A Appendix

A.1 Robot Setup

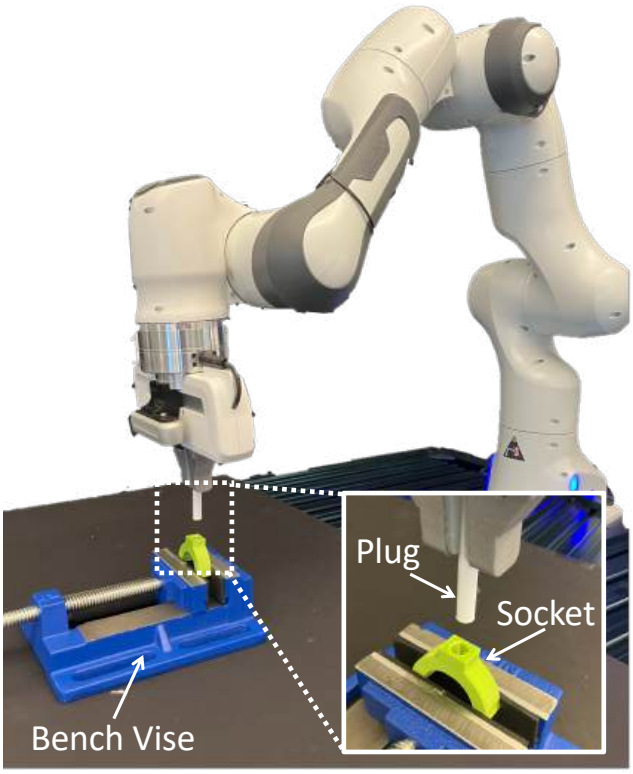


Figure 10: **Real-world experimental setup**. A Franka Panda robot and a bench vise are mounted to a tabletop. At the beginning of each episode, a 3D-printed plug is grasped by the robot gripper and and a 3D-printed socket is haphazardly placed in the bench vise. The task is to control the robot arm and fully insert the plug into the socket.

A.2 Motivation with Theoretical Perspective

Transferring knowledge from a source task to a target task can improve training efficiency and asymptotic performance. Consider a source task T_j and target task T_i, which are MDPs that share state space $\mathcal{S}$, action space $\mathcal{A}$, and reward function r, but have distinct transition functions p_i, p_j and initial state distributions ρ_i, ρ_j. To measure the transferability of a policy, we apply the same policy on both tasks and examine the difference in their expected values. Here we note that the value difference depends primarily on the difference in their transition functions p_i, p_j and initial state distributions ρ_i, ρ_j (Proposition 1).

Proposition 1. *Let $T_i = \{\mathcal{S}, \mathcal{A}, p_i, r, \gamma, \rho_i\}$ and $T_j = \{\mathcal{S}, \mathcal{A}, p_j, r, \gamma, \rho_j\}$ be two MDPs in the task space $\mathcal{T}$. Applying a policy π on T_i and T_j, we have a function f to describe the value difference:*

$$V^{\pi}(\rho_i, T_i) - V^{\pi}(\rho_j, T_j) = f(p_i - p_j, \rho_i - \rho_j)$$

Proof.

$$
\begin{aligned}
V^\pi(\rho_i, T_i) - V^\pi(\rho_j, T_j) &= \mathbb{E}_{s\sim\rho_i(\cdot)}\mathbb{E}_{a\sim\pi(\cdot|s)}Q^\pi(s,a,T_i) - \mathbb{E}_{s\sim\rho_j(\cdot)}\mathbb{E}_{a\sim\pi(\cdot|s)}Q^\pi(s,a,T_j) \\
&= \mathbb{E}_{s\sim\rho_i(\cdot)}\mathbb{E}_{a\sim\pi(\cdot|s)}[Q^\pi(s,a,T_i) - Q^\pi(s,a,T_j)] \\
&\quad + \mathbb{E}_{s\sim\rho_i(\cdot)}\mathbb{E}_{a\sim\pi(\cdot|s)}Q^\pi(s,a,T_j) - \mathbb{E}_{s\sim\rho_j(\cdot)}\mathbb{E}_{a\sim\pi(\cdot|s)}Q^\pi(s,a,T_j) \\
&= \mathbb{E}_{s\sim\rho_i(\cdot)}\mathbb{E}_{a\sim\pi(\cdot|s)}[Q^\pi(s,a,T_i) - Q^\pi(s,a,T_j)] \\
&\quad + \mathbb{E}_{s\sim\rho_i(\cdot)}V^\pi(s,T_j) - \mathbb{E}_{s\sim\rho_j(\cdot)}V^\pi(s,T_j) \\
&= \mathbb{E}_{s\sim\rho_i(\cdot)}\mathbb{E}_{a\sim\pi(\cdot|s)}[Q^\pi(s,a,T_i) - Q^\pi(s,a,T_j)] + \sum_s(\rho_i - \rho_j)V^\pi(s,T_j)
\end{aligned}
$$

For the Q-value difference $Q^\pi(s,a,T_i) - Q^\pi(s,a,T_j)$, we refer to the simulation lemma in (Agarwal et al., 2019):

$$
Q^\pi(T_i) - Q^\pi(T_j) = \gamma(I - \gamma P^\pi(T_j))^{-1}(p_i - p_j)V^\pi(T_i)
$$

where $P^\pi(T_j)$ denotes the transition matrix on state-action pairs induced by the policy π on the task T_j, i.e., $P^\pi_{(s,a),(s',a')}(T_j) = p_j(s'|s,a)\pi(a'|s')$.

Consequently, $Q^\pi(s,a,T_i) - Q^\pi(s,a,T_j)$ is the (s,a) item in the matrix $Q^\pi(T_i) - Q^\pi(T_j)$, and $Q^\pi(s,a,T_i) - Q^\pi(s,a,T_j)$ can be expressed as a function of $(p_i - p_j)$.

Overall, the value difference $V^\pi(\rho_i, T_i) - V^\pi(\rho_j, T_j)$ depends primarily on $(p_i - p_j)$ and $(\rho_i - \rho_j)$. □

Assume the reward function r is a sparse, binary term indicating task success at the end of an episode. The success rate of applying a policy π to a task T can be represented as $V^\pi(\rho) = \mathbb{E}_{s_0\sim\rho}\mathbb{E}_{\tau\sim p^\pi(\tau|s=s_0)}[\sum_{t=0}^\infty \gamma^t r_t]$. Here, our success rate $V^\pi(\rho_j, T_j)$ will naturally be high, because the source policy π is already an expert policy for the source task T_j. When the success rate of applying the source policy to target task T_i is also high, i.e., $V^\pi(\rho_i, T_i)$ is close to $V^\pi(\rho_j, T_j)$, then Proposition 1 implies that the transition functions p_i and p_j might be similar, as are the initial state distributions ρ_i and ρ_j. Consequently, if a source policy can achieve high zero-shot transfer success on a target task, the target task might have a similar transition function and initial state distribution as the source task. Hence, we hypothesize that fine-tuning the source policy on the target task will be efficient.

However, it is important to note that achieving a similarly high success rate on two tasks with a single policy does not necessarily indicate similar dynamics between the tasks. Proposition 1 establishes that similar dynamics and initial state distributions lead to similar expected values for a given policy, but the reverse is not guaranteed. We use the high transfer success rate as a heuristic indicator of similar dynamics, serving as intuitive motivation rather than strict theoretical justification.

A.3 METHOD

Algorithm 1 Policy finetuning with Self-imitation Learning

Initialize parameter θ for policy π_θ and value function V_θ with retrieved skill
Initialize replay buffer $\mathcal{D} \leftarrow \emptyset$
Initialize episode buffer $\mathcal{E} \leftarrow \emptyset$
for each iteration **do**
  # Collect training samples
  for each step **do**
    Execute an action $s_t, a_t, r_t, s_{t+1} \sim \pi_\theta(a_t|s_t)$
    Store transition $\mathcal{E} \leftarrow \mathcal{E} \cup \{(s_t, a_t, r_t)\}$
  end for
  if s_{t+1} is terminal **then**
    # Update replay buffer
    Compute returns $R_t = \sum_k^\infty \gamma^{k-t} r_k$ for all t in $\mathcal{E}$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, R_t)\}$ for all t in $\mathcal{E}$
    Clear episode buffer $\mathcal{E} \leftarrow \emptyset$
  end if
  # Update parameter θ using PPO objective
  $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}^{ppo}$ (Schulman et al., 2017)
  # Perform self-imitation learning
  for $m = 1$ to M **do**
    Sample a mini-batch $\{(s, a, R)\}$ from $\mathcal{D}$
    $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}^{sil}$
  end for
end for

Algorithm 2 Continual Learning with Skill Library Expansion

Require: Prior tasks $\mathcal{T}_{prior} = \{T_1, T_2, \cdots, T_n\}$; Skill library $\Pi_{prior} = \{\pi_1, \pi_2, \cdots, \pi_n\}$
while given newly coming batch of tasks $\mathcal{T}^j = \{T_1, T_2, \cdots, T_k\}$ **do**
  for each task T_i **do**
    Retrieve a policy π_{src} from the skill library Π_{prior}
    Finetune π_{src} to get a policy π_i solving the task T_i
    Expand the skill library, $\mathcal{T}_{prior} = \mathcal{T}_{prior} \cup \{T_i\}$; $\Pi_{prior} = \Pi_{prior} \cup \{\pi_i\}$
  end for
end while

A.4 Implementation Detail

A.4.1 Task Feature Learning in SRSA

Geometry Features As shown in Fig. 3(a), we employ a PointNet-based (Qi et al., 2017) autoencoder E_G and D_G to minimize the difference between input point cloud P and reconstructed point cloud $D_G(E_G(P))$. The autoencoder is trained using point clouds of parts from all tasks.

We follow the implementation details outlined in (Tang et al., 2024). In a large set of meshes M for various assembly parts, each mesh $m_i \in M$ consists of (V_i, E_i), where V denotes the vertices and E represents the (undirected) edges. During each training iteration, we sample a batch of meshes $B \subset M$. For each $m_i \in B$, we generate a point cloud P_i from the mesh, with each point located on the surface of m_i. The point cloud P_i ghdn is passed through a PointNet encoder (Qi et al., 2017) based on the implementation from (Mu et al., 2021) to produce a latent vector. The latent vector $z_{G,i}$ is subsequently fed into a fully-convolutional decoder, following the implementation from (Wan et al., 2023) to produce the reconstructed point cloud P_i'.

The network is trained to minimize reconstruction loss, defined here as the Chamfer distance between P_i and P_i':

$$\mathcal{L}_{CD} = \frac{1}{\|P_i\|} \sum_{p \in P_i} \min_{q \in Q_i} \|p - q\|_2^2 + \frac{1}{\|Q_i\|} \sum_{q \in Q_i} \min_{p \in P_i} \|p - q\|_2^2$$

Across 100 two-parts assembly tasks, we utilize a total of 200 meshes for the plug and socket components with $|M| = 200$. Each sampled point cloud P_i contains 2000 points and the dimension of learned embedding is $|z_{G,i}| = 32$. The autoencoder is trained for a total of 23,000 epochs, using a batch size of 64 and a learning rate of 0.001.

To represent the feature of one task, we gather the geometry features for the meshes of plug, socket, and the assembled state of the plug inserted in the socket. Therefore, the geometry feature of one task is concatenation of these three features, resulting in a dismensionality of, $|z_{G,i}| = 96$.

Dynamics Features We build upon prior work in context-based meta-RL (Rakelly et al., 2019; Lee et al., 2020) to utilize a context encoder E_D that produces a latent vector from transition segments $\tau_{t-1} = \{s_{t-h}, a_{t-h}, s_{t-h+1}, a_{t-h+1}, \cdots, s_{t-1}, a_{t-1}\}$, as shown in Fig. 3(b). We sample the transition segments from disassembly trajectories, compute the latent vector $E_D(\tau_{t-1})$, and feed the latent vector from transition segments to a forward dynamics model D_D across all tasks. For any transition samples from any task, the forward dynamics model is trained to predict the next state $s'_{t+1} = D_D(E_D(\tau_{t-1}), s_t, a_t)$ to be close to the ground-truth next state s_{t+1}.

As described in (Tang et al., 2024), for each task, we generate disassembly paths by initializing the robot hand to grasp the plug in the assembled state, where the plug is fully inserted in the socket. Using a low-level controller, we lift the plug from the socket and move it to a randomized pose. We repeat this process until collecting 100 successful disassembly trajectories. We store the state of end-effector position and the action of moving end-effector at each timestep in the disassembly trajectories. Each task has a total of 100 disassembly trajectories, with each trajectory spanning 128 timesteps.

We sample the transition segment $\tau_{t-1} = \{s_{t-h}, a_{t-h}, s_{t-h+1}, a_{t-h+1}, \cdots, s_{t-1}, a_{t-1}\}$ for $h = 10$ timesteps. The context encoder is modeled as multi-layer perceptrons (MLPs) with 3 hidden layers of sizes $(256, 128, 64)$, producing a 32-dimensional vector $z_{D,t}$. Then, the forward dynamics model D_D receives the context vector as an additional input, where the input consists of a concatenation of state s_t, action a_t, and context vector $z_{D,t}$. The forward dynamics model comprises four fully-connected layers of sizes $(200, 200, 200, 200)$ with ReLU activation functions, outputing the prediction of the next state s'_{t+1}. The objective is to minimize L2-distance between the ground-truth next state s_{t+1} and the predicted next state s'_{t+1}. For the entire set of disassembly trajectories across 100 tasks, we train the encoder and forward dynamics model for 200 epochs, using a batch size of 128 and a learning rate of 0.001.

Expert Action Features We utilize the disassembly trajectories as reverse expert demonstrations for assembly tasks and aim to capture expert action information in an embedding space. As illustrated in Fig. 3(c), we sample a transition segment τ_{t-1} from the disassembly trajectories, map it

to the action embedding $E_A(\tau_{t-1})$, and reconstruct the action sequence $\{a_{t-h}, a_{t-h+1}, \cdots, a_{t-1}\}$ using decoder D_A. We train both the encoder and decoder with transition segments from all tasks. This embedding effectively extracts the strategy for solving the task by reconstructing the expert actions from the disassembly trajectories.

We sample the transition segment $\tau_{t-1} = \{s_{t-h}, a_{t-h}, s_{t-h+1}, a_{t-h+1}, \cdots, s_{t-1}, a_{t-1}\}$ for 10 timesteps (i.e., $h = 10$). The action encoder E_A is modeled as multi-layer perceptrons (MLPs) with three hidden layers of sizes $(256, 128, 64)$, producing a 32-dimensional vector $z_{A,t}$. The action decoder D_A is an MLP with four hidden layers of sizes $(200, 200, 200, 200)$ that predicts the sequence of expert actions $\{a'_{t-h}, a'_{t-h+1}, \cdots, a'_{t-1}\}$. We minimize the L2-distance between input action sequence $\{a_{t-h}, a_{t-h+1}, \cdots, a_{t-1}\}$ and the reconstructed action sequence $\{a'_{t-h}, a'_{t-h+1}, \cdots, a'_{t-1}\}$. The encoder and decoder are trained for 200 epochs, using a batch size of 128 and a learning rate of 0.001.

A.4.2 Transfer Success Prediction in SRSA

We learn the function $F(\pi_{src}, T_{trg})$ to predict the transfer success. For any pair of source policy and target task in the skill library, we execute the source policy in the target task for 1000 episodes and average the success rate to obtain the ground-truth label for F. For any task T in the prior task set, we sample the point cloud P_i of plug, socket and assembly state to extract the geometry feature $z_{G,i}$ with a dimension of 96. Then we sample transition segment τ_i to obtain the dynamics feature $z_{D,i}$ with a dimension of 32 and action feature $z_{A,i}$ with a dimension of 32. By concatenating these features, we create a task feature z_i with a dimension of 160 for the sampled point clouds and transition segment. With both the task features $z_{src,i}$ and $z_{trg,i}$ for source and target tasks, we feed them into an MLP with one hidden layer of size 128 to predict the transfer success. We optimize the MLP while jointly finetuning the feature encoders E_G, E_D, and E_A to learn the transfer success prediction. The training is conducted for 50 epochs across all source-target pairs in the prior task set.

A.4.3 Baselines of Skill Retrieval Approaches

Signature : path signature can represent trajectories as a collection of path integrals and also quantify distances between trajectories. Inspired by (Tang et al., 2024), we find the closest path signature for skill retrieval. For each disassembly trajectory τ_k on the target task T, we calculate the path signature z_k and search all disassembly trajectories over all source tasks to identify a source disassembly trajectory τ_j with the path signature z_j closest to z_k. The source disassembly trajectory τ_j belongs to a source task in $\mathcal{T}_{prior}$, and thus we match the target trajectory τ_k to this source task, denoted as T_k. We count the times that one source task $T_{src} \in \mathcal{T}_{prior}$ is assigned as the source task for a target disassembly trajectory, $C(T_{src}) = \sum_{k=1}^{n} [T_k = T_{src}]$. Then we retrieve the source policy for one source task with the highest count, i.e. $\arg\max_{T_{src}} C(T_{src})$

Behavior : Inspired by (Du et al., 2023), we employ state-action pairs on disassembly trajectories across all tasks and learn a state-action embedding with a VAE for skill retrieval. For any state-action pair (s_k, a_k) on the target task, we infer the embedding $z_{sa,k}$ and look for one state-action pair (s_j, a_j) from the disassembly trajectories in source tasks with the embedding $z_{sa,j}$ closest to $z_{sa,k}$. The target state-action pair (s_k, a_k) is matched to one source task, which (s_j, a_j) belongs to. We denote this source task as T_k. Similar to the method above, we count the times that one source task $T_{src} \in \mathcal{T}_{prior}$ is assigned as the source task for a target state-action pair, $C(T_{src}) = \sum_{k=1}^{n} [T_k = T_{src}]$. Then we retrieve the source policy for one source task with the highest count, i.e. $\arg\max_{T_{src}} C(T_{src})$

Forward : As explained above, we learn the latent vector for transition sequence τ on disassembly trajectories. In order to retrieve one source task according to the distances between task embeddings, we average embedding for all transition sequences from the same task to obtain the task embedding, similar to (Guo et al., 2022). We retrieve the policy for the source task that has the closest task embedding.

Hyperparameters	Value
Policy Network Architecture	[256, 128, 64]
Value Function Architecture	[256, 128, 64]
LSTM network size	256
Horizon length (T)	32
Adam learning rate	1e-4
Discount factor (γ)	0.99
GAE parameter (λ)	0.95
Entropy coefficient	0.0
Critic coefficient	2
Minibatch size	8192
Minibatch epochs	8
Clipping parameter (ϵ)	0.2
LSTM network size	256
SIL update per iteration	1
SIL batch size	8192
SIL loss weight	1
SIL value loss weight (β)	0.01
Replay buffer size	10^5
Exponent for prioritization	0.6

Table 1: Hyperparameters in PPO and Self-imitation learning

Geometry : As explained above, we learn an autoencoder for the point clouds of the assembly assets to minimize the reconstruction loss, as conducted in (Tang et al., 2024). We retrieve the policy for the source task with the closest point-cloud embedding.

A.4.4 Skill Adaptation in SRSA

Following (Tang et al., 2024), we use PPO to train the stochastic policy π_θ (i.e., actor) and an approximation of the value function V_θ (i.e., critic), parameterized by a neural networks with weights θ. While the policy is stochastic following a multivariant normal distribution with the learned mean and standard deviation, at evaluation and deployment time, the action output from well-trained policy is deterministic.

The input state for the policy network consists of the robot arm's joint angles, the end-effector pose, the goal end-effector pose, and the relative pose of the end effector to the goal. The state has a dimensionality of 28.

Due to the asymmetric actor-critic strategy, the states provided to the value function include privileged information not available to the policy. The states for the critic include joint velocities, end-effector velocities, and the plug pose, resulting in an input dimensionality of 44 for the value function.

The action space consists of incremental pose targets, representing the position and orientation differences between the current pose and the target pose. We use incremental targets instead of absolute targets to restrict selection to a small, bounded spatial range. The action dimensionality is 6.

SRSA combines PPO with a self-imitation learning mechanism for policy fine-tuning. We maintain a replay buffer $\mathcal{D}$ for transitions encountered during training, defined as $\mathcal{D} = \{s_i, a_i, R_i\}$. The data samples in the buffer are prioritized based on the discounted accumulated reward.

As shown in Algorithm 1, each iteration includes one PPO update for the policy and value function, along with a batch sampling from $\mathcal{D}$ to perform one self-imitation learning update. This update aims to minimize the loss function $\mathcal{L}_{sil}$ defined in Sec. 4.2. For details on network architectures and hyperparameters, refer to Tab. 1.

We follow prior work to use object poses rather than visual observations as input to the policy. Incorporating vision-based observations would introduce additional challenges for zero-shot sim-to-real transfer, as it requires a camera. In contrast, the current policy only relies on the fixed socket pose and the robot's proprioceptive features (including the end-effector pose), making it more straightforward to execute the policy in real-world settings.

Using visual observations or object pose is orthogonal to our proposed method (i.e., SRSA is independent of the observation modality). The high-level idea of retrieving a relevant skill and fine-tuning the retrieved policy remains applicable in scenarios involving vision-based policies. The geometry features derived from point clouds in our task representation can partially capture visual similarities between tasks. This enables the retrieval of source tasks that are visually similar to the target task to some degree.

At the same time, SRSA may require modifications to better support vision-based policies. Here is no guarantee that the retrieved source and target tasks are visually similar enough and the features extracted by the vision encoder in policy might differ significantly on source and target tasks. This could pose challenges for fine-tuning the policy on the target task.

To address this, we consider two distinct directions: 1. how to perform retrieval to better account for visual similarity; 2. how to train specialist policies with visual encoders such that the current SRSA retrieval strategy is still likely to work. Below, we propose specific approaches for these two directions.

1. Enhancing retrieval by incorporating features from visual observations: For example, integrating a Variational Autoencoder (VAE) to extract features from visual observations (as in BehaviorRetrieval (Du et al., 2023)) and combining these with other task representations might improve the retrieval process. Additionally, learning dynamics features, such as predicting future visual observations, could implicitly encode relevant visual information in task features for retrieval.

2. Improving the robustness of the visual encoder in policy: Training the specialist source policy with significant data augmentation (e.g., randomizing colors, poses, backgrounds, etc.) could make the visual encoder in the source policy more robust to diverse visual observations. It is more likely to extract similar features from geometrically similar tasks. Alternatively, leveraging state-of-the-art visual foundation models (e.g., DINOv2 (Oquab et al., 2023)) as visual encoders in specialist policies could further enhance generalization and robustness. These models have demonstrated strong performance in handling diverse observations and sim-to-real challenges, as shown in PoliFormer (Zeng et al., 2024). Consequently, we believe that features extracted by such visual encoders are likely to remain consistent for visual observations across geometrically similar tasks.

A.5 Experiments

A.5.1 Skill Retrieval

We first replicate the specialist policy learning for 100 assembly tasks as described in (Tang et al., 2024). Then, these 100 tasks are split into 90 prior tasks and 10 test tasks. For the 90 prior tasks, we use the well-trained specialist policies to build the skill library.

We train the skill-retrieval method on the prior tasks and evaluate its performance on the test tasks. In Fig. 4 in main text, 11, and 12 in Appendix, we present the test results for three different ways of splitting the 100 tasks. Overall, SRSA demonstrates superior performance in identifying relevant policies from the skill library, achieving a high success rate in zero-shot transfer.

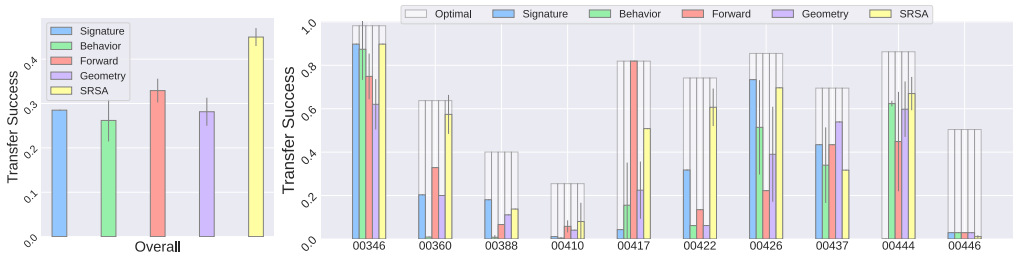


Figure 11: **Transfer success of retrieved skills applied to test tasks**. For each of the test tasks, we retrieve a policy from the prior skill library using 5 different approaches. For each approach, if it involves training neural networks, we train it for 3 random seeds. **Left**: we illustrate the mean result over 10 test tasks. **Right**: For each test task, we show the mean and standard deviation of transfer success over 3 seeds. Overall, SRSA clearly outperforms baselines.

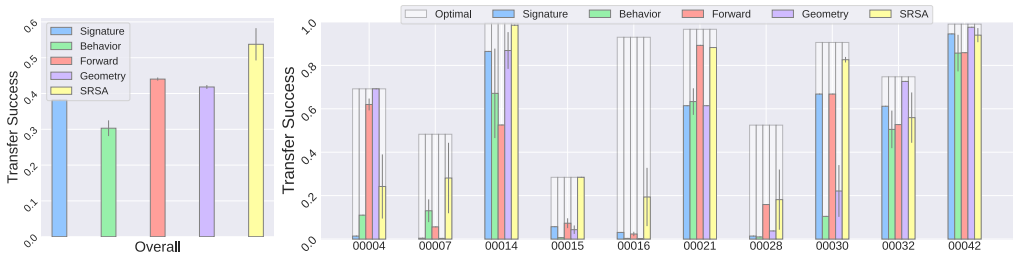


Figure 12: **Transfer success of retrieved skills applied to test tasks**. For each of the test tasks, we retrieve a policy from the prior skill library using 5 different approaches. For each approach, if it involves training neural networks, we train it for 3 random seeds. **Left**: we illustrate the mean result over 10 test tasks. **Right**: For each test task, we show the mean and standard deviation of transfer success over 3 seeds. Overall, SRSA clearly outperforms baselines.

A.5.2 Skill Adaptation

We show the learning curves in Fig. 5 in main text. At the end of 1000 training epochs, we record the success rate of the learned policies on 10 test tasks. For AutoMate, the policies are learned from scratch using PPO on the 10 test tasks. In contrast, SRSA initializes the policies with retrieved skills and fine-tunes them using PPO combined with self-imitation learning. The retrieval mechanism is trained on a skill library of 90 prior tasks, where the skills were pre-trained by AutoMate.

Compared to the baseline success rate of 69.4%, SRSA achieves a significantly higher success rate of 84.7%, corresponding to an absolute improvement of 15.3 percentage points and a relative improvement of approximately 22.0%. By leveraging the knowledge from the skill library, SRSA also obtains 3.7x lower standard deviation compared to AutoMate (Tab. 2). This advantage becomes even more pronounced in sparse-reward scenarios, where SRSA shows an absolute improvement of 41.9 percentage points and a relative improvement of 139% in comparison with baseline. (Tab. 3).

Task ID	01029	01036	01041	01053	01079	01092	01102	01125	01129	01136	Average
AutoMate	53.4 (27.4)	89.0 (7.7)	79.1 (8.4)	49.1 (15.3)	74.3 (32.9)	59.4 (13.1)	76.4 (11.4)	49.6 (3.2)	76.0 (3.0)	87.3 (4.2)	69.4 (12.7)
SRSA	98.5 (0.4)	91.3 (6.0)	83.3 (4.4)	75.4 (6.4)	93.60 (3.6)	78.3 (6.3)	92.5 (0.5)	50.6 (1.6)	85.8 (4.0)	98.4 (0.4)	84.7 (3.4)

Table 2: **Mean (standard deviation) of success rate (%) on each test task, in dense-reward setting**. We calculate the mean and standard deviation over 5 runs of different random seeds, at the last training epoch (i.e. 1000 epochs).

Task ID	01029	01036	01041	01053	01079	01092	01102	01125	01129	01136	Average
AutoMate	61.3 (26.5)	37.2 (31.4)	14.4 (1.6)	0 (0.5)	81.7 (15.1)	0 (0.5)	1.4 (1.0)	9.8 (2.0)	55.6 (6.0)	39.7 (5.4)	30.1 (9.0)
SRSA	95.1 (1.1)	78.7 (8.9)	33.7 (6.4)	92.5 (2.2)	96.1 (1.7)	51.4 (5.5)	70.7 (2.9)	51.2 (9.3)	90.3 (7.2)	60.5 (2.6)	72.0 (4.8)

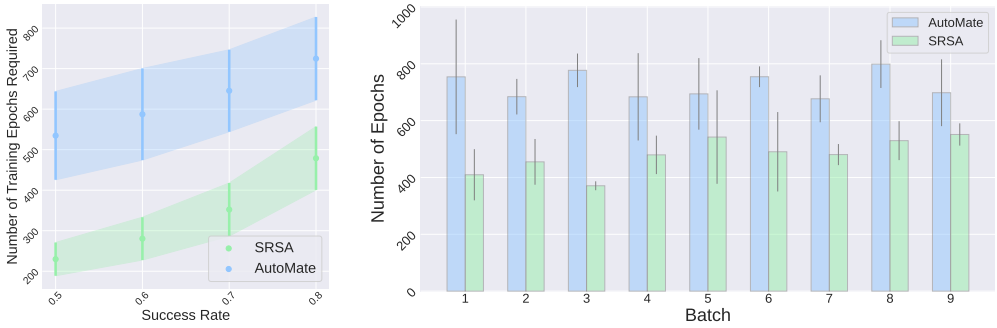
Table 3: **Mean (standard deviation) of success rate (%) on each test task, in sparse-reward setting**. We calculate the mean and standard deviation over 5 runs of different random seeds, at the last training epoch (i.e., 1000 epochs).

A.5.3 Continual Learning

We begin with an initial skill library containing 10 policies and expand its size by 10 policies per round over 9 rounds, eventually reaching 100 policies. When the skill library contains fewer than 40 policies, the number of source-target task pairs from the prior task set is limited. During this phase,

we retrieve skills solely based on geometry embeddings. Once the skill library reaches 40 or more policies, we train the transfer success prediction function F to guide skill retrieval for new tasks.

In the continual learning setting, Fig. 8 in main text and Fig. 13 in Appendix show the efficiency of SRSA and AutoMate under two different task batch orderings. In both cases, SRSA demonstrates significantly better sample efficiency compared to AutoMate.



(a) (b)

Figure 13: **(a) Sample efficiency of policy learning in a continual-learning setting**. We report how many training epochs are required to reach desired success rates (0.5, 0.6, 0.7, 0.8). We calculate the mean and standard deviation of training epochs over 5 runs, and report the average over 90 tasks.**(b) Number of training epochs required for different batches**. In the continual-learning scenario, we proceed through 9 batches of new tasks for policy learning, with each batch containing 10 new tasks. For each batch, we show the mean and standard deviation of training epochs required to reach a success rate of 0.8. SRSA requires less number of training epochs to reach a good success rate.

Additionally, we compare SRSA and AutoMate based on the best checkpoint, measured by the highest rewards achieved over 5 runs for each task. In our replication of AutoMate, we achieved an average success rate of 70% across 100 assembly tasks, which is lower than the 80% reported in the original paper. This discrepancy may be due to differences in simulator versions, asset meshes, implementation details, and other factors.

On average, SRSA achieves a success rate of 79% in Fig. 14 and 73% in Fig. 15, for two cases of task ordering, respectively. SRSA demonstrates a higher success rate and better sample efficiency than the baseline AutoMate.

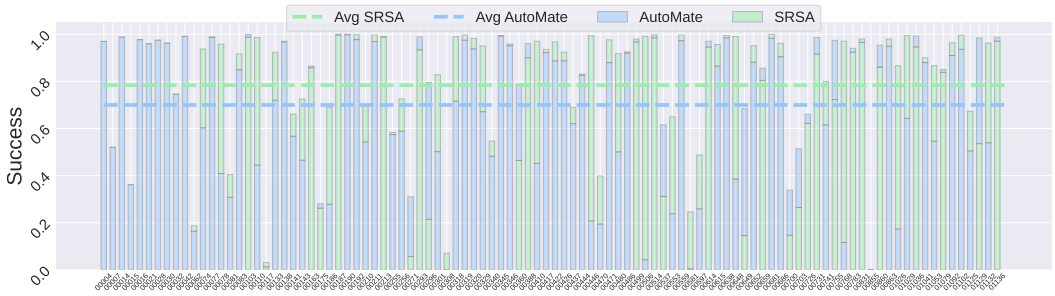


Figure 14: **Comparison of SRSA and AutoMate success rate over 100 tasks**. We replicate the specialist policy learning in the AutoMate paper over all tasks, and run SRSA with the continual-learning approach to train 90 specialist policies with initial skill library of 10 policies. For both approaches, for each task, we select the best checkpoint among 5 runs with different random seeds. We compare the success rate on all the tasks. On average, SRSA achieves a higher success rate.

A.5.4 Ablation Study

Fig. 16 illustrates the learning curves of different SRSA variations across 10 test tasks.

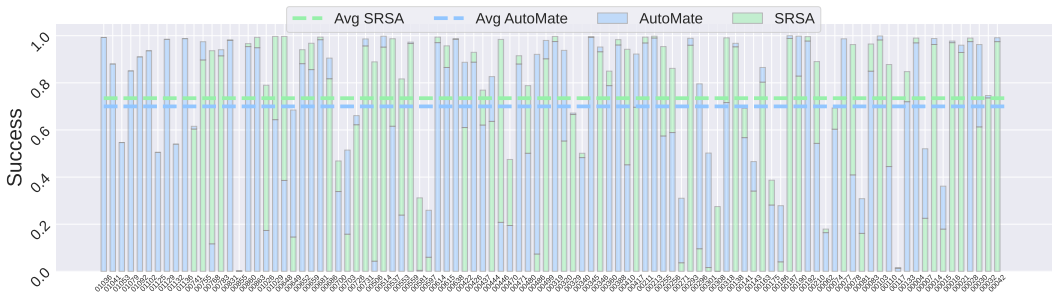


Figure 15: **Comparison of SRSA and AutoMate success rate over 100 tasks**. We replicate the specialist policy learning in the AutoMate paper over all tasks, and run SRSA with the continual-learning approach to train 90 specialist policies with the initial skill library of 10 policies. For both approaches, for each task, we select the best checkpoint among 5 runs with different random seeds. We compare the success rate on all the tasks. On average, SRSA achieves a higher success rate.

Skills retrieved based solely on geometry embeddings may face challenges during adaptation due to dynamic differences between the source and target tasks. As a result, the learning curves of SRSA-Geom tend to be less efficient and more unstable than SRSA.

When self-imitation learning is removed (SRSA-noSIL) from SRSA, the learning curves show increased fluctuation and higher variance across runs.

For the generalist policy, which was trained on 20 tasks from AutoMate (including tasks 01036, 01041, 01129, 01136), fine-tuning on these tasks yields strong performance since the policy was already optimized for them. However, on other test tasks, the generalist policy is not as effective for efficient policy learning compared to the skills retrieved by SRSA.

Fine-tuning a state-based generalist policy does not perform well because the generalist policy has limited capacity and it cannot cover more than 20 training tasks.

As prior work AutoMate (Tang et al., 2024) has shown, the training success rate of a state-based generalist policy decreases significantly when the number of training tasks exceeds 20, given a fixed policy architecture of RNN and MLP. We believe that this may be because each task requires precise control across distinct geometric features, and their single policy cannot capture the strategies for all these challenging tasks.

While "increasing model capacity" or moving toward a "large data and large model" regime might help mitigate this problem, it might introduce other challenges. Simply scaling model capacity could result in a generalist policy that works well in-domain but operates more like a "switching circuit," effectively storing task-specific strategies without generalizing to out-of-domain tasks. This approach is suboptimal as it prioritizes in-domain performance at the expense of out-of-distribution generalization. So we do not want to increase the model capacity indefinitely. Instead, we may need a more advanced architecture (e.g. diffusion policy) or model-based RL approach with planning to better handle diverse tasks.

That said, several open questions remain for state-based generalist policy: How to design policy architectures capable of high-precision control across many tasks? How to train the generalist policy efficiently on many assembly tasks considering possible gradient conflicts? How many training tasks are needed to achieve strong out-of-distribution generalization performance for new assembly tasks?

Fine-tuning a vision-based generalist policy presents additional challenges, such as effectively learning a generalist policy across multiple prior tasks with high-dimensional vision observations, fine-tuning on new tasks without forgetting prior ones, and addressing continual learning scenarios, including whether to fine-tune the original generalist policy or one already fine-tuned on other tasks. We made an initial attempt to train a vision-based generalist policy with PPO and fine-tune it. Given 90 prior tasks, it can only reach around 10% average success rate after training for two days. We expect such a generalist policy would perform no better than random initialization when fine-tuned for new tasks. Vision-based RL for generalist policy on assembly tasks is a relevantly new topic, and the development of such policies lies beyond the scope of SRSA. We leave this direction for future research.

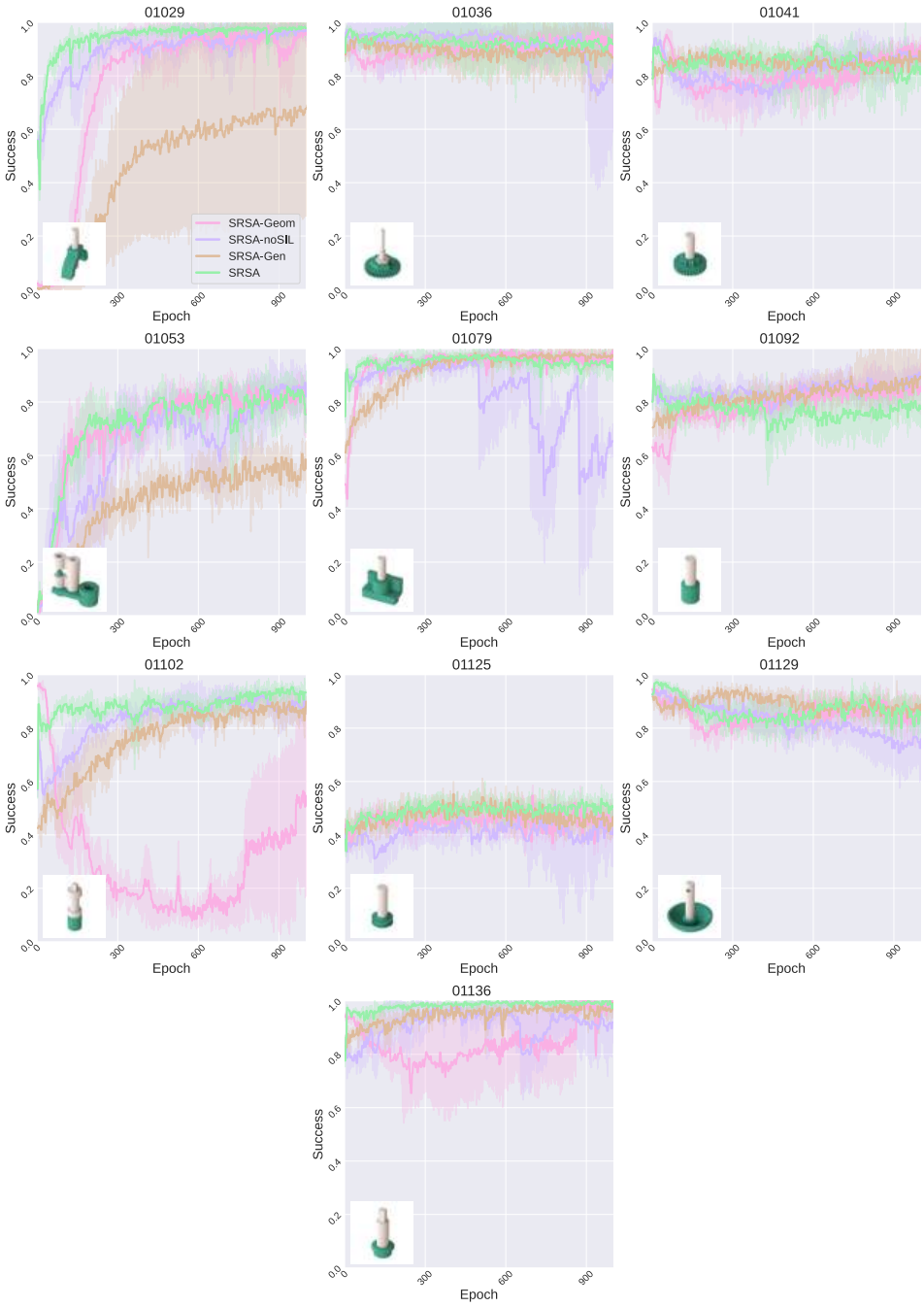


Figure 16: **Comparison for variants of SRSA with different ablated components**. For each method, we have 5 runs with different random seeds. The learning curves show mean and standard deviation of success rate over these runs.

A.6 Comparison with Geometry-based Retrieval

During adaptation, the final performance of SRSA-geom looks close to SRSA in some cases (see Fig. 16). However, it is statistically worse than SRSA, especially when there is a smaller number of training epochs. To provide a more comprehensive evaluation, we run SRSA-geom and SRSA across additional target tasks with three random seeds. The table below summarizes statistics of success rate at different numbers of training epochs, showing that SRSA consistently achieves higher success rates with lower variance. In industrial settings, a 3–9% difference in success rate can be significant.

	Test task set 1		Test task set 2	
Success rate (%)	Epoch 500	Epoch 1000	Epoch 500	Epoch 1000
SRSA-geom	73.6 (± 6.9)	81.0 (±7.7)	67.7(±7.1)	71.4(±8.1)
SRSA	**82.8(±4.2)**	**84.3(±3.4)**	**76.2(±3.0)**	**77.6(±3.5)**

Geometry-based retrieval alone is not always sufficient. When tasks share similar geometry but have different dynamics, SRSA-geom struggles to transfer as effectively as SRSA.

For example, for the target task 01092, SRSA-geom retrieves source task 00686, achieving a transfer success rate of only 61.1%, whereas SRSA retrieves task 00213 with a higher success rate of 76.7%. While the overall shapes of 01092 and 00686 are similar (see below), the lower part of plug in task 01092 is thinner than the upper part, and there is only a short distance to insert this lower part into the socket. These features closely resemble task 00213, i.e., a narrow plug to be inserted a short distance to accomplish assembly. These shared physical characteristics and similar task-solving strategies make 00213 better suited for transfer. In assembly tasks, the dynamics of the contact region are often more critical than overall geometry for task success. Therefore, source task 00213 works better than 00686 when transferring to the target task 01092.

(a) Assembly tasks in the first example

(b) Assembly tasks in the second example

Additionally, we examine assembly tasks with identical geometry but differing physical parameters. For instance, consider the target task 01136 with a friction value of 10.0. One source task has the same geometry as 01136 but a significantly lower friction value of 0.5. SRSA-geom selects this source task due to its geometric similarity; however, the corresponding source policy achieves only 88.9% transfer success on the target task, due to the friction mismatch (despite achieving a 99.3% success rate on its original source task). In contrast, SRSA selects the source task 00213, whose policy better aligns with the target task's dynamics, resulting in a higher transfer success rate of 93.2%

A.7 Analysis of Source Policy Success as Input for Retrieval

The success rate of the source policy on the source task is meaningful information to represent the source policy. To see whether it is practically beneficial for retrieval, we modify our approach. We simply concatenate this source success rate information with the task features of source and target tasks. We train the transfer success predictor F with these features as inputs.

We consider three random splits between the prior task set (90 tasks) and test task set (10 tasks). For each split, we train F on the prior task set over three random seeds. For each seed, we test the trained function F on the test task set for retrieval. We report the mean transfer success rate of the retrieved skills on 10 test tasks, with the standard deviation reported over three seeds. Empirically, the source success rate as input to F only slightly improves the retrieval results.

Average transfer success (%)	Test task set 1	Test task set 2	Test task set 3
SRSA	62.7 (+-5.7)	53.7 (+-5.5)	**44.9 (+-2.4)**
SRSA+source success rate	**66.7 (+-0.3)**	**53.7(+-2.6)**	**43.7 (+-3.7)**

A.8 Analysis of Out-of-Distribution Test Tasks

For out-of-distribution (OOD) tasks where no skill transfers zero-shot, SRSA may indeed struggle, and the initialization from a retrieved skill might not help much. To tackle this, it's essential to build a skill library that's as diverse as possible. When the target task falls outside the current library's distribution, we can use SRSA's continual learning approach (section 4.3 & 5.4) to expand the library with new tasks. By building a larger, more varied skill library, we increase the likelihood that this target task will align better with tasks in the skill library.

We run experiments for target tasks with IDs 00004, 00015, 00016, 00028, 00030. These tasks suffer from low transfer success rate given a small skill library with only 10 prior tasks. However, when we have a larger and larger skill library, the retrieved skill has a higher transfer success rate on the target task.

Transfer success rate (%)	00004	00015	00016	00028	00030
10-task library	15.9	6.9	0.2	12.2	39.1
50-task library	12.7	8.4	0.3	**27.5**	49.4
90-task library	**24.2**	**28.4**	**19.3**	18.1	**82.6**

As demonstrated, continual learning to expand the skill library is a promising step; however, generalizing to OOD tasks is a longstanding challenge in robotics, and it is still an open question how to optimally construct the curriculum that governs the expansion of the skill library.

A.9 Analysis of Other Metrics for Retrieval

We acknowledge that zero-shot transfer success rate may not be a perfect proxy for retrieval. We can consider several other possible metrics for retrieval: (1) Ground-truth success rate after adaptation (2) Predicted success rate after adaptation (3) Predicted success rate in zero-shot manner (i.e. SRSA) (4) Predicted dense rewards in zero-shot manner.

Option 1 is the ideal metric to identify the best skill for retrieval, as our final goal is to obtain the highest success rate on the target task after adaptation. However, it introduces a chicken-and-egg problem, as we cannot get this metric without fine-tuning all candidate policies on the target task.

Option 2 requires training a predictor for the success rate after adapting any source policy on any target task. We need the training labels of the ground-truth success rate after adaptation. Unfortunately, collecting this training data would require extensive computational resources. For each source-target pair, we need at least 20 GPU hours to finish adaptation; given a skill library of 100 tasks, 200,000 GPU hours would be required to collect training data. Furthermore, it will remain intractable as the skill library becomes larger.

Option 3 (SRSA) requires much less resources to collect training data for the predictor. We only need 20 minutes on a GPU to evaluate one source policy on a target task. It thus requires 3,000 GPU hours to collect training labels. We conduct an experiment to compare the performance of Option 1 and Option 3 on two test tasks. To collect experimental results for Option 1, for each test task, we sweep all 90 source policies in our skill library. We finetune each source policy with one random seed to adapt to the target task and identify the best success rate after adaptation. We only afford the computational resources for two test tasks to sweep fine-tuning for Option 1. Below we report the success rate of Option 1 and Option 3 after fine-tuning for 1500 epochs

Success rate after adaptation (%)	Test task 1036	Test task 1041
Option 3 (SRSA)	95.9	89.1
Option 1	98.3	94.0

Option 1 is the perfect but intractable metric for retrieval. The difference of success rate between the SRSA-retrieved skill (Option 3) and the best source skill (Option 1) is less than 5% after adaptation. Therefore, although zero-shot transfer success rate is not a perfect metric for retrieval, it is a high-quality metric for retrieval in terms of both performance and computational efficiency.

Furthermore, we consider using dense reward information to guide retrieval (Option 4). We learn to predict the accumulated reward rather than success rate on the target task when executing the source policies in a zero-shot manner; then we retrieve the source policy with the highest predicted transfer reward. In the table below, we show the performance of retrieved skills when they are applied on the target tasks.

In the AutoMate task set, Option 3 (SRSA) yields slightly better skill retrievals, especially with higher transfer success on the target task. However, success rate may not accurately reflect the expected value for tasks with dense rewards. The higher transfer success rate does not mean higher transfer reward in test task set 2. Therefore, if it is critical to prioritize the reward achieved on the

	Test task set 1		Test task set 2	
	Transfer reward	Transfer success (%)	Transfer reward	Transfer success (%)
Option 3 (SRSA)	**8134**	**62.7**	7722	**53.7**
Option 4	7976	54.8	**7935**	32.6

target task, using the transfer-reward predictor for retrieval is a reasonable choice. Conversely, if the success rate on the target task is more critical (as in our assembly tasks), the transfer success would be the preferred choice as a retrieval metric.

A.10 Analysis of Distance Metrics for Task Features

We concatenate the features of geometry, dynamics and expert actions as the task features, and apply some distance metrics between the vectors as the metrics for retrieval. We consider three different ways to split the prior task set (90 tasks) and test task set (10 tasks). We consider L2 distance, L1 distance, and negative cosine similarity as distance metrics. For each test task, we retrieve the source task with the closest task feature to the target task. However, the retrieval result is worse than SRSA on three different test task sets.

Transfer success rate (%)	L2 distance	L1 distance	Cosine similarity	SRSA
Test task set 1	51.6	50.8	52.6	**62.7**
Test task set 2	47.1	49.0	46.5	**53.7**
Test task set 3	35.3	35.0	36.1	**44.9**

We jointly learn features from geometry, dynamics and expert actions to represent tasks, and predict transfer success to implicitly capture other transfer-related factors from tasks. SRSA learning function F aims to capture additional information for transfer success prediction. Therefore, the prediction function F provides a better metric to identify the source task with higher zero-shot transfer success.

A.11 Ablation Study on Policy Initialization and Self-Imitation Learning

As for policy learning, AutoMate is PPO from random policy initialization, and SRSA is PPO with self-imitation learning (SIL) after initialization with the retrieved skill. Thus, the main difference between SRSA and AutoMate lies in (1) strong initialization from retrieval and (2) SIL. In section 6, we compared SRSA and SRSA-noSIL to show the effect of SIL. Below, we additionally compare with SRSA with random initialization (SRSA-noRetr) to show the effect of initialization from retrieval.

Comparing AutoMate with SRSA-noRetr, we see the difference between PPO and PPO+SIL when learning a policy from scratch. Both approaches started from poor performance, but SIL has greater learning efficiency and stability. Comparing SRSA-noRetr and SRSA, we see the difference between random initialization and initialization from retrieval. Policy retrieval provides a good start with a reasonable success rate. As a result, SRSA more efficiently reaches higher performance on the target task.

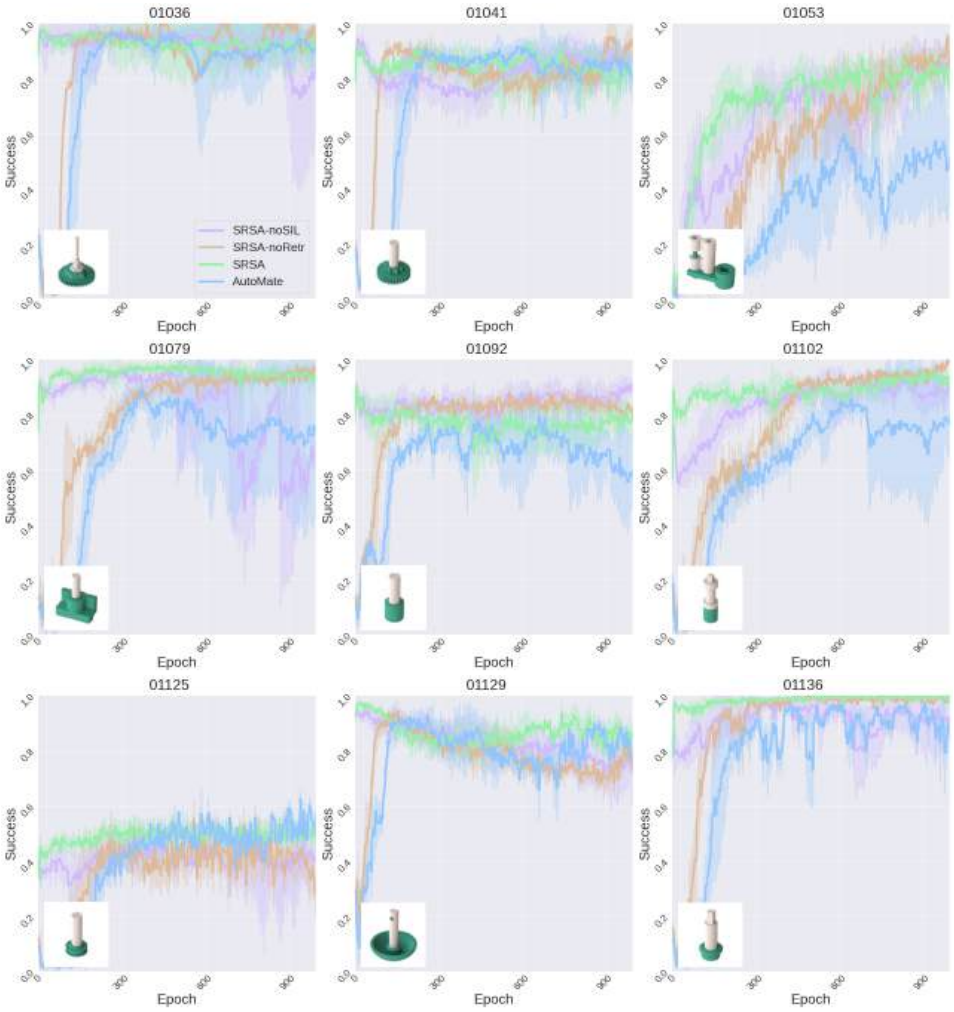


Figure 17: **Comparison for variants of SRSA with different ablated components**. For each method, we have 5 runs with different random seeds. The learning curves show mean and standard deviation of success rate over these runs.