Master's Project Report [2023-24 WS]

SAC-N-GMM: Robot Skill Refining and Sequencing for Long-Horizon Manipulation Tasks

Akshay L Chandra, Iman Nematollahi, Tim Welschehold
{lagandua, nematoli, twelsche}@cs.uni-freiburg.de

Abstract

Despite access to expert data, most long-horizon imitation-learning (IL) agents suffer from distribution shifts, compounding errors, and expert dependency. Several previous works show that refining IL agents in the world with reinforcement learning (RL) alleviates some of these problem by making the agents more robust to noisy perception and stochasticity in dynamics with much helpful real-world exposure. SAC-GMM [8] does this efficiently by first learning a task from demonstrations with a classical robotics technique (e.g., Gaussian Mixture Model) and then refines it with a deep RL (Soft Actor-Critic) agent with sparse task-completion rewards. One could further dampen the side effects of long-horizon IL agents by breaking down complex tasks into short-horizon skills. This simplifies the learning goal into a hierarchy of agents, i.e. high-level planning agent (skill sequencer) and low-level control agent (skill executor). To this end, we propose the Soft Actor-Critic-N-Gaussian Mixture Model (SAC-N-GMM), a novel hybrid RL approach that learns to simultaneously refine and sequence a repertoire of low-level skills to perform numerous combinations of long-horizon tasks. Our approach extends SAC-GMM (1) by learning N low-level robot skills with Riemannian Manifold GMMs that learn both robot positions and orientations (2) by learning a single RL agent to refine and sequence multiple manifold-aware GMM skills. Extensive evaluations in the CALVIN simulation environment demonstrate that our approach leverages high-dimensional sensory data, minimal expert demonstrations, minimal physical interactions, and sparse task-completion rewards efficiently to achieve superior long-horizon task performance compared to baselines. Code is available at https://github.com/acl21/sac_n_gmm

1. Introduction & Related Works

Learning robust manipulation tasks, either zero-shot or one-shot, is a trait that benchmarks human intelligence.

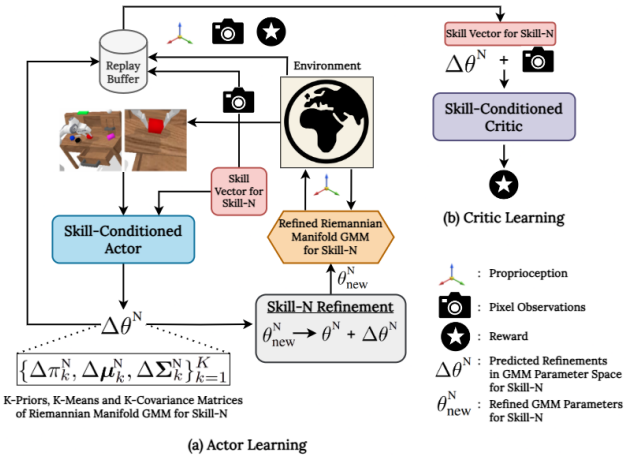

Figure 1. SAC-N-GMM RL Fine-Tuning Pipeline. In (a), an Actor conditioned on a skill vector (unique to a given skill) predicts refinements in Riemannian Manifold GMM parameter space. These refinements adapt the chosen skill's GMM before deploying in the environment. (b) highlights Critic learning in a supervised fashion.

In this context, sample efficiency is a good indicator of how good an IL approach is. While deep neural network-based IL agents may be good at predicting actions directly from pixel observations of the world, they require massive amounts of expensive data to show any promising performance. Under these circumstances, adopting classical dynamical systems to learn IL agents may be suitable. Despite their strengths, such as sample efficiency, robustness to environmental perturbations, and provably reliable behaviour, they fail to tackle the complexities of real-world robotics, where an accurate state of the environment is unavailable, and the environment observations are too high-dimensional and impossible to model. Previous works like SAC-GMM [8] addressed this issue by marrying a classical dynamical learning system with a deep RL agent. SAC-GMM demonstrated with their real-world experiments that it is possible to achieve sample efficiency by learning a GMM in the low-dimensional trajectory space and then refining the GMM in

its parameter space with a deep RL agent. The deep RL agent conveniently allows the method to work with rich, high-dimensional observations of the environment.

Most IL agents also suffer from distribution shifts and compounding errors during rollouts. These issues are further exacerbated when dealing with long-horizon tasks. Previous works have addressed the long-horizon task learning problems by decomposing tasks into temporally-extended skills. This creates a natural hierarchy of IL agents, i.e., a high-level planning agent for skill sequencing and a low-level control agent for skill execution in the environment. We adopt this hierarchy in our work as it allows for a clear division of labour and responsibility between agents, simplifying the complexities of long-horizon task learning to some extent.

Fine-tuning IL agents with model-based RL is one alternative paradigm that achieves sample efficiency - by learning a policy and a model of the world. Access to a model allows planning within itself, ultimately reducing the real-world interactions needed to learn high-performing policies. Several works have also relied on unlabeled offline robot *play data* to learn a model. [5] describes *play data* as a *"continuous logs of low-level observations and actions collected while a human teleoperates the robot and engages in behaviour that satisfies their own curiosity"*. Despite the lack of labels, robot *play data* is often full of rich physical environment interactions and tends to span states of the world magnitudes more than expert data in a standard tabletop setting. To this end, Skimo leverages the offline play dataset to first learn a model of the world, then learn hierarchical policies (high-level skill predictor and low-level skill executor) by planning with the model, enabling superior long-horizon task completion compared to their model-free alternatives. In this setting, the high-level agents learn to predict skill vectors in a high-dimensional latent space, which are then passed to a low-level agent to predict environment actions. The skill vectors learned by the high-level agent are unreadable, abstract and implicit.

We argue and show through our experiments that decomposing tasks into readable explicit skills instead of learning implicit tasks can be a powerful tool, surpassing even the model-based methods in sample efficiency. We hypothesize that learning to stitch known explicit skills can be more sample-efficient than learning implicit skills first and then learning to stitch them with rewards. To this end, we extend SAC-GMM (1) by first learning a repertoire of temporally-extended explicit skills that compose together to form long-horizon tasks (2) by learning skills with Riemannian Manifold GMMs (RM-GMMs) to model trajectories of both robot positions and orientations, on a *Eucliean* $\times$ *Unit-Quaternion* product manifold (3) by introducing a novel and intuitive way of refining RM-GMMs (4) by learning a single SAC agent to simultaneously refine and sequence multiple skills at once. A single agent trained with our method performs superior to baselines on several tasks, an advantage of learning to sequence explicit skills. This allows us to delegate the role of high-level planning agent to a large or a vision language model; we leave this thread to future work. Note that we report the task performance of our approach on all possible skill pairs (see Section 4).

2. Preliminaries

Our method naturally breaks into two phases, first learning a repertoire of robot skills from expert demonstrations and then fine-tuning the robot skills with RL. Following SAC-GMM's approach, we consider robot skills directly defined in robot trajectory space. Learning a dynamical system like a Gaussian Mixture Model (GMM) exploits the trajectory space's structure sample efficiently compared to neural network-based alternatives. We later refine these structured models through the robot's physical interactions in the world. Specifically, we are interested in learning the GMMs on Riemannian manifolds to model both robot positions and orientations. Riemannian manifolds are smooth, non-euclidean spaces where one cannot directly use Euclidean operations, such as vector sum or scalar multiplication. Zeestraten et al. [11] showed that one can learn robot skills from demonstrations on Riemannian manifolds by performing Euclidean operations on Euclidean tangent spaces instead. To that end, [11] extended Expectation Maximization (EM) algorithms to Riemannian manifolds. We use [11]'s official implementation in this work and refer the readers to our Section 3 and Section II of [11] for a more detailed discussion on Riemannian manifolds.

With access to EM algorithms that model and regress *Gaussians* on Riemannian manifolds, we can now consider robot skill as trajectories in 7D space, i.e. $\mathbb{R}^3 \times \mathbb{H}$, or as points lying on a 7D product manifold. We denote a trajectory by $\Xi = \{\langle \xi_t^p, \xi_t^o \rangle\}_{t=1}^T$ where $\xi_t^p \in \mathbb{R}^3$ is the robot's geometric position in Euclidean space and $\xi_t^o \in \mathbb{H}$ is the robot's geometric orientation (as a quaternion[1]) in Quaternion space, at time step t. Moreover, each execution of a robot skill in the environment induces observations, $\mathbf{O} = \{o_t\}_{t=1}^T$, of the robot's geometric poses as well as high-dimensional static and gripper images, depth maps, and tactile measurements. The robot's goal is to respond to these high-dimensional observations by steering through the geometric route an expert would take. The geometric route, in essence, is the space spanned by the set of D demonstrations $\mathbf{X} = \{\Xi_d\}_{d=1}^D$, trajectories collected by an expert through robot teleoperation.

Our goal is to first learn a repertoire of robot skills (RM-GMMs), where each skill is a parametric function that maps

[1] Quaternions must be of unit norm to represent valid orientations. From here on, we will use quaternion to refer to unit quaternions.

the robot's position to its next position and orientation in time, forming a state-transitioning dynamical system. For each skill,

$$\mathcal{P}(\mathbf{X}) = p_\theta(\xi_t^p) = \langle \xi_{t+1}^p, \xi_{t+1}^o \rangle \tag{1}$$

where θ refers to RM-GMM's parameters. We learn a repertoire of N such skills $\{p_{\theta_n}\}_{n=1}^N$. Note that this part only requires trajectories of robot poses and does not require high-dimensional observations, as they are infeasible for GMMs to parameterize. In the refining phase, we learn a deep RL policy with sparse task completion rewards. Conditioned on the robot's pose $\langle \xi_t^p, \xi_t^o \rangle$, the environment's high-dimensional observations $\mathbf{o}_t$ and a unique skill identifier $\bar{\nu}_n$ (e.g., a one-hot vector), the policy learns to predict refinements in skill n's parameter space – $\pi_\phi(\xi_t^p, \xi_t^o, \mathbf{o}_t, \bar{\nu}_n) = \Delta\theta^n$. This allows a single deep RL policy to learn refinements for multiple skills.

3. SAC-N-GMM

The proposed Soft Actor-Critic-N-Gaussian Mixture Models (SAC-N-GMM), N referring to the number of skills, is divided into two distinct phases - in (I) we learn a dynamical system from a few demonstrations and in (II), we refine these dynamical systems with a skill-conditioned actor learned with the soft actor-critic algorithm through interactions with the environment.

Phase I: Learning a Repertoire of Robot Skills

A dynamical system typically models how a system evolves over time based on its current state (and often, optionally, some other input). In the context of robot skill learning, Gaussian Mixture Models could be used to model a state-evolving dynamical system of robot joint positions, velocities, end-effector poses and even environmental variables [4, 8]. In this work, we formulate a robot skill as a control law driven by an autonomous dynamical system, defined by the robot pose:

$$\langle \xi_{t+1}^p \xi_{t+1}^o \rangle = p_\theta(\xi_t^p) + \epsilon \tag{2}$$

where p_θ is a robot skill, a parametric, non-linear, steady, continuously differentiable policy, and ϵ is a zero-mean additive Gaussian noise. Learning p_θ then becomes a straightforward regression problem and can be modelled by GMMs. Precisely, given a set of expert demonstrations $\mathbf{X}$ for a robot skill, we first estimate the joint probability density $\mathcal{P}(\xi_t^p, \xi_{t+1}^p, \xi_{t+1}^o)$ as a mixture of Gaussians on a 10-dimensional product manifold $\mathcal{M}$ i.e., $\mathbb{R}^6 \times \mathbb{H}$. Similar to GMMs in Euclidean space, an RM-GMM is defined by weighted sums of Gaussians on $\mathcal{M}$, so a robot skill p_θ is parameterized by $\theta = \{\pi_k, \mu_k, \Sigma_k\}_{k=1}^K$ where π_k are the priors ($\Sigma_{k=1}^K \pi_k = 1$), μ_k the mean matrix and Σ_k the covariance matrix of the k-th Gaussian on $\mathcal{M}$.

With an estimate of a joint probability, one can easily obtain future robot pose $\langle \xi_{t+1}^p, \xi_{t+1}^o \rangle$ conditioned on current robot position ξ_t^p with the help of Riemannian manifold Gaussian Mixture Regression (RM-GMR) [11]. RM-GMR allows us to obtain the closed-form conditional distribution $\mathcal{P}(\xi_{t+1}^p, \xi_{t+1}^o | \xi_t^p)$ for output Gaussians on a sub-Manifold $\mathcal{M}^s$ i.e., $\in \mathbb{R}^3 \times \mathbb{H}$. Our robot skill predicts the next robot pose, given the current robot position. Updating ξ_t^p with previous predictions allows us to iteratively generate full robot trajectories. We learn a repertoire of N such robot skills this way $\{p_{\theta_n}\}_{n=1}^N$. For a detailed explanation of using both Euclidean GMMs and RM-GMMs as dynamical systems for imitation learning, we refer the readers to [11].

Phase II: Refining the Robot Skills with Reinforcement Learning

With access to a repertoire of robot skills $\{p_{\theta_n}\}_{n=1}^N$, we can now interact with the world and learn a single neural network-based policy that can refine all N robot skills. Like SAC-GMM, we formulate this refinement as an RL problem in which the agent must learn to modify several GMM robot skills in their parameters space with access to sparse skill-completion rewards. Concretely, our RL problem can be defined as a policy search in a Partially Observable Markov decision process (POMDP), defined by the following – an observation space $\mathcal{O}$, a state space $\mathcal{S}$ and an action space $\mathcal{A}$. In this phase, our agent receives high-dimensional sensory measurements such as RGB images (optionally coupled with depth maps, tactile measures, etc.), first encoded to a latent representation space z by an autoencoder. The latent z concatenated with the robot end-effector position ξ^p and a one-hot skill vector $\bar{\nu}_n$ form our continuous state space $\mathcal{S}$. The continuous action space $\mathcal{A}$ consists of the refinements we want to make to skill n in its parameters' space $\Delta\theta$. The environment emits a sparse non-zero reward only when the robot executes the skill correctly. Namely, at time t, if s_t is the environment's inferred state, a_t is the agent's action, z_t is latent representation of environment observation, $\bar{\nu}$ is the skill vector corresponding to a target skill to be refined, $\langle \xi_t^p, \xi_t^o \rangle$ is the robot end-effector's position and orientation respectively, then for a skill N:

$$\begin{aligned} s_t &:= \{\xi_t^p, \xi_t^o, z_t, \bar{\nu}_N\}, \\ a_t &:= \Delta\theta^N = \texttt{Flatten}(\{\Delta\pi_k, \Delta\mu_k, \Delta\Sigma_k\}_{k=1}^K) \\ \therefore \Delta\theta^N &= \pi_\phi(a_t | s_t) \end{aligned} \tag{3}$$

where π_ϕ is the skill-refinement policy. We choose the soft actor-critic (SAC) algorithm [1] as our RL framework. SAC is an off-policy actor-critic method that optimizes for maximum expected total reward while also maximizing the actor's entropy. This interplay between reward and

entropy maximization forces SAC to explore sufficiently while learning high-performing stochastic policies. SAC's sample efficiency and stability align well with our project goals; hence, it is a good choice for our projects. SAC's objective is as follows:

$$J(\pi_\phi) = \sum_{t=0}^{T} \gamma^t \mathbb{E}_{(s_t,a_t)\sim\rho_{\pi_\phi}}[r(s_t,a_t) + \alpha\mathcal{H}(\pi_\phi(\cdot|s_t))] \tag{4}$$

where α is a tunable penalty parameter on entropy term - it regulates the stochasticity of π_ϕ. Our SAC agent stores $\{s_t, o_t, a_t, r_t.s_{t+1}, o_{t+1}\}_{t=1}^{T}$ transition tuples in the replay buffer $\mathcal{D}$. The replay buffer aids in joint learning of an autoencoder q_ω, the policy π_ϕ, two Q-functions Q_{φ_1} and Q_{φ_2} and their target functions. We learn an autoencoder to map high-dimensional observations o_t to a low-dimensional latent representation z_t, with an L2 penalty on z_t to encourage robust feature learning. All networks are trained with mini-batches sampled from the replay buffer and the gradients are backpropagated with stochastic gradient descent, minimizing the following losses:

$$\begin{aligned}
\mathcal{L}(\omega,\mathcal{D}) &= \mathbb{E}_{o_t\sim\mathcal{D}}\Big[\log q_\omega(o_t|z_t) + \lambda_z||z_t||^2\Big] \\
\mathcal{L}(\varphi_i,\mathcal{D}) &= \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}}\Big[||Q_{\varphi_i}(s_t,a_t) - Q^{target}(s_t,a_t)||^2\Big] \\
\mathcal{L}(\phi,\mathcal{D}) &= \mathbb{E}_{(s_t\sim\mathcal{D},a_t\sim\pi_\phi)}\Big[\alpha\log\pi_\phi(a_t|s_t) - \min_{i=1,2} Q_{\varphi_i}(s_t,a_t)\Big]
\end{aligned} \tag{5}$$

Where Q^{target} is the target for the Q functions and is computed using the immediate reward, the value estimate of target Q network and an entropy regularization term. Note that the SAC agent predicts refinements conditioned on s_t, which contains significantly more information about the environment than the RM-GMMs, through latent representation z_t. Also, the SAC agent knows which skill to refine through the one-hot skill vector $\bar{\nu}_n$.

Skill Refinement

The SAC agent π_ϕ receives the current state $s_t = \{\xi_t^p, \xi_t^o, z_t, \bar{\nu}_N\}$ and predicts refinements as action $\Delta\theta^N$ conditioned on s_t. Recall from Phase I, our RM-GMMs learn a joint probability density $\mathcal{P}(\xi_t^p, \xi_{t+1}^p, \xi_{t+1}^o)$ as a mixture of Gaussians on a 10-dimensional product manifold $\mathcal{M}$ i.e., $\mathbb{R}^6 \times \mathbb{H}$. The predicted refinements $\Delta\theta^N$, however, are in Euclidean space[2]. How one can refine Gaussians lying on such a product manifold with Euclidean refinements is unclear. Through trial and error, we learned that splitting the 10-dimensional product manifold $\mathcal{M}$ into two manifolds and refining them separately a posteriori yields stable Gaussians in $\mathcal{M}$. In that context, we split $\mathcal{M}$ into – a Euclidean Manifold $\mathcal{M}^E$ corresponding to ξ_t^p and ξ_{t+1}^p (first six dimensions) and a Quaternion Manifold $\mathcal{M}^Q$ corresponding to ξ_{t+1}^o (last four dimensions).

Therefore, we naively refine all elements in θ^N corresponding to $\mathcal{M}^E$ with the following update rule:

$$\theta^N_{new}[i] \rightarrow \theta^N[i] + \Delta\theta^N[i] \quad \forall i \in \mathcal{E} \tag{6}$$

where $\mathcal{E}$ is a set of all the indices that correspond to elements relevant to ξ_t^p and ξ_{t+1}^p in θ^N. Note that this update rule only applies to elements form $\{\pi_k, \mu_k\}_{k=1}^K$. To update elements from $\{\Sigma_k\}_{k=1}^K$, we must ensure the updated Σ_k is a valid symmetric positive definite matrix. This is out of the scope of this project. We only refine $\{\pi_k, \mu_k\}_{k=1}^K$ in our experiments.

To refine Gaussians on $\mathcal{M}^Q$, we turn to a fundamental manifold mapping operation – the *exponential map*. For a d-dimensional smooth manifold $\mathcal{M}$, there exists a tangent space $\mathcal{T}_{\mathbf{p}}\mathcal{M}$ at each point $\mathbf{p} \in \mathcal{M}$. The *exponential map* $\text{Exp}_{\mathbf{g}}(\cdot) : \mathcal{T}_{\mathbf{g}}\mathcal{M} \rightarrow \mathcal{M}$ is a distance preserving map between the tangent space and the manifold. As shown in Figure 2, $\text{Exp}_{\mathbf{g}}(\mathfrak{p})$ maps $\mathfrak{p}$ to $\mathbf{p}$ so that $\mathbf{p}$ lies on the geodesic through $\mathbf{g}$ with direction $\mathfrak{p}$[3]. Once again, we refer the readers to [11] for a more detailed discussion.

The distance preserving Exp operation allows us to indirectly perform computations on the manifold by navigating in the tangent space. To that end, we assume that the relevant elements of $\Delta\theta^N$ span the tangent spaces at points $\{\mu_k[i:j]\}_{k=1}^K$ where $[i:j]$ is a section of μ_k that corresponds to ξ_{t+1}^o on $\mathcal{M}^Q$. π_ϕ essentially searches for the best vectors in the tangent spaces at points $\{\mu_k[i:j]\}_{k=1}^K$ on $\mathcal{M}^Q$. The update rule for a Gaussian K's mean is as follows:

$$\mu_k[i:j]_{new} \rightarrow \text{Exp}_{\mu_k[i:j]}(\Delta\mu_k[i:j]) \tag{7}$$

This quaternion mean refinement is a novel contribution of this work. The update rule for quaternion covariances in Σ_k is out of this project's scope. In all our experiments, we only refine quaternion means.

Stochastic Skill-Pretraining (SSP)

While RM-GMM robot skills are data-efficient, they can be brittle when queried on out-of-training-distribution samples. The goal of SAC-N-GMM is to refine and sequence many robot skills. Therefore, a given RM-GMM robot skill p_{θ_n} should be robust to robot position queries ξ_t^p unseen in its own expert trajectories. One way to address this problem is to train π_ϕ to refine each skill, starting from every other skill's final timestep robot position. More generally,

[2] In SAC-GMM, the authors learned GMMs entirely in Euclidean space, so the refinements were straightforward

[3] The *logarithmic map* is the inverse mapping, and it exists if there is only one geodesic through $\mathbf{g}$ and $\mathbf{p}$

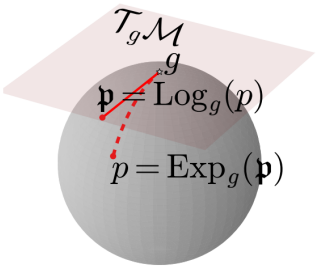

Figure 2. The exponential and logarithmic map between the manifold $\mathcal{S}^{\in}$ and its tangent space at point }. Figure is taken unmodified from [11].

we can train π_ϕ to refine all RM-GMM robot skills from *any* feasible point in the environment. In that context, we first pretrain π_ϕ to refine all robot skills by randomly initializing the robot end-effector in any *feasible* point in the environment. Precisely, both the one-hot skill vector $\bar{\nu}_n$ and the robot end-effector position are randomly sampled. We call this Stochastic Skill-Pretraining (SSP) a key contributor to the skill-sequencing ability of SAC-N-GMM; see Section 4 for details. Figure 3 shows such randomly sampled robot positions in the environment. Note that a reset-free RL curriculum similar to [7] can be implemented in this scenario, but we intentionally ignore it in this work. Also, once pretrained with SSP, we can easily fine-tune π_ϕ and the observation encoder on a single task with sparse task-completion rewards.

4. Experiments

We evaluate SAC-N-GMM's ability to refine and stitch robot skills in simulation. The goals of the experiments are to verify (i) if our hybrid RL agent is capable of refining and sequencing multiple skills in realistic noisy environments, (ii) if learning a single agent contributes to a lower exploration budget, (iii) if quaternion refinement is necessary for high performance (iv) if our agent's performance scales consistently in skills and physical interactions.

4.1. Setup

We investigate SAC-N-GMM's performance on four table-top robot skills in CALVIN's Env D [6]: Open Drawer (**A**), Turn On Lightbulb (**B**), Move Slider Left (**C**) and Turn on LED (**D**). The environment signals a sparse skill-completion reward – for **A** when the drawer is opened at least 12 cm from its initial position, for **B** when the lightbulb turns on, for **C** when the slider moves 15 cm to the left from its initial position, and for **D** when the LED turns on. We define a single robot task as a unique combination of four skills. Therefore, skills **A**, **B**, **C**, **D** could permute to form 24 unique tasks. However, we limit all our experiments to the following eight tasks: **ABCD**, **BACD**, **CBAD**, **DCBA**, **ACBD**, **BADC**, **CADB**, **DACB**. These eight tasks are chosen so that all possible combinations of skill pairs appear together. E.g., skill **A** appears after **B**, **C**, **D** and skill **B** appears after **A**, **C**, **D** and so on. To that end, we report the number of successful skills executed while performing a task.

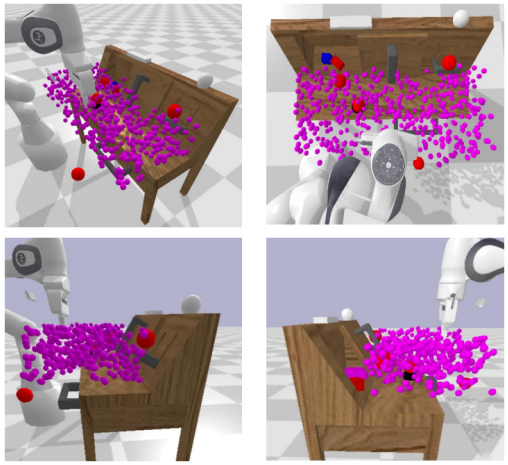

Figure 3. Spheres in magenta show randomly sampled robot positions from multiple viewing angles in CALVIN's Env D. For reference, spheres in red show average last timestep robot positions of ABCD skills.

4.2. Baselines

We compare SAC-N-GMM against several sample efficient baselines:

TD-MPC: In TD-MPC [3], an environment model, a value function and an actor are simultaneously learned. TD-MPC uses CEM [9] to plan within the model with the value functions as guidance. This imaginative exploration allows TD-MPC to sample efficiently and take only high-reward-producing actions in the environment. TD-MPC learns a single-step dynamics model.

DreamerV3: DreamerV3 [2] is another model-based RL method similar to TD-MPC. However, it differs significantly in several carefully taken modelling choices for state estimation and offline planning. The actor and critic are learned entirely in imagination. Similar to TD-MPC, DreamerV3 also learns a single-step dynamics model.

Skimo: In Skimo [10], a model is first learned from a large unlabeled robot play data, and the actor and critic are learned online in the second phase with environment interactions. Skimo also uses hierarchical agents; a high-level planning agent predicts a latent skill, while a low-level control agent predicts robot actions conditioned on the latent skill. In contrast to TD-MPC and DreamerV3, Skimo learns a multi-step dynamics model and actor. For a fair comparison, we set multi-steps similar to SAC-N-GMM's – 16.

Method	Wall-Clock Runtimes (Hours)
SAC-N-GMM	8.5
N-SAC-GMM	$2.5 \times 4 = 10$
N-GMM	-
Skimo (Vector)	$20 + 2 \times 8 = 36$
DreamerV3 (Vector)	$2 \times 8 = 16$
DreamerV3 (Static)	$10 \times 8 = 80$
TD-MPC (Vector)	$23.5 \times 8 = 188$

Table 1. Wall-Clock runtimes (in hours) taken to run a single seed of experiments on one NVIDIA GeForce RTX 2080 Ti.

N-SAC-GMM: We train N different SAC-GMMs, one for each robot skill trained on the same expert trajectories. Note that SAC-GMM was originally designed to handle only single skills. We integrate SAC-GMM's pipeline with our SSP and quaternion refinement for a fair comparison.

N-GMM: In this, raw, unrefined RM-GMM robot skills are used in the environment during a task's rollout.

Skimo and TD-MPC underperformed severely when trained on RGB data, so we only reported their results of experiments with state vectors (proprioception + scene) as input. DreamerV3 surprisingly underperformed when trained on both static and gripper images as input; however, it worked reasonably well when just static images were provided as input, so we only reported those results. In all our experiments, we train and report SAC-N-GMM with only gripper images as input. Our early experiments showed that the SAC-N-GMM variant trained with both static and gripper images underperforms marginally compared to the only gripper variant, so we ignore it entirely.

4.3. Results

Quantitative results of the average number of skills over 50 trials performed per two different random seeds are shown in Figure 4. Our single SAC-N-GMM-SSP-500K agent, which is pre-trained with SSP with 500K environment interactions, successfully outperforms all baselines in six out of eight tasks without ever having to fine-tune on individual tasks. Our SAC-N-GMM-250K agent pre-trained with SSP with half as many environment interactions outperforms all baselines in seven out of eight tasks. Note that all baseline methods were explicitly trained to fine-tune each task separately. However, in our case, a single SAC-N-GMM pre-trained with SSP performs well on seven of eight tasks. This makes our wallclock runtimes magnitudes lower than the baselines. Table 1 compares the wallclock times taken to obtain results in Figure 4 on a single NVIDIA GeForce RTX 2080 Ti.

5. Ablations

In ablations, we evaluate and quantify how much SSP and Quaternion refinement contribute to SAC-N-GMM's performance. We also discuss task-specific fine-tuning to show that SAC-N-GMM can fine-tune. At the end, we evaluate if SAC-N-GMM can scale up in the number of skills and the environment interactions.

5.1. Role of SSP and Quaternion Refinement

In our early experiments, directly doing Phase II on individual tasks with either skill or task completion rewards yielded poor results. Despite several hyperparameter tuning attempts, SAC-N-GMM could not stitch even two skills together. Introducing SSP into the pipeline changed that. We want to emphasize that SSP plays a major role in SAC-N-GMM's performance. As evidenced in Figure 4, SSP alone can allow SAC-N-GMM to perform well on most evaluation tasks.

Figure 6 compares the performance of SAC-N-GMM with and without quaternion refinement. The difference, while marginal, across eight tasks makes it clear that quaternion refinement contributes to better overall task performance in expectation.

5.2. Task-Specific Fine-Tuning

In Figure 4, we can see that SAC-N-GMM is unable to convincingly solve tasks **BDCA** and **ACBD**. Upon closer examination of policy rollouts, we noticed that SAC-N-GMM struggled to sequence skills **B** and **D** in that order. It was clear that just SSP, even for 500K environment steps, was not enough to learn this stitching. To that end, we could fine-tune SAC-N-GMM, which is pre-trained with SSP first. Figure 5 compares how fine-tuning SAC-N-GMM on specific tasks with sparse task-completion rewards after pre-training with SSP helps improve performance on **BDCA** and **ACBD**.

5.3. Does SAC-N-GMM scale well?

We evaluate SAC-N-GMM's ability to scale in terms of the number of skills to refine and stitch and the number of environmental steps. To that end, we introduce three new CALVIN's Env D skills - Close Drawer (**A'**), Turn Off Lightbulb (**B'**) and Move Slider Right (**C'**)[4]. The environment signals a sparse skill-completion reward – for **A'** when the drawer is closed at least 12 cm from its initial position, for **B'** when the lightbulb turns off, for **C'** when the slider moves 15 cm to the right from its initial position.

Figure 7 shows that SAC-N-GMM outperforms the strongest baseline (N-SAC-GMM) across four arbitrarily

[4]Note that we intentionally refrain from adding Turn Off LED skill as it has exactly identical geometry to Turn On LED (**D**) skill.

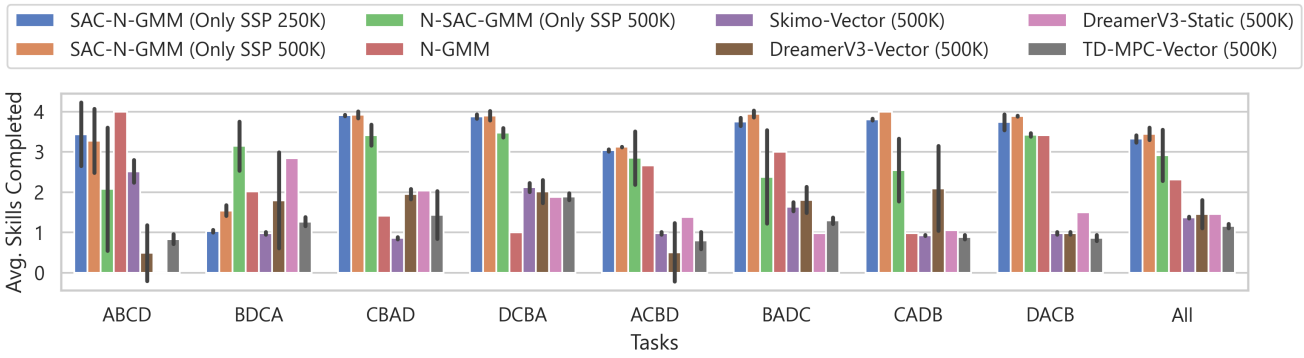

Figure 4. Main Results: SAC-N-GMM vs baseline methods for all eight tasks.

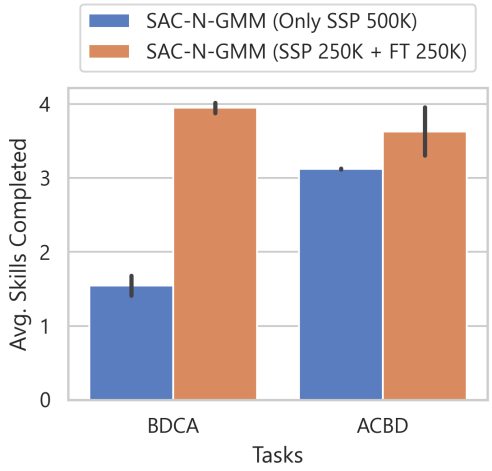

Figure 5. SAC-N-GMM with only 500K of SSP vs SAC-N-GMM with 250K of SSP and Fine-tuning of 250K environment steps.

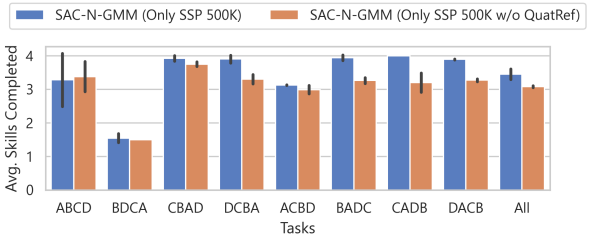

Figure 6. SAC-N-GMM with and without Quaternion Refinement

chosen seven-skill tasks. We can also see that, with increased environment steps, SAC-N-GMM's performance also increases, displaying stable refinement ability with scale in skills and steps.

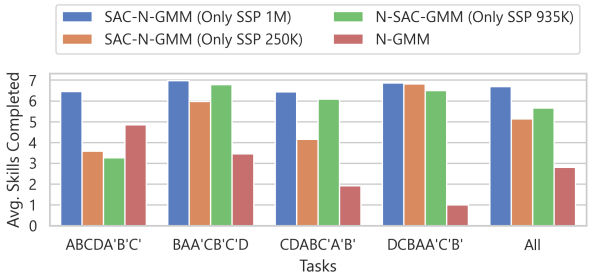

Figure 7. SAC-N-GMM on seven skills

6. Limitations and Future Work

Despite strong task performance, SAC-N-GMM suffers from a few unavoidable drawbacks. Firstly, SAC-N-GMM expects expert demos to train RM-GMMs in Phase I, this can be particularly expensive for some environments. Secondly, despite the ability to explore in Phase II, SAC-N-GMM's diversity in exploratory behaviour is rather limited compared to other end-to-end RL methods. While this makes it suitable for real robots where exploration is random and unrealistic, it could limit SAC-N-GMM's application space. Using geometric skills in Phase I also hinders SAC-N-GMM's ability to tackle skills that involve diverse behaviours like pick-place of objects in complex environments.

SAC-N-GMM also demands a stable geometric skill-learning library like [11]. Before fully adopting [11]'s official implementation, we spent hundreds of hours making RM-GMMs work with CALVIN skills. Moreover, our proposed quaternion refinement update rule, despite being stable and intuitive, is an ad-hoc approach with no theoretical guarantees. Recent attempts to address this using Wasserstein Gradient Flows [12] may be a good future work to change this.

7. Conclusion

In this work, we present SAC-N-GMM, a robot learning framework to refine and sequence multiple robot skills to perform tasks. This hybrid model combines data-efficient, robust Riemannian manifold dynamical systems with flexible, high-dimensional deep RL. Extensive experiments and ablations in the CALVIN simulator show that our proposed method (1) learns a single RL agent to refine and sequence multiple robot skills through physical interactions, (2) refines not just robot positions but also robot orientations, (3) performs significantly better sample efficiently than several model-based RL baselines.

References

[1] Tuomas Haarnoja, Aurick Zhou, P. Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *ArXiv*, abs/1801.01290, 2018. 3

[2] Danijar Hafner, J. Pa*vs*.ukonis, Jimmy Ba, and Timothy P. Lillicrap. Mastering diverse domains through world models. *ArXiv*, abs/2301.04104, 2023. 5

[3] Nicklas Hansen, Xiaolong Wang, and Hao Su. Temporal difference learning for model predictive control. In *International Conference on Machine Learning*, 2022. 5

[4] Seyed Mohammad Khansari-Zadeh and Aude Billard. Learning stable nonlinear dynamical systems with gaussian mixture models. *IEEE Transactions on Robotics*, 27:943–957, 2011. 3

[5] Corey Lynch, Mohi Khansari, Ted Xiao, Vikash Kumar, Jonathan Tompson, Sergey Levine, and Pierre Sermanet. Learning latent plans from play. *CoRR*, abs/1903.01973, 2019. 2

[6] Oier Mees, Lukás Hermann, Erick Rosete-Beas, and Wolfram Burgard. Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks. *IEEE Robotics and Automation Letters*, 7:7327–7334, 2021. 5

[7] William Montgomery, Anurag Ajay, Chelsea Finn, Pieter Abbeel, and Sergey Levine. Reset-free guided policy search: Efficient deep reinforcement learning with stochastic initial states. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3373–3380, 2017. 5

[8] Iman Nematollahi, Erick Rosete-Beas, Adrian Röfer, Tim Welschehold, Abhinav Valada, and Wolfram Burgard. Robot skill adaptation via soft actor-critic gaussian mixture models. *2022 International Conference on Robotics and Automation (ICRA)*, pages 8651–8657, 2021. 1, 3

[9] J Schulman. Deep reinforcement learning. https://www.youtube.com/watch?v=aUrX-rPss4, 2016. 5

[10] Lu Shi, Joseph J. Lim, and Youngwoon Lee. Skill-based model-based reinforcement learning. *CoRL*, abs/2207.07560, 2022. 5

[11] Martijn J. A. Zeestraten, Ioannis Havoutis, João Silvério, Sylvain Calinon, and Darwin Gordon Caldwell. An approach for imitation learning on riemannian manifolds. *IEEE Robotics and Automation Letters*, 2:1240–1247, 2017. 2, 3, 4, 5, 7

[12] Hanna Ziesche and Leonel Dario Rozo. Wasserstein gradient flows for optimizing gaussian mixture policies. *NeurIPS*, abs/2305.10411, 2023. 7