WINFlowNets: Warm-up Integrated Networks Training of Generative Flow Networks for Robotics and Machine Fault Adaptation

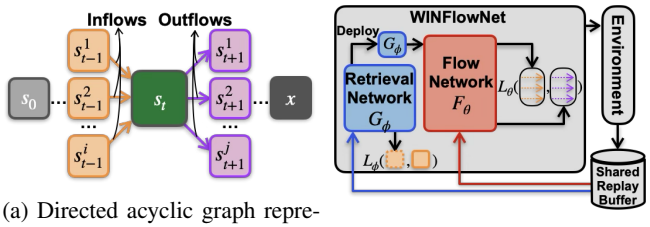
Zahin Sufiyan[1], Shadan Golestan[2], Yoshihiro Mitsuka[3], Shotaro Miwa[4], and Osmar Zaiane[1,2]

***Abstract*— Generative Flow Networks for continuous scenarios (CFlowNets) have shown promise in solving sequential decision-making tasks by learning stochastic policies using a flow and a retrieval network. Despite their demonstrated efficiency compared to state-of-the-art Reinforcement Learning (RL) algorithms, their practical application in robotic control tasks is constrained by the reliance on pre-training the retrieval network. This dependency poses challenges in dynamic robotic environments, where pre-training data may not be readily available or representative of the current environment. This paper introduces WINFlowNets, a novel CFlowNets framework that enables the co-training of flow and retrieval networks. WINFlowNets begins with a warm-up phase for the retrieval network to bootstrap its policy, followed by a shared training architecture and a shared replay buffer for co-training both networks. Experiments in simulated robotic environments demonstrate that WINFlowNets surpasses CFlowNets and state-of-the-art RL algorithms in terms of average reward and training stability. Furthermore, WINFlowNets exhibits strong adaptive capability in fault environments, making it suitable for tasks that demand quick adaptation with limited sample data. These findings highlight WINFlowNets' potential for deployment in dynamic and malfunction-prone robotic systems, where traditional pre-training can be impractical and continual adaptation is required.**

I. Introduction

Generative flow networks (GFlowNets) [1] is a class of generative algorithms designed to learn a stochastic policy for generating (sampling) discrete structured objects (such as graphs, and sequences) $x \in \mathcal{X}$, in the space of objects $\mathcal{X}$ to maximize a reward function $R(x)$. To construct an object x, GFlowNets starts from an initial null state s_0 and generates a trajectory by iteratively taking actions that add simple building blocks, passing through intermediate states until it reaches the final state x with a probability proportional to $R(x)$. The process can be shown using directed acyclic graphs (depicted in Figure 1a), where nodes and edges denote the states and actions, respectively. This problem can be framed as a sequential decision-making problem, making GFlowNets analogous to Reinforcement Learning (RL) [2]. GFlowNets learns a distribution over trajectories, enabling efficient exploration and reducing the risk of local optima.

GFlowNets has demonstrated potential in solving sequential decision-making problems [3], [4], [5], [6] and has been compared with RL algorithms across various problem settings, such as combinatorial optimization problems [7], [8], HyperGrid modeling [1], [9], and operation scheduling [10]. In several real-world domains, where the space of objects $\mathcal{X}$ is typically continuous, a variation of GFlowNet, known as Continuous Flow Networks (CFlowNets) [11] has been proposed. CFlowNets generates a distribution over all possible trajectories, and samples from it in proportion to the trajectories' performance. As shown in various studies [11], [1], [12], GFlowNet/CFlowNets can be more sample-efficient than certain state-of-the-art RL algorithms such as Proximal Policy Optimization (PPO) and Soft Actor Critic (SAC).



(a) Directed acyclic graph representation of the decision-making process in GFlowNet.

(b) An overview of our proposed decision-making framework.

Fig. 1: Main concepts of WINFlowNets.

Nevertheless, significant challenges remain in the training of CFlowNets [13]. While CFlowNets is effective for continuous spaces [14], its application can be impractical particularly in dynamic robotic control tasks [15], [12] such as object manipulation, where evolving environments require continual adaptation and where the space of objects $\mathcal{X}$ is exponentially large or composed of long trajectories [1], [16], making repeated re-training computationally expensive and inefficient. To generate an object x, CFlowNets operates using two networks, i.e., a flow network F_θ, parametrized by θ, and a retrieval network G_ϕ, parametrized by ϕ. F_θ computes the flow, i.e., the probabilistic transitions between consecutive states conditioned on actions (inflows and outflows in Figure 1a). This computation relies heavily on G_ϕ, which estimates the possible parent states of a given state. In the original CFlowNets training architecture, G_ϕ is pre-trained prior to training F_θ, assuming a fixed data distribution. However, this assumption limits the practical applicability of CFlowNets in continuous robotic control tasks.

In robotic control tasks, environments often change due to evolving conditions and unexpected factors—particularly hardware faults and malfunctions [17]—resulting in out-of-distribution (OOD) scenarios. Without adaptation, robots are prone to errors in OOD situations, as they encounter previously unseen states more frequently. Such errors can slow down the learning process or gradually degrade the overall

*This work was not supported by any organization

[1]Computing Science Department, University of Alberta zsufiyan@ualberta.ca

[2]Alberta Machine Intelligence Institute (Amii)

[3]Information Technology R&D Center, Mitsubishi Electric

[4]Advanced Technology R&D Center, Mitsubishi Electric

performance of CFlowNets. This is especially problematic because G_ϕ requires a separate pre-training dataset, which may not be readily available in novel or dynamic environments. If G_ϕ is pre-trained on data that does not fully represent the space of objects $\mathcal{X}$, it can lead to both incorrect inflow and outflow estimations. Moreover, pre-training G_ϕ introduces an extra step into the workflow, increasing resource requirements such as time, memory, and computational power. Integrating the training of G_ϕ with F_θ is essential for enabling G_ϕ to dynamically adapt to OOD situations. Our target setting is not arbitrary real-world robotics, but continuous control problems in which the environment may change after deployment and where collecting a representative pre-training dataset for G_ϕ is difficult. In such settings, repeatedly pre-training or retraining the retrieval network from scratch can become computationally inefficient, especially when the object space $\mathcal{X}$ consists of long trajectories. WINFlowNets is designed for this regime: it trades some early-stage sample efficiency for the ability to update the retrieval and flow networks continually from newly collected interactions.

This paper proposes WINFlowNets (Figure 1b), a novel CFlowNets framework that enables simultaneous training of both G_ϕ and F_θ networks within a unified training loop. To improve the training efficiency, both networks utilize a shared replay buffer. WINFlowNets begins with a Warm-Up phase, wherein G_ϕ independently interacts with the environment to gather initial experience and refine its policy. The Warm-Up phase is followed by a Dual-Training phase, where both G_ϕ and F_θ actively interact with the environment, collect experience, and contribute to the shared replay buffer to support collaborative learning. Experiments in simulated robotic environments demonstrate that WINFlowNets improves final policy performance and adaptation under OOD situations, while revealing a trade-off with early-stage sample efficiency. We also find that while WINFlowNets surpasses state-of-the-art RL algorithms (PPO, SAC) in performance, its training efficiency remains comparatively lower. Our contributions are: 1) We propose WINFlowNets, a novel training framework tailored for dynamic, continuous robotic control tasks. 2) We introduce and validate a two-phase training strategy—comprising a Warm-Up and Dual-Training phase—that enables efficient adaptation without requiring a separately pre-trained retrieval network G_ϕ. 3) We empirically show that WINFlowNets outperforms standard CFlowNets and state-of-the-art RL algorithms in both standard and OOD (faulty) environments.

II. Related Work

Reinforcement Learning (RL) for adaptation in robotic tasks. There is a rich body of literature proposing various RL [18], [19] and Meta-RL [20], [21], [22]. algorithms for adapting to OOD situations in robotic tasks. In a previous research, Meta-reinforcement learning (Meta-RL) frameworks have been developed to enable adaptation to unseen tasks using minimal data from task distributions. For instance, Neghabadi et al. [22] proposed a model-based Meta-RL approach for fault adaptation tasks, utilizing two adaptive learners: a gradient-based adaptive learner (GrBAL) and a recurrence-based adaptive learner (ReBAL). GrBAL, based on model-agnostic meta-learning (MAML), uses neural networks to learn generalized dynamics, while ReBAL employs recurrent neural networks for task adaptation. These methods demonstrated significant sample efficiency, outperforming baselines like TRPO and MAML-RL, which required 1000 times more data. Comparative analyses on continuous control tasks, such as Ant and HalfCheetah in OpenAI Gym, and real-life robots, showed faster adaptation to dynamic environments, underscoring the effectiveness of Meta-RL for fault adaptation. Unlike many Meta-RL approaches, which typically assume access to a distribution of related training tasks or fault conditions, our setting focuses on unexpected OOD changes where the fault type may not be known in advance. WINFlowNets therefore does not rely on meta-training over a predefined family of faults; instead, it adapts online by updating both the retrieval and flow networks using newly collected experience.

GFlowNets or CFlowNets for adaptation in robotic tasks. Few works have proposed flow networks for robotic control tasks [15], [23], [11]. The closest work to ours is by Li et al. [11], where they applied CFlowNets to continuous control tasks like Point-Robot-Sparse, Reacher-Goal-Sparse, and Swimmer-Sparse, comparing them to RL algorithms such as PPO and SAC. In Point-Robot-Sparse and Reacher-Goal-Sparse, CFlowNets achieved higher average returns in fewer timesteps due to their superior exploration capabilities and ability to fit reward distributions. However, in Swimmer-Sparse, performance declined due to its steep reward gradient, which limited the exploration. The study also highlighted that CFlowNets generated thousands of valid and distinct trajectories, surpassing RL baselines, which struggled with exploration. RL algorithms like SAC and TD3 either generated fewer trajectories or saw performance drop as training progressed. These results underscore the strength of CFlowNets in solving sparse-reward tasks requiring diverse exploration but reveal limitations in tasks with dense rewards. The original CFlowNets framework also allows the retrieval network to be updated during training; therefore, the novelty of WINFlowNets is not online updating of G_ϕ in isolation, but the integration of an online Warm-Up phase, Dual-Training phase, and shared replay buffer to reduce dependence on a separately prepared pre-training dataset.

Moreover, Sufiyan et al. [12] investigated the application of CFlowNets for robotic fault adaptation. Existing approaches often rely on RL, which can suffer from sample inefficiency and can struggle in OOD situations. The study introduced CFlowNets in the Reacher-v2 environment with four fault scenarios mimicking real-world robotic malfunctions. The authors compared CFlowNets with RL baselines, i.e., PPO, SAC, DDPG, and TD3 highlighting their superior adaptation speed and asymptotic performance in most fault scenarios. We consider CFlowNets as a baseline in our study.

Training Frameworks for Generative Flow Networks. Several works have identified challenges in training GFlowNets and proposed suitable frameworks. Pan et al.[24] introduced

local credit assignment, which assigns partial rewards along trajectories and enables learning from incomplete trajectories. Malkin et al.[25] proposed a global credit assignment approach that sums flow constraints over entire trajectories, improving robustness to long action sequences and large action spaces. Shen et al. [13] improved GFlowNet training in limited-data settings using efficient evaluation sampling and relative edge flow parameterization. However, training GFlowNets/CFlowNets in dynamic robotic environments, where OOD situations such as faults or environmental changes can disrupt learning, remains largely unexplored.

Novelty of this work. We cast continuous robotic control tasks with OOD situations as a generative flow networks problem. We then introduce a novel training architecture for CFlowNets, called WINFlowNets, and compare the resulting adaptation performance with those found by CFlowNets, PPO, SAC, and DDPG. We show that the combination of Warm-Up phase, Dual-Training phase, and a shared replay buffer for both G_ϕ and F_θ leads to improved performance and more effective adaptation in dynamic environments.

III. Method

A. Problem Setup

Continuous robotic tasks are typically framed as sequential decision-making problems, where the environment is modeled as a Markov Decision Process (MDP), denoted as $\mathcal{M}{=}(\mathcal{S}, \mathcal{A}, T, R, \gamma)$, where $\mathcal{S}$ is the state space; $\mathcal{A}$ is the action space; T is the environment's transition dynamics; R is the reward function; and $\gamma{\in}[0,1]$ is the discount factor. At each time $t{=}\{0,1,2,...,h\}$, with h being the horizon, the agent interacts with the MDP via the allowed actions $\mathcal{A}$, and the state of the environment changes from $s_t{\in}\mathcal{S}$ to $s_{t+1}{\in}\mathcal{S}$ based on a transition dynamics $T(s_{t+1}|s_t, a_t)$, where $a_t{\in}\mathcal{A}$ is the action taken at time t. Finally, the agent receives a feedback from the reward function $R(s_{t+1})$. The objective of the agent is to find a policy π to maximize an expected discounted sum of rewards $J(\pi){=}\mathbb{E}_{\tau\sim\pi}[\sum_{t=0}^{h}\gamma^t R(s_t)]$ via a trial and error process, expressed as a trajectory $\tau{=}(s_0{\rightarrow}s_1{\rightarrow}...{\rightarrow}x)$. The agent's objective is to learn the optimal policy $\pi^*{=}\operatorname{argmax}_\pi J(\pi)$. We use $J(\pi)$ to compare the policies identified by our proposed method against those from baseline approaches.

B. CFlowNets for Continuous Robotic Tasks

The CFlowNets framework consists of three main components: action selection procedure, flow matching approximation, and continuous flow matching loss function. In *action selection procedure*, the agent interacts with the environment by iteratively sampling actions from a probability distribution $a_t{\sim}\pi(a_t|s_t)$. Due to the continuous nature of the action space in continuous robotic tasks, the agent uniformly samples M actions from the policy π and generates an action probability buffer using the flow network F_θ, defined as $\mathcal{P}{=}[F_\theta(s_t, a_i)]_{i=1}^{M}$. Actions with higher $F_\theta(s_t, a_i)$ values are sampled with higher probability. The process repeats until a set of complete trajectories is constructed and stored in a replay buffer.

The purpose of *flow matching approximation* is to satisfy, for any state s_t, inflows ($f^+(s_t)$) and outflows ($f^-(s_t)$) are equal. This condition can be expressed as:

$$f^+(s_t) = f^-(s_t)$$
$$\int_{a:T(s,a)=s_t} F_\theta(s,a)\,da = \int_{a\in\mathcal{A}} F_\theta(s_t,a)\,da \qquad (1)$$

These integrals are approximated by sampling K actions from the action space $\mathcal{A}$. To approximate inflows, the retrieval network G_ϕ is used, and then we calculate $\hat{f}^+{=}\frac{\mu(\mathcal{A})}{K}\sum_{k=1}^{K}F_\theta(G_\phi(s_t,a_k),a_k)$. To approximate outflows, we calculate $\hat{f}^-{=}\frac{\mu(\mathcal{A})}{K}\sum_{k=1}^{K}F_\theta(s_t,a_k)$, where $\mu(A)$ indicates the measure of the continuous action space $\mathcal{A}$. Finally, *continuous flow matching loss function* is:

$$\mathcal{L}_\theta = \sum_{s_t=s_0}^{x}[\hat{f}^+ - \hat{f}^-]$$
$$= \sum_{s_t=s_0}^{x}\left[\sum_{k=1}^{K}F_\theta(G_\phi(s_t,a_k),a_k) - \lambda\tilde{R}(s_t) - \sum_{k=1}^{K}F_\theta(s_t,a_k)\right]^2$$
$$= \sum_{s_t=s_0}^{x}\left[\log\left(\epsilon + \sum_{k=1}^{K}\exp(F_{log\theta}(G_{log\phi}(s_t,a_k),a_k))\right)\right.$$
$$\left. - \log\left(\epsilon + \lambda\tilde{R}(s_t) + \sum_{k=1}^{K}\exp(F_{log\theta}(s_t,a_k))\right)\right]^2 \qquad (2)$$

where λ is the scaling factor used to scale the summation of the sampled flows appropriately, considering the size of the continuous action space $\mathcal{A}$, and $\tilde{R}(s_t)$ denotes the non-negative reward measure used in the flow-matching objective. In our experiments, $\tilde{R}$ is obtained by applying an exponential transformation to the raw environment reward, while the untransformed raw reward is used for reporting average reward.

C. WINFlowNets Architecture

We now introduce our proposed WINFlowNets architecture. The training mechanism of WINFlowNets removes the requirement for a separately pre-trained retrieval network G_ϕ by replacing external pre-training with an online Warm-Up phase using interactions collected directly from the environment. Figure 2 provides an overview of the WINFlowNets architecture, including the training of the retrieval network G_ϕ (module 1 in Figure 2), and the training of both the retrieval network G_ϕ and the flow network F_θ (module 2 in Figure 2), and its interaction with the environment at a given time t. During the environment interaction, WINFlowNets samples an action a_t from an action probability distribution obtained by the forward propagation of WINFlowNets. Afterwards, WINFlowNets interacts with the environment by taking action a_t, which in turn updates the current state from s_t to s_{t+1}, and yields a reward of $R(s_{t+1})$. After advancing to the next timestep ($t = t{+}1$), we store a tuple of $<s_{t-1}, a_{t-1}, R(s_t), s_t>$ in a shared replay buffer, denoted as $\mathcal{B}$.

We modify the standard separated training loops of the G_ϕ and F_θ into a unified loop structure. In WINFlowNets, both

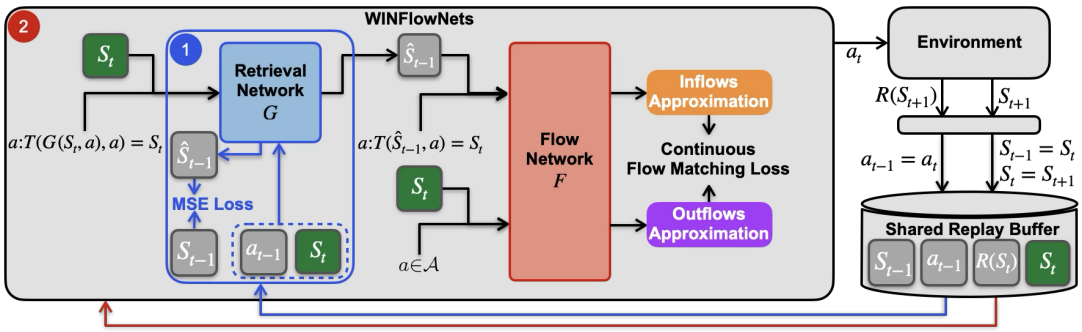


Fig. 2: The architecture of WINFlowNets.

networks access the shared replay buffer $\mathcal{B}$, which contains state-action sequences that both networks encountered while interacting with the environment during the training process. The shared replay buffer $\mathcal{B}$ facilitates the training of both the retrieval network G_ϕ and the flow network F_θ to estimate inflows and outflows of every state, respectively.

Module 1: Warm-Up Phase of the Retrieval Network $\mathbf{G}_\phi$. The Warm-Up phase is the foundational stage in the WINFlowNets for training G_ϕ. The retrieval network G_ϕ is responsible for estimating the predecessor states (s_{t-1}) given the current state s_t and action a_{t-1}. To train G_ϕ, WINFlowNets introduces a dedicated *Warm-Up* phase during which training of G_ϕ and F_θ remain active and inactive, respectively. As described in Algorithm 1, the agent begins each episode from an initial state s_0, and repeatedly (until reaching to a terminal state x or time horizon T) samples a set of M actions uniformly from the action space $\mathcal{A}$. An action a_t is randomly selected from this set and executed in the environment, yielding the next state s_{t+1} and its corresponding reward $R(s_{t+1})$. After advancing to the next timestep, The resulting tuple is stored in $\mathcal{B}$.

After each interaction episode, G_ϕ is trained using a batch of N tuples sampled from $\mathcal{B}$. For each tuple i, $i \in \{1, ..., N\}$, we compute the predicted predecessor state as $\hat{s}^i_{t-1}{=}G_\phi(s^i_t, a^i_{t-1})$, and the prediction is compared to the ground-truth s^i_{t-1} using Mean Squared Error (MSE) loss:

$$\mathcal{L}_\phi = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{s}^i_{t-1} - s^i_{t-1}\right)^2 \tag{3}$$

where $\hat{s}^i_{t-1} - s^i_{t-1}$ is the element-wise error across the robot's state vector (e.g., joint angles, velocities and fingertip-to-target position). Standard gradient descent with the Adam optimizer is utilized to minimize Equation 3. Note that the shared replay buffer $\mathcal{B}$ continuously incorporates new data during the training process, ensuring that G_ϕ can adapt to the dynamics of the environment.

Note that during the Warm-Up phase the learning rate η begins at a low value and is gradually increased to accelerate learning as training progresses. This is because early in this phase, G_ϕ operates with limited predictive accuracy and high uncertainty. This conservative strategy ensures more stable updates on noisy gradient signals, while gradually increasing it enables faster convergence as the agent gathers more diverse and representative data from the environment. We also define

Algorithm 1 WINFlowNets (Warm-Up phase)

Input: F_θ, G_ϕ, β
Initialize: Warm-up duration $\omega = 100k$, dynamic learning rate η
for $t = 1$ **to** ω **do**
  Set $s_t = s_0$
  while $s_t \neq x$ **and** $t < T$ **do**
    Uniformly sample M actions $\{a_i\}_{i=1}^{M}$ from $\mathcal{A}$
    Select $a_t \sim \mathcal{U}(\{a_i\}_{i=1}^{M})$
    Take a_t and observe $(s_{t+1}, R(s_{t+1}))$
    $t \leftarrow t + 1$
    Store $(s_{t-1}, a_{t-1}, R(s_t), s_t)$ in β
  end while
  Train G_ϕ using $\mathcal{L}_\phi = \frac{1}{N}\sum_{i=1}^{N}(\hat{s}^i_{t-1} - s^i_{t-1})^2$
  $\eta = \eta + \epsilon$ ▷ Increase η to accelerate learning
end for

w as the number of Warm-Up steps, and set w=100k based on preliminary experiments, which gave G_ϕ enough time to learn reliable predictions without excessive overhead.

Module 2: Dual Training of the Flow Network $\mathbf{F}_\theta$ and the Retrieval Network $\mathbf{G}_\phi$. The flow network F_θ serves to approximate the inflows and outflows distributions associated with a given state s_t, a key step in maintaining the flow-matching condition expressed in Equation 1. After the Warm-Up phase, during the Dual-Training phase, both F_θ and G_ϕ are jointly optimized using data collected from agent-environment interactions. As depicted in Algorithm 2, each episode begins at an initial state s_0 and continues until reaching a terminal state x or a time horizon T. At each timestep, the agent samples a set of candidate actions uniformly from the action space $\mathcal{A}$, and uses F_θ to construct the action probability buffer $\mathcal{P}$. An action a_t (selected based on $\mathcal{P}$), is executed in the environment, and the resulting tuple is stored in $\mathcal{B}$.

After each episode, a minibatch of transitions is sampled from $\mathcal{B}$. For each state s_t in the batch, a set of parent state is predicted using G_ϕ, and these are used to compute the inflow approximation $(\hat{f}^+(s_t))$. The outflow approximation $(\hat{f}^-(s_t))$ is computed directly using F_θ and the current state's candidate actions. These inflow and outflow estimates are used to update F_θ via Equation 2, while both the ground-truth and estimated parent states are used to update G_ϕ through Equation 3. The Dual-Training phase allows the networks to reinforce each other's learning, improving overall stability and performance in dynamic, OOD-prone environments.

Algorithm 2 WINFlowNets (Dual-Training phase)

while not converged **do**:
  Set $s_t = s_0$
  while $s_t \neq x$ **and** $t < T$ **do**
    Uniformly sample M actions $\{a_i\}_{i=1}^{M}$ from $\mathcal{A}$
    Compute action probability buffer $\mathcal{P}$ via $\{F_\theta(s_t, a_i)\}_{i=1}^{M}$
    Sample action $a_t \sim \mathcal{P}$
    Take a_t and observe $(s_{t+1}, R(s_{t+1}))$
    $t \leftarrow t + 1$
    Store $(s_{t-1}, a_{t-1}, R(s_t), s_t)$ in β
  end while
  Sample minibatch B from β
  for each state s in B **do**
    Uniformly sample K actions $\{a_i\}_{i=1}^{K}$ from $\mathcal{A}$
    Compute parent states $\{G_\phi(s, a_k)\}_{i=1}^{K}$
    Compute inflows:
    $\hat{f}^+ = \log\left(\epsilon + \sum_{i=1}^{K} \exp\left(F_{\log\theta}(G_\phi(s_t, a_i), a_i)\right)\right)$
    Compute outflows:
    $\hat{f}^- = \log\left(\epsilon + \lambda R(s_t) + \sum_{k=1}^{K} \exp\left(F_{\log\theta}(s_t, a_k)\right)\right)$
  end for
  Update F_θ via $\mathcal{L}_\theta = \sum_{s_t=s_0}^{x} [f^+ - f^-]$
  Update G_ϕ via $\mathcal{L}_\phi = \frac{1}{N}\sum_{i=1}^{N} (\hat{s}_{t-1}^i - s_{t-1}^i)^2$
end while

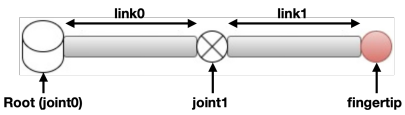


Fig. 3: Visualization of the Reacher-v2 robotic arm.

IV. Experiments

A. Robot Control Setup

Similar to several related works [11], [26], [27], we opted for Reacher-V2[1], a robotic simulation environment featuring a two-joint robotic arm with two degrees of freedom (Figure 3). Each state in this environment is represented as a vector comprising joint angles, joint velocities, and the position of a target. The target is a spatial point the robot must reach. The two joints, named `joint0` and `joint1`, allow for a wide range of motion. The first anchors the first segment of the arm (`link0`) to a fixed base (root), and the latter connects the second segment (`link1`) to `link0`. The robotic arm uses actuators to control its joints by applying torque. Thus, the action space $\mathcal{A}$ is continuous and represented as a 2-dimensional vector of type `float32`, with each element ranging from -1.0 to 1.0.

The state space includes the angles of the joints, the position of the end effector (fingertip), and the relative position to the target. We denote the state space $\mathcal{S}$ as `Box`$(-\infty, \infty, (11,),$ `float64`$)$, representing an 11-dimensional continuous space where each element is a real-valued 64-bit floating-point number.

The Reacher-v2 environment returns a raw reward, which we denote as $r(s, a)$ to distinguish it from the non-negative reward measure used in the flow objective. The raw reward takes as input a state $s \in \mathcal{S}$ and an action $a \in \mathcal{A}$, and comprises two components: a distance penalty term, i.e., $(-|p_{\text{fingertip}} - p_{\text{target}}|)$, where $p_{\text{fingertip}}$ and p_{target} are the positions of the fingertip and the target, respectively, and a control penalty term, i.e., $(-\alpha \cdot |a|^2)$ (with α ranging from 0 to 1), which discourages excessive action magnitudes. Consequently, the raw environment reward is given by:

$$r(s, a) = -|p_{\text{fingertip}} - p_{\text{target}}| - \alpha \cdot |a|^2. \tag{4}$$

Since the flow-matching objective requires a non-negative reward measure, we use a transformed reward for training CFlowNets and WINFlowNets, while reporting the original raw environment reward $r(s, a)$ for evaluation. Specifically, the reward used in the flow-matching objective is

$$\tilde{R}(s, a) = \exp(r(s, a)). \tag{5}$$

This transformation maps the raw negative rewards to positive values while preserving their ordering, so trajectories with higher raw environment reward are assigned larger reward values in the flow objective.

[1] `https://gymnasium.farama.org/environments/mujoco/reacher/`

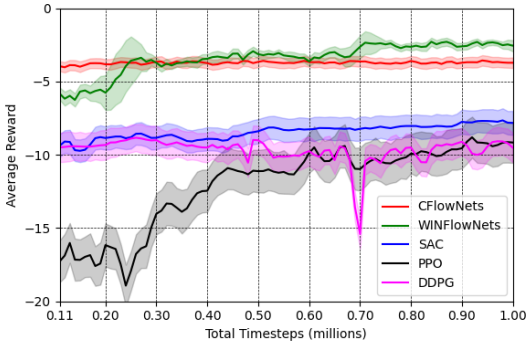


Fig. 4: Average reward of CFlowNets, WINFlowNets, and RL algorithms in the normal Reacher-v2 environment.

B. Normal Environment Case Study

To establish a performance baseline, we evaluate WINFlowNets and all baselines in the standard Reacher-V2 environment, where no faults are introduced. This setting serves as a controlled baseline to evaluate the agent's performance in a stable and fully functional scenario. We compare the performance of WINFlowNets with CFlowNets and three standard RL algorithms (PPO, SAC, and DDPG), typically used as baselines for continuous robotic tasks.

Baseline implementation details. All methods were trained for one million timesteps and evaluated under the same protocol. PPO, SAC, and DDPG used standard continuous-control implementations with fixed hyperparameters across the normal and faulty environments, and were evaluated using the original Reacher-v2 reward without additional reward shaping.

C. Faulty Environment Case Study

We evaluate the adaptability of WINFlowNets and all baselines under OOD situations, by introducing two common types of robotic faults: *Actuator Damage* (AD), and reduced *Range Of Motion* (ROM). AD can arise from overheating, gear failure, or electrical issues [28], [29], which leads to weaker or unstable movements. To simulate this, we

TABLE I: Final performance ($\pm$ std.) and sample efficiency (M timesteps) in Reacher-v2. Bold = best.

Model	Final Perf.	Sample Eff.
SAC	-7.89 ± 0.16	0.67
PPO	-9.50 ± 0.37	3.39
DDPG	-9.55 ± 0.44	5.20
CFlowNets	-3.70 ± 0.05	**0.10**
WINFlowNets	$\mathbf{-2.39\pm0.17}$	0.72

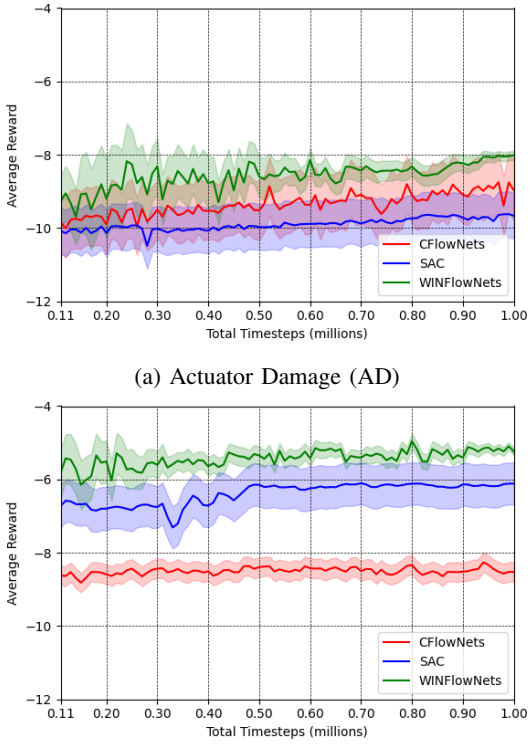
reduce the actuator power (torque output) to one-fourth of its original value. This constraint affects the robot's ability to perform precise or forceful movements, resulting in degraded control. ROM can arise from gear tear and wear, mechanical restrictions, or malfunctioning software [30], [31]. In Reacher-V2, we simulate this fault by constraining the angular range of *joint1* from its original $[-3.0, 3.0]$ radians to a narrower range of $[-1.5, 1.5]$. This fault limits the robot's precision, flexibility and overall maneuverability. In this case study, we first train the models in the normal environment. The learned model parameters, along with the replay buffer, are then transferred to resume training under the new fault condition.

D. Results

We collect empirical data in a span of 1 million timesteps. Following related works (e.g., [32], [12]) we evaluate the performance of the models in terms of average reward, average final performance, and sample efficiency which are commonly used to assess the performance of agents. We calculate *average reward* ($\pm$ standard deviation) over 10 episodes, showing the learning behavior of the models during 1 million timesteps. To obtain *average final performance*, we calculate both average and standard deviation of the average reward over the last 20 evaluations of 1 million timesteps. This metric reflects the models behavior as training progresses towards the final steps, providing insight into the quality of the final policy learned. We define *sample efficiency* as the timesteps required for each model to achieve their asymptotic performance, i.e., the average reward a model achieves once training has stabilized, where stabilization is identified by monitoring the variance of rewards across recent evaluations and observing no significant fluctuations[2].

Normal environment. Figure 4 illustrates the average reward of WINFlowNets and our baselines across the normal environment with shaded regions representing the 95% confidence interval to indicate performance variability. Note that the first evaluation begins at 110k timesteps, following the 100k Warm-Up phase of WINFlowNets. According to Figure 4, we make the following observations: N1) WINFlowNets outperforms CFlowNets and the RL baselines by a significant margin in the normal environment; N2) WINFlowNets, which does not rely on pre-training for G_ϕ, shows a slower initial reward increase compared to CFlowNets. However, it exhibits steady improvement and surpasses CFlowNets after approximately 240k timesteps, ultimately achieving higher average reward by the end of training; N3) CFlowNets exhibits a rapid rise in average reward, with the narrow-shaded region highlighting the reliability and robustness provided by the pre-training of G_ϕ; N4) Among RL baselines, SAC follows a more stable learning curve, and outperforms other RL models; N5) DDPG and PPO show unstable learning curves, with PPO in particular demonstrating an early performance dip and requiring more timesteps (greater than 1 million) to stabilize.

[2]In our experiments, models reach stabilization within 5 million timesteps



(a) Actuator Damage (AD)

(b) Reduced Range of Motion (ROM)

Fig. 5: Average reward of CFlowNets, WINFlowNets, baselines in the Reacher-v2 environment under faulty conditions.

To investigate these observations further, we measure the average final performance and sample efficiency of the models in Table I. It can be seen that WINFlowNets shows significantly better final performance, as confirmed by a two-tailed t-test ($p<0.05$). However, it requires more samples than CFlowNets and SAC to converge to its asymptotic performance. This is primarily because in WINFlowNets, both G_ϕ and F_θ co-adapt using a shared replay buffer, allowing them to continuously learn from the most recent interactions, and drive ongoing policy improvements. For instance, note the two distinct surges in average reward around 0.2 and 0.8 million timesteps in Figure 4, which highlight this behavior.

Faulty environments. For faulty environments, we choose SAC as our RL baseline, as it demonstrates the most stable performance among the RL models. In contrast, based on observation N5, PPO and DDPG are less suitable for comparison in faulty environments. Figure 5 illustrates the average reward of WINFlowNets and our baselines across AD and reduced ROM environments. Our observations in faulty environments are: F1) WINFlowNets outperforms CFlowNets and SAC; F2) In both faulty environments, WINFlowNets shows unsteady performance during the initial 400k timesteps due to its more extensive exploration enabled by the Warm-Up

TABLE II: Average final performance ($\pm$ std.) and sample efficiency (M timesteps) of WINFlowNets, CFlowNets, and SAC in two faulty environments. Bold indicates the best result in each setting.

Model	AD		ROM	
	Final Perf.	Sample Eff.	Final Perf.	Sample Eff.
SAC	-9.69 ± 0.19	**0.11**	-6.16 ± 0.03	0.37
CFlowNets	-9.01 ± 0.17	0.32	-8.50 ± 0.08	**0.11**
WINFlowNets	$\mathbf{-8.25\pm0.19}$	0.24	$\mathbf{-5.25\pm0.12}$	0.12

and Dual-Training phases (which are absent in CFlowNets), allowing G_ϕ and F_θ to continually adapt with new OOD data and ultimately find policies with higher average reward; F3) WINFlowNets initially shows high variance in performance, comparable to SAC and CFlowNets, but this variance progressively diminishes as training advances, indicating increased stability in learning.

We present the models' final performance and sample efficiency in Table II. As shown, SAC and CFlowNets are the most sample-efficient models in AD and reduced ROM environments, respectively. This complements observations F2 and F3 by highlighting the trade-off in WINFlowNets: although it requires more samples, it ultimately achieves better performance via continual adaptation in OOD situations. We believe this gain comes from the Warm-Up and Dual-Training phases, together with the shared replay buffer.

V. Discussion

We have found that WINFlowNets and CFlowNets outperform standard RL baselines (SAC, PPO, and DDPG) in average reward and sample efficiency, leading to better policies for continuous robotic control. These results suggest that the combination of the Warm-Up phase, Dual-Training phase, and shared replay buffer can improve policy performance in continuous robotic control tasks.

We argue that WINFlowNets removes the need for a separately pre-trained G_ϕ by using an online Warm-Up phase with data collected directly from the current environment. After this phase, WINFlowNets learns to adapt to the dynamic nature of tasks, such as OOD situations, by jointly training G_ϕ and F_θ, enabling continual policy improvements based on recent shared interactions with the environment. It is worth mentioning that WINFlowNets requires a substantial number of timesteps to match the performance of CFlowNets. This is because inflows and outflows approximations are not accurate, as G_ϕ requires training on more experiences. Although the initial training period showed a wider confidence interval region, it quickly stabilized as more improved versions of the retrieval network were deployed as the training continued. Note that the wider shaded region at the beginning of the post-Warm-Up phase reflects the adjustment period for the first few retrieval network instances. Thus, in applications that safety of the policies are critical [33], such as human-robot interaction scenarios [34],WINFlowNets may not be feasible, as the initial performance of the model could pose risks.

To show this, we calculate in Figure 6 the 95% confidence interval width ($CI = 2 \times \frac{STD}{\sqrt{n}}$, where $n = 10$ is the number

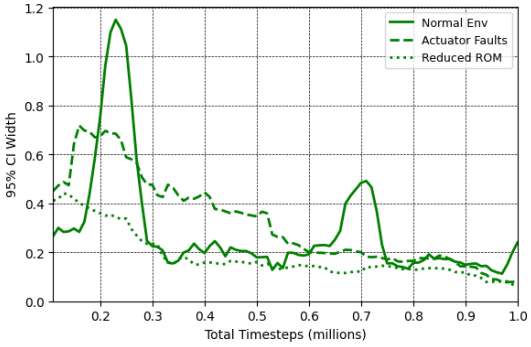


Fig. 6: Confidence interval width of WINFlowNets average reward over time in three environments.

of evaluation rollouts per point), of WINFlowNets average reward across training timesteps in all environments, providing insights into how its performance uncertainty evolves over time. According to the figure, the variance in performance is high initially across all environments, especially in the normal environment. However, the uncertainty decreases as training progresses, indicating stabilization and improved reliability of the learned policies. Note that although the initial variance is high, it remains comparable to those observed in our baselines.

The rapid improvement of CFlowNets is closely tied to their strong ability to learn effectively with fewer samples. This feature becomes apparent when contrasted with RL algorithms like PPO, which tend to struggle during early learning phases. making them better suited for more deterministic environments, where pre-training G_ϕ is feasible.

CFlowNets and WINFlowNets consistently outperformed RL algorithms in both reward trends and sample efficiency on all of the environments, except reduced ROM, where SAC outperformed CFlowNets. A key reason for this superior performance is both WINFlowNets and CFlowNets' ability to generate a distribution over trajectories, sampling paths with higher rewards more frequently. This distribution-based exploration reduces the risk of getting stuck in local optima, enabling WINFlowNets and CFlowNets to converge toward globally optimal solutions. Also, CFlowNets' off-policy nature allows for efficient reuse of stored experiences, enabling faster convergence to asymptotic performance with minimal variance across runs.

In terms of RL algorithms, PPO's weaker performance may stem from its on-policy nature, which relies on potentially suboptimal initial experiences. Additionally, PPO's clipping mechanism, while improving stability, can limit policy updates and slow learning. Although SAC achieved lower average reward than WINFlowNets, it showed greater stability and better sample efficiency than PPO and DDPG, likely due to its entropy-regularized exploration strategy. Lastly, DDPG's sensitivity to hyperparameters and overestimation bias in its Q-value estimates can lead to poor and inefficient learning.

Limitations, future work, and potential applications. A limitation of this study is that WINFlowNets is evaluated as a complete training framework, combining the Warm-Up phase, online updating of the retrieval network, and the shared replay buffer. Therefore, the current experiments do

not isolate the individual contribution of each component. Future work should include ablation studies comparing variants such as CFlowNets with the retrieval-network fine-tuning option enabled, CFlowNets trained using the same Warm-Up interactions, WINFlowNets without the Warm-Up phase, WINFlowNets without the shared replay buffer, and different Warm-Up durations. Such experiments would help clarify which components contribute most strongly to final performance, sample efficiency, and adaptation under fault conditions. These ablations should also quantify the amount of data required for CFlowNets pre-training relative to the $100k$ interactions used in the WINFlowNets Warm-Up phase.

We wish to emphasize that WINFlowNets demands significant computational overhead. The shared replay buffer and Dual-Training phase increase GPU memory usage. As a result, WINFlowNets is more resource-intensive than CFlowNets, which was already noted to be costly in [12]. Consequently, WINFlowNets is less suitable for highly resource-constrained systems. In contrast, certain RL models may be more practical for edge-device deployment [35], due to their lower computational footprint at inference time.

Another limitation is that our evaluation is limited to simulated Reacher-v2 variants; although this setup provides controlled and repeatable fault conditions, future work should validate WINFlowNets on distinct locomotion and manipulation tasks, such as Hopper, HalfCheetah, Ant, and FetchPickAndPlace, as well as on physical robotic hardware to assess the sim-to-real gap.

While WINFlowNets illustrates strong performance, its sample efficiency can be improved. This limitation is due to its Dual-Training phase, which requires sufficient data for both networks. One potential enhancement is to incorporate ideas from prioritized shared replay buffer, where experiences are sampled more frequently with higher learning potential, such as sampling those visited experiences proportional to their absolute Temporal Difference (TD) errors. Prioritizing the shared replay buffer can also shorten the duration of the Warm-Up phase.

VI. Conclusion

This paper introduced WINFlowNets, a novel framework for training CFlowNets in continuous control tasks. It integrates the training of the flow and retrieval networks via a shared replay buffer (Dual-Training phase) and a Warm-Up phase for the retrieval network. We hypothesized that WINFlowNets could improve exploration of the action space and help identify policies with higher average reward. To test this, we evaluated WINFlowNets in three simulated Reacher-v2 environments: one standard setting and two faulty conditions representing out-of-distribution scenarios. We compared WINFlowNets with CFlowNets and standard RL algorithms commonly used for continuous control tasks. Our results show that WINFlowNets outperforms CFlowNets and RL baselines (SAC, PPO, DDPG) in average reward, while also yielding policies with progressively lower uncertainty over time. However, like CFlowNets, WINFlowNets remains more resource-intensive than RL algorithms, highlighting the need for further optimization.

References

[1] E. Bengio, M. Jain, M. Korablyov, D. Precup, and Y. Bengio, "Flow network based generative models for non-iterative diverse candidate generation," *Advances in Neural Information Processing Systems*, vol. 34, pp. 27 381–27 394, 2021.

[2] P. Niu, S. Wu, M. Fan, and X. Qian, "Gflownet training by policy gradients," in *Proceedings of the 41st International Conference on Machine Learning*, ser. ICML'24. JMLR.org, 2024.

[3] T. Deleu, M. Nishikawa-Toomey, J. Subramanian, N. Malkin, L. Charlin, and Y. Bengio, "Joint bayesian inference of graphical structure and parameters with a single generative flow network," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 31 204–31 231. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/639a9a172c044fbb64175b5fad42e9a5-Paper-Conference.pdf

[4] T. da Silva, D. A. de Souza, and D. Mesquita, "Streaming bayes gflownets," in *Advances in Neural Information Processing Systems*, A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, Eds., vol. 37. Curran Associates, Inc., 2024, pp. 27 153–27 177. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2024/file/2fb57276bfbaf1b832d7bfcba36bb41c-Paper-Conference.pdf

[5] L. Pan, D. Zhang, M. Jain, L. Huang, and Y. Bengio, "Stochastic generative flow networks," in *Uncertainty in Artificial Intelligence*. PMLR, 2023, pp. 1628–1638.

[6] L. Pan, D. Zhang, A. Courville, L. Huang, and Y. Bengio, "Generative augmented flow networks," in *The Eleventh ICLR*, 2023. [Online]. Available: https://openreview.net/forum?id=urF_CBK5XC0

[7] M. Jain, E. Bengio, A. Hernandez-Garcia, J. Rector-Brooks, B. F. Dossou, C. A. Ekbote, J. Fu, T. Zhang, M. Kilgour, D. Zhang *et al.*, "Biological sequence design with gflownets," in *International Conference on Machine Learning*. PMLR, 2022, pp. 9786–9801.

[8] D. Zhang, H. Dai, N. Malkin, A. C. Courville, Y. Bengio, and L. Pan, "Let the flows tell: Solving graph combinatorial problems with gflownets," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 11 952–11 969. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/27571b74d6cd650b8eb6cf1837953ae8-Paper-Conference.pdf

[9] P. Niu, S. Wu, M. Fan, and X. Qian, "Gflownet training by policy gradients," in *Forty-first International Conference on Machine Learning*, 2024.

[10] D. W. Zhang, C. Rainone, M. Peschl, and R. Bondesan, "Robust scheduling with GFlownets," in *ICLR*, 2023. [Online]. Available: https://openreview.net/forum?id=ZBUthI6wK9h

[11] Y. Li, S. Luo, H. Wang, and J. Hao, "Cflownets: Continuous control with generative flow networks," *International Conference on Learning Representations*, 2023.

[12] Z. Sufiyan, S. Golestan, S. Miwa, Y. Mitsuka, and O. Zaiane, "A study of the efficacy of generative flow networks for robotics and machine fault-adaptation," *arXiv preprint arXiv:2501.03405*, 2025.

[13] M. W. Shen, E. Bengio, E. Hajiramezanali, A. Loukas, K. Cho, and T. Biancalani, "Towards understanding and improving gflownet training," in *International Conference on Machine Learning*. PMLR, 2023, pp. 30 956–30 975.

[14] S. Lahlou, T. Deleu, P. Lemos, D. Zhang, A. Volokhova, A. Hernández-García, L. N. Ezzine, Y. Bengio, and N. Malkin, "A theory of continuous generative flow networks," in *Proceedings of the 40th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, Eds., vol. 202. PMLR, 23–29 Jul 2023, pp. 18 269–18 300. [Online]. Available: https://proceedings.mlr.press/v202/lahlou23a.html

[15] S. Luo, Y. Li, S. Liu, X. Zhang, Y. Shao, and C. Wu, "Multi-agent continuous control with generative flow networks," *Neural Networks*, vol. 174, p. 106243, 2024.

[16] N. Malkin, S. Lahlou, T. Deleu, X. Ji, E. J. Hu, K. E. Everett, D. Zhang, and Y. Bengio, "GFlownets and variational inference," in *The Eleventh International Conference on Learning Representations*, 2023. [Online]. Available: https://openreview.net/forum?id=uKiE0VIluA-

[17] A. Farid, S. Veer, and A. Majumdar, "Task-driven out-of-distribution detection with statistical guarantees for robot learning," in *Conference on Robot Learning*. PMLR, 2022, pp. 970–980.

[18] F. Yang, C. Yang, D. Guo, H. Liu, and F. Sun, "Fault-aware robust control via adversarial reinforcement learning," in *IEEE 11th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems*, 2021, pp. 109–115.

[19] A. S. Chen, G. Chada, L. Smith, A. Sharma, Z. Fu, S. Levine, and C. Finn, "Adapt on-the-go: Behavior modulation for single-life robot deployment," *arXiv preprint arXiv:2311.01059*, 2023.

[20] I. Ahmed, M. Quinones-Grueiro, and G. Biswas, "Complementary meta-reinforcement learning for fault-adaptive control," *arXiv preprint arXiv:2009.12634*, 2020.

[21] M. Luo, A. Balakrishna, B. Thananjeyan, S. Nair, J. Ibarz, J. Tan, C. Finn, I. Stoica, and K. Goldberg, "Mesa: Offline meta-rl for safe adaptation and fault tolerance," *arXiv preprint arXiv:2112.03575*, 2021.

[22] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," 2019.

[23] L. Brunswic, Y. Li, Y. Xu, Y. Feng, S. Jui, and L. Ma, "A theory of non-acyclic generative flow networks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 10, 2024, pp. 11 124–11 131.

[24] L. Pan, N. Malkin, D. Zhang, and Y. Bengio, "Better training of GFlowNets with local credit and incomplete trajectories," in *Proceedings of the 40th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, A. Krause, E. Brunskill, K. Cho, B. Engelhardt, S. Sabato, and J. Scarlett, Eds., vol. 202. PMLR, 23–29 Jul 2023, pp. 26 878–26 890. [Online]. Available: https://proceedings.mlr.press/v202/pan23c.html

[25] N. Malkin, M. Jain, E. Bengio, C. Sun, and Y. Bengio, "Trajectory balance: Improved credit assignment in gflownets," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35. Curran Associates, Inc., 2022, pp. 5955–5967. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2022/file/27b51baca8377a0cf109f6ecc15a0f70-Paper-Conference.pdf

[26] P. Fournier, O. Sigaud, M. Chetouani, and P.-Y. Oudeyer, "Accuracy-based curriculum learning in deep reinforcement learning," *arXiv preprint arXiv:1806.09614*, 2018.

[27] E. Parisotto, S. Ghosh, S. B. Yalamanchi, V. Chinnaobireddy, Y. Wu, and R. Salakhutdinov, "Concurrent meta reinforcement learning," *arXiv preprint arXiv:1903.02710*, 2019.

[28] I. A. Co., "12 causes of servo motor failure," *Industrial Automation Insights*, 2023, accessed: 2024-04-15. [Online]. Available: https://www.industrialautomationco.com/servo-motor-failure

[29] K. America, "5 servo motor failure causes & how to prevent them," *KEB America Insights*, 2023, accessed: 2024-04-15. [Online]. Available: https://www.kebamerica.com/servo-motor-failures

[30] M. C. Tips, "Environmental effects on motion components in robots," *Motion Control Tips*, 2023. [Online]. Available: https://www.motioncontroltips.com

[31] U. Robots, "Working of robotic arm: How does a robotic arm work?" 2022, accessed: 2023-04-15. [Online]. Available: https://www.universal-robots.com

[32] G. Sokar, E. Mocanu, D. C. Mocanu, M. Pechenizkiy, and P. Stone, "Dynamic sparse training for deep reinforcement learning," in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, L. D. Raedt, Ed. International Joint Conferences on Artificial Intelligence Organization, 7 2022, pp. 3437–3443, main Track. [Online]. Available: https://doi.org/10.24963/ijcai.2022/477

[33] N. Xiong, Y. Du, and L. Huang, "Provably safe reinforcement learning with step-wise violation constraints," *Advances in Neural Information Processing Systems*, vol. 36, 2024.

[34] J. Varley, S. Singh, D. Jain, K. Choromanski, A. Zeng, S. B. R. Chowdhury, A. Dubey, and V. Sindhwani, "Embodied ai with two arms: Zero-shot learning, safety and modularity," *arXiv preprint arXiv:2404.03570*, 2024.

[35] N. Yang, S. Chen, H. Zhang, and R. Berry, "Beyond the edge: An advanced exploration of reinforcement learning for mobile edge computing, its applications, and future research trajectories," *IEEE Communications Surveys & Tutorials*, 2024.