Robot lifelong learning with composable diffusion models on edge devices

TIANQI ZHU

Enabling lifelong learning in robots requires models that can continuously adapt to evolving tasks, environments, and user preferences while operating under strict computational and privacy constraints. We propose a framework for *robot lifelong learning with composable diffusion models on edge devices*, where complex robot behaviors are represented as compositions of lightweight diffusion modules trained incrementally over time. Each module captures a reusable skill, preference, or environmental dynamic, and compositions are formed through learned conditioning and guidance mechanisms without retraining the full system. To support on-device deployment, we introduce parameter-efficient adaptation strategies and selective memory replay that bound compute, memory, and energy usage on edge hardware. The resulting system mitigates catastrophic forgetting, enables rapid skill recombination, and preserves data locality by keeping learning and inference fully on-device.

1 Related Work

1.1 Continual and Reinforcement Learning for Robotics

Continual learning in robotics seeks to enable agents to acquire new skills over time while retaining previously learned behaviors under non-stationary task and environment distributions. Early approaches adapt continual learning techniques from supervised learning to reinforcement learning (RL), including regularization-based methods that constrain parameter updates to preserve prior knowledge [15], rehearsal and experience replay strategies [21], and dynamic network expansion methods [22]. While effective in mitigating catastrophic forgetting, many of these approaches assume centralized training, access to large replay buffers, or clearly defined task boundaries, which are often unrealistic in long-term robotic deployments.

In reinforcement learning for robotics, continual learning has been studied through lifelong policy learning [24], multi-task and multi-goal RL [8, 13], and hierarchical or option-based frameworks [4, 23]. Skill discovery and reuse play a central role in these methods, with learned options or sub-policies serving as reusable building blocks for more complex behaviors. However, hierarchical and option-based approaches typically rely on discrete skill abstractions and fixed temporal scales, limiting their flexibility when tasks evolve continuously or when fine-grained behavioral composition is required.

More recent work explores modular and compositional architectures for continual RL, in which policies are decomposed into reusable components that can be selectively adapted or recombined [1, 2]. Such designs improve transfer and scalability but often depend on hand-designed module interfaces or require retraining when introducing new compositions. Moreover, most continual RL approaches focus on policy optimization objectives and do not explicitly model uncertainty or multimodality in action distributions, which are critical for robust robotic control in dynamic and partially observed environments.

Generative policy representations provide an alternative to purely reactive policies by modeling distributions over trajectories or actions. Generative models have been applied to imitation learning and offline RL for robotics [9, 12], demonstrating improved robustness and expressivity. However, their application to continual and lifelong reinforcement learning remains limited, particularly in resource-constrained, on-device settings. Our work builds on these lines of research by integrating continual reinforcement learning with compositional generative models, enabling incremental skill acquisition and recombination without centralized retraining.

Author's Contact Information: Tianqi Zhu, tianqizhu@u.nus.edu.

1.2 Composable Robot Learning

A long-standing approach to composability in robot learning is to build reusable *skills* and *temporal abstractions* that can be sequenced or invoked as higher-level actions. Early work in lifelong robot learning emphasized accumulating knowledge over time to improve future performance across tasks [25]. In reinforcement learning, the options framework formalized temporally extended actions (initiation sets, policies, and termination conditions), enabling hierarchical composition and reuse [23]. Building on this foundation, skill discovery and library-building methods aim to automatically induce reusable subpolicies that can be chained to solve long-horizon problems, such as skill chaining in continuous domains [16]. Beyond RL-only pipelines, skill libraries can also be expanded from demonstrations by segmenting unstructured behavior into reusable primitives and composing them into higher-level task representations [19]. Parameterized skills further generalize composition by conditioning primitives on continuous task parameters (e.g., goal pose or object location), enabling interpolation and transfer across task instances [7].

A complementary line of work studies *architectural* modularity: instead of composing actions, it composes functional modules within a network. Modular neural architectures dynamically assemble reusable components to solve compositional queries [3], inspiring analogous designs in robotics where perception, grounding, and control can be recombined across tasks. Such modularity is appealing for lifelong/continual settings because it localizes updates and reduces interference between tasks, while preserving reuse.

Recently, diffusion models have introduced a generative perspective on composition. In vision, composable diffusion approaches show that combining generative components can support systematic generalization to more complex compositions at test time [18]. For robotics, diffusion-based policies cast action generation as conditional denoising over action sequences, offering a natural mechanism to represent multi-modal behaviors and to integrate priors and constraints at inference [5]. These ideas motivate *composable diffusion policies* for robots: a library of generative skill components that can be combined (e.g., via energy/score composition, guidance, or constraint projection) to synthesize behaviors for new task compositions, while remaining compatible with edge deployment constraints through modular updates and selective on-device adaptation.

1.3 Lifelong and Continual Learning on Distributed Edge Devices

Lifelong and continual learning (CL) aim to enable models to adapt to evolving, non-stationary data distributions while mitigating catastrophic forgetting. These challenges are amplified in distributed edge environments, where learning systems must operate under strict constraints on compute, memory, energy, and communication, while often lacking reliable access to centralized cloud resources. As a result, edge-based continual learning has emerged as a distinct research direction at the intersection of systems, machine learning, and distributed optimization.

Early work on edge intelligence primarily focused on efficient inference and cloud-assisted training pipelines, limiting the degree of long-term on-device adaptation. More recent on-device CL frameworks explicitly target incremental learning under resource constraints. For example, *Etuner* introduces convergence-aware layer freezing and incremental updates to reduce training overhead and energy consumption on edge hardware. *Miro* further improves efficiency through adaptive memory allocation and energy-aware configuration selection, dynamically balancing episodic replay buffers and streaming data to optimize performance under varying workloads. Complementary sparsity-driven approaches such as *SparCL* apply task-aware pruning and gradient masking to preserve essential parameters while reducing computation and communication costs.

Beyond single-device settings, distributed and federated continual learning frameworks address scalability and heterogeneity across multiple edge clients. Methods such as *FedWeIT* [28] introduce parameter isolation and selective sharing to mitigate interference across tasks and devices, while *CLAD* [26] leverages adaptive distillation and replay-based coordination to stabilize learning in non-iid, distributed environments. These approaches demonstrate that modularity, selective knowledge transfer, and replay are critical for maintaining performance in large-scale edge deployments. However, most existing distributed CL systems rely on monolithic model architectures and episodic synchronization phases, limiting flexibility for fine-grained skill composition and continuous adaptation.

In robotics and embodied AI, these limitations are particularly pronounced due to real-time control requirements, safety constraints, and highly non-stationary sensory streams. While prior work establishes key building blocks—incremental updates, adaptive memory management, sparsity, and federated coordination—there remains a need for unified frameworks that combine these principles with expressive, modular policy representations. Our work addresses this gap by enabling lifelong learning on edge devices through composable generative models, allowing behaviors to be incrementally acquired, recombined, and updated while respecting the computational and memory constraints inherent to distributed edge systems.

2 System Design Discussion

There are several possible ways to model human preference. Some of these could be characterized by a simple rule, for example, when a human is left handed, we call it rule-based preference. There other preferences could be more continuous more difficult to characterize , for example, someone would prefer things put on the shelf to space evenly from each other to take easily, while others would prefer pushing things together for more empty space. These preference could be modeled as latent vectors [17], natural-language representations [6, 20], or generative models that capture a distribution over preferences from which latent vectors can be sampled [10, 14, 27]. We denote those preferences as $P = \{p_1, p_2, ..., p_n\}$. Though these preferences might not be totally independent from each other, we denote the set of all the preference vectors as the preference dictionary P. Suppose a household robot has several components, such as navigating across different rooms, looking for misplaced objects, rearranging objects to tidy up rooms etc. Each component for performing the corresponding tasks could be affected by a subset of all preferences. If a model (or a module) for performing a task could be denoted as $M : O \to A$, where O denotes the observation, and A denotes the space of actions, the model could be conditioned on the preference vectors involved to take the preferences into consideration: $M : (O, P) \to A$. There is another possibility how preference could be included by the lower level models (or modules) that perform the specific actions by model coupling operations, which we represent using $\otimes$. i.e. $M_T : O \to A$ and $M_T = M_a \otimes M_p$, where M_T represents the task performing module, M_a is action performing module with only dynamics and physics information without human parameters, and M_p could be either a generative model representing distribution of human preference taken into consideration, or another model incorporating human factors but in a better format for the coupling operation with the action performing model. For the system to be able to adapt to changes to human preferences and environment factors, while M_T drives a robot to perform a certain task, the submodules that M_T composed of would also need to be continually updated.

In order to protect users' privacy, we would prefer the models encoding user preference, and the user specific training data to be stored only locally.

There are several sources where the preferences could be learned from, learning from LLM (or obtaining directly from LLM), and learning from specific user data when actions being performed online. For the former, we would like to minimize the LLM query costs on compute and API charges. For the latter, we would like to achieve high sample efficiency, and adapt quickly to changing human factors. We would also want the action models themselves to be preference agnostic, but could be conditioned on preference, or be coupled with preference models.

3 Problem Statement

We consider the problem of enabling autonomous service robots operating in a smart-home environment to learn and adapt to user preferences over long periods of time, while satisfying strict constraints on safety, privacy, latency, and computational resources. The smart home contains heterogeneous devices with varying compute capabilities (e.g., smart TVs, gaming PCs, laptops) in addition to the robots themselves. These devices may intermittently provide spare GPU or accelerator capacity. The central challenge is to design a system that performs both *real-time inference* and *continual preference learning* using a collection of modular learned components (e.g., trajectory models, preference models, constraint models), without relying on cloud-based computation for sensitive data or model updates.

Formally, let $\mathcal{M} = \{M_1, \ldots, M_K\}$ denote a set of composable model components that govern robot behavior. Each component M_i exposes a common interface for inference and training, abstracted as

$$\texttt{forward}(x; \theta_i), \qquad \texttt{update}(\theta_i, \mathcal{B}),$$

where θ_i are learnable parameters and $\mathcal{B}$ is a batch of data derived from robot experience or user feedback. Components may represent trajectory diffusion models, preference diffusion models, safety constraints, energy-efficiency models, or future algorithmic modules.

The problem is to design:

1. A **distributed execution architecture** that assigns inference and training operations on $\{M_i\}$ to robots and home devices with minimal latency and minimal interference with real-time control.
2. A **privacy-preserving learning mechanism** that ensures all user preference data and model updates remain strictly within the home environment, without reliance on external cloud services.
3. A **workload scheduling and orchestration strategy** that leverages heterogeneous and intermittently available GPUs to accelerate learning and high-level inference while ensuring safe fallback when such devices become unavailable.
4. A **modular composition framework** that supports combining models in $\mathcal{M}$ to generate robot behavior, and remains extensible to new types of composable models beyond diffusion-based ones.
5. A **lifelong learning objective** that continuously adapts user-specific parameters (e.g., LoRA adapters) based on feedback data $\mathcal{D}_t$, while avoiding catastrophic forgetting and preserving stable performance over time.

The system must produce robot trajectories τ_t that satisfy:

$$\tau_t = \texttt{compose}\big(\{M_i.\texttt{forward}(\cdot)\}_{i=1}^{K}\big),$$

subject to the constraints that:

- inference on τ_t must meet the robot's real-time safety and control deadlines,
- training on $\{\theta_i\}$ must not leak sensitive preference information beyond the home,
- inference and training must coexist without degrading responsiveness or safety,
- updates to user-specific models must remain computationally feasible on available in-home hardware.

In summary, the core problem addressed in this work is how to architect and orchestrate a privacy-preserving, distributed, and modular lifelong-learning system for smart-home robots, such that both inference and training workloads are effectively partitioned and scheduled across robots and heterogeneous local compute resources. The solution must generalize beyond diffusion models and remain robust under real-world constraints of intermittent compute, network variability, and user-driven preference evolution.

4 Composable diffusion models for robot lifelong learning system

4.1 Introduction

Robotic systems deployed in household, industrial, rehabilitation, or assistive settings must adapt continuously to human users. User preferences—e.g., trajectory style, comfort, speed, safety margins, or interaction modality—change over time.

Traditional reinforcement learning (RL) or classical control systems struggle with:

- catastrophic forgetting when continually adapting,
- difficulty composing multiple constraints cleanly,
- limited ability to represent multi-modal behavior,
- reliance on brittle, hand-tuned reward functions.

Diffusion models provide an alternative by generating full trajectories and evaluating trajectories in a way that is:

- multi-modal,
- gradient-based,
- modular and compositional,
- stable for continual (online) learning.

This opens the door to lifelong preference-adaptive robots whose behavior emerges from the composition of multiple diffusion models, each encoding a specific constraint or preference.

4.2 System Overview

We consider a robot system built from four core diffusion-based modules, each independent but composable:

(1) **Trajectory Diffusion Model (TDM)**: Models expert-like or task-optimal trajectories,

$$p_\theta(\tau), \tag{1}$$

where τ denotes a full trajectory (states and possibly actions).

(2) **Preference Diffusion Model (PDM)**: Encodes user-specific preferences via a score model,

$$s_\phi(\tau) = \nabla_\tau \log p_\phi(\text{pref} \mid \tau). \tag{2}$$

(3) **Constraint Diffusion Modules (CDMs)**: Each constraint type—e.g., safety distance, energy efficiency, social comfort—has its own diffusion-based score,

$$s_{\psi_k}(\tau), \quad k = 1, \ldots, K. \tag{3}$$

(4) **Continual Preference Adapter (CPA)**: Small low-rank (LoRA-style) parameter modules updated online to incorporate new user feedback without overwriting previous knowledge.

The robot ultimately plans by sampling a trajectory from a composed diffusion system with effective score

$$\tilde{s}(\tau) = s_\theta(\tau) + \alpha\, s_\phi(\tau) + \sum_{k=1}^{K} \beta_k\, s_{\psi_k}(\tau), \tag{4}$$

where α and β_k are scalar weights that control the relative strength of preferences and constraints.

This score guides the generative denoising process that produces a trajectory satisfying all relevant constraints.

4.3 Diffusion-Based Trajectory Planning

4.3.1 Trajectory Diffusion (Diffuser-Style Models). The trajectory diffusion model learns a denoising process over full trajectories. We denote the trajectory at diffusion time t by x_t, and a goal or task condition by goal. The denoising model is trained to reverse a forward noising process:

$$x_t = \text{denoise}_\theta(x_t, t, \text{goal}). \tag{5}$$

Advantages of trajectory diffusion include:

- natural handling of long-horizon dependencies,
- support for goal conditioning and context,
- generation of multi-modal plans,
- avoidance of RL training instability.

4.3.2 Score-Based Planning. At inference time, planning is implemented as integration of a reverse-time stochastic differential equation (SDE) or ordinary differential equation (ODE). For an SDE formulation, the reverse process can be written as:

$$d\tau = \tilde{s}(\tau, t)\, dt + \sigma(t)\, dW_t, \tag{6}$$

where W_t is a Wiener process, $\sigma(t)$ is a noise schedule, and $\tilde{s}(\tau, t)$ is the composed score that includes trajectory plausibility, user preferences, and other constraints.

In an ODE formulation, the dynamics reduce to:

$$\frac{d\tau}{dt} = \tilde{s}(\tau, t). \tag{7}$$

4.4 Preference Diffusion for Reward-Like Guidance

4.4.1 Preference Modeling. A preference diffusion model learns a distribution over user judgments conditioned on trajectories. For example, it may model:

$$p_\phi(\text{pref} \mid \tau), \tag{8}$$

or pairwise comparisons such as

$$p_\phi(\tau_A \succ \tau_B), \tag{9}$$

where $\tau_A \succ \tau_B$ denotes that trajectory τ_A is preferred to τ_B.

The model outputs score gradients with respect to the trajectory:

$$s_\phi(\tau) = \nabla_\tau \log p_\phi(\text{pref} \mid \tau), \tag{10}$$

which act like reward gradients but avoid committing to a single scalar reward.

4.4.2 Continual Learning with a Preference ODE. User feedback arrives over time. We adapt the preference diffusion parameters $\phi(t)$ via a continual learning ODE of the form:

$$\frac{d\phi}{dt} = -\eta\, \nabla_\phi L_{\text{pref}}(\phi; \mathcal{D}_{\text{new}}(t)) - \lambda\, \nabla_\phi L_{\text{replay}}(\phi; \mathcal{B}) - \gamma\, (\phi - \phi_0) + \sigma\, \xi(t), \tag{11}$$

where:

- $\mathcal{D}_{\text{new}}(t)$ is the stream of new preference data at time t,
- $\mathcal{B}$ is a replay buffer of past preference examples,
- L_{pref} is the loss on new data (e.g., score-matching or pairwise preference loss),
- L_{replay} is the loss on replayed examples,
- ϕ_0 is a stable base set of parameters or a previous snapshot,
- $\xi(t)$ is a noise term supporting diffusion-style stability.

This ODE implements online preference adaptation with replay-based lifelong learning and regularization toward a stable base model.

4.5 Modular Diffusion Constraints

4.5.1 Safety Diffusion Module. A safety diffusion module is trained on safe trajectories or on labels indicating safe vs. unsafe behavior. It defines a score

$$s_{\psi_1}(\tau) = \nabla_\tau \log p_{\psi_1}(\text{safe} \mid \tau), \tag{12}$$

which pushes the generated trajectory into regions of high safety probability.

4.5.2 Energy and Smoothness Module. Energy or smoothness preferences can likewise be encoded by a diffusion model trained on low-energy or smooth trajectories:

$$s_{\psi_2}(\tau) = \nabla_\tau \log p_{\psi_2}(\text{low-energy} \mid \tau). \tag{13}$$

4.5.3 Social and Human Comfort Module. Social norms, such as maintaining a certain distance from humans or limiting sudden accelerations, can be modeled by a module capturing social comfort:

$$s_{\psi_3}(\tau) = \nabla_\tau \log p_{\psi_3}(\text{comfortable} \mid \tau). \tag{14}$$

4.5.4 Environmental Affordance Module. Environmental affordances, such as feasible motions given a map or obstacle configuration, can be handled by a diffusion module conditioned on environment state e:

$$s_{\psi_4}(\tau, e) = \nabla_\tau \log p_{\psi_4}(\tau \mid e). \tag{15}$$

Each constraint module is separate and can be plugged into the overall system without retraining the others.

4.6 Modular Score Composition

During denoising, the robot uses a composed score function that combines all modules:

$$\tilde{s}(\tau, t) = s_\theta(\tau, t) + \alpha\, s_\phi(\tau, t) + \sum_{k=1}^{K} \beta_k\, s_{\psi_k}(\tau, t). \tag{16}$$

This is analogous to classifier or guidance-based conditioning in image diffusion models, and it allows:

- plug-and-play constraint modules,

- adjustable constraint weights,
- gradient-based preference shaping,
- multi-objective control without explicit scalar reward design.

Adding, removing, or modifying constraints is modular and does not require retraining the trajectory diffusion model.

4.7 Lifelong Learning Architecture

4.7.1 Sources of User Feedback. User feedback can come from multiple channels, including:

- explicit ratings of trajectories,
- pairwise comparisons (trajectory A vs. trajectory B),
- kinesthetic demonstrations,
- natural language descriptions of preferences,
- implicit metrics, such as measured comfort or jerk tolerance.

4.7.2 Updating the Preference Diffusion Model. A central requirement of our system is to update user-specific preference models continuously while maintaining real-time inference performance, avoiding catastrophic forgetting, and preserving user privacy. To achieve this, we use low-rank adapters (LoRA) [11] as the primary mechanism for online parameter updates in the preference diffusion model and, optionally, in other composable modules such as safety or social-comfort components.

LoRA provides a low-overhead method for injecting user-specific corrections into large neural models without modifying the full parameter matrices. Let W denote a weight matrix of a model component M_i (e.g., an attention or fully connected layer within a diffusion score network). LoRA decomposes updates into a low-rank residual

$$W_{\text{eff}} = W + \Delta W, \qquad \Delta W = BA, \tag{17}$$

where $A \in \mathbb{R}^{r\times d}$ and $B \in \mathbb{R}^{d\times r}$ are trainable matrices of rank $r \ll d$. During lifelong preference learning, the base weight matrix W remains frozen and only (A, B) are updated. This design provides several advantages for a distributed smart-home environment.

Using LoRA or adapter-based updates, the system can:

- keep the base preference diffusion model frozen,
- train small adapter modules on new preference data,
- maintain a replay buffer to avoid catastrophic forgetting.

The continual update of adapters approximates integration of the preference ODE in discrete time.

4.7.3 Ongoing Personalization Loop. A high-level loop for lifelong personalization is:

(1) Generate candidate trajectories using the composed diffusion system.
(2) Present options to the user or observe implicit feedback.
(3) Update the preference diffusion model (and its adapters) using the new feedback.
(4) Recompose the overall score with the updated s_ϕ.
(5) Use the updated composition to generate future trajectories.

Over time, the robot's behavior becomes increasingly aligned with the user's preferences.

4.8 Benefits of Diffusion Composition in Robotics

4.8.1 Multi-Constraint Satisfaction. Diffusion score composition allows simultaneous handling of:

- task goals,
- user preferences,
- safety norms,
- dynamics and environmental constraints,

without hand-engineering scalar rewards or complex multi-objective optimization schemes.

4.8.2 Incremental User Adaptation. Preferences can be updated:

- in real time,
- in a non-destructive way,
- without retraining the core planner.

4.8.3 Uncertainty-Aware and Multi-Modal Behavior. Diffusion models naturally represent uncertainty and multiple viable trajectories, enabling robust behavior in ambiguous or novel situations.

4.8.4 Modularity and Extensibility. Each new constraint (safety, energy, comfort, task-specific rules) can be added as a new diffusion module. The system remains modular and extensible as more constraints or preference dimensions are introduced.

4.9 Case Study: Household Object Pickup

Consider a household robot performing object pickup and delivery:

- The trajectory diffusion model generates candidate navigation and manipulation paths.
- The safety module ensures collision-free behavior and respect of safety zones.
- The preference diffusion module biases trajectories toward slower motion for an elderly user, or faster paths for an experienced user.
- Continuous user feedback (e.g., “too fast”, “too close”) updates the preference adapters over days or weeks.

Over time, the robot learns a personalized style of motion that balances efficiency, safety, and comfort in a user-specific way.

4.10 Conclusion

Composing diffusion models provides a principled, modular, and flexible architecture for lifelong preference-adaptive robots. By representing trajectory generation, safety, energy efficiency, and human preferences as diffusion-based score functions, robot behavior becomes the result of modular score superposition.

Continual updating of the preference diffusion model via LoRA or continual gradient flow supports real-time, on-device adaptation without catastrophic forgetting. This framework enables robots that:

- learn continuously from user feedback,
- respect multiple constraints simultaneously,
- adapt safely and personally over long-term deployment.

5 Smart-Home System Design for Private Distributed Lifelong Learning

We now specialize our system design to a smart-home deployment in which one or more household robots operate alongside a heterogeneous set of local compute devices (smart TVs, desktops, laptops, gaming consoles). Our primary goal is to keep user preference learning entirely private by confining all training data and optimization to the home network, while opportunistically exploiting idle GPUs across devices to accelerate diffusion-based training. Under this design, the home environment forms a small, private edge cluster that jointly supports both inference and lifelong preference adaptation.

5.1 In-Home Edge Cluster Architecture

The smart home comprises three logical components:

(1) **Robotic Agents.** One or more robots equipped with sensors, actuators, and a lightweight onboard compute module. They perform local perception, low-level control, and latency-critical planning.
(2) **Home Orchestrator.** A central coordination service, typically running on a home server, router-attached mini-PC, or any always-on device. It discovers available GPUs/accelerators on the local network, schedules training and heavy inference workloads, and manages model and adapter versions.
(3) **Auxiliary Compute Nodes.** General-purpose devices in the home (e.g., smart TV, gaming PC, laptop) that may expose a GPU or NPU. These join the home cluster when powered and idle, contributing spare capacity for training and batch inference.

Crucially, all data and model updates remain within the home network. The cloud is used only for optional retrieval of *generic* base models and software updates; no user-specific preference gradients or logs are transmitted outside.

5.2 Workload Placement Under Privacy and Resource Constraints

We allocate workloads across the robot and the in-home cluster with two priorities: (i) respecting strict real-time requirements for control and safety, and (ii) exploiting the cluster's pooled compute for training and heavy diffusion inference.

On-Robot Workloads. Each robot maintains a self-sufficient inference loop, independent of external connectivity:

- **Real-time inference.** A distilled trajectory diffusion model and safety-critical constraint modules (collision envelopes, joint limits) run at high frequency, supporting short-horizon replanning and local reflexes.
- **Preference application.** The robot holds a compact copy of user-specific preference adapters (e.g., LoRA modules) and applies them during inference without running any heavy optimization.
- **Data logging.** The robot logs user feedback and interaction traces (e.g., accepted / rejected trajectories, overrides, comfort events) into local encrypted storage.

These workloads are bounded to ensure the robot can satisfy 10–1000 Hz control loops and remain safe even if no other devices are available.

Cluster Training and Heavy Inference. The home orchestrator discovers and manages auxiliary compute nodes that expose GPUs. On these nodes, it schedules:

- **Full-capacity diffusion inference.** When latency permits, long-horizon trajectory diffusion and non-critical constraint modules (energy, social navigation, multi-agent coordination) are executed on cluster GPUs to generate coarse global plans or candidate trajectories.
- **Preference model training.** All optimization for user-specific preference diffusion—including adapter updates and small replay buffers—is performed on devices within the home. The training data originate from robot logs and are transmitted over the local network, optionally preprocessed or anonymized, but never leave the home boundary.
- **Periodic constraint refinement.** If desired, constraint diffusion modules can also be fine-tuned using in-home experience (e.g., particular furniture layouts or habitual human traffic patterns).

This design transforms the home into a small private cluster for lifelong learning, while keeping the robot's on-board workload limited and predictable.

5.3 Private In-Home Lifelong Preference Learning

Lifelong learning proceeds as a continuous, privacy-preserving process entirely inside the home network.

Local Data Collection and Sharding. Robots collect preference-relevant data (trajectory rollouts, user corrections, scalar feedback) and store them locally. The orchestrator periodically queries the robots for new training-ready batches. Data are sharded across cluster nodes to balance load, but at no point are they uploaded to external services.

Distributed Edge Training of Preference Models. We adopt an "in-home federated" pattern: each cluster node trains on its assigned shard of preference data using a frozen base preference diffusion model and trainable adapters. The orchestrator aggregates adapter updates or gradients within the home, producing a new set of user-specific parameters. Because all communication occurs locally and models never cross the home boundary, privacy is preserved by construction.

Adapter Deployment and Versioning. Once training completes for a given round, updated adapters are pushed back to the robots. The robots maintain a small cache of recent adapter versions, enabling rollback in case of instability. Updates are applied asynchronously and non-blockingly; inference continues using the previous adapter until the new version is safely loaded.

5.4 Concurrent Training and Inference in the Home Cluster

Training and inference must coexist on constrained hardware without violating the robot's timing requirements or saturating shared devices.

Priority Scheduling and Throttling. The orchestrator assigns highest priority to inference-related cluster tasks (e.g., long-horizon planning) requested by robots. Preference training jobs are scheduled at lower priority and can be throttled or paused when devices are actively used by humans (e.g., gaming on the PC, streaming on the TV). This prevents learning from interfering with user experience or robot responsiveness.

Asynchronous Parameter Updates. Robots never block their control loop waiting for training. Instead, they periodically poll the orchestrator or receive push notifications when new adapters or constraint parameters become available. The integration of new parameters is designed to be lightweight (e.g., swapping a small adapter matrix), so that updates can occur between control cycles.

Fallback Modes. If no auxiliary compute is available (e.g., all devices powered off), the robot falls back to purely on-board inference using its distilled diffusion models and the last-known adapter. Lifelong learning pauses transparently and resumes when the cluster becomes available again.

5.5 Resource Discovery, Load Balancing, and Energy Awareness

Because smart-home devices are heterogeneous and intermittently available, the orchestrator must dynamically manage resources:

- **Resource discovery.** Devices periodically advertise their capabilities (GPU type, memory, current load, energy policy) over the local network. The orchestrator maintains a live inventory of active nodes.
- **Load balancing.** Training jobs are partitioned across nodes according to their capacity and current utilization. Heavy preference diffusion training may be assigned to a gaming PC, while lighter jobs execute on a smart TV SoC GPU.
- **Energy-aware scheduling.** To avoid unnecessary power consumption, training is biased toward times when devices are idle and plugged in (e.g., overnight). Users may configure schedules and caps to limit background training.

These mechanisms allow the system to continuously adapt user preferences while respecting the household's comfort and energy constraints.

5.6 Privacy, Trust, and Design Rationale

The new design is driven by three key considerations:

Local Privacy by Construction. All user-specific preference data and model updates remain within the home network; only generic, non-personalized base models may be fetched from the cloud. This reduces privacy risk and regulatory burden, and aligns with user expectations for smart-home systems.

Maximal Use of Existing Hardware. Instead of relying solely on the robot's limited compute, the system opportunistically harnesses idle GPUs across smart-home devices. This yields faster training and richer models without requiring dedicated hardware.

Robustness and Graceful Degradation. Robots retain full autonomy based on on-board models and the last available adapters, and training simply slows or pauses when cluster resources are unavailable. Thus, the system can exploit additional compute when present but does not depend on it for safety or basic functionality.

Overall, the smart-home design reframes the environment as a private, cooperative edge cluster that jointly supports diffusion-based planning and lifelong preference learning, while keeping all sensitive computation and data on-premise.

6 Algorithmic Workload Distribution for Lifelong Learning on a Home Cluster

We now describe the algorithmic mechanisms that distribute training and inference workloads among robots and heterogeneous home compute nodes. While our concrete instantiation uses diffusion-based trajectory, preference, and constraint modules, the design is intentionally model-agnostic: it supports any library of *composable models* that expose a common interface for scoring, generating, and updating behaviors.

6.1 Abstraction Layers and Model-Agnostic Interfaces

To ensure generality, we introduce three abstraction layers:

Model Components. Each behavioral module (trajectory, preference, safety, energy, social, etc.) is represented as a *model component* M implementing a unified interface:

- `forward`$(x; \theta)$: inference, e.g., score evaluation, next-step prediction, or trajectory refinement.
- `update`$(\theta, \mathcal{B})$: parameter update given a batch $\mathcal{B}$ of training data.
- `compose`$(\{M_i\})$: optional method specifying how outputs of components are combined.

For diffusion models, `forward` corresponds to score or denoising evaluation; for future models (e.g., transformers, energy-based models), `forward` may return logits, energies, or other differentiable outputs.

Composable Model Graph. Robot behavior is defined by a directed acyclic graph (DAG) of components:

$$\mathcal{G} = (\mathcal{V}, \mathcal{E}), \quad \mathcal{V} = \{M_{\text{traj}}, M_{\text{pref}}, M_{\text{safety}}, \dots\}.$$

Edges encode conditioning or composition (e.g., a planner node conditioned on goal and preference embeddings). The graph describes *what* should be computed, but not *where*. This decoupling allows the same high-level behavior to be deployed across different hardware configurations.

Execution Plans and Schedules. The home orchestrator transforms $\mathcal{G}$ into an *execution plan* Π by mapping each node M_i and its operations (`forward`, `update`) to a device in the home cluster. The schedule specifies:

$$\Pi = \big\{(M_i, \text{op}, \text{device}, \text{priority})\big\},$$

where `op` is either inference or training, and priority encodes latency constraints (e.g., real-time, near-real-time, or best-effort).

This separation between component definitions, graph structure, and execution plans ensures that the same algorithmic logic applies whether components are diffusers, transformers, or other composable models.

6.2 Inference Workload Distribution

Inference proceeds as a distributed evaluation of the composable model graph under latency and resource constraints.

6.2.1 *Planning Request Handling.*

When a robot requires a plan (e.g., for a navigation or manipulation task), it:

(1) Constructs a compact *planning request* R containing:

$$R = (\text{task descriptor}, \text{goal}, \text{local env summary}, \text{current adapter version}).$$

(2) Locally evaluates time-critical nodes of $\mathcal{G}$ using on-board models, such as:

$$M_{\text{safety}}.\texttt{forward}(\cdot), \quad M_{\text{traj}}^{\text{distilled}}.\texttt{forward}(\cdot).$$

(3) Transmits R to the orchestrator for optional cluster-assisted planning.

6.2.2 *Cluster-Side Inference Scheduling.*

Upon receiving R, the orchestrator:

(1) **Constructs a subgraph** $\mathcal{G}_{\text{cluster}} \subseteq \mathcal{G}$ that contains components deemed beneficial for offloading, such as full-capacity trajectory and preference models and non-critical constraints.

Algorithm 1 Distributed Inference over Home Cluster

```
Require: Composable graph $\mathcal{G}$, request $R$, device set $\mathcal{D}$
1: $\mathcal{G}_{\text{cluster}} \leftarrow$ select_offload_subgraph($\mathcal{G}$)
2: $\Pi_{\text{inf}} \leftarrow$ plan_inference($\mathcal{G}_{\text{cluster}}, \mathcal{D}, R$)
3: Initialize intermediate cache $C \leftarrow \{\}$
4: for node $M_i$ in topological order of $\mathcal{G}_{\text{cluster}}$ do
5:     $(\text{device}, \text{priority}) \leftarrow \Pi_{\text{inf}}[M_i]$
6:     Submit job $J_i = (M_i.\texttt{forward}, R, C)$ to device with priority
7:     Wait for $J_i$ completion, store outputs in $C$
8: end for
9: Extract cluster proposal(s) $P$ from $C$
10: Robot refines $P$ locally using on-board graph nodes
11: return Final trajectory $\tau$ for execution
```

(2) **Selects devices** for nodes in $\mathcal{G}_{\text{cluster}}$ based on:
   - device capability (GPU/CPU, memory),
   - current utilization,
   - energy policy (e.g., only use TV GPU when idle).

(3) **Builds an execution plan** Π_{inf} assigning each `forward` call to a device, with high priority for steps on the critical path.

Inference then executes as a distributed pipeline:

(1) Nodes are evaluated in topological order respecting data dependencies.
(2) Intermediate representations (e.g., latent trajectories, embeddings) are transmitted between nodes, not raw sensor data.
(3) The cluster produces one or more candidate plans (e.g., coarse trajectories or high-level waypoints).
(4) The robot receives these proposals and performs final refinement using its local components, enforcing local perception and safety constraints.

Algorithm 1 sketches this process.

This algorithm is agnostic to whether `forward` calls correspond to diffusion denoising steps, transformer decoding, or any other compositional model; the only requirement is that components provide a consistent interface and can be ordered into a DAG.

6.3 Training Workload Distribution

Lifelong learning is expressed as distributed `update` operations on selected parameters (primarily adapters) of chosen components, orchestrated across the home cluster while respecting device availability and user constraints.

6.3.1 Data Flow and Sharding. Each robot maintains a local buffer of preference and interaction data:

$$\mathcal{D}_{\text{robot}} = \{(\tau, f, c, \dots)\},$$

where τ are trajectories, f are feedback signals (ratings, comparisons), and c are context variables. Periodically, the orchestrator:

(1) Requests new training examples from robots.

Algorithm 2 In-Home Distributed Adapter Training

```
Require: Components $\{M_j\}$ with trainable adapters $\{\theta_j^{\text{adapt}}\}$, home data $\mathcal{D}_{\text{home}}$, devices $\mathcal{D}$
1: $\{\mathcal{B}_k\} \leftarrow$ shard($\mathcal{D}_{\text{home}}, \mathcal{D}$)
2: for each device $d \in \mathcal{D}$ in parallel do
3:     Assign batch $\mathcal{B}_d$ and adapter snapshot $\{\theta_j^{\text{adapt}}\}$ to $d$
4:     On $d$, perform $T$ local steps:
           $\Delta\theta_{j,d}^{\text{adapt}} \leftarrow$ local_update($M_j, \theta_j^{\text{adapt}}, \mathcal{B}_d$)
5:     Return deltas $\{\Delta\theta_{j,d}^{\text{adapt}}\}$ to orchestrator
6: end for
7: Aggregate updates:
           $\theta_j^{\text{adapt}} \leftarrow \theta_j^{\text{adapt}}$ + aggregate($\{\Delta\theta_{j,d}^{\text{adapt}}\}_d$)
8: Validate adapters on held-out data (optionally on a designated node)
9: If stable, deploy new adapters to robots
```

(2) Partitions the aggregated dataset $\mathcal{D}_{\text{home}}$ into shards $\mathcal{B}_1, \ldots, \mathcal{B}_K$ based on device capacity and locality.
(3) Assigns each shard $\mathcal{B}_k$ to a compute node for training.

6.3.2 In-Home Federated Adapter Training. We treat preference learning as an *in-home federated* process over adapters of relevant components (typically the preference model, but potentially also trajectory or constraint components). Each node performs local optimization steps on its shard and returns parameter deltas to the orchestrator, which aggregates them.

Algorithm 2 outlines one training round.

The `local_update` and `aggregate` routines are model-agnostic and may implement SGD, Adam, or any preferred optimizer and averaging scheme. For diffusion models, `local_update` might perform score-matching or preference comparison losses; for transformers, it may minimize cross-entropy or ranking losses. The orchestration logic remains unchanged.

6.4 Concurrency and Priority Policies

Training and inference tasks share the same physical hardware but differ in priority:

- **Inference jobs** (e.g., planning requests) receive *high* priority and preempt or delay training jobs if necessary.
- **Training jobs** are best-effort, scheduled when devices are idle or lightly loaded, and possibly constrained by user-configured energy policies.

The orchestrator maintains separate queues for inference and training jobs per device and uses a priority-aware scheduler, for example:

- highest-priority queue: cluster-assisted planning for active robots,
- medium-priority queue: batch evaluation tasks (e.g., simulation rollouts),
- low-priority queue: adapter training and constraint refinement.

This scheduling strategy is independent of the underlying model family; any component that exposes `forward` and `update` can be slotted into the same framework.

6.5 Generalization Beyond Diffusion Models

Although we have framed our system around diffusion-based planning and preference learning, the abstractions are deliberately generic:

- The composable model graph can represent any hierarchical or factorized policy, including transformer-based planners, energy-based controllers, or value/policy ensembles.
- The composition operation need not be linear score addition; `compose` can implement more general fusion operators (e.g., attention-based aggregation, gating networks) as long as dependencies can be expressed in a DAG.
- Training and inference distribution algorithms depend only on the existence of `forward` and `update` interfaces and do not encode diffusion-specific assumptions.

Therefore, future components can be introduced (e.g., language-conditioned instruction models, symbolic planners) without requiring structural changes to the workload distribution mechanism. Only their computational profiles and priority requirements affect how the orchestrator maps them to devices, preserving the core architectural principles of privacy, modularity, and graceful degradation.

7 Acknowledgment

A version of the report has been submitted for CS6216 project at National University of Singapore.

References

[1] Ferran Alet, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2018. Modular meta-learning. In *Conference on Robot Learning*. 856–868.

[2] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Neural module networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 39–48.

[3] Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. 2016. Neural Module Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 39–48.

[4] Pierre-Luc Bacon, Jean Harb, and Doina Precup. 2017. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 31.

[5] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin C. M. Burchfiel, Russ Tedrake, and Shuran Song. 2023. Diffusion Policy: Visuomotor Policy Learning via Action Diffusion. *Robotics: Science and Systems (RSS)* (2023). https://arxiv.org/abs/2303.04137

[6] Paul F. Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep Reinforcement Learning from Human Preferences. In *Advances in Neural Information Processing Systems*, Vol. 30. 4299–4307.

[7] Bruno C. Da Silva, George D. Konidaris, and Andrew G. Barto. 2012. Learning Parameterized Skills. *arXiv preprint arXiv:1206.6398* (2012). https://arxiv.org/abs/1206.6398

[8] Tuomas Haarnoja, Vivian Pong, Aurick Zhou, Murtaza Dalal, Pieter Abbeel, and Sergey Levine. 2018. Latent space policies for hierarchical reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning*. 1851–1860.

[9] Jonathan Ho and Stefano Ermon. 2016. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems*, Vol. 29.

[10] Neil Houlsby, Ferenc Huszár, Zoubin Ghahramani, and Máté Lengyel. 2012. Collaborative Gaussian Processes for Preference Learning. In *Advances in Neural Information Processing Systems*, Vol. 25.

[11] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen Wang. 2022. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv preprint arXiv:2106.09685* (2022).

[12] Michael Janner, Qiyang Li, and Sergey Levine. 2021. Offline reinforcement learning as one big sequence modeling problem. *Advances in Neural Information Processing Systems* 34 (2021).

[13] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4 (1996), 237–285.

[14] Raza Khan, David Nott, and Christopher Drovandi. 2014. Fast Preference Learning with Bayesian Mallows Models. *Journal of Machine Learning Research* 15, 1 (2014), 2487–2517.

[15] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. 2017. Overcoming Catastrophic Forgetting in Neural Networks. *Proceedings of the National Academy of Sciences* 114, 13 (2017), 3521–3526. doi:10.1073/pnas.1611835114

[16] George Konidaris and Andrew G. Barto. 2009. Skill Discovery in Continuous Reinforcement Learning Domains using Skill Chaining. In *Advances in Neural Information Processing Systems*, Vol. 22.

[17] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (2009), 30–37. doi:10.1109/MC.2009.263

[18] Nan Liu, Shuang Li, Yilun Du, Antonio Torralba, and Joshua B. Tenenbaum. 2022. Compositional Visual Generation with Composable Diffusion Models. *arXiv preprint arXiv:2206.01714* (2022). https://arxiv.org/abs/2206.01714

[19] Scott Niekum, Sarah Osentoski, George D. Konidaris, Sachin Chitta, Bhaskara Marthi, and Andrew G. Barto. 2015. Learning Grounded Finite-State Representations from Unstructured Demonstrations. *The International Journal of Robotics Research* 34, 2 (2015), 131–157. doi:10.1177/0278364914554471

[20] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, et al. 2022. Training Language Models to Follow Instructions with Human Feedback. *arXiv preprint arXiv:2203.02155* (2022).

[21] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy Lillicrap, and Greg Wayne. 2019. Experience replay for continual learning. *Advances in Neural Information Processing Systems* 32 (2019).

[22] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. 2016. Progressive neural networks. In *International Conference on Learning Representations*.

[23] Richard S. Sutton, Doina Precup, and Satinder Singh. 1999. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence* 112, 1–2 (1999), 181–211.

[24] Sebastian Thrun. 1998. Lifelong Learning Algorithms. In *Learning to Learn*, Sebastian Thrun and Lorien Pratt (Eds.). Springer, Boston, MA, 181–209. doi:10.1007/978-1-4615-5529-2_8

[25] Sebastian Thrun and Tom M. Mitchell. 1995. Lifelong Robot Learning. *Robotics and Autonomous Systems* 15, 1–2 (1995), 25–46. doi:10.1016/0921-8890(95)00004-Y

[26] Eliott Verwimp, Marc Masana, and Tinne Tuytelaars. 2023. CLAD: Continual Learning with Adaptive Distillation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 18956–18965.

[27] Andrew Wilson and Alan Fern. 2012. A Bayesian Approach to Preference Learning. In *Advances in Neural Information Processing Systems*. 1133–1141.

[28] Jaehong Yoon, Woonhyuk Jeong, Giwoong Lee, Eunho Yang, and Sung Ju Hwang. 2021. Federated Continual Learning with Weighted Inter-client Transfer. In *Proceedings of the 38th International Conference on Machine Learning*. 12073–12086.