Meta-Reinforcement Learning in Nonstationary and Nonparametric Environments

Zhenshan Bing, *Member, IEEE*, Lukas Knak, Long Cheng, Fabrice O. Morin, Kai Huang, *Member, IEEE*, and Alois Knoll, *Fellow, IEEE*

***Abstract*— Recent state-of-the-art artificial agents lack the ability to adapt rapidly to new tasks, as they are trained exclusively for specific objectives and require massive amounts of interaction to learn new skills. Meta-reinforcement learning (meta-RL) addresses this challenge by leveraging knowledge learned from training tasks to perform well in previously unseen tasks. However, current meta-RL approaches limit themselves to narrow parametric and stationary task distributions, ignoring qualitative differences and nonstationary changes between tasks that occur in the real world. In this article, we introduce a Task-Inference-based meta-RL algorithm using explicitly parameterized Gaussian variational autoencoders (VAEs) and gated Recurrent units (TIGR), designed for nonparametric and nonstationary environments. We employ a generative model involving a VAE to capture the multimodality of the tasks. We decouple the policy training from the task-inference learning and efficiently train the inference mechanism on the basis of an unsupervised reconstruction objective. We establish a zero-shot adaptation procedure to enable the agent to adapt to nonstationary task changes. We provide a benchmark with qualitatively distinct tasks based on the *half-cheetah* environment and demonstrate the superior performance of TIGR compared with state-of-the-art meta-RL approaches in terms of sample efficiency (three to ten times faster), asymptotic performance, and applicability in nonparametric and nonstationary environments with zero-shot adaptation. Videos can be viewed at https://videoviewsite.wixsite.com/tigr.**

***Index Terms*— Gaussian variational autoencoder (VAE), meta-reinforcement learning (meta-RL), robotic control, task adaptation, task inference.**

Manuscript received 17 September 2022; revised 25 January 2023; accepted 22 April 2023. This work was supported in part by the European Union's Horizon 2020 Framework Program for Research and Innovation within the "Human Brain Project Specific Grant Agreement (SGA3)" under Agreement 945539 and in part by the National Natural Science Foundation of China under Grant 61902442. *(Zhenshan Bing and Lukas Knak contributed equally to this work.) (Corresponding author: Long Cheng.)*

Zhenshan Bing, Lukas Knak, Fabrice O. Morin, and Alois Knoll are with the Department of Informatics, Technical University of Munich, 85748 Munich, Germany (e-mail: bing@in.tum.de; lukas.knak@tum.de; morinf@in.tum.de; knoll@in.tum.de).

Long Cheng is with the College of Computer Science and Artificial Intelligence, Wenzhou University, Wenzhou 325035, China (e-mail: chenglong@wzu.edu.cn).

Kai Huang is with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou 543000, China.

This article has supplementary material provided by the authors and color versions of one or more figures available at https://doi.org/10.1109/TNNLS.2023.3270298.

Digital Object Identifier 10.1109/TNNLS.2023.3270298

I. Introduction

HUMANS have the ability to learn new skills by transferring previously acquired knowledge, which enables them to quickly and easily adapt to new challenges. Reinforcement learning (RL) methods have successfully used self-learning algorithms to create agents that exhibit superhuman performance in board or computer games [1], [2], enable a robot to walk [3], or to manipulate objects, such as a Rubik's Cube [4]. However, state-of-the-art artificial agents lack the ability to adapt to differing tasks or to reuse existing experiences, since they are commonly trained from scratch on specific tasks only. This renders the application of RL to real-world problems difficult, since the algorithms require massive amounts of interaction experience with the environment for every task, resulting in severe wear and tear of the joints and gears of the used machines. For instance, to imbue a robotic hand with the dexterity to solve a Rubik's Cube, OpenAI reported a cumulative experience of 13 000 years [4]. In contrast, adult humans are able to manipulate the cube almost instantaneously, as they possess prior knowledge regarding generic object manipulation.

As a promising approach, meta-RL reinterprets this open challenge of adapting to new and yet related tasks as a learning-to-learn problem [5]. Specifically, meta-RL aims to learn new skills by first learning a prior from a set of similar tasks and then reusing this policy to succeed after few or zero trials in the new target environment. This bridges the gap between simulation and the real world by allowing the transfer of skills that are not tailored to a specific application, enabling the agent to succeed under new circumstances [3], [4]. Recent studies in meta-RL can be divided into three main categories. Gradient-based meta-RL approaches, such as model-agnostic meta-learning (MAML) [6], aim to learn a set of highly sensitive model parameters, so that the agent can quickly adapt to new tasks with only few gradient descent steps. Recurrence-based methods aim to learn how to implicitly store task information in the hidden states during meta-training and utilize the resulting mechanism during meta-testing [7]. While these two concepts can adapt to new tasks in only a few trials, they adopt on-policy RL algorithms during meta-training, which require massive amounts of data and lead to sample inefficiency. To address this issue, probabilistic embeddings for actor-critic RL (PEARL) [8], a model-free and off-policy method, achieves state-of-the-art results and

significantly outperforms prior studies in terms of sample efficiency and asymptotic performance, by representing the task with a single Gaussian distribution through an encoder, which outputs the probabilistic task embeddings.

However, most previous approaches, including PEARL, are severely limited to narrow task distributions, as they have only been applied to parametric environments [6], [8], [9], [10], [11], in which only certain parameters of the tasks are varied. This ignores the fact that humans are usually faced with qualitatively different tasks in their daily lives, which happen to share some common structure. For example, grasping a cup and turning a handle both require the dexterity of a hand. However, the nonparametric variability introduced by the two different objects makes it much more difficult to solve the tasks when compared with the sole use of parametric variations, such as turning a handle to different angles. In addition, algorithms, such as PEARL [8] and MAML [6], are designed to adapt to the task in few trials, which excludes them from being applicable to nonstationary environments, where task changes can occur at any time during the interaction. For a meta-RL approach to be applicable in common real-world applications, the challenge of nonparametric and nonstationary environments must be overcome, since any real-world task, such as refilling a cup with tap water, consists of many qualitatively different (i.e., nonparametric) subtasks (e.g., grasping the cup, putting the cup under the tap, turning the handle of the tap, and so on), and the overall goal can only be achieved by correctly solving and switching between the subtasks (i.e., the environment is nonstationary). Despite this importance, there is currently no study that explicitly focuses on both nonparametric and nonstationary environments while providing the benefits of model-free and off-policy algorithms, such as the superior data efficiency and good asymptotic performance.

In this article, we establish an approach that addresses the challenge of learning how to behave in nonparametric and broad task distributions with nonstationary task changes. We argue that only with these requirements can an algorithm be truly applied to real-world scenarios, as mentioned above. We leverage insights from PEARL [8] and introduce a **T**ask-**I**nference-based meta-RL algorithm using explicitly parameterized **G**aussian variational autoencoders (VAEs) and gated **R**ecurrent units (TIGR), which is sample-efficient, adapts in a zero-shot manner to nonstationary task changes, and achieves good asymptotic performance in nonparametric tasks. We propose a novel algorithm composed of four concepts. First, we use a generative model, leveraging a combination of Gaussians to cluster the information on each qualitatively different base task. Second, we decouple the task-inference training from the RL algorithm by reconstructing the tasks' Markov decision processes (MDPs) in an unsupervised setup. Third, we propose a zero-shot adaptation mechanism by extracting features from recent transition history and infer task information at each timestep to enable the agent to adapt to task changes at any time. Finally, we provide a novel benchmark with nonparametric tasks based on the commonly used *half-cheetah* environment. While the aforementioned techniques have been utilized to address RL challenges several times before [8], [9], [12], [13], we show that through modifications and thoroughly investigated interaction, we can use them in a novel way that allows us to tackle previously unsolved meta-RL problems. Experiment results demonstrate that the TIGR significantly outperforms state-of-the-art methods with three to ten times faster sample efficiency, substantially increased asymptotic performance, and unmatched task-inference capabilities under zero-shot adaptation in nonparametric and nonstationary environments for the first time. To the best of the authors' knowledge, TIGR is the first model-free meta-RL algorithm to solve nonparametric and nonstationary environments with zero-shot adaptation.

II. Background

A. Meta-Reinforcement Learning

The learning problem of meta-RL is extended to an agent that has to solve different tasks from a distribution $p(\mathcal{T})$ [14]. Each task $\mathcal{T}$ is defined as an individual MDP specifying its properties. A meta-RL agent is not given any task information other than the experience it gathers while interacting with the environment. A standard meta-RL setup consists of two task sets: a meta-training task set $\mathcal{D}_{\mathcal{T}}^{\text{train}}$ used to train the agent and a meta-test task set $\mathcal{D}_{\mathcal{T}}^{\text{test}}$ used to evaluate the agent. Both sets are drawn from the same distribution $p(\mathcal{T})$, but $\mathcal{D}_{\mathcal{T}}^{\text{test}}$ may differ from $\mathcal{D}_{\mathcal{T}}^{\text{train}}$. The objective is to train a policy π_θ on $\mathcal{D}_{\mathcal{T}}^{\text{train}}$ that maximizes rewards on $\mathcal{D}_{\mathcal{T}}^{\text{test}}$, which is defined as follows:

$$\theta^* = \arg\max_\theta \mathbb{E}_{\mathcal{T}\sim\mathcal{D}_{\mathcal{T}}^{\text{test}}}\left[\mathbb{E}_{\tau\sim p(\tau|\pi_\theta)}\left[\sum_t \gamma^t r_t\right]\right]. \tag{1}$$

B. Meta-Training and Meta-Testing

Meta-RL consists of two stages: meta-training and meta-testing. During meta-training, each training epoch consists of a data collection and optimization phase. In the data collection phase, interaction experiences for each task are collected and stored in the replay buffer. In the optimization phase, the losses for the policy are computed, and the gradient of the averaged losses is used to update the parameters of the policy. During meta-testing, the policy is adapted to new tasks with either few trials, meaning that the agent can experience the presented environment and adapt before the final evaluation, or in zero-shot manner, which means that the agent must solve the environment at first sight.

C. Stationary and Nonstationary Environments

The meta-RL setting can involve stationary or nonstationary environments. In stationary environments, each episode consists of one task; i.e., the underlying MDP of the environment is fixed during an episode. In such cases, an algorithm can use a few-shot (episode-wise) mechanism to adapt to the task by collecting experiences for a few episodes and adjusting before the final evaluation in the last trial. In nonstationary environments, on the other hand, the underlying MDP can change at any timestep. Here, episode-wise adaptation fails, and a zero-shot procedure with online (or continuous) adaptation at the transition level is required. The environments are regulated in a way that between switching two goals, there are at least

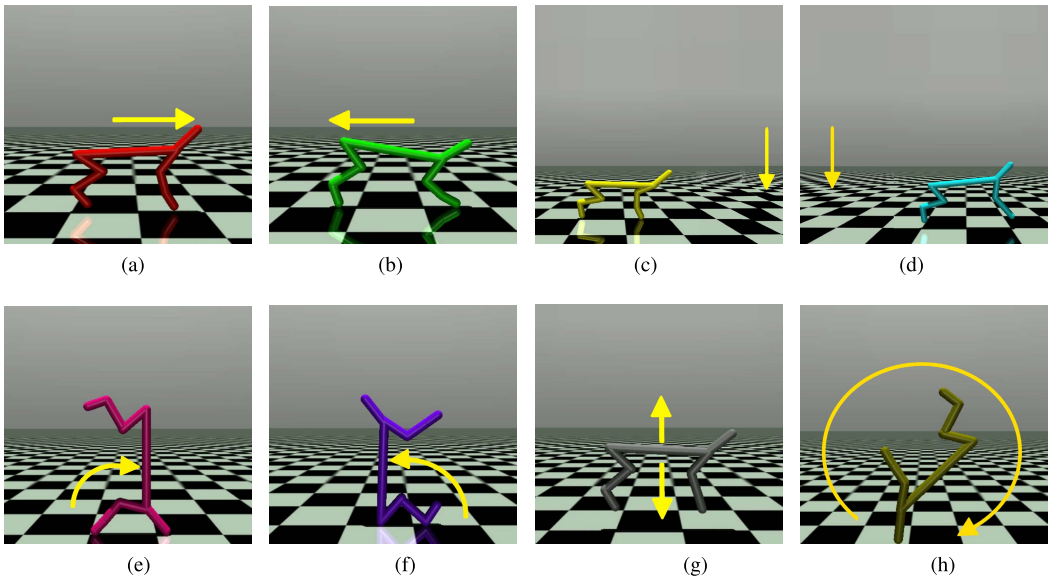


Fig. 1. Visualization of eight base environment tasks (yellow arrows show the movement direction). These eight tasks share similar dynamic models, but are qualitatively distinct. Each task contains parametric variations, e.g., different goal velocities in the run forward/backward task. (a) Run forward. (b) Run backward. (c) Reach front goal. (d) Reach back goal. (e) Front stand. (f) Back stand. (g) Jump. (h) Front flip.

some number of timesteps (e.g., 50 timesteps), so the agent can use the information from recent transitions to figure out the current objective [15] (i.e., tasks must not be switched too rapidly). Thus, we can describe nonstationary environments as a set of several stationary subtasks that the agent must adapt to. This is a particularly common scenario in the real world, for example, when a motor malfunction occurs in a robot's joint that changes its dynamics and, therefore, impacts the MDP underlying the task. Due to the importance of this challenge for real world application of meta-RL, we also tackle this problem in this study.

D. Parametric and Nonparametric Variability in Meta-RL

Two key properties define the underlying structure of task distributions in meta-RL [14]: parametric and nonparametric variability. Parametric variability describes tasks that qualitatively share the same properties (i.e., their semantic task descriptions are similar), but the parameterization of the tasks varies. Parametric task distributions tend to be more homogeneous and narrower, which limits the generalization ability of the trained agent to new tasks. Nonparametric variability, however, describes tasks that are qualitatively distinct but share a common structure, so a single meta-RL agent can succeed (see Fig. 1 for a visual example of nonparametric tasks). Nonparametric task distributions are considerably more challenging, since each distinct task may contain parametric variations [14].

E. Continual Learning

In continual learning settings, as described in [16], the tasks that an agent has to solve change during the training process. The agent can use information learned from previous, simpler tasks to succeed in new, more complex environments. Retaining learned abilities from earlier time steps is a critical skill for the agent in a continual learning process. Lack of this skill can lead to forgetting of older abilities on the one hand, but can also benefit overfitting on earlier tasks on the other. We consider a continual learning setting in which access to different nonparametric tasks changes over time, but the agent is given the total number of nonparametric tasks at the start of training. We explore two different possibilities to regulate access to the nonparametric tasks: 1) the agent starts with access to only one nonparametric task, and the number of accessible tasks increases during the training process ("linear" setting) and 2) the agent starts with access to only one nonparametric task, and with each new task, access to the previous task is removed, but the gained experience can still be used ("cut" setting).

F. Probabilistic Embeddings for Actor-Critic RL

In task-inference-based meta-RL, the task information that the agent lacks to enable it to behave optimally given a problem $\mathcal{T} \sim p(\mathcal{T})$ is modeled explicitly [10]. PEARL [8] learns a probabilistic latent variable $\boldsymbol{z}$ that encodes the salient task information given by a fixed-length context variable $\boldsymbol{c}^{\mathcal{T}}_{1:N}$ containing N recently collected experiences of a task $\mathcal{T}$, which is fed into the policy $\pi_\theta(\boldsymbol{a}|\boldsymbol{s}, \boldsymbol{z})$ trained via soft actor-critic (SAC) [17] to solve the presented task. To encode the task information, an inference network $q_\phi(\boldsymbol{z}|\boldsymbol{c}^{\mathcal{T}}_{1:N})$ is learned with the variational lower bound objective

$$\mathbb{E}_{\mathcal{T}\sim p(\mathcal{T})}\Big[\mathbb{E}_{\boldsymbol{z}\sim q_\phi(\boldsymbol{z}|\boldsymbol{c}^{\mathcal{T}}_{1:N})}\big[R(\mathcal{T}, \boldsymbol{z}) + \beta D_{\mathrm{KL}}\big(q_\phi\big(\boldsymbol{z}|\boldsymbol{c}^{\mathcal{T}}_{1:N}\big)\big\|p(\boldsymbol{z})\big)\big]\Big] \tag{2}$$

where $p(\boldsymbol{z})$ is a Gaussian prior used as a bottleneck constraint on the information of $\boldsymbol{z}$, given context $\boldsymbol{c}^{\mathcal{T}}_{1:N}$ using the Kullback-Leibler (KL)-divergence D_{KL}. $R(\mathcal{T}, \boldsymbol{z})$ is an objective used to train the encoder via the Bellman critics loss. β is a hyperparameter for weighting the KL-divergence. The probabilistic encoder q_ϕ in PEARL is modeled as a product of independent Gaussian factors assuming permutation invariance for the N transitions in the context with $q_\phi(\boldsymbol{z}|\boldsymbol{c}^{\mathcal{T}}_{1:N}) \propto \prod_n \Psi_\phi(\boldsymbol{z}|\boldsymbol{c}^{\mathcal{T}}_n)$,

where $\Psi_\phi(\boldsymbol{z}|\boldsymbol{c}_n^{\mathcal{T}}) \sim \mathcal{N}(f_\phi^\mu(\boldsymbol{c}_n^{\mathcal{T}}), f_\phi^{\sigma^2}(\boldsymbol{c}_n^{\mathcal{T}}))$ and f_ϕ is represented as a neural network with parameters ϕ that outputs the mean μ and variance σ^2 of the Gaussian conditioned on the context $\boldsymbol{c}_n^{\mathcal{T}}$. The parameters of the network f_ϕ are optimized during meta-training. Here, in each data collection phase, previous experiences are iteratively added to the context $\boldsymbol{c}_n^{\mathcal{T}}$ to predict the new latent task representation $\boldsymbol{z}$ used in the policy for the next action. Then, during the optimization phase, the Bellman critics loss is used to jointly update the parameters of the encoder and actor-critic model using contexts sampled from the replay buffer as inputs. During meta-testing, PEARL performs a few-shot adaptation by first collecting experiences and then computing a posterior for the latent representation $\boldsymbol{z}$ obtained through the encoder. This latent representation remains unchanged during the entire rollout. The few-shot mechanism and the assumption of permutation invariance of the collected transitions during a rollout renders PEARL inapplicable to nonstationary task change.

III. Related Work

The recent work in the domain of meta-RL can be divided into three groups according to the approach taken: gradient-based, recurrence-based, and task-inference-based.

A. Gradient-Based

Gradient-based meta-RL approaches, such as MAML [6] and follow-up methods [11], [15], [18], are based on finding a set of model parameters during meta-training that can rapidly adapt to achieve large improvements on tasks sampled from a distribution $p(\mathcal{T})$. During the meta-test phase, the initial learned parameters are adjusted to succeed in the task with few gradient steps [6]. Gradient-based approaches can be applied to nonstationary environments only if the parameter adaption is performed after every timestep as in [11]. Inspired by MAML [6], some works utilized the trained policy from meta-training as an initializer and performed gradient descent to achieve fast and continuous adaption in nonstationary environments [18], [19]. Gradient-based methods do not involve reasoning about the task properties, which distinguishes them from our method.

B. Recurrence-Based

The key element in recurrence-based methods, as in [4], [5], [7], and [20], is the implementation of a recurrent model that uses previous interactions to implicitly store information that the policy can exploit to perform well on a task distribution. By resetting the model at the beginning of each rollout and recurrently feeding states, actions, and rewards back to the model, the agent can track the interaction history over the entire path in its hidden state and learn how to memorize relevant task information [7]. In meta-training, the model is trained via backpropagation through time. In meta-testing, the parameters are fixed, but the agent's internal state adapts to the new task in zero-shot manner [7]. Examples, such as fast reinforcement learning via slow reinforcement learning (RL2) [5] and variational Bayes-adaptive deep RL via meta-learning (VariBAD) [21], are also implemented in nonparametric tasks, but tasks are only sampled from a narrow task distribution (only one base task).

C. Task-Inference-Based

In task-inference-based meta-RL, as in PEARL [8], the information that the agent lacks to enable it to behave optimally is modeled explicitly [10]. Model agnostic exploration with structured noise (MAESN) [22], for example, learns a task-dependent latent space that is used to introduce structured noise into the observations to guide the policy's exploration. While we also feed a latent description of the task to the policy to learn task-dependent behavior, we do not keep the task embedding stationary during policy execution as we want to enable our agent to adapt to the task online. Lan et al. [12] build on this idea and improve on MAESN by modeling the task explicitly using an encoder consisting of a gated recurrent unit (GRU) to extract information from a history of transitions, which is given to the policy in addition to the observations. Hu et al. [23] extended PEARL by adding a dynamic task-adaptiveness distillation module to describe how the meta-learners adjust the exploration strategy in the meta-training process. The distillation module can measure the meta-learner's awareness of how it has adapted among the distribution of tasks, which can encourage self-oriented exploration in new tasks. Empirical results show improved performance compared with PEARL in terms of averaged return. Jiang et al. [24] also extended PEARL by adding two exploration terms in action and task embedding space. Moreover, meta-RL algorithms have also been implemented to solve the control tasks, such as tracking control of autonomous vehicles [25] and path planning tasks [26]. In contrast to their approach, we do not directly feed the aggregated task information of the recurrent network to the policy, but instead learn a Gaussian distribution over the latent task information from which we can sample embedded task descriptors. The benefits of using Gaussian distributions to describe the latent task information have also been shown in [8] and [9] (see Section II-F). In [9], the feature extraction is improved by leveraging a graph neural network that aggregates task information over time and outputs a Gaussian distribution over the latent representation. We build on the idea of using Gaussian distributions and show that single Gaussian distributions, although valid for narrow task distributions, fail to capture the variance between tasks in nonparametric environments. Thus, similar to Ren et al. [13], who further use a combination of a Dirichlet and a Gaussian distribution to model different base tasks with style factors, we also adapt the Gaussian model for nonparametric tasks, but in contrast we use a generative model of explicitly parameterized Gaussian VAEs. Moreover, our method reduces the complexity of the latent space, while [13] was only examined in toy goal-reaching environments.

IV. Problem Statement

This work aims to solve nonparametric meta-RL tasks, in which an agent is trained to maximize the expected discounted return across multiple test tasks from a nonparametric task distribution in a nonstationary setting. Specifically, we aim to achieve the following goals: first, our algorithm should be applicable to nonparametric and broad task distributions in a meta-RL setting. Second, the developed algorithm must be able to perform zero-shot adaptation to nonstationary

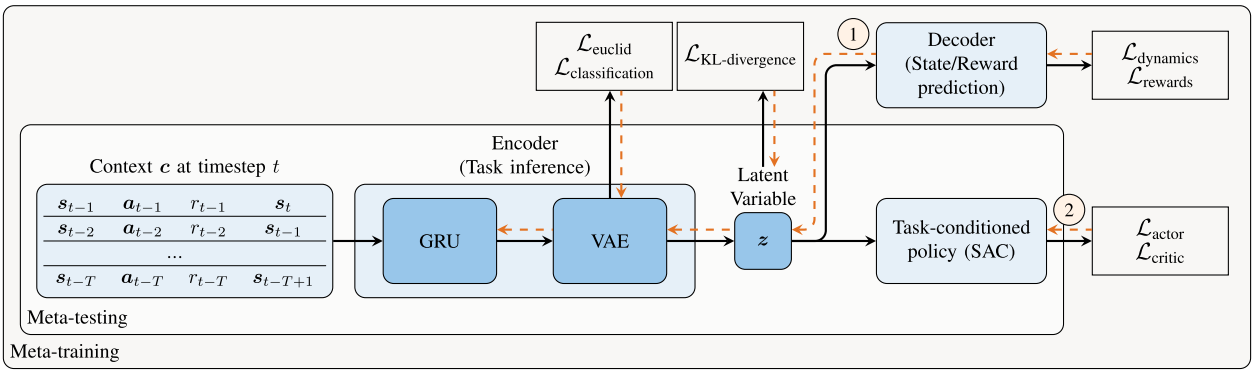


Fig. 2. Meta-training and meta-testing procedure. The encoder learns a task encoding z from the recent context with gradients from the decoder and provides z for the task-conditioned policy trained via SAC during meta-training. In meta-testing, the latent task description z is fed into the task-conditioned policy to perform the actions. Orange arrows outline the gradient flow.

task changes. Finally, the method should provide the sample efficiency and asymptotic performance of model-free and off-policy algorithms. To the best of our knowledge, there is no approach that provides the advantages of model-free and off-policy algorithms and is applicable to nonparametric environments in a zero-shot manner. Currently, the only benchmark that satisfies the environment requirements is Meta-world [14]. Metaworld does not offer well-defined rewards that are normalized across all environments, which can lead to strong bias toward dominant tasks. Clear evidence can be found in the first version of Metaworld [14], in which very poor performance of state-of-the-art metal RL algorithms [5], [6], [8] is discussed. Thus, we provide our own benchmark *half-cheetah-eight* to evaluate different meta-RL approaches. The detailed description of how the agent interacts with the environment is described in Appendix A and Table I. Note that different from model pretraining algorithms that aim to have an initializer to perform gradient steps during testing, our method aims to learn a task-dependent latent space that is used to identify the MDP of the task. During meta-test phase, the latent space is once again used to update the agent's belief about the task and solve it accordingly.

V. Methodology

In this section, we first give an overview of our TIGR algorithm. We then explain the strategy for making TIGR applicable in nonparametric and nonstationary environments, derive the generative model, and explain how we implement the encoder and decoder. Finally, we summarize TIGR with its pseudocode.

A. Overview

In this article, we leverage the notion of meta-RL as task inference. Similar to PEARL, we also extract information from the transition history and use an encoder to generate task embeddings, which are provided to a task-conditioned policy learned via SAC. We design a generative model for the task inference to succeed in nonparametric environments. Unlike PEARL, however, we first decouple the training of the probabilistic encoder from the training of the policy by introducing a decoder that reconstructs the underlying MDP of the environment. Second, we modify the training and testing procedure to encode task information from the recent transition history on a per-time-step basis, enabling zero-shot adaptation to nonstationary task changes. The structure of our method is shown in Fig. 2. Our algorithm is briefly explained as follows.

1) During meta-training, we first gather interaction experiences from the training tasks and store them in the replay buffer. At each interaction, we infer a task representation, such that the policy can behave according to the objective in (1). We feed the recent transition history into a GRU [Section V-B1.b], which merges the extracted information and forward the features to the VAE [Section V-B1.a] to generate the overall task representation z. The task representation is given to the policy with the current observation to predict the corresponding action.
2) Second, we optimize the task-inference and policy networks, in two sequential stages.
   a) We first train the GRU-VAE encoder networks for task inference by reconstructing the underlying MDP. For this, we use two additional neural networks that predict the dynamics and reward for each transition [see orange gradient ① in Fig. 2 and Section V-B2.a]. This gives the encoder the information required to generate an informative task representation. To improve the performance of the task inference, we employ two additional losses, namely, $\mathcal{L}_{\text{classification}}$ and $\mathcal{L}_{\text{Euclid}}$ [see Section V-B2.c]. We do not use any gradients from SAC (see orange gradient ②), which enables us to train the encoder independently of the task-conditioned policy.
   b) In policy training, we compute the task representation for the sampled transitions online using our GRU-VAE encoder. We feed this information to the task-conditioned policy and train it via SAC independently of the task-inference mechanism.
3) During meta-testing, our method infers the task representation at each timestep, selects actions with the task-conditioned policy, and adapts to the task in zero-shot manner.

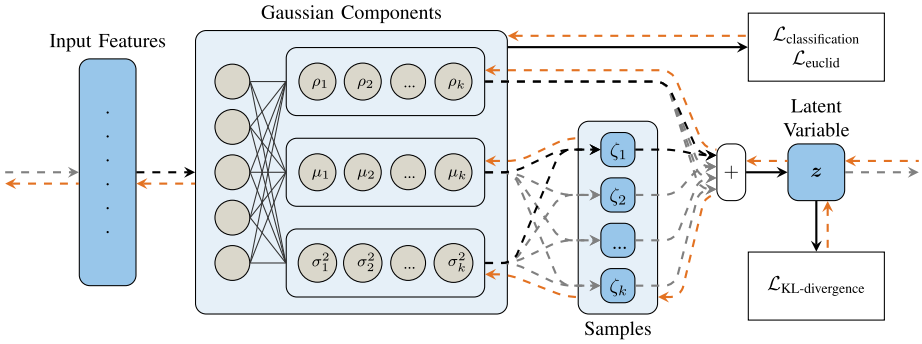


Fig. 3. Overview of the VAE network. The statistics for each Gaussian, including mean $\mu(\boldsymbol{c}, k)$, variance $\sigma^2(\boldsymbol{c}, k)$, and activation $\rho(\boldsymbol{c}, k)$, are computed in parallel. The values are processed, and the task representation $\boldsymbol{z}$ is calculated as the weighted sum denoted by the $+$ as described in Section V-B1.a. Orange arrows outline the gradient flow.

B. Task Inference

The nonparametric environments that we consider describe a broad task distribution, with different clusters representing the nonparametric base tasks, and the intra-cluster variance describing the parametric variability for each objective. We improve the generative model of PEARL and propose an expressive generative model that captures the multimodality of the environments and produces reasonable task representations that account for both nonparametric and parametric variability.

1) Generative Model: Given the sequence of the recent transition history in the last T timesteps as the context $\boldsymbol{c} = (\boldsymbol{s}_{t-T}, \boldsymbol{a}_{t-T}, r_{t-T}, s_{t-T+1}, \ldots, \boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1}, r_{t-1}, s_t)$, we aim to extract features and find a latent representation $\boldsymbol{z}$ that explains $\boldsymbol{c} \sim p(\boldsymbol{c}|\boldsymbol{z})$ in a generative model, such that $p(\boldsymbol{c}, \boldsymbol{z}) = p(\boldsymbol{c}|\boldsymbol{z})p(\boldsymbol{z})$. Using the last T timesteps to infer the current task representation z at each interaction allows the model to adapt to task changes online and in zero-shot manner. We reinterpret the joint structure of the meta-RL task distribution as a combination of different features in a latent space that represent the properties of a particular objective. Following this idea, we model $\boldsymbol{z}$ as a combination of Gaussians that can express both nonparametric variability with the different Gaussian modes and parametric variability using the variance of a particular Gaussian k with its statistics $\mu(\boldsymbol{c}, k)$ and $\sigma^2(\boldsymbol{c}, k)$. We introduce an activation $\rho(\boldsymbol{c}, k)$ that determines the impact of each Gaussian k on $\boldsymbol{z}$ subject to $\sum_k \rho(\boldsymbol{c}, k) = 1$. We sample representatives

$$\zeta_k \sim \mathcal{N}\big(\mu(\boldsymbol{c}, k), \sigma^2(\boldsymbol{c}, k)\big) \tag{3}$$

from each Gaussian, which are then fused to represent the latent task encoding. We obtain a linear combination of random variables ζ_k representing different latent task features, which describes a distribution with

$$p(\boldsymbol{z}|\boldsymbol{c}) = \sum_k \rho(\boldsymbol{c}, k) \cdot \zeta_k. \tag{4}$$

Given a shared structure between tasks, the weights $\rho(\boldsymbol{c}, k)$, thus, represent the activation for the latent features that explain the tasks' properties. The final distribution corresponds to a VAE with a single Gaussian distribution, but we show that the explicit parameterization with multiple Gaussian modes is advantageous for representing nonparametric task distributions. First, linear combination using the activations $\rho(\boldsymbol{c}, k)$ allows us to permanently assign a Gaussian to each task, which is beneficial in continual learning settings, because the separate parameters for each Gaussian prevent forgetting of older skills. Second, the explicit parameterization allows us to use prior information about the nonparametric task distribution, since we can train the algorithm to discriminate between nonparametric tasks using a categorical regularizer similar to [16] for the activations $\rho(\boldsymbol{c}, k)$. We argue that the use of this prior during training does not violate the principle of meta-RL, because it is not given directly to the agent. We compute $\mu(\boldsymbol{c}, k), \sigma^2(\boldsymbol{c}, k)$, and $\rho(\boldsymbol{c}, k)$ as described in Section V-B1.a.

a) VAE architecture: Using the variational inference approach, we approximate the intractable posterior $p(\boldsymbol{z}|\boldsymbol{c})$ using a variational posterior $q_\theta(\boldsymbol{z}|\boldsymbol{c})$, parameterized by neural networks with parameters θ. The neural network that implements the VAE is designed as a multilayer perceptron (MLP) that predicts the statistics for each Gaussian component, including the mean $\mu(\boldsymbol{c}, k)$, variance $\sigma^2(\boldsymbol{c}, k)$, and activation $\rho(\boldsymbol{c}, k)$ as a function of the input features derived from the context $\boldsymbol{c}$ (see Fig. 3). The standard model is given as a two-layer network and an output layer size of $K \times (\dim(\boldsymbol{z}) \times 2 + 1)$, where $\dim(\boldsymbol{z})$ is the latent dimensionality required for mean $\mu(\boldsymbol{c}, k)$ and variance $\sigma^2(\boldsymbol{c}, k)$, and the Gaussian activation value is $\rho(\boldsymbol{c}, k)$. K is the number of Gaussian components, which we set equal to the number of nonparametric tasks. This gives the algorithm prior information about the task distribution, which could be circumvented in future work by determining K online as described in [16]. Finally, we represent the VAE components as multivariate Gaussian distributions with mean $\mu(\boldsymbol{c}, k)$ and covariance $\Sigma_k = \boldsymbol{I} \odot \sigma^2(\boldsymbol{c}, k)$, where the diagonal of Σ_k consists of the entries of $\sigma^2(\boldsymbol{c}, k)$, while every other value is 0. This assumes that there are no statistical effects between the tasks. We apply a softplus operation to enforce that the network output $\sigma^2(\boldsymbol{c}, k)$ contains only positive values. We sample from the multivariate Gaussian distributions and obtain representatives ζ_k for each Gaussian component. We enforce $\sum_k \rho(\boldsymbol{c}, k) = 1$ by computing the softmax over the VAE's output for the $\rho(\boldsymbol{c}, k)$ values. Using the computed activations $\rho(\boldsymbol{c}, k)$ and the representatives, we obtain the final latent task representation as in (4).

b) Feature extraction: In this article, we consider RL environments that are described as high-dimensional MDPs. To enable our VAE to produce an informative task representation from the high-dimensional input data, we first employ a

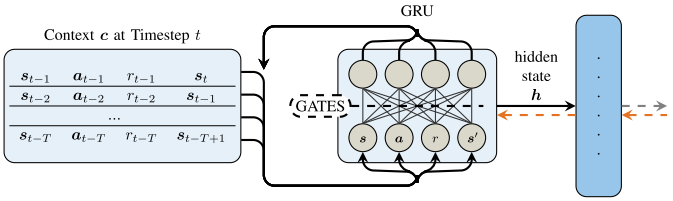


Fig. 4. Overview of GRU feature extraction. Transitions from the context $\boldsymbol{c}$ are fed in recurrently, and the last hidden state is extracted. Orange arrows outline the gradient flow.

feature-extraction mechanism to find the relevant information contained in the context. We employ a GRU to process the sequential input data (see Fig. 4). We recurrently feed in the transitions of the context $\boldsymbol{c}$ and, thereby, combine the features internally in the GRU's hidden state. We extract this hidden state after the last transition is processed and forward it into the VAE. We implement two more feature-extraction architectures for comparison. First, a shared MLP architecture similar to PEARL [8] that processes each transition of the context in parallel. The extracted features are passed into the VAE, where we combine the Gaussians for each timestep using the standard Gaussian multiplication with $\mu = (\mu_1\sigma_2^2 + \mu_2\sigma_1^2/\sigma_2^2 + \sigma_1^2)$ and $\sigma^2 = (\sigma_1^2\sigma_2^2/\sigma_1^2 + \sigma_2^2)$. Second, a Transformer architecture [27] that creates a key-value embedding for each transition in the context. We extract features from the embedding with a linear layer and forward them to the VAE, where we combine the Gaussians for each timestep using the standard Gaussian multiplication (see MLP). We provide an ablation study of the feature-extraction configurations in Section VII.

2) Encoder–Decoder Strategy: Our generative model follows the idea of a VAE. The setup employs an encoder, to describe the latent task information given a history of transitions from an MDP as $q_\theta(\boldsymbol{z}|\boldsymbol{c})$, and a decoder to reconstruct the MDP from the latent task information given by the encoder as $p_\phi(\boldsymbol{c}|\boldsymbol{z})$. The encoder is modeled as a VAE, which involves the prior use of a shared feature-extraction method, as described in Section V-B1. The decoder implements the generating function $p_\phi(\boldsymbol{c}|\boldsymbol{z})$, parameterized as neural networks with parameters ϕ. Following the variational approach in [28], we derive the evidence lower bound objective (ELBO) for the encoder and decoder and obtain:

$$\begin{aligned}\log p_\phi(\boldsymbol{c}) &\geq \mathcal{L}(\theta,\phi;\boldsymbol{c})\\ &= \mathbb{E}_{q_\theta(\boldsymbol{z}|\boldsymbol{c})}\big[\log p_\phi(\boldsymbol{c},\boldsymbol{z}) - \log q_\theta(\boldsymbol{z}|\boldsymbol{c})\big]\\ &= \mathbb{E}_{q_\theta(\boldsymbol{z}|\boldsymbol{c})}\big[\log p_\phi(\boldsymbol{c}|\boldsymbol{z})\big] - D_{\mathrm{KL}}\big(q_\theta(\boldsymbol{z}|\boldsymbol{c})\,\big\|\,p_\phi(\boldsymbol{z})\big). \quad (5)\end{aligned}$$

We use the reparameterization trick and combine the k Gaussian components to arrive at

$$\tilde{\boldsymbol{z}} = \sum_k \rho_{q_\theta}(\boldsymbol{c},k)\big(\mu_{q_\theta}(\boldsymbol{c},k) + \epsilon\cdot\sigma^2_{q_\theta}(\boldsymbol{c},k)\big) \quad (6)$$

with $\epsilon \sim \mathcal{N}(0,1)$ and apply Monte Carlo sampling to arrive at the objective

$$\mathcal{L}(\theta,\phi;\boldsymbol{c}) \approx \big[\log p_\phi\big(\boldsymbol{c}|\tilde{\boldsymbol{z}}\big) - D_{\mathrm{KL}}\big(q_\theta\big(\tilde{\boldsymbol{z}}|\boldsymbol{c}\big)\,\big\|\,p_\phi\big(\tilde{\boldsymbol{z}}\big)\big)\big] \quad (7)$$

where $\log p_\phi(\boldsymbol{c}|\tilde{\boldsymbol{z}})$ is a reconstruction objective of the input data $\boldsymbol{c}$. $D_{\mathrm{KL}}\big(q_\theta(\tilde{\boldsymbol{z}}|\boldsymbol{c})\,\big\|\,p_\phi(\tilde{\boldsymbol{z}})\big)$ introduces a regularization conditioned on the prior $p_\phi(\tilde{\boldsymbol{z}})$. Since the lower bound objective

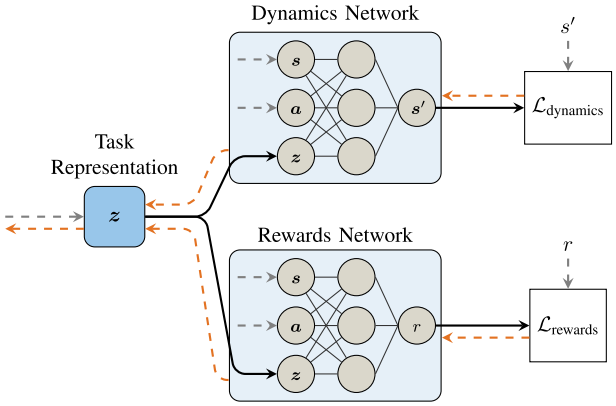


Fig. 5. Overview of dynamics and reward prediction networks. The latent task representation $\boldsymbol{z}$ in addition to the state $\boldsymbol{s}$ and action $\boldsymbol{a}$ is used as the input to both networks. Orange arrows outline the gradient flow.

requires maximization, we denote the optimization objective as minimizing $-\mathcal{L}(\theta,\phi;\boldsymbol{c})$, which results in a negative log-likelihood objective for the reconstruction term.

a) Reconstruction objective: The reconstruction objective provides the information necessary for the encoder and VAE to extract and compress relevant task information from the context. It can take many forms, as suggested in [8], such as reducing the Bellman critic's loss, maximizing the actor's returns, and reconstructing states and rewards. We follow the third proposal and extend the negative log-likelihood objective of reconstructing states and rewards to predicting the environment dynamics and the reward function for the underlying MDP (see Fig. 5). We split the decoder into two parts $p_{\phi_{\text{dynamics}}}$ and $p_{\phi_{\text{rewards}}}$ modeled as MLPs (see Fig. 5), which predict the next state $\boldsymbol{s}'$ and reward r given $\boldsymbol{s}, \boldsymbol{a}$, and $\boldsymbol{z}$, and train them with the following loss:

$$\begin{aligned}\log p_\phi\big(\boldsymbol{c}|\tilde{\boldsymbol{z}}\big) &= \log p_\phi\big(\boldsymbol{s}', r|\boldsymbol{s},\boldsymbol{a},\tilde{\boldsymbol{z}}\big)\\ &= \log p_{\phi_{\text{dynamics}}}\big(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a},\tilde{\boldsymbol{z}}\big) + \log p_{\phi_{\text{rewards}}}\big(r|\boldsymbol{s},\boldsymbol{a},\tilde{\boldsymbol{z}}\big).\end{aligned} \quad (8)$$

We model both parts as regression networks, in which the data are modeled as a normal distribution. Thus, the loss function is defined as the sum of $\mathcal{L}_{\text{dynamics}}(\phi)$ and $\mathcal{L}_{\text{rewards}}(\phi)$ as follows:

$$\begin{aligned}\mathcal{L}_{\text{prediction}}(\phi) = &\frac{1}{\dim(\boldsymbol{s})}||\boldsymbol{s}' - p_{\phi_{\text{dynamics}}}\big(\boldsymbol{s}'|\boldsymbol{s},\boldsymbol{a},\tilde{\boldsymbol{z}}\big)||^2\\ &+ \frac{1}{\dim(r)}||r - p_{\phi_{\text{rewards}}}\big(r|\boldsymbol{s},\boldsymbol{a},\tilde{\boldsymbol{z}}\big)||^2\end{aligned} \quad (9)$$

where both components are normalized by the number of their dimensions.

b) Information bottleneck: The regularization introduced by the KL-divergence in (7) serves as an information bottleneck that helps the VAE compress the input to a compact format. Due to the potential over-regularization of this term [29], we control its impact on the ELBO as follows:

$$-\mathcal{L}(\theta,\phi;\boldsymbol{c}) \approx \mathcal{L}_{\text{NLL}}(\theta,\phi) + \alpha\cdot\mathcal{L}_{\text{KL-divergence}}(\theta) \quad (10)$$

with a factor $\alpha < 1$ to allow expressive latent representations.

c) Clustering losses: To improve the task inference performance for nonparametric environments, we employ two additional losses that represent the following ideas.

Algorithm 1 TIGR Meta-Training

Require: Encoder q_θ, decoder p_ϕ, policy π_ψ, Q-network Q_ω, task distribution $p(\mathcal{T})$, replay buffer $\mathcal{D}$

1: **for** each epoch **do**
2: Perform roll-out for each task $\mathcal{T} \sim p(\mathcal{T})$, store in $\mathcal{D}$
3: **for** each task-inference training step **do**
4: Sample context $\boldsymbol{c} \sim \mathcal{D}$
5: Compute $\boldsymbol{z} = q_\theta(\boldsymbol{c})$ ▷ (See Section V-B1.a)
6: Calculate losses $\mathcal{L}_{\text{KL-divergence}}$, $\mathcal{L}_{\text{Euclid}}$, $\mathcal{L}_{\text{classification}}$ ▷ (See Section V-B2.d)
7: Compute $(\boldsymbol{s}', r) = p_\phi(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{z})$ ▷ (See Section V-B2.a)
8: Calculate loss $\mathcal{L}_{\text{prediction}}$
9: Derive gradients for losses with respect to q_θ, p_ϕ and perform optimization step
10: **for** each policy training step **do**
11: Sample RL batch $d \sim \mathcal{D}$ and corresponding context $\boldsymbol{c}$, infer $\boldsymbol{z} = q_\theta(\boldsymbol{c})$
12: Perform SAC algorithm for π_ψ, Q_ω

return q_θ, π_ψ

1) Assign each component of the VAE to one base task only. This objective is related to component-constraint learning as in [16]. The idea is to enforce each Gaussian to represent one base task, containing only task-specific features. We can feed prior information about a true base task y to the algorithm and use a supervised classification learning approach for the activations $\rho(\boldsymbol{c}, k)$ with the standard cross-entropy formulation as follows:

$$\mathcal{L}_{\text{classification}}(\theta) = -\log\left(\frac{\exp(\rho(\boldsymbol{c}, k = y))}{\sum_k \exp(\rho(\boldsymbol{c}, k))}\right). \quad (11)$$

Cross-entropy enforces that task activations correspond to the base task distribution, such that $\rho_k \to 1$ for $k \triangleq y_k$. By introducing this secondary objective, we have the possibility to constrain the components to represent task-specific features instead of shared features among tasks.

2) Push the components of the VAE away from each other to achieve a clear distinction between base tasks. To be able to further distinguish the features and prevent overlap, we confine them to separate clusters. This can be realized by an objective that seeks to maximize the Euclidean distance between the means $\mu(\boldsymbol{c}, k)$ of the K components scaled by the sum of variances $\sigma^2(\boldsymbol{c}, k)$ with

$$\mathcal{L}_{\text{Euclid}}(\theta) = \sum_{k_1=1}^{K} \sum_{k_2=k_1+1}^{K} \frac{\sigma^2(\boldsymbol{c}, k_1) + \sigma^2(\boldsymbol{c}, k_2)}{||\mu(\boldsymbol{c}, k_1) - \mu(\boldsymbol{c}, k_2)||^2}. \quad (12)$$

The Euclidean distance is replaced by the sum of squares, which avoids the computationally unstable calculation of the square root, but has the same effect.

We provide an evaluation of the impact of the clustering losses in Section V-C.

d) Final objective: The final objective derived from the reconstruction objective, information bottleneck, and clustering losses is used to jointly train the encoder–decoder setup as described in Section V-B2. We combine the different loss functions to enable the encoder to produce informative embeddings, which are used in the task-conditioned policy. The resulting overall loss is denoted as follows:

$$\mathcal{L}(\theta, \phi) = \mathcal{L}_{\text{NLL (prediction)}}(\theta, \phi) + \alpha \cdot \mathcal{L}_{\text{KL-divergence}}(\theta) + \beta \cdot \mathcal{L}_{\text{Euclid}}(\theta) + \gamma \cdot \mathcal{L}_{\text{classification}}(\theta) \quad (13)$$

where α, β, and γ are hyperparameters that weigh the importance of each term.

C. Algorithm Overview

The TIGR algorithm is summarized in pseudocode (see Algorithm 1). The task-inference mechanism is implemented from lines 3–9. Lines 10 and 12 implement the standard SAC [17]. A list of the most important hyperparameters of the algorithm and their values is given in Appendix B.

VI. Experiments

We evaluate the performance of our method on the non-parametric *half-cheetah-eight* benchmark that we provide and verify its wide applicability on a series of other environments. The *half-cheetah-eight* environments are shown in Fig. 1, and their detailed descriptions are introduced in Appendix A. In each experiment, we iteratively gather training transitions in the data collection phase, where the agent interacts with the environments following the policy. These data are stored inside a replay buffer (i.e., the meta-RL training dataset) from which we sample transitions during the optimization phase to train the encoder and decoder, as well as the actor and critic, according to their respective objectives. The experiment finishes after enough training iterations were completed, and the exact training parameters can be seen in the Appendix. The evaluation metric is the average reward during the meta testing phase. We first compare the sample efficiency and asymptotic performance against state-of-the-art meta-RL algorithms, including PEARL. Second, we visualize the latent space encoding of the VAE. Third, we evaluate the task-inference capabilities of our algorithm in the zero-shot setup. Fourth, we evaluate the applicability of the algorithm to nonstationary task changes in the *half-cheetah-eight* benchmark. Finally, we provide videos displaying the distinct learned behaviors in the Supplementary Material, along with our code.

A. Asymptotic Performance and Sample Efficiency

We first demonstrate the performance of PEARL and our method in standard parametric environments, namely, *half-cheetah-vel* and *half-cheetah-dir* tasks [8], to verify both approaches. It should be noted that PEARL achieves reported performances in few-shot manner, while our method is tested at first sight in zero-shot fashion. Fig. 6 shows that both methods achieve similar performance and can solve the tasks. However, TIGR significantly outperforms PEARL in terms of sample efficiency across both tasks, even in zero-shot manner.

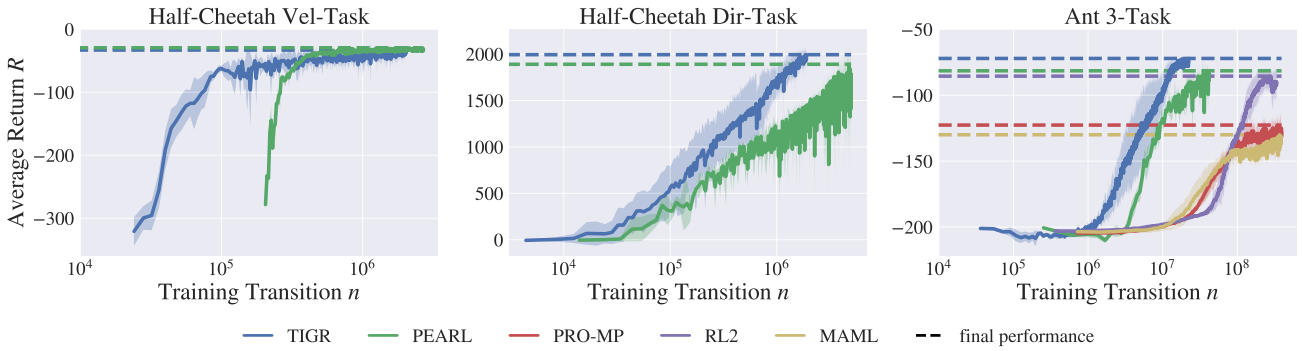
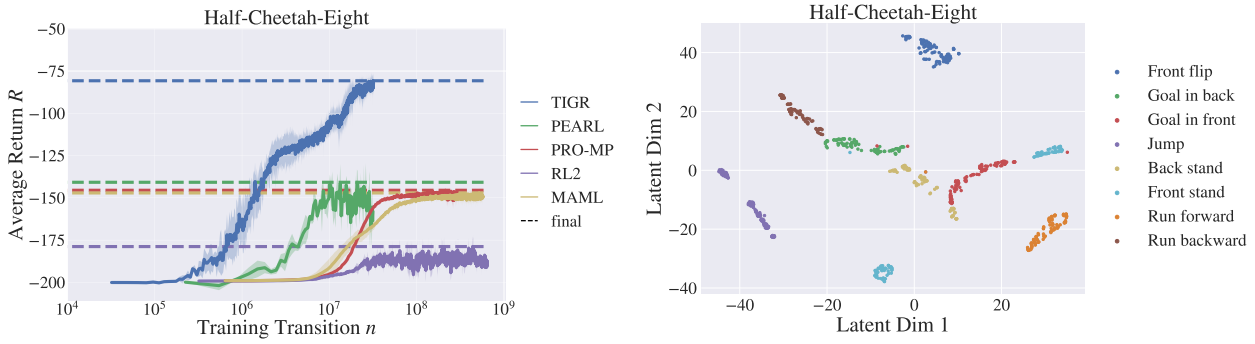


Fig. 6. Meta-testing performance over environment interactions evaluated periodically during the meta-training phase. As PEARL [8] outperforms ProMP [30], RL2 [5], and MAML [6] in *half-cheetah-vel* and *half-cheetah-dir*, we only compare TIGR with PEARL in these two environments. The blue line shows the performance of our method. We show the mean performance over three independent runs, evaluated in the *half-cheetah-velocity*, *half-cheetah-direction*, and *ant-three* environments. Note that the x-axis is in **log scale**.



(a) (b)

Fig. 7. (a) Meta-testing performance over environment interactions evaluated periodically during the meta-training phase. We show the mean performance from three independent runs. (b) Final encoding of the eight tasks visualized using t-SNE [31] in two dimensions. Note that the x-axis is in **log scale**.

We then progress to a slightly broader task distribution and evaluate the performance of PEARL, other state-of-the-art algorithms, and our method, on a modification of the *ant* environment (a detailed description is introduced in Appendix A). We use three different tasks, namely, goal tasks, velocity tasks, and a jumping task. The goal and velocity tasks are each split into left, right, up, and down and include different parametrizations. We take the original code and parameters provided from PEARL [8]. For a fair comparison, we adjust the dimensionality of the latent variable to be the same. For the other meta-RL algorithms, we use the code provided by Rothfuss et al. [30] of proximal metapolicy search (ProMP).[1] Fig. 6 (right) shows the performances of these five methods. We see that although PEARL and RL2 can solve the different tasks, our method greatly outperforms the others in sample efficiency and has a slight advantage in asymptotic performance.

Finally, we evaluate the approaches on the *half-cheetah-eight* benchmark. The average reward during meta-testing is shown in Fig. 7(a). We can see that TIGR outperforms prior methods in terms of sample efficiency and demonstrates superior asymptotic performance. Looking at the behaviors showcased in the video provided with the Supplementary Material, we find that with a final average return of -150, the PEARL agent is not able to distinguish the tasks. We observe a goal-directed behavior of the agent for the goal tasks but no generalization of the forward/backward movement to velocity tasks. It learns how to stand in the front but fails in *stand back*, *jump*, and *front flip*. For TIGR, we can see that every base task except the *front flip* is learned correctly. It should be noted that for the customized flip task, we expect the agent to flip at different angular velocities, which is much harder than the standard flip task, in which the rotation speed is simply maximized.

[1]Repository available at https://github.com/jonasrothfuss/ProMP/tree/full_code

B. Clustering Losses

We evaluate the impact of the clustering losses on the meta-testing performance of TIGR on the *half-cheetah-eight* benchmark in Fig. 8. We see that, when leaving out any one of the losses, the performance is weaker and less stable when one of the losses is omitted. Nevertheless, the algorithm significantly outperforms the prior meta-RL methods in both cases. Thus, each loss has a beneficial impact on the meta-RL objective, but the additional prior information about the true base tasks during meta-training is not mandatory for the algorithm to succeed in the nonparametric environment.

C. Latent Space Encoding

We evaluate the latent task representation by sampling transition histories from the replay buffer that belong to individual rollouts. We extract features from the context using the GRU encoder and obtain the compressed representation from the VAE. The latent task encoding is visualized in Fig. 7(b). We use t-distributed stochastic neighbor embedding (t-SNE) [31] to visualize the 8-D encoding in two dimensions. The representations are centered around 0, which demonstrates

Fig. 8. Evaluation of the impact of the proposed clustering losses on the meta-testing performance of the algorithm on the *half-cheetah-eight* benchmark. We remove each of the losses in turn and compare with the setup with all losses involved.

the information bottleneck imposed by the KL-divergence. We can see that qualitatively different tasks are clustered into different regions (e.g., *run forward*), verifying that the VAE is able to separate different base tasks from each other. Some base tasks show clear directions along which the representations are spread (e.g., *run backward*), suggesting how the parametric variations in each base task are encoded.

D. Task Inference

We evaluate whether the algorithm infers the correct task by examining the evolution of the velocity, distance, or angle during an episode. A task is correctly inferred when the current value approaches the target specification. We can see in Fig. 9 that the target specification is reached within different time spans for the distinct tasks. This is because goal-based tasks take longer to perform. Nevertheless, we can see that the task inference is successful, as the current value approaches the target specification in the displayed settings. In Fig. 9, the dashed line with a different color represents the goal of the task, and the solid line with the same color represents the performance of the agent. Taking the *run forward* task as an example, the goal velocities are 1–4 m/s, respectively.

For *jump*, the vertical velocity oscillates due to gravity and cannot be kept steady at the target. But, we can see that the peak of each color is close to the corresponding desired goal velocity. For the *run forward* and *run backward* task, the velocity slightly oscillates about the target with increasing strength for larger velocities. We suggest that this is because during running, the cheetah periodically accelerates when its feet touch the ground and shortly after decelerates due to air resistance. Since its feet need to be moved back to the starting position before the next acceleration, the velocity cannot be kept steady at its target but is prone to oscillate due to the physical aspects of the simulation. For the *goal in front*, *goal in back*, *front stand*, and *back stand* tasks, the agent is able to stop at the desired goal location. Due to the complexity of the dynamics, there are some tracking errors, but the order of the colors shows that the tasks are correctly inferred according to the specification of the goal. The *front flip* task remains unsolved. We have fully tested the *front flip* task and find out it is too difficult to be solved in a single task setting using SAC. Therefore, it is reasonable that it cannot be solved in a nonparametric task setting. However, as shown in Fig. 7(b), the *front flip* task can still be inferred as one base task.

E. Applicability to Nonstationary Environments

The zero-shot adaptation mechanism of the TIGR algorithm allows us to evaluate its performance for nonstationary task changes in the *half-cheetah-eight* benchmark, which is not possible for few-shot methods as PEARL [8]. First, we evaluate the adaptation to parametric task changes for the eight base tasks in the *half-cheetah-eight* benchmark. We use three consecutive parameterizations each, without resetting the state of the environment or agent in-between. We evaluate whether the algorithm infers the correct tasks by examining the evolution of the velocity, goal distance, or angle during the episode. The nonstationary adaptation to the tasks is successful when the current value repeatedly approaches the target specification. The results are shown in Fig. 10. Each subfigure depicts a different base task. We find that nonstationary adaptation to the different parameterizations is successful for each base task, as the inspected value repeatedly approaches the target specification the displayed settings, exhibiting similar task-inference patterns as described in Section VI-D. The vertical velocity for the *jump* task oscillates due to gravity, but the peak of each color shows a correct inference of the task. The *front flip* task remains unsolved, and the potential reason is given as above.

Second, we evaluate the adaptation to nonparametric task changes in the *half-cheetah-eight* benchmark. We set a fixed order for the base tasks, as visualized in Fig. 11, and iterate through them online after executing 80 steps for each environment, without resetting its state in-between. We evaluate whether the algorithm infers the correct tasks by examining the evolution of the velocity, goal distance, or angle during the entire episode. The nonstationary adaptation to the tasks is successful when the current value repeatedly approaches the target specification. The results are shown in Fig. 11. Each subfigure describes the evolution of the values for the given base task and its specified target parameterization with successive timesteps across all environments. We find that nonstationary adaptation to the tasks is successful, as the inspected value repeatedly approaches the target specification in all of the displayed settings.

F. Applicability to Continual Learning

We evaluate the applicability of the TIGR algorithm to two different continual learning settings derived from the *half-cheetah-eight* benchmark: 1) the agent starts with access to only one nonparametric task, and the number of accessible tasks increases during the training process ("linear" setting) and 2) the agent starts with access to only one nonparametric task, and with each new task, access to the previous task is removed, but the experience gained with the previous tasks can still be used ("cut" setting). The total amount of environment interactions gathered in training is the same for all the displayed settings. We set the number of Gaussians in the VAE equal to the total number of nonparametric tasks presented during the learning process. We use the following order of tasks: *run forward*, *run backward*, *reach front goal*, *reach back goal*, *front stand*, *back stand*, *jump*, and *front flip*. The results are shown in Fig. 12. Each subfigure represents

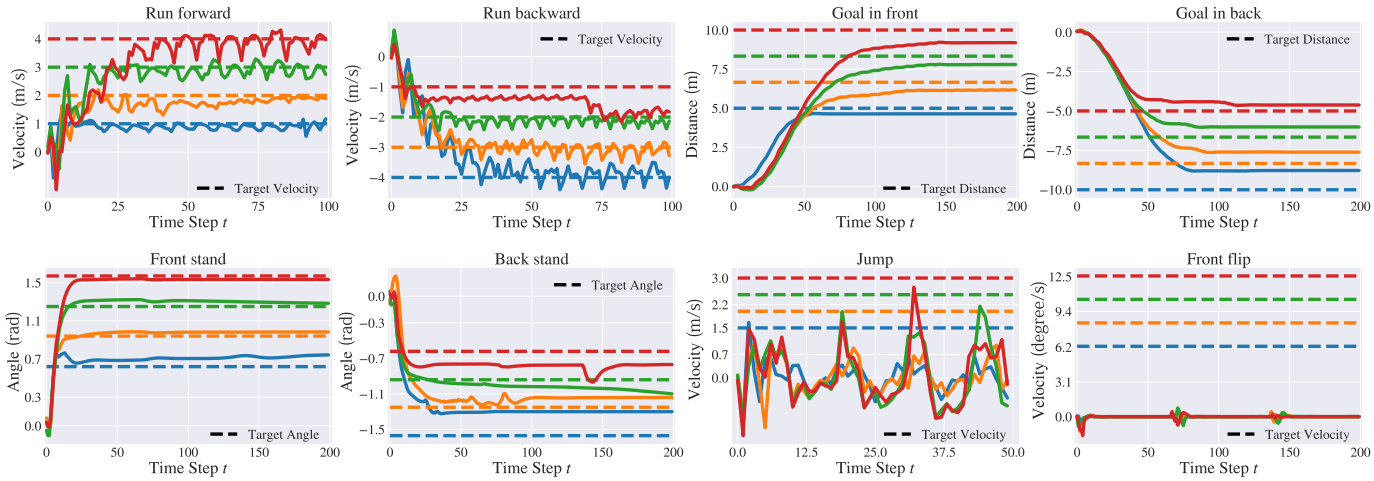


Fig. 9. Task-inference response during one episode for the *half-cheetah-eight* benchmark after 2000 training epochs. Each task is evaluated under different parametric variations, which are each represented using the different colors. The target for each task variation is marked with the dashed line. The task is inferred correctly when the solid line approaches the target. The vertical velocity for the *jump* task oscillates due to gravity. For the *run forward* and *run backward* task, the velocity slightly oscillates about the target with increasing strength for larger velocities due to the physical aspects of the simulation. The *front flip* task remains unsolved. In each plot, different colors represent tasks with different targets. The targets are depicted with dashed lines. Taking the *run forward* as an example, the goal velocities are 1–4 m/s, respectively.

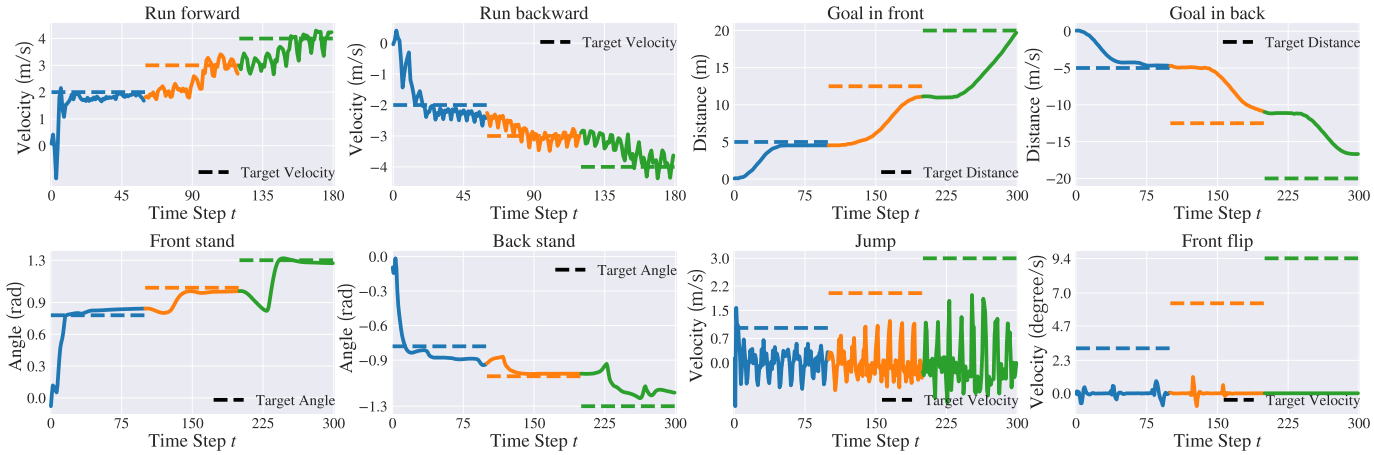


Fig. 10. Evaluation of the performance of the TIGR algorithm for parametric nonstationary task changes for each base task in the *half-cheetah-eight* benchmark. The targets (specifications) for each of the environments in the subfigures are marked as the dashed line. The nonstationary adaptation to the tasks is successful when the solid line for the current value repeatedly approaches the targets. The vertical velocity for the *jump* task oscillates due to gravity. The *front flip* task remains unsolved. In each plot, different colors represent tasks with different targets. The targets are depicted with dashed lines.

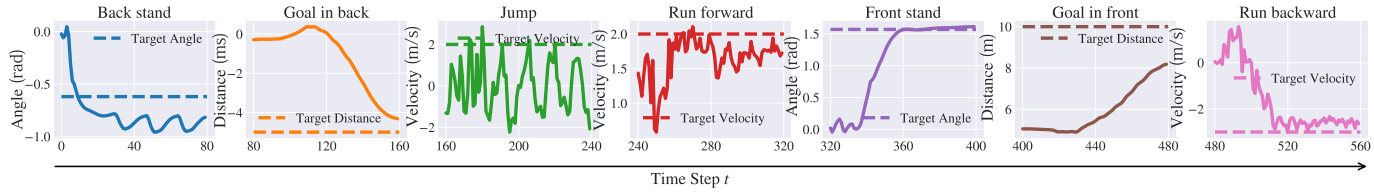


Fig. 11. Evaluation of the performance of the TIGR algorithm for nonstationary and nonparametric task changes in the *half-cheetah-eight* benchmark. The target (specification) for each of the environments in the subfigures is marked as the dashed line. The nonstationary adaptation to the tasks is successful when the solid line for the current value repeatedly approaches the target. In each plot, different colors represent tasks with different targets. The targets are depicted with dashed lines.

a different base task. After each 12.5% of training progress, a new base task is provided in the given order. We note that the agent does not perform as well in the continual learning setting as in the noncontinual setting, because the reduced training time for difficult tasks that are not accessed until the later stages of training, such as the *back stand* or *jump* tasks, does not give the algorithm enough time to fully learn the desired behavior. Nevertheless, the algorithm does not exhibit forgetting or overfitting on old tasks, thus fulfilling these critical abilities necessary for continual learning.

VII. Discussion and Ablation Study

We perform an ablation study of different configurations of our method on the environments that we provide. We first compare the sample efficiency and asymptotic performance for the different clustering losses. Second, we evaluate the performances of the three implemented feature-extraction configurations. Third, we provide an ablation study of TIGR on solving out-of-distribution (OOD) tasks. Finally, we discuss the performance and limitations of the TIGR algorithm.

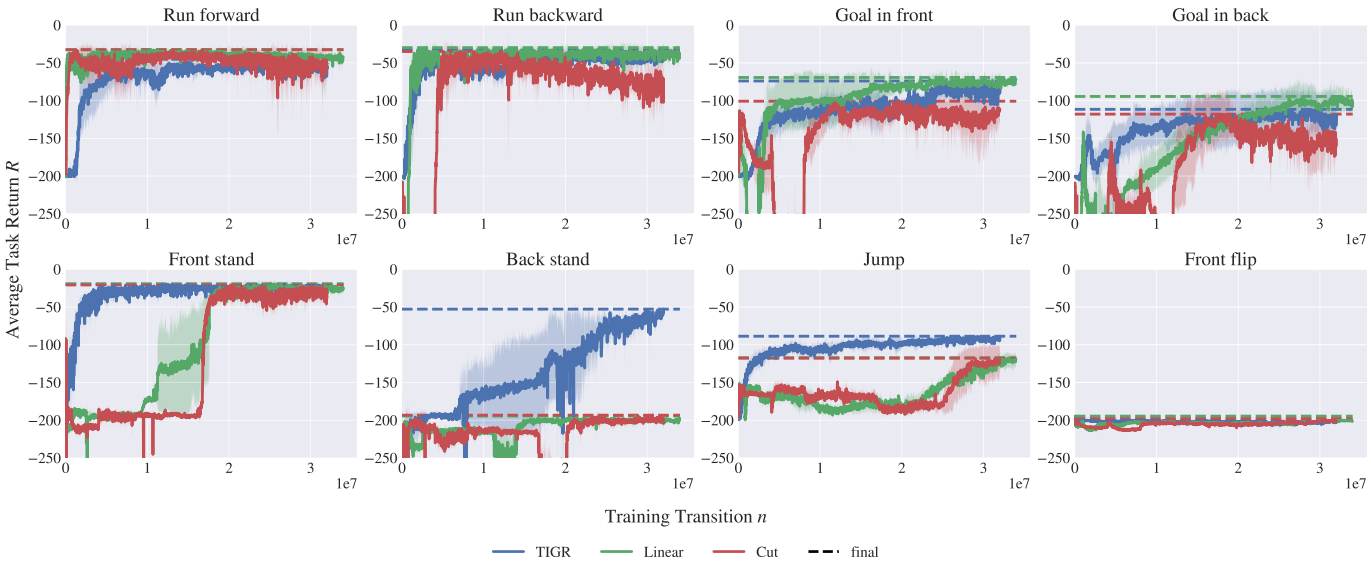


Fig. 12. Evaluation of the performance of the TIGR algorithm for the two continual learning settings in the *half-cheetah-eight* benchmark. The noncontinual learning curve for TIGR is shown for comparison in blue. The continual learning is successful when the curve for the "linear" (green) or "cut" (red) setting approaches the TIGR curve.

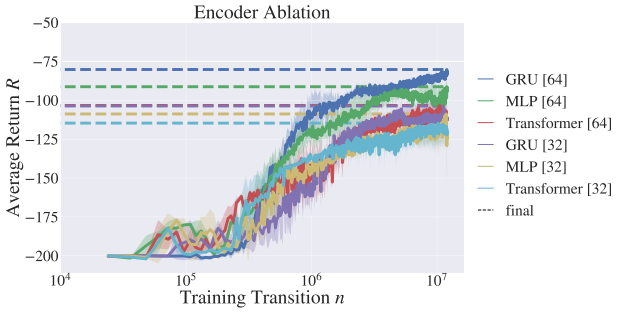


Fig. 13. Evaluation of meta-testing performance on the *half-cheetah-six environment* of different feature-extraction configurations in addition to different context lengths in the brackets used to infer the task.

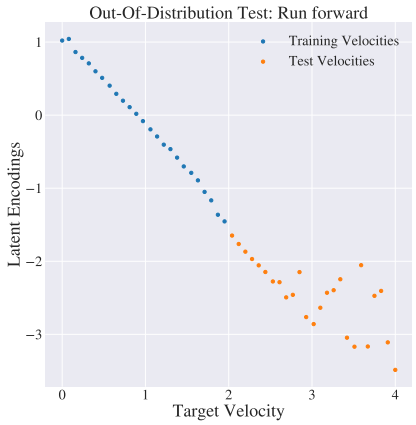


Fig. 14. Out-of distribution experiment with velocities between 0 and 2 during meta-training, and velocities greater than 3 and up to 4 during meta-testing.

A. Feature-Extraction Ablation

To study the effect of different VAE architectures, we have performed the following ablation study. We evaluate the three feature-extraction architectures GRU, MLP, and Transformer on the *half-cheetah-six* environment, i.e., omitting the *jump* and *front flip* tasks. We compare the three methods using 32 and 64 timesteps in the context. The results are shown in Fig. 13. We can see that all methods show improved performance when using more timesteps in the context. As the GRU outperforms the other methods when 64 timesteps are used, we use this architecture in our study.

OOD Tasks: We also examine TIGR's capability on solving OOD tasks. For the *half-cheetah-vel* task, the goal velocities are set as 0–2 during meta-training, but is challenged with 3–4 during meta-testing, which is out of the distribution of the training dataset. We visualize the latent space structure of run forward task in Fig. 14 to demonstrate our assumption; here, we reduce the dimension of latent representations to 1-D using the t-SNE visualization to intuitively illustrate the relationship between representations and the ground-truth goal value. We can observe that the latent variable shows a linear relationship against the target velocity. Blue points indicate the training velocities, and the orange points indicate the testing velocities. Our algorithm is able to accurately adapt to the goal velocities between 2 and 3 and roughly follow the same trend for velocities between 3 and 4. The characteristics of the OOD test tasks are well reflected in the latent space even though the task inference module is not trained with a goal velocity between 2 and 4. The result supports our claim that our task inference module can generalize to unseen test tasks without additional training and, thus, benefits the downstream policy training.

B. Limitations

The results of our experiments demonstrate that TIGR is applicable to broad and nonparametric environments with zero-shot adaptation to nonstationary task changes, where prior methods even fail with few-shot adaptation. However, our method is not applicable to sparse reward settings, since it assumes that the environment gives a feedback to the agent following a dense reward function. In general, this drawback can presumably lead to weaker performance for problems that

TABLE I

NONPARAMETRIC VARIABILITY PROPOSED FOR THE *Half-Cheetah-Eight* ENVIRONMENT

Behaviour	**Task Properties**				**Objective**
Run forward	Horizontal velocity	$1 \leq$	v_x^*	≤ 5	$r = -\lvert v_x^* - v_x \rvert$
Run backward	Horizontal velocity	$-5 \leq$	v_x^*	≤ -1	
Reach goal in front	Horizontal position	$5 \leq$	p_x^*	≤ 25	$r = -\lvert p_x^* - p_x \rvert$
Reach goal in back	Horizontal position	$-25 \leq$	p_x^*	≤ -5	
Front stand	Angular position	$\frac{\pi}{6} \leq$	p_y^*	$\leq \frac{\pi}{2}$	$r = -\lvert p_y^* - p_y \rvert$
Back stand	Angular position	$-\frac{\pi}{2} \leq$	p_y^*	$\leq -\frac{\pi}{6}$	
Front flip	Angular velocity	$2\pi \leq$	v_y^*	$\leq 4\pi$	$r = -\lvert v_y^* - v_y \rvert$
Jump	Vertical velocity	$1.5 \leq$	v_z^*	≤ 3.0	$r = -\lvert v_z^* - \lvert v_z \rvert \rvert$

TABLE II

NONPARAMETRIC VARIABILITY PROPOSED FOR THE *Ant* ENVIRONMENT

Behaviour	**Task Properties**				**Objective**
Run	Velocity *up*, *down*, *left*, *right*	$1 \leq$	$v_{x/y}^*$	≤ 3	$r = -\lvert v_{x/y}^* - v_{x/y} \rvert$
Reach goal	Position *up*, *down*, *left*, *right*	$5 \leq$	$p_{x/y}^*$	≤ 15	$r = -\lvert p_{x/y}^* - p_{x/y} \rvert$
Jump	Velocity	$0.5 \leq$	v_z^*	≤ 2	$r = -\lvert v_z^* - v_z \rvert$

TABLE III

GENERAL HYPERPARAMETERS

Hyperparameter	**Value**
Optimizer	ADAM
Learning rate encoder, decoder, SAC	3e-4
Discount factor γ	0.99
Entropy target $\mathcal{H}$	$-\dim(\mathcal{A})$
SAC network size	3×300 units
Net complex c_n	5
GRU input dim	$\dim(\mathcal{S}) + \dim(\mathcal{A}) + 1 + \dim(\mathcal{S}')$
GRU hidden layer size	$c_n \times$ GRU input dim
VAE network size	2 layers [GRU hidden layer size & Num Classes $\times$ (2 $\times$ Latent Dim + 1)]
Dynamics network size	2 layers [$c_n \times (\dim(\mathcal{S}) + \dim(\mathcal{A}) + \dim(\mathcal{Z}))$]
Reward network size	3 layers [$c_n \times (\dim(\mathcal{S}) + \dim(\mathcal{A}) + \dim(\mathcal{Z}))$]
Non-linearity (all networks)	ReLU
SAC target smoothing coefficient	0.005
Evaluation trajectories per task per epoch	1
Task inference training steps per epoch	128
Task inference training batch size	4096
Policy training steps per epoch	2048
Policy training batch size	256
Train-validation split encoder training	0.8 / 0.2
Loss weights:	
- $\alpha_{\text{KL-divergence}}$	0.001
- β_{euclid}	5e-4
- $\gamma_{\text{classification}}$	0.1

do not follow well-shaped reward functions, and we suppose that the *front flip* task might also not be solved due to ill-defined rewards.

VIII. CONCLUSION

In this article, we presented TIGR, an efficient meta-RL algorithm for solving nonparametric and nonstationary task distributions. Using our task representation learning strategy, TIGR is able to learn behaviors in nonparametric environments using zero-shot adaptation to nonstationary task changes. Our encoder is based on a generative model represented as a VAE and trained by unsupervised MDP reconstruction. This makes it possible to capture the multimodality of the nonparametric task distributions. We report three to ten times better sample efficiency and superior performance compared with prior methods on a series of environments, including the novel nonparametric *half-cheetah-eight* benchmark.

APPENDIX A
EXPERIMENT ENVIRONMENTS

The OpenAI Gym toolkit [32] provides many environments for RL setups that can be easily modified to meet our desired properties.

A. Half-Cheetah-Eight

An environment that is often used in meta-RL is the *half-cheetah* [6], [11], [20], [33], and therefore, we have chosen it to demonstrate the performance of our proposed approach. We provide a new benchmark consisting of eight nonparametric tasks requiring qualitatively distinct behavior, as defined in Table I and visualized in Fig. 1. Each environment contains internal parametric variability, in which the desired velocity or goal is sampled from a range of possible values. Each task was verified individually to show that the correct behavior is

TABLE IV
Half-Cheetah-Eight HYPERPARAMETERS

Hyperparameter	Value
Training tasks	80
Test tasks	40
Maximal trajectory length	200
Training epochs	2000
Data collection: initial samples per task	200
Data collection: training tasks for sampling per epoch	80
Data collection: samples per task per epoch	200
Encoder context timesteps	64
Encoder latent dimension dim(z)	8
Encoder VAE components	8

learned when a high return is achieved by the algorithm. The environments are pseudo-normalized, such that the maximum possible reward is 0 (i.e., when there is no deviation from the desired velocity/position), and the agent starts with a reward of -1 on each episode. We suggest that this is a very important feature of the environments, since the agent cannot distinguish tasks based on the magnitude of the reward alone. We assume that this increases the difficulty of the challenge, as some kind of exploratory movement is required at the beginning of each episode to deduce what behavior is needed.

B. Ant-Three

The second environment that is often used in meta-RL to demonstrate the generalization ability to multiple agents is the *ant* [8]. We modify the standard *ant* environment and introduce three base tasks, such as *run*, *reach goal*, and *jump*. *Run* and *reach goal* are divided into the four directions, such as *up*, *down*, *left*, and *right*, but are considered as a single base task due to the ant's symmetricity. Task specifications are defined in Table II.

APPENDIX B
EVALUATION DETAILS

We carried out the experiments on a 32-core machine with 252 GB of RAM and Eight Tesla V100 GPUs. We implemented TIGR in PyTorch (version 1.7.0) and ran it on Ubuntu 18.04 with Python 3.7.7. The implementation of TIGR is based on the PEARL implementation given by Rakelly et al. [8].

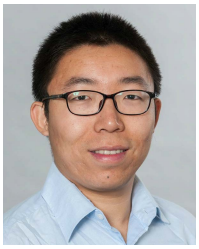
1) All curves in this work are plotted from three runs with random task initializations and seeds.
2) Shaded regions indicate one standard deviation around the mean.

We give an overview of important hyperparameters of the method and the values we used during our experiments in Table III. The settings for the *half-cheetah-eight* environment can be seen in Table IV. Detailed code can be found in the Supplementary Materials.

REFERENCES

[1] D. Silver et al., "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
[2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.
[3] T. Haarnoja, A. Zhou, S. Ha, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," 2019, *arXiv:1812.11103*.
[4] I. Akkaya et al., "Solving Rubik's cube with a robot hand," 2019, *arXiv:1910.07113*.
[5] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "RL^2: Fast reinforcement learning via slow reinforcement learning," 2016, *arXiv:1611.02779*.
[6] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, Aug. 2017, pp. 1126–1135.
[7] J. X. Wang et al., "Learning to reinforcement learn," 2016, *arXiv:1611.05763*.
[8] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen, "Efficient off-policy meta-reinforcement learning via probabilistic context variables," in *Proc. 36th Int. Conf. Mach. Learn.*, vol. 97, K. Chaudhuri and R. Salakhutdinov, Eds., Jun. 2019, pp. 5331–5340.
[9] H. Wang, J. Zhou, and X. He, "Learning context-aware task reasoning for efficient meta-reinforcement learning," in *Proc. AAMAS*, 2020, pp. 1–10.
[10] J. Humplik, A. Galashov, L. Hasenclever, P. A. Ortega, Y. W. Teh, and N. Heess, "Meta reinforcement learning as task inference," 2019, *arXiv:1905.06424*.
[11] A. Nagabandi, C. Finn, and S. Levine, "Deep online learning via meta-learning: Continual adaptation for model-based RL," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–15.
[12] L. Lan, Z. Li, X. Guan, and P. Wang, "Meta reinforcement learning with task embedding and shared policy," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, Aug. 2019, pp. 2794–2800.
[13] H. Ren, A. Garg, and A. Anandkumar, "Context-based meta-reinforcement learning with structured latent space," in *Proc. Skills Workshop NeurIPS*, 2019, p. 5.
[14] T. Yu et al., "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning," 2019, *arXiv:1910.10897*.
[15] I. Clavera et al., "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–17.
[16] D. Rao, F. Visin, A. A. Rusu, Y. Teh, R. Pascanu, and R. Hadsell, "Continual unsupervised representation learning," in *Proc. NeurIPS*, 2019, pp. 1–11.
[17] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *Proc. ICML*, vol. 80, Jul. 2018, pp. 1861–1870.
[18] M. Al-Shedivat, T. Bansal, Y. Burda, I. Sutskever, I. Mordatch, and P. Abbeel, "Continuous adaptation via meta-learning in nonstationary and competitive environments," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–21.
[19] A. Nagabandi et al., "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," 2018, *arXiv:1803.11347*.
[20] V. Mnih et al., "Asynchronous methods for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn.*, vol. 48, M. F. Balcan and K. Q. Weinberger, Eds., New York, NY, USA, Jun. 2016, pp. 1928–1937.
[21] L. Zintgraf et al., "VariBAD: A very good method for Bayes-adaptive deep RL via meta-learning," in *Proc. ICLR*, 2020, pp. 1–20.
[22] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine, "Meta-reinforcement learning of structured exploration strategies," in *Proc. 32nd Int. Conf. Neural Inf. Process. Syst.* Red Hook, NY, USA: Curran Associates Inc., 2018, pp. 5307–5316.
[23] H. Hu, G. Huang, X. Li, and S. Song, "Meta-reinforcement learning with dynamic adaptiveness distillation," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 3, pp. 1454–1464, Mar. 2021.
[24] P. Jiang, S. Song, and G. Huang, "Exploration with task information for meta reinforcement learning," *IEEE Trans. Neural Netw. Learn. Syst.*, early access, Nov. 5, 2021, doi: 10.1109/TNNLS.2021.3121432.
[25] P. Jiang, S. Song, and G. Huang, "Attention-based meta-reinforcement learning for tracking control of AUV with time-varying dynamics," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 11, pp. 6388–6401, Nov. 2022.
[26] L. Chen, B. Hu, Z.-H. Guan, L. Zhao, and X. Shen, "Multiagent meta-reinforcement learning for adaptive multipath routing optimization," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 10, pp. 5374–5386, Oct. 2022.
[27] A. Vaswani et al., "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds. Red Hook, NY, USA: Curran Associates, 2017.

[28] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," 2013, *arXiv:1312.6114*.
[29] N. Dilokthanakul et al., "Deep unsupervised clustering with Gaussian mixture variational autoencoders," 2016, *arXiv:1611.02648*.
[30] J. Rothfuss, D. Lee, I. Clavera, T. Asfour, and P. Abbeel, "ProMP: Proximal meta-policy search," in *Proc. Int. Conf. Learn. Represent.*, 2019, pp. 1–10.
[31] L. Van Der Maaten and G. Hinton, "Visualizing data using t-SNE.," *J. Mach. Learn. Res.*, vol. 9, no. 11, pp. 1–27, 2008.
[32] G. Brockman et al., "OpenAI Gym," 2016, *arXiv:1606.01540*.
[33] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, "A simple neural attentive meta-learner," in *Proc. Int. Conf. Learn. Represent.*, 2018, pp. 1–17.



Zhenshan Bing (Member, IEEE) received the B.S. degree in mechanical design manufacturing and automation and the M.Eng. degree in mechanical engineering from the Harbin Institute of Technology, Harbin, China, in 2013 and 2015, respectively, and the Doctorate degree in computer science from the Technical University of Munich, Munich, Germany, in 2019.

He is currently a Post-Doctoral Researcher with Informatics 6, Technical University of Munich. His research interests include bio-robots, which are controlled by artificial neural networks and related applications.

Lukas Knak received the B.Sc. degree in cognitive sciences from the University of Tübingen, Tübingen, Germany, in 2018, and the M.Sc. degree in robotics, cognition, intelligence from the Technical University of Munich, Munich, Germany, in 2021.

His research interests include various applications of artificial intelligence in robotics.

Long Cheng received the B.S. degree in mechanical design manufacturing and automation and the M.Eng. degree in mechanical engineering from the Harbin Institute of Technology, Harbin, China, in 2011 and 2013, respectively, and the Doctorate degree in computer science from the Technical University of Munich, Munich, Germany, in 2018.

In 2021, he joined Wenzhou University, Wenzhou, China, as an Assistant Professor. His research interests include snake-like robots, artificial neural networks, and real-time systems.

Fabrice O. Morin received the Engineering degree from the Ecole Nationale Supérieure des Mines de Nancy, Nancy, France, in 1999, the master's degree in bioengineering from the University of Strathclyde, Glasgow, U.K., in 2000, and the Ph.D. degree in materials science from the Japanese Advanced Institute of Science and Technology, Nomi-Shi, Japan, in 2004.

After several post-docs at the University of Tokyo, Tokyo, Japan, and the Integration: from Material to Systems (IMS) Laboratory, University of Bordeaux, Nouvelle-Aquitaine, France, in 2008, he joined Tecnalia, San Sebastián, Spain, a nonprofit Research and Technology Organization (RTO), first as a Senior Researcher and then as a Group Leader. There, he worked on various projects in neurotechnology and biomaterials. Since 2017, he has been working as a Scientific Coordinator with the Technical University of Munich, Munich, Germany, where, in the framework of the Human Brain Project, he oversees the development of software tools for embodied simulation applied to neuroscience and artificial intelligence.

Kai Huang (Member, IEEE) received the B.Sc. degree from Fudan University, Shanghai, China, in 1999, the M.Sc. degree from the University of Leiden, Leiden, The Netherlands, in 2005, and the Ph.D. degree from ETH Zürich, Zürich, Switzerland, in 2010.

He was a Research Group Leader with fortiss GmbH, Munich, Germany, in 2011. He was a Senior Researcher with the Computer Science Department, Technical University of Munich, Munich, from 2012 to 2015. In 2015, he joined Sun Yat-sen University, Guangzhou, China, as a Professor. He was appointed as the Director of the Institute of Unmanned Systems of School of Data and Computer Science, Guangzhou, in 2016. His research interests include techniques for the analysis, design, and optimization of embedded systems, particularly in the automotive and robotic domains.

Dr. Huang has been serving as a member for the Technical Committee on Cybernetics for Cyber-Physical Systems of the IEEE SMC Society since 2015. He was awarded the Program of Chinese Global Youth Experts 2014 and was granted the Chinese Government Award for Outstanding Self-Financed Students Abroad 2010. He was a recipient of the Best Paper Awards National Conference on Embedded Systems Technology (ESTC) 2017, IEEE Workshop on Embedded Systems for Real-Time Multimedia (ESTIMedia) 2013, SAMOS 2009, Best Paper Candidate ROBIO 2017, ESTMedia 2009, and General Chairs' Recognition Award for Interactive Papers in IEEE Conference on Decision and Control (CDC) 2009.

Alois Knoll (Fellow, IEEE) received the M.Sc. degree in electrical/communications engineering from the University of Stuttgart, Stuttgart, Germany, in 1985, and the Ph.D. degree (summa cum laude) in computer science from the Technical University of Berlin (TU Berlin), Berlin, Germany, in 1988.

He served on the faculty of the Computer Science Department, TU Berlin, until 1993. He joined the University of Bielefeld, Bielefeld, Germany, as a Full Professor, where he served as the Director of the Technical Informatics Research Group until 2001. Since 2001, he has been a Professor with the Department of Informatics, Technical University of Munich (TUM), Munich, Germany. His research interests include cognitive, medical, and sensor-based robotics; multiagent systems; data fusion; adaptive systems; multimedia information retrieval; model-driven development of embedded systems with applications to automotive software and electric transportation; and simulation systems for robotics and traffic.

Dr. Knoll was a member of the EU's highest advisory board on information technology, the Information Society Technology Advisory Group (ISTAG), from 2007 to 2009, and its subgroup on Future and Emerging Technologies (FETs). In this capacity, he was actively involved in developing the concept of the European Unions (EU's) FET flagship projects.