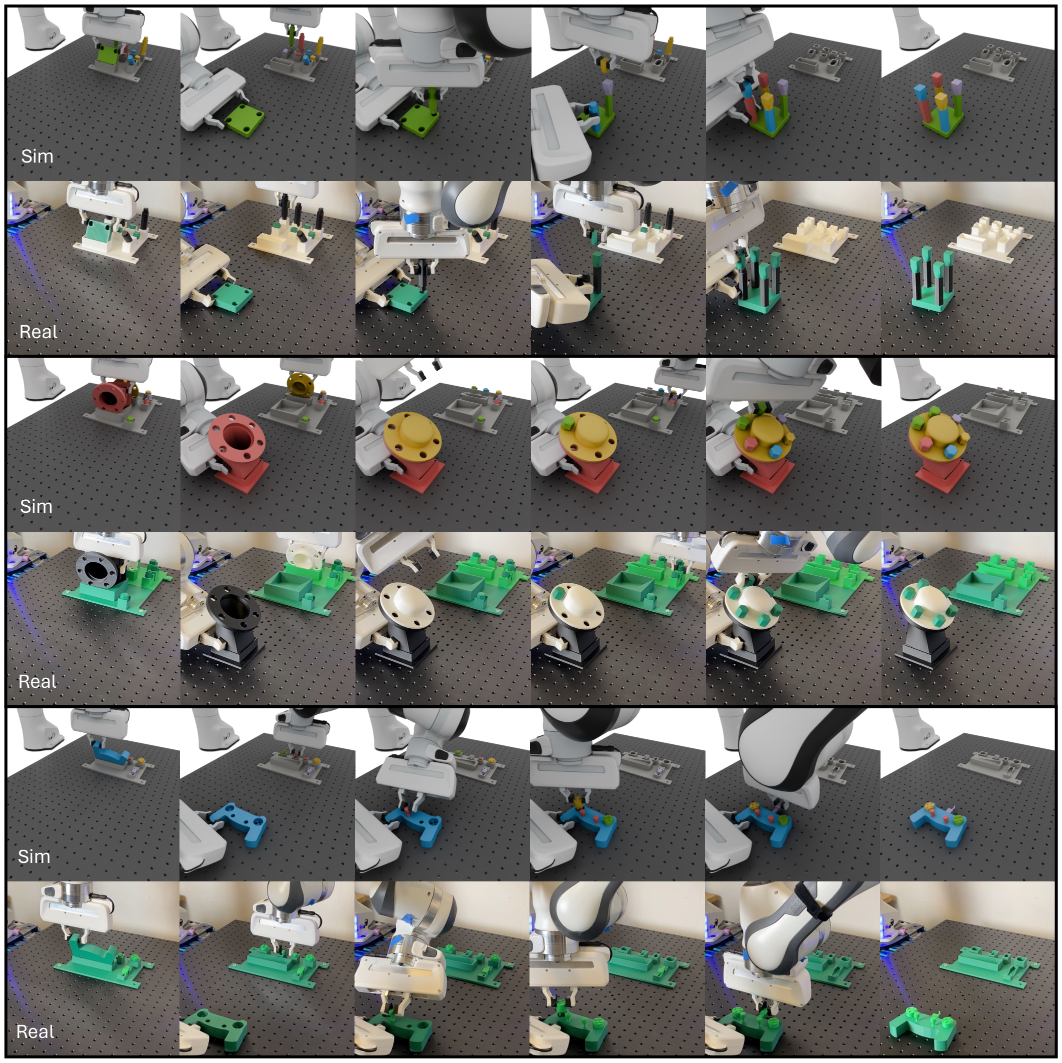


Figure 2: Step-by-step real-world assembly executions on different assemblies with Panda robots, with side-by-side correspondences between simulation and real.

Appendix B Problem Formulation

As in Sec. 3, we formulate planning as optimizing assembly-hold sequences ϕ, grasps σ, and robot motions π:

$$\min_{\phi,\sigma,\pi} E\big(\Phi_{i=1}^{n}\vec{f}(\phi_{1:i},\sigma_{i-1:i},\pi_i)\big) \quad \text{s.t.} \quad C_{\text{prec}}(\phi)\leq 0,\ C_{\text{kin}}(\phi,\sigma,\pi)=0,\ C_{\text{col}}(\phi,\sigma,\pi)\leq 0 \tag{2}$$

A more detailed breakdown of constraints $C_{\text{prec}}(\phi)$, $C_{\text{kin}}(\phi)$, $C_{\text{col}}(\phi)$ is shown below.

$$
\begin{aligned}
\min_{\phi,\sigma,\pi} \quad & \vec{F}(\phi,\sigma) \\
\text{s.t.} \quad & \forall i \in [1,n], \\
& C_{\text{prec}}(\phi_{1:i}) \leq 0 && \text{(3a)} \\
& C_{\text{kin}}(o_{a,i}, g_{a,i}, \tau^{g}_{a,i}(0), p^{0}_{o_i}) = 0 && \text{(3b)} \\
& C_{\text{kin}}(o_{a,i}, g_{a,i}, \tau^{f}_{a,i}(1), p^{G}_{o_i}) = 0 && \text{(3c)} \\
& C_{\text{kin}}(o_{h,i}, g_{h,i}, \tau^{f}_{h,i}(1), p^{G}_{o_i}) = 0 && \text{(3d)} \\
& C_{\text{col}}(\phi_{1:i}, \pi[a,i], \pi[h,i]) \leq 0 && \text{(3e)}
\end{aligned}
$$

where each motion $\tau : [0,1] \rightarrow \mathcal{Q}$ is a time-parametrized joint trajectory in the robot's configuration space $\mathcal{Q}$. Eq. (3a) are the precedence constraints that ensure the partially assembled parts are connected and do not collapse under gravity. Eqs. (3b) to (3d) ensure the robot configuration, grasp, and the grasped object pose are kinematically consistent when picking, assembling, and holding. Eq. (3e) ensures that both robots' trajectories do not collide with previously assembled parts and other static obstacles.

The key planning stages in Sec. 3 are further summarized here.

- Sec. 3.1: Starting from a multi-part mesh model, we construct a precedence graph representing a minimum constraint set for any feasible sequence, considering only collision among parts.
- Sec. 3.2: To reduce online collision checking overhead during search, we pre-compute a discretized, collision-free grasp set for assembling and holding for all part pairs. Each feasible grasp is associated with a corresponding robot trajectory.
- Sec. 3.3: We leverage the precedence graph and precomputed grasp pairs to expand a state tree that contains all feasible part-grasp sequences and then search for an optimal part-grasp sequence that minimizes a grasp stability cost.
- Sec. 3.4: After the grasps are determined, we develop an automatic design algorithm to generate a pickup fixture, so that the planned grasp can be achieved easily without the need of a re-grasp.
- Sec. 3.5: With the assembly sequence fixed and all robot configurations determined for kinematic switches, i.e., pick-up, assembly, and hold, we plan for all transit and transfer motions.

Appendix C Algorithmic Details

We now present detailed mathematical formulations of each algorithm introduced in Sec. 3.

C.1 Part Precedence Planning

Alg. 1 and the following paragraphs provide the details of the part precedence planning algorithm.

Precedence Tier Generation Initially, an empty list of tiers T_{prec} is initialized. We use O_r to represent all the remaining assembled parts, which starts from all parts O. Although O_r is not empty, the algorithm constructs a new tier t_{prec} by evaluating each part $o_i \in O_r$ to determine the feasibility of disassembly. Using motion planning, a disassembly path q_i is computed for o_i. Specifically, we apply Assemble-Them-All [10], a physics-based method for efficient disassembly motion planning given the highly constrained search space, and determine success based on a given timeout. If o_i can be disassembled, the pair (o_i, q_i) is added to t_{prec}. Once all feasible parts are processed, the constructed tier t_{prec} is added to T_{prec}, and all $o_i \in t_{\text{prec}}$ is removed from O_r. This process repeats until all parts are assigned to tiers.

Precedence Graph Generation The graph construction phase generates a directed graph G_{prec} that encodes precedence constraints among parts. An empty set O_e is initialized to track parts in earlier tiers (i.e., parts that are supposed to be disassembled earlier). For each tier $t_{\text{prec}} \in T_{\text{prec}}$, the algorithm processes every part $o_i \in t_{\text{prec}}$ by checking its disassembly path τ_{o_i} for collisions with parts in earlier tiers O_e. A collision check function identifies the set of colliding parts $O_c \subseteq O_e$. For each $o_c \in O_c$, an edge (o_i, o_c) is added to G_{prec}, indicating that o_i's disassembly depends on o_c's prior removal. After processing all parts in t_{prec}, the disassembled parts O_t are added to O_e. The algorithm proceeds until all parts in all precedence tiers are added to the precedence graph.

Algorithm 1 Part Precedence Planning

```
 1: **Input:** All parts $O$ with goal poses $p_O^G$
 2: **Output:** Directed graph $G_{\text{prec}}$ representing precedence constraints
 3: Initialize an empty list of tiers: $T_{\text{prec}} \leftarrow []$
 4: Initialize an empty directed graph: $G_{\text{prec}} \leftarrow \text{DiGraph}()$
 5: Initialize all remaining parts $O_r \leftarrow O$
 6: **while** $O_r \neq \emptyset$ **do**                                   ▷ Tier generation
 7:     Initialize an empty tier: $t_{\text{prec}} \leftarrow \{\}$
 8:     **for** each part $o_i \in O_r$ **do**
 9:         $\tau_{o_i} \leftarrow \text{DisassemblyPath}(o_i, O_r)$
10:         **if** feasible $\tau_{o_i}$ is found **then**
11:             $t_{\text{prec}} \leftarrow t_{\text{prec}} \cup \{(o_i, \tau_{o_i})\}$
12:     $O_r \leftarrow O_r \setminus \{o_i \mid (o_i, \tau_{o_i}) \in t_{\text{prec}}\}$
13:     $T_{\text{prec}}.\text{Append}(t_{\text{prec}})$
14: Initialize an empty set of parts in earlier tiers: $O_e \leftarrow \emptyset$
15: **for** each tier $t_{\text{prec}} \in T_{\text{prec}}$ **do**                  ▷ Graph construction
16:     Let $O_t \leftarrow \{o_i \mid (o_i, \tau_{o_i}) \in t_{\text{prec}}\}$                ▷ Parts in tier $t_{\text{prec}}$
17:     **for** each $(o_i, \tau_{o_i}) \in t_{\text{prec}}$ **do**
18:         $G_{\text{prec}}.\text{AddNode}(o_i, \text{path} = \tau_{o_i})$
19:         $O_c \leftarrow \text{CheckPathCollision}(o_i, \tau_{o_i}, O_e)$       ▷ $O_c$: colliding parts in $O_e$
20:         **for** each $o_c \in O_c$ **do**
21:             $G_{\text{prec}}.\text{AddEdge}(o_i, o_c)$
22:     $O_e \leftarrow O_{\text{e}} \cup O_t$                                    ▷ Update parts in earlier tiers
23: **return** $G_{\text{prec}}$
```

C.2 Dual-Arm Grasp Filtering

This section provides a detailed description of the sub-steps involved in the grasp filtering algorithm.

Single-Pose Grasp Feasibility Check Alg. 2 evaluates whether a specific grasp configuration g is feasible for a target part o in its current pose p_o. The algorithm first determines the set of preceding parts O_{prec} from the precedence graph G_{prec}. For each gripper aperture (a_{grasp} and a_{release}), collision checks are performed involving the robot body, the gripper, the target part o, the preceding parts O_{prec}, and the environment obstacles E. If any collision occurs between them, the grasp g is not feasible. Additional collision checks are performed between the gripper and other non-preceding parts $O_{\text{all}} \setminus O_{\text{prec}}$, and collision information is added to the grasp. Such collisions are not hard constraints because the non-preceding parts may get assembled later than the target part depending on the specific assembly sequence. In this case, the collision does not matter, but the information has to be recorded to find collision-free assembly sequences in the later stage.

Assembling Grasp Feasibility Check Alg. 3 determines the feasibility of using a grasp g to disassemble a target part o along its disassembly path τ_o, derived from G_{prec}. For each pose $p_{o,t}$ along τ_o, the grasp is transformed accordingly and its feasibility is validated using Alg. 2. The aggregated collision and IK information across all poses is stored in g. The grasp is feasible if all poses along τ_o are validated.

Algorithm 2 Single-Pose Grasp Feasibility Check

```
 1: Input: Grasp $g$, robot $R$, target part $o$ with pose $p_o$, all parts $o \in O$ with their corresponding
    goal poses $p_o^G$, precedence graph $G_{\text{prec}}$
 2: Output: Feasibility (True/False), updated grasp $g$ with collision and IK information
 3: $O_{\text{prec}} \leftarrow G_{\text{prec}}$.PrecedingParts$(o)$
 4: $E \leftarrow$ environment obstacles
 5: $q_g^R \leftarrow \text{IK}(R, g, o, p_o)$            ▷ IK for robot to grasp part $o$ under pose $p_o$ with grasp $g$
 6: if feasible $q_g^R$ is found then
 7:     Set robot $R$ configuration to $q_g^R$
 8:     for each gripper aperture $a \in \{a_{\text{grasp}}, a_{\text{release}}\}$ do
 9:         Set gripper with aperture $a$
10:         if CheckCollision$(R, o, O_{\text{prec}}, E)$ then
11:             return False, $g$
12:         CheckCollision$(R, O \setminus O_{\text{prec}})$
13:     Record collision and IK information to $g$
14:     return True, $g$
15: else
16:     return False, $g$
```

Algorithm 3 Assembling Grasp Feasibility Check

```
 1: Input: Grasp $g$, robot $R$, target part $o$, all parts $O$ with goal poses $p_O^G$, precedence graph $G_{\text{prec}}$
 2: Output: Feasibility (True/False), updated grasp $g$ with collision and IK information
 3: $\tau_o \leftarrow G_{\text{prec}}$.GetPath$(o)$                          ▷ Disassembly path of part $o$
 4: for each part pose $p_{o,t} \in \tau_o$ do                    ▷ Disassembling $o$ while grasping $o$
 5:     Transform grasp $g$ according to $p_o^t$ to obtain $g_t$
 6:     feas, $g_t \leftarrow$ CheckGraspFeas($g_t$, $o$, $p_{o,t}$, ...)                                  ▷ Alg. 2
 7:     if not feas then
 8:         return False, $g$
 9:     Gather collision and IK information from $g_t$ to $g$
10: return True, $g$
```

Holding Grasp Feasibility Check Alg. 4 evaluates whether a grasp g can securely hold a part o while allowing other parts o_i to be disassembled. The feasibility of g is first validated using Alg. 2.

Algorithm 4 Holding Grasp Feasibility Check

```
 1: Input: Grasp $g$, robot $R$, target part $o$, all parts $O$ with goal poses $p_O^G$, precedence graph $G_{\text{prec}}$
 2: Output: Feasibility (True/False), updated grasp $g$ with collision and IK information
 3: feas, $g \leftarrow$ CheckGraspFeas($g$, $o$, $p_o^G$, ...)                                   ▷ Alg. 2
 4: if not feas then
 5:     return False, $g$
 6: for each part $o_i \in O_{\text{all}} \setminus \{o\}$ do
 7:     $q_{o_i} \leftarrow G_{\text{prec}}$.GetPath$(o_i)$                          ▷ Disassembly path of $o_i$
 8:     for each part pose $p_{o_i,t} \in q_{o_i}$ do                ▷ Disassembling $o_i$ while grasping $o$
 9:         CheckCollision($R$, $o_i$) and gather collision information to $g$
10: return True, $g$
```

Grasp Pair Filtering Putting the assembling and holding feasibility checks together, Alg. 5 generates and filters the dual-arm grasp pairs. For each part o_i, a set of candidate grasp poses $\{g_k\}_{k=1}^{N_g}$ is generated. Each grasp is evaluated for assembling and holding feasibility using Algs. 3 and 4, respectively, and feasible grasps are stored in $\mathcal{G}^a[o_i]$ and $\mathcal{G}^h[o_i]$. Finally, iterating through all assembly-hold part pairs, the set of feasible assembly-hold grasp pairs $\mathcal{G}^{a\times h}[o_a, o_h]$ only contains those that do not lead to collisions between the two robots.

Algorithm 5 Dual-Arm Grasp Pair Filtering

```
1: **Input:** All parts $O$ with goal poses $p_O^G$, robots $R_a, R_h$, precedence graph $G_{\text{prec}}$
2: **Output:** Assembling grasps $\mathcal{G}^a$, holding grasps $\mathcal{G}^h$, assembling-holding grasp pairs $\mathcal{G}^{a\times h}$
3: Initialize empty dictionaries $\mathcal{G}^a, \mathcal{G}^h, \mathcal{G}^{a\times h} \leftarrow \{:\}, \{:\}, \{:\}$
4: **for each** part $o_i$ in $O$ **do**                                  ▷ Feasible grasp generation
5:     $\mathcal{G}^a[o_i], \mathcal{G}^h[o_i] \leftarrow \{\}, \{\}$
6:     Generate $N_g$ grasp poses $\{g_k\}_{k=1}^{N_g}$ on part $o_i$
7:     **for each** grasp pose $g_k$ **do**
8:         $\text{feas}_a, g_a \leftarrow \text{CheckAssemGraspFeas}(g_k, R_a, o_i, ...)$
9:         **if** $\text{feas}_a$ **then**
10:            $\mathcal{G}^a[o_i] \leftarrow \mathcal{G}^a[o_i] \cup \{g_a\}$
11:        $\text{feas}_h, g_h \leftarrow \text{CheckHoldGraspFeas}(g_k, R_h, o_i, ...)$
12:        **if** $\text{feas}_h$ **then**
13:            $\mathcal{G}^h[o_i] \leftarrow \mathcal{G}^h[o_i] \cup \{g_h\}$
14: **for each** part $o_a \in O$ **do**                              ▷ Feasible grasp pair filtering
15:     **for each** part $o_h \in O$ **do**
16:         **if** $o_a \in G_{\text{prec}}.\text{PrecedingParts}(o_h)$ **then**
17:             **continue**
            $\mathcal{G}^{a\times h}[(o_a, o_h)] \leftarrow \{\}$
18:         **for each** grasp $g_a \in \mathcal{G}^a[o_a]$ **do**
19:             **for each** grasp $g_h \in \mathcal{G}^h[o_h]$ **do**
20:                 Set robot $R_a$ configuration to $q_{g_a}^{R_a}$
21:                 Set robot $R_h$ configuration to $q_{g_h}^{R_h}$
22:                 **if not** $\text{CheckCollision}(R_a, R_h)$ **then**
23:                     $\mathcal{G}^{a\times h}[(o_a, o_h)] \leftarrow \mathcal{G}^{a\times h}[(o_a, o_h)] \cup \{(g_a, g_h)\}$
24: **return** $\mathcal{G}^a, \mathcal{G}^h, \mathcal{G}^{a\times h}$
```

C.3 Dual-arm Sequence-Grasp Optimization

Alg. 6 provides the pseudocode for the sequence-grasp optimization algorithm, followed by the formulas used to evaluate the transition edge cost.

Objective evaluation

- Maximizing the number of supportive parts held (f_1): Part A is supportive to part B if A is in the preceding parts of B in G_{prec}.
- Minimizing the number of holding grasp transitions (f_2): Holding grasp transitions can be counted simply by comparing whether the holding grasps in consecutive steps are the same.
- Minimizing approximated torque for assembling grasps (f_3):

$$\frac{\|\boldsymbol{\tau}_{\text{part}} + \boldsymbol{\tau}_{\text{grasp}}\|}{N_{\text{grasp}}} \tag{4}$$

  Where:

$$\boldsymbol{\tau}_{\text{part}} = \sum_{i=1}^{N_{\text{part}}} (\mathbf{r}_i - \mathbf{c}_{\text{part}}) \times \frac{\mathbf{d}_{\text{contact}}}{N_{\text{part}}}$$

$$\boldsymbol{\tau}_{\text{grasp}} = \sum_{j=1}^{N_{\text{grasp}}} (\mathbf{r}_j - \mathbf{c}_{\text{part}}) \times \frac{-\mathbf{d}_{\text{contact}}}{N_{\text{grasp}}}$$

  Where:

  - $\mathbf{r}_i, \mathbf{r}_j$ are the position vectors of contact points on the part and grasp, respectively.
  - $\mathbf{c}_{\text{part}}$ is the center of mass of the part.

Algorithm 6 Dual-Arm Assembly Sequence Planning

```
1: Input: All parts $O$, precedence graph $G_{\text{prec}}$, assembling grasps $\mathcal{G}^a$, assembling-holding grasp pairs $\mathcal{G}^{a\times h}$
2: Output: Optimal assembly part-grasp sequence $S^*$
3: Initialize an empty tree $T_G \leftarrow$ DiGraph()
4: S ← []                                                    ▷ Initialize an empty search stack.
5: for each $o \in O$ do
6:     if $G_{\text{prec}}$.SucceedingParts$(o) = \emptyset$ then
7:         for each $g \in \mathcal{G}^a[o]$ do
8:             $T_G$.AddNode($(a, O \setminus \{o\}, o, g)$)                         ▷ Root nodes
9:             push(S, $(a, O \setminus \{o\}, o, g)$)
10: while S not empty do
11:     $(t_i, O_i, o_i, g_i) \leftarrow$ pop(S)
12:     $t_{i+1} \leftarrow h$ if $t_i = a$ else $t_{i+1} \leftarrow a$
13:     for each $o_{i+1} \in O_i$ do
14:         $O_{i+1} \leftarrow O_i \setminus \{o_i\}$ if $t_{i+1} = a$ else $O_{i+1} \leftarrow O_i$
15:         if $(O_i \cup \{o_i\}) \cap G_{\text{prec}}$.PrecedingParts$(o_{i+1}) \neq \emptyset$ then
16:             continue                                                   ▷ Precedence check
17:         for each $g_{i+1} \in \mathcal{G}^{t_{i+1}}[o_{i+1}]$ do
18:             $O_c \leftarrow \mathcal{G}^{t_{i+1}}[o_{i+1}][g_{i+1}]$.CollidingSet         ▷ Precomputed collision set
19:             if $O_c \cap O_{i+1} \neq \emptyset$ then
20:                 continue                                               ▷ State collison check
21:             if $(g_{i+1}, g_i) \in \mathcal{G}^{t_{i+1}\times t_i}[(o_{i+1}, o_i)]$ then
22:                 $T_G$.AddEdge(($(t_i, O_i, o_i, g_i)$, $(t_{i+1}, O_{i+1}, o_{i+1}, g_{i+1})$))    ▷ Grasp pair validity check
23:                 push(S, $(t_{i+1}, O_{i+1}, o_{i+1}, g_{i+1})$)
24: for each $e_i \in T_G$.Edges() do                                  ▷ Objective evaluation
25:     $((t_i, O_i, o_i, g_i), (t_j, O_j, o_j, g_j)) \leftarrow e_i$
26:     if $t_i = a, t_j = h$ then                                     ▷ Assembling-holding step
27:         $e_i.\vec{f} \leftarrow \vec{f}(O_i, o_i, g_i, o_j, g_j)$
28:     else                                                           ▷ Holding transition step
29:         $e_i.\vec{f} \leftarrow \vec{0}$
30: $S^* \leftarrow [o^*_{a,1}, g^*_{a,1}, o^*_{h,2}, g^*_{h,2}, ..., o^*_{a,n}, g^*_{a,n}] \leftarrow$ DP$(T_G, \Phi)$     ▷ Dynamic programming
31: return $S^*$
```

  - $\mathbf{d}_{\text{contact}}$ is the contact direction vector.
  - N_{grasp} and N_{part} are the number of contact points for the grasp and part, respectively.

- Maximizing part contact area for holding grasps, weighted by the orientation difference from the paired assembling grasps (f_4):

$$\Delta\theta_{\text{rotation}} \times N_{\text{hold}} \tag{5}$$

  Where:

$$\Delta\theta_{\text{rotation}} = \left\|\mathbf{R}_{\text{hold}}^{-1} \cdot \mathbf{R}_{\text{assemble}}\right\|$$

  Where:

  - $\Delta\theta_{\text{rotation}}$ represents the angular difference between the holding and assembling grasps.
  - $\mathbf{R}_{\text{hold}}$ and $\mathbf{R}_{\text{assemble}}$ are the rotation matrices derived from the quaternions of the holding and assembling grasps, respectively.
  - N_{hold} is the number of contact points in the holding grasp.

C.4 Grasp-Oriented Pickup Fixture Generation

The fixture generation process begins with the computation of an appropriate pickup pose for each part. Our goal is to enforce a top-down pickup grasp without requiring re-grasping during the transition from pickup to assembly. Therefore, we can derive the pickup orientation of each part given the final assembled pose of them and the optimal grasp planned at the assembled pose, since we maintain the same relative transformation between the part and the gripper from pickup to assembly.

Next, we determine the pickup position for all parts. Since all pickup motions follow a top-down path, parts are arranged to prevent any overlap along the Z-axis, the vertical direction in the world coordinate frame. This constraint ensures unobstructed pickup paths and simplifies the design of the supporting fixture. Along the Z-axis, parts are positioned at the lowest possible height while ensuring that there are no collisions with the ground based on their orientation. In addition, a minimum base height is maintained for the fixture board to provide structural stability and support.

Determining the XY positions of the parts is more challenging, as the layout directly impacts the fixture area, which should ideally be minimized to reduce material costs for fixture fabrication and maximize the available workspace within the workcell. Additionally, incorrect part layout on the XY plane can lead to potential collisions between the gripper and the remaining parts during pickup. Since the orientation of each object is locked, we can represent each part using a rectangular bounding box of its 2D-projected contour. Then, the problem becomes a 2D bin-packing problem, a classic problem with both heuristic and exact algorithms exist [51]. We use a simple algorithm that iterates through the following phases:

1. We use the Maximal Rectangles algorithm [52] to pack the bounding boxes into an initial bounding area;
2. We check the collisions between the gripper and the precedent parts of the grasped part;
3. If any collisions are detected, the colliding parts are buffered with additional spacing and the bin-packing process is performed again;
4. We increase the bin area once it is not enough to find a packing solution given the increased rectangle sizes.

Once an optimal packing configuration is determined, the fixture is generated by creating mold cavities that accommodate the part shapes. A minimal mold depth is calculated to ensure gravitational stability of part placement, where the Z-axis projection of each part's center of mass lies within the convex hull of the contact points between the fixture and the part. The fixture cavity is generated by projecting the part's 3D geometry onto a 2D plane perpendicular to the pickup direction and extruding it to the calculated depth. Additional cavity is generated by creating free space for the grasp motion for every part based on the gripper geometry and the grasping motion. Finally, the generated fixture is enhanced with countersunk pads to assist in accurate positioning. By automating the entire fixture generation process, our approach provides a flexible and scalable solution for diverse part geometries and assembly sequences.

Appendix D Experimental Setup

D.1 Hardware Setup

We conduct real-world experiments using a dual-arm setup composed of two Franka Emika Panda robots, each equipped with parallel-jaw grippers. The arms are mounted on one side of a shared table, facing the user, and their relative positions are calibrated via a common reference frame. The workspace is divided into a pickup area and an assembly area, both fixed and pre-defined based on the available workspace area. The system uses internal encoders for joint sensing, without external force-torque sensors or visual feedback.

D.2 Simulation Environment

We use RedMax, the same simulator used in Tian et al. [11], for simulation-based planning, and Isaac Gym for reinforcement learning. Grasp feasibility is determined by sampling 100 candidate grasps per part using an antipodal grasp planner, followed by inverse kinematics validation and collision checking in RedMax. Precedence tier planning uses a physics-based disassembly planner [10] with orthogonal force directions for motion planning.

D.3 Training Configuration

Assembly policies are trained using PPO from the RL Games framework with key hyperparameters for RL presented in Table 1, which we use for all reported RL experiments. Our generalist policies are trained for a maximum of 5×10^7 steps (or equivalently 1500 iterations) with a parallel rollout setup using 1024 environments, which takes less than 2 hours on a single NVIDIA RTX A6000 GPU.

Table 1: Key RL Hyperparameters.

Parameter	**Value**
Algorithm Name	PPO
MLP Units	[256, 128, 64]
MLP Activation	elu
Learning Rate	1e-4
Gamma	0.99
Tau	0.95
Entropy Coefficient	0.003
Gradient Norm	1.0
Horizon Length	32
Minibatch Size	512
Mini Epochs	8
Critic Coefficient	2
KL Threshold	0.016

D.4 Real-World Deployment

Robot control alternates between executing planned transit motions in joint space and reactive policy execution for insertion steps. For policy execution, a task-space impedance controller is used with gains K_p = [800 N/m, 800 N/m, 400 N/m] and $K_d = 2\sqrt{K_p}$, transformed into the path-centric frame to ensure consistent compliance. Control runs at 30 Hz. During deployment, we use the leaky Policy-Level Action Integrator (leaky PLAI) scheme for improved stability with an action scale of 0.001 and an error threshold of 0.02. We use the same set of control parameters across all benchmark assemblies. Interventions are requested after three consecutive failures to insert, detected via joint deviation thresholds and motion stagnation.

Appendix E Ablation Studies

E.1 Impact of Coordinate and Action Design

We conducted ablation studies to evaluate the impact of coordinate frame selection (world vs. path-centric) and action representation (nominal vs. residual) on the assembly specialist (AS) policy's performance in simulation. Table 2 presents the average % of successful steps without intervention across 1024 randomized simulation evaluations for each assembly task under four configurations.

Overall, the combination of path-centric coordinates and residual actions proves particularly effective for assemblies involving diverse insertion directions, such as the Plumbers Block and Gamepad. Individually, each technique provides a structured prior that simplifies learning: path-centric coordinates standardize insertion directions by reorienting each assembly motion into a canonical frame, while residual actions leverages the planned trajectory and allow for fine-grained corrective adjustments on top of it. When applied together, they complement each other, enabling both directional consistency and precise control, leading to significantly higher success rates across most assemblies in our benchmark.

Table 2: Ablation studies on the impact of coordinate and action design choices on the assembly specialist (AS) policy's performance across 1024 randomized simulation evaluations.

Coordinate	Action	% of Successful Steps without Intervention (Simulation)						
		Beam	Plumber Block	Car	Gamepad	Cooling Manifold	Duct	Stool
World	Nominal	**99.71%**	49.32%	**73.24%**	76.95%	**94.92%**	**96.97%**	70.41%
World	Residual	**99.71%**	95.02%	71.09%	73.44%	**95.02%**	**97.75%**	74.80%
Path-Centric	Nominal	**99.71%**	93.95%	60.35%	85.35%	92.58%	**97.17%**	70.41%
Path-Centric	Residual	**99.12%**	**97.46%**	70.12%	**88.87%**	**95.02%**	**96.58%**	**76.66%**

Table 3: Scaling effect of number of policy trials across 3 complete end-to-end multi-step real-world evaluations.

Method	# Trials	% of Successful Steps without Intervention (Real World)						
		Beam	Plumber Block	Car	Gamepad	Cooling Manifold	Duct	Stool
Open-Loop Tracking	1	42%	25%	20%	20%	17%	14%	21%
Assembly Specialist Policy (AS)	1	58%	58%	60%	40%	67%	43%	75%
	2	67%	58%	80%	60%	67%	79%	75%
	3	75%	67%	87%	73%	83%	79%	88%

E.2 Effect of Number of Trials

In real-world experiments, we evaluate the assembly specialist policy with varying numbers of allowed trials (1, 2, or 3) per step, due to the stochastic nature of the policy. Table 3 summarizes the average % of successful steps without intervention across three end-to-end multi-step assembly runs. Results indicate that permitting additional trials substantially boosts success rates. The policy consistently improves as trial count increases, suggesting that the policy benefits from repeated attempts to refine alignment and correct minor positional errors.

The results further demonstrate open-loop baseline's inability to recover from failures and adapt to variations in the real world environment, even though the planned path is accurate and the real robot is well calibrated. In contrast, the RL policy consistently demonstrates improved progress, with notable gains observed as the number of trials increases. The results show a clear trend: with 1 trial per step, the policy can achieve moderate progress but still faces challenges in most of the assemblies. Introducing retries significantly enhance progress, enabling the system to correct minor errors and overcome small disturbances.

Appendix F Integrating Vision Feedback: VLM for Insertion Alignment

To further address insertion misalignments observed during the initial insertion attempt, we integrate a vision-language model (VLM) to provide corrective alignment feedback in the form of discrete actions. The model, a state-of-the-art version of Gemini (`gemini-2.5-pro-preview-03-25`), is leveraged to assess spatial alignment between the grasped part and the insertion hole based on visual input.

While training visuomotor policies directly from visual input is a common approach for alignment tasks, it requires extensive data collection, task-specific training, and continuous fine-tuning to generalize across diverse parts and insertion scenarios. In contrast, leveraging a VLM for alignment feedback offers significant advantages. VLMs are pre-trained on diverse visual contexts, enabling them to generalize across varied geometries and occlusions without extensive task-specific data collection. Additionally, they provide interpretable, discrete corrective actions (e.g., "move right") accompanied by concise explanations, enhancing both robustness and transparency in alignment tasks, especially under low-cost, fixed-focus camera setups.

F.1 Physical Setup

The vision integration requires only RGB input without high imaging quality, allowing for the use of low-cost cameras. We utilize an Arducam B0205 USB camera ($34.99), shown in Fig. 3, mounted on the robot wrist at an angle optimized to capture the insertion area. The camera is equipped with an IR-CUT filter and infrared LEDs for low-light conditions, but lacks focus adjustment, resulting in reduced image clarity when the insertion part is out of focus.

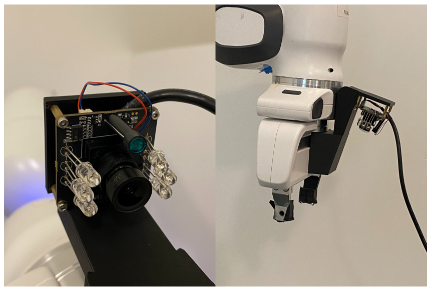


Figure 3: Physical setup for integerating vision feedback. Left: Camera details. Right: The mounted configuration on the robot wrist.

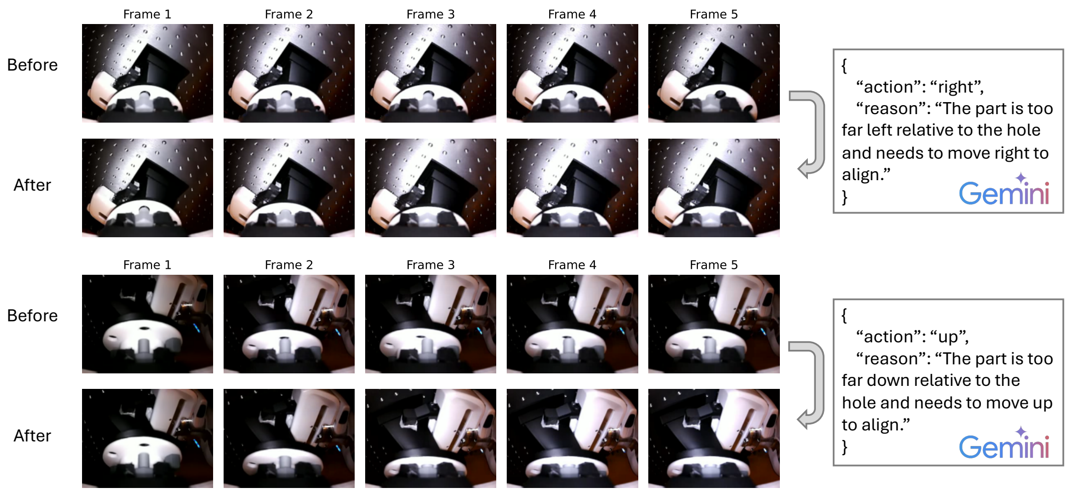
F.2 Methodology

If the insertion policy fails on the first attempt, the entire video of the failed attempt is recorded and segmented into ten key frames sampled uniformly across the video. These frames are passed to the VLM, which is prompted to recommend the best corrective action (`up`, `down`, `left`, or `right`) to align the part with the hole. The VLM is additionally prompted to provide a concise, step-by-step reasoning for the recommended action to mitigate potential hallucinations. The exact prompt we use is detailed below.

```
'''
You are assisting a robot in aligning a grasped part for insertion
    using visual feedback from a camera mounted on the robot's wrist.


Task:
- The part is grasped by the robot and can move in four directions: ["
    up", "down", "left", "right"], each by 2 mm in the camera frame.
- The goal is to move the part to align it precisely with the hole for
     insertion.


Instructions:
- Carefully observe the video frames. Focus only on the position of
    the part relative to the hole.
- Determine the single best action to move the part to align with the
    hole.
- Focus only on spatial cues: Is the part too far left, right, above,
    or below the hole?


Response format:
{
"action": "right",
"reason": "The part is too far left relative to the hole and needs to
    move right to align."
}


Only output the single best action based on spatial cues. If the part
    is already aligned, output "hold".
What is the best action to move the part to align with the hole?
'''
```

After receiving the VLM feedback, the robot executes the recommended action by adjusting the gripper in the task frame by 2 mm. The insertion policy is then restarted from this new position. If the insertion still fails, the entire process is repeated with the newly recorded video sequence, allowing for multiple VLM interventions as necessary.

F.3 Results and Analysis



```
{
  "action": "right",
  "reason": "The part is too far left relative to the hole and needs to move right to align."
}
```

```
{
  "action": "up",
  "reason": "The part is too far down relative to the hole and needs to move up to align."
}
```

Figure 4: Example outputs from VLM during corrective alignment. The VLM identifies spatial misalignments in the camera frame and recommends discrete corrective actions (e.g., "right" and "up") with concise reasoning.

The VLM integration demonstrated notable improvements in alignment accuracy, particularly in cases where the insertion policy initially failed due to misalignment. Figure 4 illustrates two representative examples where the VLM provided corrective actions that successfully guided the arm to the intended alignment.

In the first example, the VLM identified a leftward misalignment and recommended the action "right", allowing the part to be re-centered relative to the insertion hole. The corrective action was executed in the tool frame, resulting in a more precise alignment before the subsequent insertion attempt. Similarly, in the second example, the VLM detected a downward offset and suggested the action "up", effectively repositioning the part closer to the target insertion point. In both cases, a single VLM intervention was sufficient to resolve the misalignment, highlighting the model's capacity to reason spatially based on minimal visual input.

Despite the low-cost camera setup and lack of focus adjustment capabilities, the VLM effectively discerned alignment cues based on coarse visual features. This is particularly noteworthy given that occlusions and visual clutter are prevalent in multi-part assemblies, where small positional errors can accumulate over successive steps. The VLM's concise, reason-based output structure further mitigates hallucination risks by constraining the response format to a single action-reason pair, reducing ambiguity and enhancing interpretability.

F.4 Limitations and Future Work

While the VLM integration demonstrated significant alignment improvements, occlusions remained a primary failure mode. Occlusions are common in complex assemblies, necessitating either a flexible/active camera setup or multiple cameras strategically positioned to cover the scene comprehensively. Furthermore, hallucination remains a concern, particularly in cluttered scenes where visual cues are ambiguous. Future work will explore improving prompt engineering and camera configurations, potentially leveraging multi-view setups and active camera movements akin to human head and body movements.