Learning Expressive Meta-Representations with Mixture of Expert Neural Processes

Anonymous Author(s)
Affiliation
Address
email

Abstract

Neural processes (NPs) formulate exchangeable stochastic processes and are promising models for meta learning that do not require gradient updates during the testing phase. However, most NP variants place a strong emphasis on a global latent variable. This weakens the approximation power and restricts the scope of applications using NP variants, especially when data generative processes are complicated. To resolve these issues, we propose to combine the **M**ixture **o**f **E**xpert models with **N**eural **P**rocesse**s** to develop more expressive exchangeable stochastic processes, referred to as Mixture of Expert Neural Processes (MoE-NPs). Then we apply MoE-NPs to both few-shot supervised learning and meta reinforcement learning tasks. Empirical results demonstrate MoE-NPs' strong generalization capability to unseen tasks in these benchmarks.

1 Introduction

Humans can naturally accommodate themselves to new environments after developing related skills, and this kind of adaptability relies on the good abstraction of environments. Similarly, *meta learning* or *learning to learn* tries to leverage past experiences, and with help of the incorporated meta learned knowledge, a new skill can be mastered rapidly with a few instances.

During the past decade, an increasing number of methods have emerged in meta learning domains. In this paper, we concentrate on a special branch of meta learning methods, referred to as context-based meta learning [1; 2]. A representative one is a neural process (NP) [1], which was initially proposed to approximate Gaussian processes with lower computational cost. The core purpose of NPs is to learn meta-representations [3], which encode context points into latent variables and represent the task in a functional form. In comparison to gradient-based meta learning algorithms, *e.g.* model-agnostic meta learning (MAML) [4], the NP directly learns a functional representation and does not require additional gradient updates in fast adaptation.

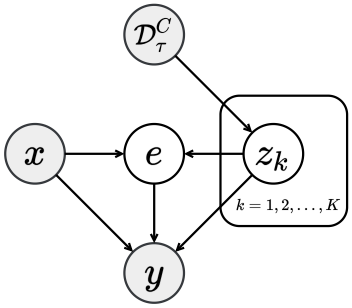

Figure 1: Generative Process of MoE-NPs. Here $\mathcal{D}_\tau^C$ refers to dataset of context points in the paper. $\{z_k\}_{k=1}^K$ are a set of expert latent variables and e is an assignment latent variable. Observed variables are grey in circles with latent variables white.

Research Motivations. Fundamentally, vanilla NPs employ global Gaussian latent variables to specify different tasks. This setting raises several concerns in some scenarios. (i) When observations originate from a mixture of stochastic processes [5], a single Gaussian latent variable is faced with deficiencies in formulating complex functional forms. (ii) In context-based meta reinforcement learning, the uncertainty of value functions revealed from latent variables encourages effective exploration in environments [6]. But when tasks are governed by

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

multiple variate, *e.g.* velocities, masses or goals in Mujoco robots [7], the use of a unimodal Gaussian latent variable restricts the flexibility of randomized value functions [8], leading to sub-optimality in performance.

Developed Methods. Instead of using a global latent variable in modeling, we employ multiple latent variables to induce a mixture of expert NPs to specify diverse functional priors. Meanwhile, the discrete latent variables as assignment variables are introduced to establish connections between a single data point and expert NPs. To optimize this model with hybrid types of latent variables, we utilize variational inference to formulate the evidence lower bound. Additionally, special modules are designed to accommodate few-shot supervised learning and meta reinforcement learning tasks.

Outline & Contributions. We overview the properties of NPs and general notations for context-based meta learning tasks in Section 3. Section 2 summarizes related work in meta learning and the NP family. In Section 4, **M**ixture **o**f **E**xpert **N**eural **P**rocesse**s** (MoE-NPs) are elaborated, together with required modules for meta learning tasks. Experimental results and analysis are reported in Section 5. Discussions and limitations are included in Conclusion Section. In principle, our contribution is two-fold:

- We introduce a new exchangeable stochastic process, referred to as MoE-NPs, to enrich the family of NPs. Our model inherits the advantages of both mixtures of experts models and NPs, enabling multi-modal meta representations for functions.
- We specify inference modules in MoE-NPs for few shot supervised learning and meta reinforcement learning tasks. Extensive meta learning experiments show that MoE-NPs can achieve competitive performance in comparison to most existing methods.

2 Literature Review

Meta Learning. Meta learning is a paradigm to enable fast learning (fast adaptation to new tasks) via slow learning (meta training in the distribution of tasks). There exist several branches of meta learning algorithms. Gradient-based meta learning algorithms, *e.g.* MAML [4] and its variants [9; 10; 11], perform gradient updates over model parameters to achieve fast adaptation with a few instances. Metrics-based meta learning algorithms try to learn representations of tasks in distance space, and models *e.g.* prototypical networks [12; 13] are popular in computer vision domains. As for context-based meta learning methods of our interest, latent variable models, *e.g.* NPs [1], are designed to learn task representations in a functional space. This family does not require gradient updates in fast adaptation.

Neural Processes Family. Apart from vanilla NPs or CNPs [2; 1], other variants are developed and these are built on various inductive biases. To address underfitting issues, attention networks [14; 15] are introduced to augment NPs. [16; 17] improve the generalization capability of NPs with help of convolutional operations. To learn distributions of group equivariant functions, [18] has proposed EquiCNP. Similarly, SteerCNPs also incorporate equivariance to approximate stochastic fields [19]. Our work is to get NPs married with Mixture of Experts (MoEs) models [20; 21], which model datasets with a collection of expert NPs. We provide more technical summary of the NP family together with other probabilistic meta learning methods [22; 23; 24; 25] in the Appendix (F).

3 Preliminaries

Notations. The paradigm of *meta learning* is considered in the distribution of tasks $p(\mathcal{T})$, and a task sampled from $p(\mathcal{T})$ is denoted by τ in this paper. The form of a task depends on applications. For example, a task of our interest in regressions can be a function f to fit, which is a realization from unknown stochastic processes [26].

Let $\mathcal{D}_\tau^C$ refer to a set of context points used to specify the underlying task, and $\mathcal{D}_\tau^T = [x_T, y_T]$ is a set of target points to predict, *e.g.* $f(x_T) = y_T$. In the context-based meta learning with latent variables, we write the probabilistic dependencies in distributions of target functions as follows,

$$p(f(x_T)|\mathcal{D}_\tau^C, x_T) = \int p(f(x_T)|z, x_T)p(z|\mathcal{D}_\tau^C)dz \tag{1}$$

where the functional prior $p(z|\mathcal{D}_\tau^C)$ is injected in modeling via latent variables z.

Neural Processes. The family of NPs [1] belongs to exchangeable stochastic processes [27]. A generative process is written as Eq. (2) with a global Gaussian latent variable z placed in Eq. (1),

$$\rho_{x_{1:N}}(y_{1:N}) = \int p(z) \prod_{i=1}^{N} \mathcal{N}(y_i | f_\theta(x_i, z), \sigma_i^2) dz \tag{2}$$

where f_θ is a mean function and σ_i^2 is the corresponding variance. In our settings, we treat the conditional neural process [2] as a special case in NPs, when the distribution of z is collapsed into a Dirac delta distribution $p(z) = \delta(z - \hat{z})$ with $\hat{z}$ a fixed real valued vector.

3.1 Few-Shot Supervised Learning

In the context-based meta learning, we formulate the few-shot supervised learning objective within the expected risk minimization principle as follows.

$$\min_{\Theta} \mathbb{E}_{\tau \sim p(\mathcal{T})} \left[\mathcal{L}(\mathcal{D}_\tau^T; \mathcal{D}_\tau^C, \Theta) \right] \tag{3}$$

The risk function $\mathcal{L}$, *e.g.* negative log-likelihoods, measures performance of meta learning structure on the task-specific dataset $\mathcal{D}_\tau^C$ and $\mathcal{D}_\tau^T$, and Θ means parameters of common knowledge shared across tasks and parameters for fast adaptation (*e.g*, Θ denotes parameters of the encoder ϕ and decoder θ, and $\mathcal{L}$ is the approximate objective in NPs).

With the set of context points $\mathcal{D}_\tau^C = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ and the target points $\mathcal{D}_\tau^T$, the posterior of a global latent variable z in Eq. (1) is approximated with a variational distribution $q_\phi(z|\mathcal{D}_\tau^T)$ and an evidence lower bound (ELBO) is derived to optimize in practice. A general meta training objective of NPs in few-shot supervised learning is $\mathcal{L}(\theta, \phi)$ in Eq. (4), where $p_\theta(\mathcal{D}_\tau^T|z) = \prod_{i=1}^{n} p_\theta(y_i|[x_i, z])$.

$$\mathbb{E}_\tau \left[\ln p(\mathcal{D}_\tau^T | \mathcal{D}_\tau^C) \right] \geq \mathbb{E}_\tau \left[\mathbb{E}_{q_\phi(z|\mathcal{D}_\tau^T)} [\ln p_\theta(\mathcal{D}_\tau^T|z)] - D_{KL}[q_\phi(z|\mathcal{D}_\tau^T) \parallel q_\phi(z|\mathcal{D}_\tau^C)] \right] \tag{4}$$

3.2 Meta Reinforcement Learning

For the context-based meta reinforcement learning, the context points $\mathcal{D}_\tau^C$ are a set of random transition samples from an environment as $\mathcal{D}_\tau^C = \{(s_1, a_1, s_2, r_1), \ldots, (s_H, a_H, s_{H+1}, r_H)\}$, where r_t is the one-step reward after performing action a_t at state s_t. Here $\mathcal{D}_\tau^C$ plays a role in task inference [28] to obtain the information bottleneck $q_\phi(z|\mathcal{D}_\tau^C)$ and $\mathcal{D}_\tau^T$ is the dataset of state action values to fit.

For example, in an off-policy meta reinforcement learning algorithm, *e.g.* PEARL [6] or FCRL [3], the general optimization objective consists of two parts: (i) to approximate distributions of task-specific optimal value functions in Eq. (5), where Q_θ is optimal Q-value with the state value $\hat{V}$ (ii) to maximize the cumulative rewards $\mathbb{E}_\tau \left[\mathbb{E}_{q_\phi(z|\mathcal{D}_\tau^C)} \left[\mathcal{R}(\tau, z; \varphi) \right] \right]$, where $\mathcal{R}$ is the expected cumulative rewards in the environment τ given policies $\pi_\varphi(a|[s, z])$.

$$\mathcal{L}(\theta, \phi) = \mathbb{E}_\tau \mathbb{E}_{\substack{(s,a,s',r) \sim \mathcal{D}_\tau^T \\ z \sim q_\phi(z|\mathcal{D}_\tau^C)}} \left[Q_\theta([s, z], a) - (r + \hat{V}([s', z])) \right]^2 + \beta \mathbb{E}_\tau \left[D_{KL}[q_\phi(z|\mathcal{D}_\tau^C) \parallel p(z)] \right] \tag{5}$$

Different from the few-shot supervised learning, here the context points are not part of fitting dataset, which means $\mathcal{D}_\tau^C \not\subset \mathcal{D}_\tau^T$. As implemented in [6], the prior distribution $p(z)$ is typically selected as a fixed one, *e.g.* $\mathcal{N}(0, I)$. The induced distribution of task-specific value functions $p(Q_\theta([s, z], a))$ enables posterior sampling [29] in meta learning scenarios, which brings additional benefits of exploration for continuous control problems.

4 Model & Algorithm

In this section, we present our developed MoE-NPs and connect them to the hierarchical Bayes framework. Then approximate objectives are derived and stochastic gradient variational Bayes [30] is used to optimize the developed model. Finally, specialized neural modules are described for MoE-NPs application to different meta learning tasks. Detailed computational diagrams in training and testing are attached in Appendix (C). For the sake of simplicity, we derive equations *w.r.t.* a task τ in the following section, but a batch of tasks are considered in training in implementation.

4.1 Mixture of Expert Neural Processes

Vanilla NPs often suffer underfitting in experiments [1; 14]. This can be attributed to expressiveness bottlenecks when employing a global latent variable in learning functional priors of tasks from unknown distributions [31].

To alleviate mentioned deficiencies, we make two modifications for the NP family. (i) Multiple functional priors are encoded in modeling with help of K expert latent variables, which can capture statistical traits, *e.g.* distributional multi-modality, in data points. This setting is also an extension of Mixture of Experts (MoEs) models [20; 32; 33; 34] to meta learning scenarios. (ii) Like the gating mechanism in [5], assignment latent variables are included in modeling to select functional forms for each data point in prediction. The resulting MoE-NPs can learn more expressive functional priors and exhibit the approximation power for local properties of the dataset.

Generative Process. As displayed in Fig. (1), the graphical model involves two types of latent variables, respectively the continuous expert latent variables $z_{1:K}$ and the discrete assignment latent variable e. Further, we can translate the generative process into equations as follows,

$$\rho_{x_{1:N}}(y_{1:N}) = \int \prod_{k=1}^{K} p(z_k) \cdot \prod_{i=1}^{N} \left[\sum_{k=1}^{K} p(y_i|x_i, z_{1:K}, e_k = 1) p(e_k = 1|x_i, z_{1:K}) \right] dz_{1:K} \tag{6}$$

where the sampled assignment variable e is in the form of K-dimensional one-hot encoding, and $e_k = 1$ in Eq. (6) indicates the k-th expert z_k is selected from $z_{1:K}$ for prediction. A more detailed probabilistic generative process can also be found in Appendix (E.1). In this way, our developed model constitutes an *exchangeable stochastic process*. And we demonstrate this claim with help of Kolmogorov Extension Theorem [35] in Appendix (E.2).

Link to Hierarchical Bayes. Note that latent variables in Eq. (6) are of hybrid types. K functional priors are incorporated in expert latent variables $z_{1:K}$, while the assignment latent variable e is input dependent. The dependencies between $z_{1:K}$ and e are reflected in modeling, and this connects our work to Hierarchical Bayes [36; 37] in a latent variable sense. Also when only one expert latent variable is used here, the hierarchical model degenerates to the vanilla (C)NPs [1; 2].

4.2 Scalable Training & Prediction

Inference Process. Given a task τ, due to existence of unknown latent variables, it is intractable to perform exact inference *w.r.t.* $p(\mathcal{D}_\tau^T|\mathcal{D}_\tau^C)$. As an alternative, we apply variational inference to our developed model. Here we use $[x, y]$ to denote a single data point from a set of target points $\mathcal{D}_\tau^T$.

$$\begin{aligned} \ln p(y|x, \mathcal{D}_\tau^C) \geq \mathbb{E}_{q_{\phi_1}} \left[\mathbb{E}_{q_{\phi_{2,1}}} \left[\ln p_\theta(y|x, z_{1:K}, e) \right] - D_{KL}[q_{\phi_{2,1}}(e|x, y, z_{1:K}) \parallel p_{\phi_{2,2}}(e|x, z_{1:K})] \right] \\ - \sum_{k=1}^{K} D_{KL}[q_{\phi_{1,k}}(z_k|\mathcal{D}_\tau^T) \parallel q_{\phi_{1,k}}(z_k|\mathcal{D}_\tau^C)] = -\mathcal{L}(\theta, \phi_1, \phi_2) \end{aligned} \tag{7}$$

An example for the k-th expert latent variable z_k can be in the form of a Gaussian distribution $\mathcal{N}(z_k; \mu_k, \Sigma_k)$. And in meta training processes, the assignment variable e is assumed to be drawn from a categorical distribution $\texttt{Cat}(e; K, \alpha(x, y, z_{1:K}))$ with parameters $\alpha(x, y, z_{1:K})$. The existence of discrete latent variables e makes it tough to optimize using traditional methods. This is because either sampling algorithms or expectation maximization algorithms are computationally intensive when utilized here (we have discussed this point in Appendix (G)) for expert latent variables. To reduce computational cost, we again utilize variational inference and the decoder directly formulates the output as a mixture of log-likelihoods $\mathbb{E}_{q_{\phi_{2,1}}}[\ln p_\theta(y|x, z_{1:K}, e)] = \sum_{k=1}^{K} \alpha_k(x, y, z_{1:K}; \phi_{2,1}) \ln p_\theta(y|x, z_k)$. This results in a general ELBO as Eq. (7) for few-shot supervised learning in meta training, where q_{ϕ_1} denotes a collection of K independent variational distribution $\{q_{\phi_{1,1}}, \ldots, q_{\phi_{1,K}}\}$. $q_{\phi_{2,1}}$ and $p_{\phi_{2,2}}$ respectively define the variational posterior and prior for assignment latent variables. Please refer to Appendix (C)/(G) for definitions and more detailed derivations.

Monte Carlo Estimates & Predictions. Meta-training processes consider a batch of tasks to optimize in iterations, and we apply Monte Carlo methods to the obtained negative ELBO $\mathcal{L}(\theta, \phi_1, \phi_2)$ as

follows.

$$\mathcal{L}_{MC}(\theta, \phi_1, \phi_2) = -\frac{1}{NB}\sum_{b=1}^{B}\sum_{i=1}^{N}\left[\sum_{k=1}^{K}\alpha_k^{(b)}\ln p(y_i^{(b)}|x_i^{(b)}, z_k^{(b)})\right]$$
$$+\frac{1}{NB}\sum_{b=1}^{B}\sum_{i=1}^{N}D_{KL}[q_{\phi_{2,1}}(e_i^{(b)}|x_i^{(b)}, y_i^{(b)}, z_{1:K}^{(b)}) \parallel p_{\phi_{2,2}}(e_i^{(b)}|x^{(b)}, z_{1:K}^{(b)})] \tag{8}$$
$$+\frac{1}{NB}\sum_{b=1}^{B}\sum_{k=1}^{K}D_{KL}[q_{\phi_{1,k}}(z_k^{(b)}|\mathcal{D}_b^T) \parallel q_{\phi_{1,k}}(z_k^{(b)}|\mathcal{D}_b^C)]$$

With the number of tasks B and the number of data points N in mini-batches, the Monte Carlo estimate with one stochastic forward pass is Eq. (8) for meta training objectives.

Like that in NPs [1], we derive the predictive distribution as Eq. (9) with one stochastic forward pass and parameters of discrete latent variables $p_{\phi_{2,2}}(e_k = 1|x_*, z_{1:K}) = \alpha_k(x, z_{1:K}; \phi_{2,2})$.

$$p(y_*|x_*, \mathcal{D}_\tau^C) \approx \sum_{k=1}^{K}\alpha_k(x, z_{1:K}; \phi_{2,2})p_\theta(y|x_*, z_k) \quad \text{with } z_{1:K} \sim q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^C) \tag{9}$$

And the point estimate in prediction $\mathbb{E}[Y|X = x, \mathcal{D}_\tau^C]$ can also be obtained in Appendix (G.4).

4.3 Module Details for Meta Learning

4.3.1 Inference Modules in MoE-NPs

The equations so far define a framework that can be implemented in different meta learning tasks. Two examples are given in Fig. (2). Note that two types of latent variables are involved in modelling, we need different structures of encoders for latent variables. Inference Modules are required for them, satisfying different conditions. In variational inference, distribution parameters of these latent variables are approximated with the output of these specialized encoders.

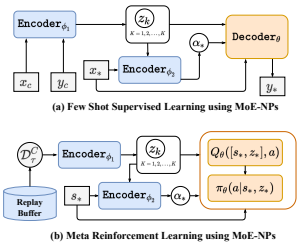

Figure 2: Computational Diagram of MoE-NPs in Meta Testing. **In (a):** The context variables are $\mathcal{D}_\tau^C = [x_C, y_C]$, and the expert latent variables $z_{1:K}$ are approximated with neural networks. For discrete assignment latent variables e_*, we learn parameters of categorical distributions α_* with neural networks. **In (b):** The context variables $\mathcal{D}_\tau^C$ are sampled transitions from a memory buffer. The selected expert latent variable z_* is a context variable in both Actor and Critic networks during policy search.

Inference Modules for Continuous Latent Variables. For neural networks to parameterize the encoder of continuous latent variables q_{ϕ_1}, we use the same architectures in (C)NPs [1; 2], which are permutation invariant to the order of context points $[x_C, y_C] = \{[x_1, y_1], \ldots, [x_N, y_N]\}$. That is, for any permutation operator σ over the set of context points, the neural network (NN) parameters of an output distribution for each expert z_k should satisfy $[\mu_k, \Sigma_k] = \texttt{NN}_{\phi_{1,k}}([x_{\sigma(1:N)}, y_{\sigma(1:N)}])$.

$$r_{k,i} = h_{\phi_{1,k}}([x_i, y_i]),\ r_k = \bigoplus_{i=1}^{N} r_{k,i},\ [\mu_k, \Sigma_k] = g_{\phi_{1,k}}(r_k) \tag{10}$$

Eq. (10) is an example, where h is the embedding function, $\bigoplus$ denotes a mean pooling operation, and g is part of encoder networks.

Inference Modules for Categorical Latent Variables. For neural networks to parameterize the encoder of discrete latent variables $q_{\phi_{2,1}}$ and $p_{\phi_{2,2}}$, we need the categorical distribution parameters α to be permutation equivariant [38] with respect to the order of $z_{1:K}$. This means for any order permutation operation σ, the condition is satisfied as $[\alpha_{\sigma(1)}, \alpha_{\sigma(2)}, \ldots, \alpha_{\sigma(K)}] = \texttt{NN}_{\phi_{2,1}}(x, y, z_{\sigma(1:K)})$.

$$b_k = h_{\phi_{2,1}}(x, y, z_k)\ \forall k \in \{1, 2, \ldots, K\}, \quad [\alpha_1, \alpha_2, \ldots, \alpha_K] = \texttt{softmax}(b/t) \tag{11}$$

An example implementation for the variational posterior $\texttt{Cat}(e; K, \alpha(x, y, z_{1:K}))$ can be Eq. (11), where the vector of logits is $b = [b_1, b_2, \ldots, b_K]$ with t a temperature parameter. And this

implementation applies to prior networks $\texttt{NN}_{\phi_{2,2}}(x, z_{\sigma(1:K)})$ to learn distribution parameters of $\texttt{Cat}(e; K, \alpha(x, z_{1:K}))$ in the same way.

4.3.2 Meta RL Modules in MoE-NPs.

When extending MoE-NPs to meta RL tasks, optimization objectives in Eq. (7) need to be modified for Actor-Critic methods, which are employed in our settings. Like that in PEARL [6] and FCRL [3], the soft actor critic (SAC) algorithm [39] is used to learn policies due to good sample efficiency.

Given a specific MDP τ, posterior distributions of optimal value functions are formulated via latent variables z in context-based meta RL. That is, $p(Q_\theta(s, a; \mathcal{M}))$ is approximated in the form $p(Q_\theta([s, z], a))$. The resulting objectives for the Actor and Critic functions are respectively in Eq. (12) and Eq. (13), where $\mathcal{Z}_\theta$ is a normalization factor.

$$\mathcal{L}_A^\tau = \mathbb{E}_{\substack{s\sim\mathcal{D}_\tau^T, a\sim\pi_\varphi \\ z\sim q_\phi}} \left[D_{KL} \left[\pi_\varphi(a|[s, z]) \parallel \frac{\exp\left(Q_\theta([s, z], a)\right)}{\mathcal{Z}_\theta(s)} \right] \right] \tag{12}$$

The variational posterior $q_\phi(z|s, \mathcal{D}_\tau^C)$ in Eq. (13) is a state dependent distribution with $\hat{V}$ a state value function, and sampling processes refer to steps in Algorithm (3).

$$\mathcal{L}_C^\tau = \mathbb{E}_{\substack{(s,a,s',r)\sim\mathcal{D}_\tau^T \\ z,z'\sim q_\phi}} \left[Q_\theta([s, z], a) - (r + \hat{V}([s', z'])) \right]^2 \tag{13}$$

A key difference from PEARL [6] lies in that several expert latent variables and assignment latent variables are involved in modeling. So we refer the Kullback–Leibler divergence term to Eq. (14) in MoE-NPs with coefficient β_0 and β_1.

$$\mathcal{L}_{KL}^\tau = \beta_1 \mathbb{E}_{\substack{(s,a,s',r)\sim\mathcal{D}_\tau^T \\ q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^C)}} \left[D_{KL}[q_{\phi_2}(e|s, z_{1:K}) \parallel p(e)] \right] + \beta_0 \sum_{k=1}^{K} D_{KL}[q_{\phi_{1,k}}(z_k|\mathcal{D}_\tau^C) \parallel p(z_k)] \tag{14}$$

The Monte Carlo estimates *w.r.t.* Eq. (12/13/14) are used in meta training, and Pseudo code to optimize these functions is listed in Appendix (A).

5 Experiments and Analysis

5.1 General Settings

The implementation of MoE-NPs in meta training can be found in Appendix Algorithms (1)/(3), and also please refer to Appendix Algorithms (2)/(4) for the corresponding meta-testing processes. We leave the details of experimental implementations (*e.g.* parameters, neural architectures, corresponding PyTorch modules and example codes) in Appendix (H).

Baselines for Learning Tasks. Apart from MoE-NPs, methods involved in comparisons are context-based methods such as CNPs [2], NPs [1] and FCRL [3], and gradient-based methods such as MAML [4] and CAVIA [9]. For FCRL, contrastive terms from SimCLR [40] are included in the objective. In meta RL, the modified NP model corresponds to PEARL [6]. Meanwhile, in Appendix (I), we include additional experimental results compared with other NPs models augmented by attentive modules [14] or convolutional modules [17].

5.2 Illustration in Toy Regression

To see different roles of latent variables, we visually show effects of stochastic function fitting and quantified uncertainty in toy dataset. Our goal is to discover potential components of distributions from limited observed data points.

The learning data points are sampled in x-domain $[-\pi, 3\pi]$ and merged from a mixture of randomized functions $f_1(x) = \sin(x) + \epsilon_1$ and $f_2(x) = \cos(x) + \epsilon_2$ with equal probability for mixture components, where $\epsilon_1 \sim \mathcal{N}(0, 0.03^2)$ and $\epsilon_2 \sim \mathcal{N}(0, 0.01^2)$. In each training iteration, we sample a batch of data points and randomly partition context points and target points for learning. In testing

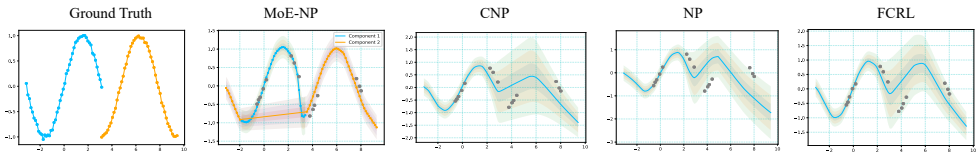

Figure 3: The Ground Truth and Predictive Distributions of Curves using NP related Models. The gray dots around curves are the context points. The shaded regions correspond to `3x` standard deviations. In MoE-NPs, two components of the sampled mixture curve in blue and orange can be identified via assignment latent variables with more than 85% accuracy.

phase, we draw up 100 data points from this mixture of distributions with 15 random data points selected as the context. The fitting results for one sampled mixture curve are shown in Fig. (3). It can be seen that both CNPs and FCRL display similar patterns, overestimate the uncertain in the mixture curve of the second component. NPs show intermediate performance and still fails to match context points well. As for MoE-NPs, with help of predicted assignment variables parameters $e_* = \texttt{one_hot}[\arg_{k\in\{1,2\}} \max \alpha_k]$, we set the number of experts as two and partition data points to visualize predictive distributions $p_\theta(y_*|x_*, z_*)$. The MoE-NP is able to precisely separate mixture components inside the dataset and provides more reliable uncertainty.

5.3 Few-Shot Supervised Learning

We evaluate the performance of models on a system identification task in Acrobot [41] and image completion task in CIFAR10 [42]. Both tasks are common benchmarks in the meta learning or NPs literature [41; 43; 1; 2; 14].

Table 1: System Identification Performance in Meta Testing Acrobot Tasks. Shown are mean square errors and standard deviations in fitting meta-testing tasks. Figures in the Table are scaled by multiplying `E-3` for means and standard deviations. The best results are in bold.

CNP	NP	FCRL	MAML	CAVIA	MoE-NP
2.3(±0.13)	7.2(±0.5)	2.0(±0.15)	2.5(±0.35)	2.0(±0.23)	**1.4(±0.06)**

System Identification. For Acrobot systems, different tasks are generated by varying masses of two pendulums. A dataset of state transitions is collected by using a complete random policy to interact with sampled environments. The state consists of continuous as angles and angular velocities $[\theta_1, \theta_1', \theta_2, \theta_2']$. The objective is to predict the resulting state after a selected Torque action from $\{-1, 0, +1\}$. For more details about meta training dataset and environment information, refer to Appendix (H.2).

In the meta testing phase, 15 episodes with the length of horizon 200 are collected for each task and we report the average predictive errors and standard deviations for all transitions. Here we use 50 transitions as the context points to identify the task. As exhibited in Table (1), gradient-based methods, *e.g.* CAVIA and MAML, beat NP in terms of predictive accuracy but show higher variances than all other models. With three experts in modeling, MoE-NPs significantly outperform other baselines in terms of dynamics prediction. Our finding is consistent with observations in [44], where multi-modal distributions are necessary for Acrobot systems. We also illustrate the asymptotic performance of MoE-NPs with the increase of the context points in the following Section (5.5) Ablation part.

Image Completion. We use CIFAR10 dataset [42] in this experiment, which is formulated with 32x32 RGB images. In the meta training process, a random number of pixels are masked to complete in images. That is, given the context pixel locations and values $[x_C, y_C]$, we need to learn a map from each 2-D pixel location $x \in [0, 1]^2$ to pixel values $y \in \mathbb{R}^3$. Here two expert latent variables are used in MoE-NPs.

In Fig. (4), we evaluate image completion performance on the test dataset and the number of context pixels is varied in three levels. It can be found that CAVIA works best in cases with 10 random context pixels or less than 500 ordered context pixels. In other cases, MoE-NP surpasses all baselines. With more observed pixels, the predictive errors of MoE-NPs can be decreased in both random and ordered context cases. An example of image completion results is displayed in Fig. (5).

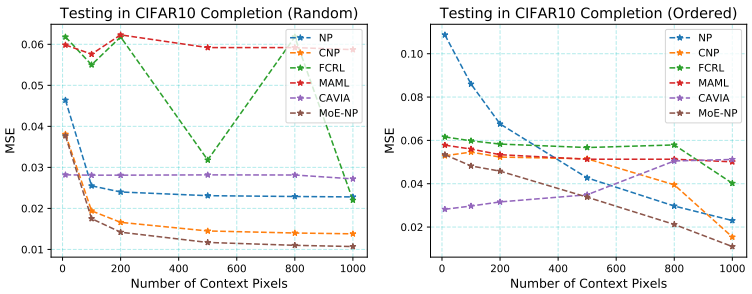

Figure 4: CIFAR10 Completion Performance with Various Number of Context Pixels. The numbers of context points used in prediction are 10, 100, 200, 500, 800, 1000. The left figure is with random context pixels while the right one is with the ordered context pixels.

For gradient-based methods, CAVIA and MAML are sensitive to the number of context points and do not exhibit asymptotic performance like that in MoE-NP. NP still suffers underfitting in performance.

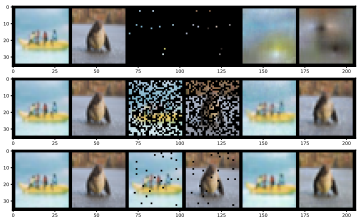

Figure 5: Image Completion Visualization using MoE-NPs. From the Top to the Bottom: the number of random context pixles are 10, 500 and 1000. From the Left to the Right (every two): original images, masked images and reconstructed images.

5.4 Meta Reinforcement Learning

To evaluate the meta RL implementation of our model, we conduct the experiments in a 2-D point robot and Mujoco environments [7]. Fig. (6) exhibits the environments used in this paper, and we leave more details in Appendix (H.1).

2D Navigation. For 2-D point robot tasks, the agent needs to navigate with sparse rewards. The navigation goal of a task is sampled from a mixture of arcs in a semi-circle in Fig. (6.a) during meta training processes.

From Fig. (7.a-c), we can observe the evaluation performance of agents over iterations. For gradient-based methods, CAVIA shows better performance than MAML in exploration but both are weaker than context-based baselines. MoE-NPs can converge earlier with less training samples and show slight advantage over vanilla PEARL. In particular, we test the asymptotic performance in out of distributions (O.O.D.) tasks and show results in Fig. (7.d). We notice O.O.D. tasks are challenging for all algorithms to generalize but average returns are gradually increased with more trials. PEARL and FCRL achieve comparable rewards, while MoE-NP behaves better in this case.

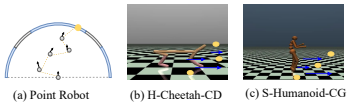

Figure 6: Environments for Meta Reinforcement Learning. **In (a)**: Blue arcs are distributions of goals in orange for the robot to reach with sparse rewards. **In (b)/(c)**: Goals in orange and directions in blue are varied in tasks.

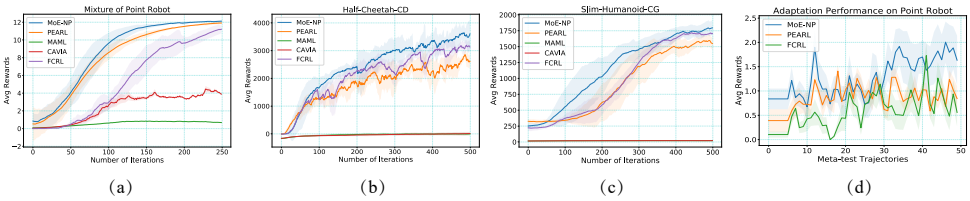

Figure 7: Results in Meta Learning Continuous Control. **In (a)/(b)/(c):** Learning curves show tested average returns with variances in 4 runs. For point robot environments, 100 transitions are randomly collected from a task specific memory buffer to infer the posterior. For Mujoco environments, 400 transitions are randomly collected from a task specific memory buffer to infer the posterior. **In (d):** Fast Adaptation Performance in Meta Testing Point Robot Environments. The collected episodes are gradually increased to 50 and the average returns together with variances are visualized. 5 goals are sampled from the white part of arcs in Fig. (6.a).

Locomotion. As for Half Cheetah-Complex-Direction (H-Cheetah-CD) and Slim Humanoid-Complex-Goal (S-Humanoid-CG) tasks, these correspond to locomotion in complicated environments. Note that multiple directions and goals are involved in tasks.

Fig. (7) illustrates the performance of learned policies in meta learning tasks. In H-Cheetah-CD, MoE-NP shows a slight advantage over FCRL, and it exhibits comparable performance in S-Humanoid-CG. In both environments, MoE-NP and FCRL outperform other baselines. This implies the importance of functional representations for task-specific value functions. Either contrastive or multiple functional priors lead to better exploration and have a potential to boost performance in continuous control. For gradient-based methods, observations show that they can easily get stuck in the local optimal [6; 45].

5.5 Ablation Studies

Number of Experts. We examine the influence of the number of experts in meta-trained MoE-NPs, and system identification in the Acrobot system is selected as an example here. As displayed in Fig. (8), the number of experts are 3, 5, 7 and 9 in different MoE-NPs. Here we test the predictive performance of meta-trained MoE-NP by varying the number of transitions. We set respectively 15, 25, 50, 100 transition samples as the number of context points to identify the system. All settings for meta testing processes are already described in Section (5.3). It can be seen when the expert number is 5, the predictive performance is largely enhanced with the increase of context points and the variance shrinks accordingly. But with more experts, *e.g.* greater than 5, MoE-NPs exhibit higher predictive errors, and show no significant performance improvement with the increase of the number of context points. These suggest increasing the number of experts beyond a certain point can deteriorate the predictive performance of the MoE-NPs.

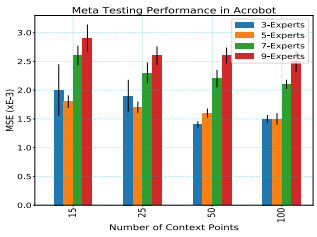

Figure 8: Predictive Performance of MoE-NPs in Acrobot Meta Testing Processes using Varying Numbers of Expert Latent Variables and Context Points. The scale for mean square errors together with standard deviations is E-3.

Latent Variables in Meta RL. As mentioned in Preliminaries Section, the use of latent variable is able to induce task-specific optimal value functions. Here we take the S-Humanoid-CG as the example, and the expert encoder of MoE-NPs (Deterministic) is set to the deterministic. In Fig. (9), we observe the performance degrades a lot using deterministic expert latent variables. These further verify findings in PEARL [6]. The randomness of value function distributions captures task uncertainty and encourages more efficient exploration.

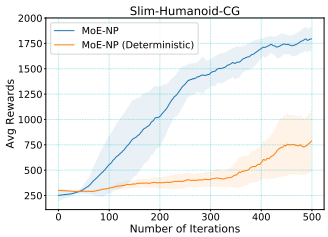

Figure 9: Ablation Performance in S-Humanoid-CG. Learning curves display tested average returns with variances in 4 runs.

6 Conclusions

Technical Discussions. In this work, we have developed a new variant of NP models by introducing multiple expert latent variables. Our work illustrates the roles of different latent variables in MoE-NPs for meta learning tasks. MoE-NPs are able to separate data points from different clusters of stochastic processes and exhibit superior performance in few-shot supervised learning tasks. Also, MoE-NPs are consistently among the best methods in meta learning continuous control.

Existing Limitations. Though the developed model provides more expressive functional priors, the appropriate number of experts is still hard to determine. Also, the mechanism of gradually incorporating new expert latent variables has not been explored and this raises concerns in additional computational cost and more effective inference.

Future Extensions. Here we provide a couple of heuristics to determine optimal number of experts for MoE-NPs in the future. Information metrics, *e.g.* Bayesian information criterion, can be incorporated in modeling. Another way is to place priors over distributions of discrete latent variables like that in hierarchical Dirichlet processes [46] and select the optimal number of experts in a Bayesian way.

References

[1] M. Garnelo, J. Schwarz, D. Rosenbaum, F. Viola, D. J. Rezende, S. Eslami, and Y. W. Teh, "Neural processes," *arXiv preprint arXiv:1807.01622*, 2018.

[2] M. Garnelo, D. Rosenbaum, C. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. Rezende, and S. A. Eslami, "Conditional neural processes," in *International Conference on Machine Learning*, pp. 1704–1713, PMLR, 2018.

[3] M. W. Gondal, S. Joshi, N. Rahaman, S. Bauer, M. Wuthrich, and B. Schölkopf, "Function contrastive learning of transferable meta-representations," in *International Conference on Machine Learning*, pp. 3755–3765, PMLR, 2021.

[4] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International Conference on Machine Learning*, pp. 1126–1135, PMLR, 2017.

[5] C. E. Rasmussen and Z. Ghahramani, "Infinite mixtures of gaussian process experts," *Advances in neural information processing systems*, vol. 2, pp. 881–888, 2002.

[6] K. Rakelly, A. Zhou, C. Finn, S. Levine, and D. Quillen, "Efficient off-policy meta-reinforcement learning via probabilistic context variables," in *International conference on machine learning*, pp. 5331–5340, PMLR, 2019.

[7] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, IEEE, 2012.

[8] I. Osband, B. Van Roy, and Z. Wen, "Generalization and exploration via randomized value functions," in *International Conference on Machine Learning*, pp. 2377–2386, PMLR, 2016.

[9] L. Zintgraf, K. Shiarli, V. Kurin, K. Hofmann, and S. Whiteson, "Fast context adaptation via meta-learning," in *International Conference on Machine Learning*, pp. 7693–7702, PMLR, 2019.

[10] A. Rajeswaran, C. Finn, S. Kakade, and S. Levine, "Meta-learning with implicit gradients," 2019.

[11] C. Finn, K. Xu, and S. Levine, "Probabilistic model-agnostic meta-learning," in *NeurIPS*, 2018.

[12] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 4080–4090, 2017.

[13] K. Allen, E. Shelhamer, H. Shin, and J. Tenenbaum, "Infinite mixture prototypes for few-shot learning," in *International Conference on Machine Learning*, pp. 232–241, PMLR, 2019.

[14] H. Kim, A. Mnih, J. Schwarz, M. Garnelo, A. Eslami, D. Rosenbaum, O. Vinyals, and Y. W. Teh, "Attentive neural processes," in *International Conference on Learning Representations*, 2019.

[15] M. Kim, K. R. Go, and S.-Y. Yun, "Neural processes with stochastic attention: Paying more attention to the context dataset," in *International Conference on Learning Representations*, 2021.

[16] A. Y. Foong, W. Bruinsma, J. Gordon, Y. Dubois, J. Requeima, and R. E. Turner, "Meta-learning stationary stochastic process prediction with convolutional neural processes," in *NeurIPS*, 2020.

[17] J. Gordon, W. P. Bruinsma, A. Y. Foong, J. Requeima, Y. Dubois, and R. E. Turner, "Convolutional conditional neural processes," in *International Conference on Learning Representations*, 2020.

[18] M. Kawano, W. Kumagai, A. Sannai, Y. Iwasawa, and Y. Matsuo, "Group equivariant conditional neural processes," in *International Conference on Learning Representations*, 2020.

[19] P. Holderrieth, M. J. Hutchinson, and Y. W. Teh, "Equivariant learning of stochastic fields: Gaussian processes and steerable conditional neural processes," in *International Conference on Machine Learning*, pp. 4297–4307, PMLR, 2021.

[20] L. Xu, M. I. Jordan, and G. E. Hinton, "An alternative model for mixtures of experts," *Advances in neural information processing systems*, pp. 633–640, 1995.

[21] S. Waterhouse, D. MacKay, T. Robinson, *et al.*, "Bayesian methods for mixtures of experts," *Advances in neural information processing systems*, pp. 351–357, 1996.

[22] J. Gordon, J. Bronskill, M. Bauer, S. Nowozin, and R. Turner, "Meta-learning probabilistic inference for prediction," in *International Conference on Learning Representations*, 2018.

[23] E. Iakovleva, J. Verbeek, and K. Alahari, "Meta-learning with shared amortized variational inference," in *International Conference on Machine Learning*, pp. 4572–4582, PMLR, 2020.

[24] J. Requeima, J. Gordon, J. Bronskill, S. Nowozin, and R. E. Turner, "Fast and flexible multi-task classification using conditional neural adaptive processes," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[25] Z. Sun, J. Wu, X. Li, W. Yang, and J.-H. Xue, "Amortized bayesian prototype meta-learning: A new probabilistic meta-learning approach to few-shot image classification," in *International Conference on Artificial Intelligence and Statistics*, pp. 1414–1422, PMLR, 2021.

[26] C. E. Rasmussen, "Gaussian processes in machine learning," in *Summer school on machine learning*, pp. 63–71, Springer, 2003.

[27] S. M. Ross, J. J. Kelly, R. J. Sullivan, W. J. Perry, D. Mercer, R. M. Davis, T. D. Washburn, E. V. Sager, J. B. Boyce, and V. L. Bristow, *Stochastic processes*, vol. 2. Wiley New York, 1996.

[28] J. Humplik, A. Galashov, L. Hasenclever, P. A. Ortega, Y. W. Teh, and N. Heess, "Meta reinforcement learning as task inference," *arXiv preprint arXiv:1905.06424*, 2019.

[29] D. Russo and B. Van Roy, "Learning to optimize via posterior sampling," *Mathematics of Operations Research*, vol. 39, no. 4, pp. 1221–1243, 2014.

[30] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[31] N. Dilokthanakul, P. A. Mediano, M. Garnelo, M. C. Lee, H. Salimbeni, K. Arulkumaran, and M. Shanahan, "Deep unsupervised clustering with gaussian mixture variational autoencoders," *arXiv preprint arXiv:1611.02648*, 2016.

[32] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," 2016.

[33] S. E. Yuksel, J. N. Wilson, and P. D. Gader, "Twenty years of mixture of experts," *IEEE transactions on neural networks and learning systems*, vol. 23, no. 8, pp. 1177–1193, 2012.

[34] J. Pei, P. Ren, C. Monz, and M. de Rijke, "Retrospective and prospective mixture-of-generators for task-oriented dialogue response generation," in *ECAI 2020*, pp. 2148–2155, IOS Press, 2020.

[35] R. N. Bhattacharya and E. C. Waymire, *Stochastic processes with applications*. SIAM, 2009.

[36] N. D. Lawrence and J. C. Platt, "Learning to learn with the informative vector machine," in *Proceedings of the twenty-first international conference on Machine learning*, p. 65, 2004.

[37] H. Daumé III, "Bayesian multitask learning with latent hierarchies," in *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pp. 135–142, 2009.

[38] M. Finzi, M. Welling, and A. G. Wilson, "A practical method for constructing equivariant multilayer perceptrons for arbitrary matrix groups," *arXiv preprint arXiv:2104.09459*, 2021.

[39] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*, pp. 1861–1870, PMLR, 2018.

[40] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton, "A simple framework for contrastive learning of visual representations," in *International conference on machine learning*, pp. 1597–1607, PMLR, 2020.

[41] T. W. Killian, S. Daulton, G. Konidaris, and F. Doshi-Velez, "Robust and efficient transfer learning with hidden parameter markov decision processes," *Advances in neural information processing systems*, vol. 30, 2017.

[42] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[43] A. Galashov, J. Schwarz, H. Kim, M. Garnelo, D. Saxton, P. Kohli, S. Eslami, and Y. W. Teh, "Meta-learning surrogate models for sequential decision making," *arXiv preprint arXiv:1903.11907*, 2019.

[44] B. Kégl, G. Hurtado, and A. Thomas, "Model-based micro-data reinforcement learning: what are the crucial model properties and which model to choose?," *arXiv preprint arXiv:2107.11587*, 2021.

[45] J. Li, Q. Vuong, S. Liu, M. Liu, K. Ciosek, K. Ross, H. I. Christensen, and H. Su, "Multi-task batch reinforcement learning with metric learning," *arXiv preprint arXiv:1909.11373*, 2019.

[46] Y. W. Teh, M. I. Jordan, M. J. Beal, and D. M. Blei, "Hierarchical dirichlet processes," *Journal of the american statistical association*, vol. 101, no. 476, pp. 1566–1581, 2006.

[47] M. Volpp, F. Flürenbrock, L. Grossberger, C. Daniel, and G. Neumann, "Bayesian context aggregation for neural processes," in *International Conference on Learning Representations*, 2020.

[48] S. A. Eslami, D. Jimenez Rezende, F. Besse, F. Viola, A. S. Morcos, M. Garnelo, A. Ruderman, A. A. Rusu, I. Danihelka, K. Gregor, *et al.*, "Neural scene representation and rendering," *Science*, vol. 360, no. 6394, pp. 1204–1210, 2018.

[49] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "Gshard: Scaling giant models with conditional computation and automatic sharding," *arXiv preprint arXiv:2006.16668*, 2020.

[50] S. Rajbhandari, C. Li, Z. Yao, M. Zhang, R. Y. Aminabadi, A. A. Awan, J. Rasley, and Y. He, "Deepspeed-moe: Advancing mixture-of-experts inference and training to power next-generation ai scale," *arXiv preprint arXiv:2201.05596*, 2022.

[51] Y. Shi, B. Paige, P. Torr, *et al.*, "Variational mixture-of-experts autoencoders for multi-modal deep generative models," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[52] P. Kidger, J. Morrill, J. Foster, and T. Lyons, "Neural controlled differential equations for irregular time series," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6696–6707, 2020.

[53] M. Schirmer, M. Eltayeb, S. Lessmann, and M. Rudolph, "Modeling irregular time series with continuous recurrent units," in *International Conference on Machine Learning*, pp. 19388–19405, PMLR, 2022.

[54] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016.

[55] K. Sohn, H. Lee, and X. Yan, "Learning structured output representation using deep conditional generative models," *Advances in neural information processing systems*, vol. 28, 2015.

[56] B. De Finetti, "La prévision: ses lois logiques, ses sources subjectives," in *Annales de l'institut Henri Poincaré*, vol. 7, pp. 1–68, 1937.

[57] H. Edwards and A. Storkey, "Towards a neural statistician," 2017.

[58] Q. Wang and H. Van Hoof, "Doubly stochastic variational inference for neural processes with hierarchical latent variables," in *International Conference on Machine Learning*, pp. 10018–10028, PMLR, 2020.

[59] C. Louizos, X. Shi, K. Schutte, and M. Welling, "The functional neural process," *Advances in Neural Information Processing Systems*, vol. 32, pp. 8746–8757, 2019.

[60] E. Mathieu, A. Foster, and Y. W. Teh, "On contrastive representations of stochastic processes," *arXiv preprint arXiv:2106.10052*, 2021.

[61] J. Yoon, G. Singh, and S. Ahn, "Robustifying sequential neural processes," in *International Conference on Machine Learning*, pp. 10861–10870, PMLR, 2020.

[62] T. K. Moon, "The expectation-maximization algorithm," *IEEE Signal processing magazine*, vol. 13, no. 6, pp. 47–60, 1996.

[63] E. J. Gumbel, *Statistical theory of extreme values and some practical applications: a series of lectures*, vol. 33. US Government Printing Office, 1954.

[64] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine learning*, vol. 8, no. 3, pp. 229–256, 1992.

[65] H. Ren, A. Garg, and A. Anandkumar, "Contextbased meta-reinforcement learning with structured latent space," in *Skills Workshop NeurIPS*, 2019.

[66] T. A. Le, H. Kim, M. Garnelo, D. Rosenbaum, J. Schwarz, and Y. W. Teh, "Empirical evaluation of neural process objectives," in *NeurIPS workshop on Bayesian Deep Learning*, p. 71, 2018.

[67] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, PMLR, 2015.

Checklist

The checklist follows the references. Please read the checklist guidelines carefully for information on how to answer these questions. For each question, change the default **[TODO]** to [Yes] , [No] , or [N/A] . You are strongly encouraged to include a **justification to your answer**, either by referencing the appropriate section of your paper or providing a brief inline description. For example:

- Did you include the license to the code and datasets? [Yes] See Section **??**.
- Did you include the license to the code and datasets? [No] The code and the data are proprietary.
- Did you include the license to the code and datasets? [N/A]

Please do not modify the questions and only use the provided macros for your answers. Note that the Checklist section does not count towards the page limit. In your paper, please delete this instructions block and only keep the Checklist section heading above along with the questions/answers below.

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

   (a) If your work uses existing assets, did you cite the creators? [Yes]

   (b) Did you mention the license of the assets? [N/A]

   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]

   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes]

   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Contents

1 Introduction **1**

2 Literature Review **2**

3 Preliminaries **2**

3.1 Few-Shot Supervised Learning 3

3.2 Meta Reinforcement Learning 3

4 Model & Algorithm **3**

4.1 Mixture of Expert Neural Processes 4

4.2 Scalable Training & Prediction 4

4.3 Module Details for Meta Learning 5

4.3.1 Inference Modules in MoE-NPs 5

4.3.2 Meta RL Modules in MoE-NPs. 6

5 Experiments and Analysis **6**

5.1 General Settings 6

5.2 Illustration in Toy Regression 6

5.3 Few-Shot Supervised Learning 7

5.4 Meta Reinforcement Learning 8

5.5 Ablation Studies 9

6 Conclusions **9**

A Pseudo Code of Algorithms in Meta Learning **17**

B Frequently Asked Questions **19**

C Probabilistic Graphs in Meta Training/Testing **19**

D More Descriptions of NP Family Models and Meta RL **20**

E MoE-NPs as Exchangeable $\mathcal{SP}$s **21**

E.1 Generative Processes 21

E.2 Consistency Properties 21

F Summary of Existing NP Related Models **22**

F.1 Comparison in Technical Details 22

F.2 Time Complexity 22

F.3 Additional Literature Review 22

G Formulation of Evidence Lower Bounds **23**

G.1 Variational Distributions 24

G.2 Lower Bound on the Evidence for Few-Shot Supervised Learning 24
G.3 Selection of Categorical Approximate Posteriors/Priors 26
G.4 Stochastic Gradient Estimates 26
G.5 Estimates of Statistics 27

H Experimental Settings and Neural Architectures **27**
H.1 Dataset & Environments 28
H.1.1 Dataset in Few-shot Supervised Learning 28
H.1.2 Environments in Meta Reinforcement Learning 28
H.2 Implementations in Meta Learning Tasks 28
H.2.1 Toy Experiments 28
H.2.2 Few-Shot Supervised Learning 29
H.2.3 Meta Reinforcement Learning 29
H.3 Neural Architectures 30

I Additional Experimental Results **35**
I.1 Additional Analysis of Learned Latent Variables 35
I.2 Additional Results of NP Variants in Toy Regression 36
I.3 Additional Results of NP Variants in System Identification 37
I.4 Comparison with Attentive Neural Processes 37
I.5 Augmenting MoE-NPs with Convolutional Modules 38
I.6 More Visualization Results 38

J Computational Devices **38**

A Pseudo Code of Algorithms in Meta Learning

Algorithm 1: MoE-NPs for Few-Shot Supervised Learning.

Input : Task distribution $p(\mathcal{T})$; Task batch size $\mathcal{B}$; Length of mini-batch instances N_{max}; Epochs m; Learning rates λ_1 and λ_2.
Output : Meta-trained parameters $\phi = [\phi_1, \phi_2]$ and θ.

Initialize parameters ϕ and θ;
for $i = 1$ ***to*** m **do**
  Sample a random value $N_C \sim U[1, N_{max}]$
  as the number of context points;
  Sample mini-batch instances $\mathcal{D}$
  to split dataset $\{(x_C, y_C, x_T, y_T)_{bs}\}_{bs=1}^{B}$;
  `// generative process`
  Sample expert latent variables $z_{1:K} \sim q_{\phi_1}(z_{1:K}|\mathcal{D}^T)$;
  Compute distribution parameters α with Eq. (11);
  Compute negative ELBOs $\mathcal{L}_{MC}(\theta, \phi)$ in Eq. (8);
  `// amortized inference process`
  $\phi \leftarrow \phi - \lambda_1 \nabla_\phi \mathcal{L}_{MC}(\theta, \phi)$ in Eq. (8);
  $\theta \leftarrow \theta - \lambda_2 \nabla_\theta \mathcal{L}_{MC}(\theta, \phi)$ in Eq. (8);
end

Algorithm 2: MoE-NPs for Few Shot Supervised Learning (Meta-Testing Phases).

Input : Task τ; Meta-trained $\phi = [\phi_1, \phi_{2,2}]$ and θ.
Output : Predictive distributions.

Initialize parameters ϕ and θ;
Set the number of context points N_C;
Split test dataset into the context/target $\{(x_C, y_C, x_T, y_T)_{bs}\}_{bs=1}^{B} \sim \mathcal{D}$;
`// generative process`
Sample expert latent variables of the mini-batch $z_{1:K} \sim q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^C)$;
if *discrete l.v.s for hard assignment* **then**
  Sample assignment latent variables of the mini-batch $e \sim p_{\phi_{2,2}}(e|x_T, z_{1:K})$;
  Output the distribution $p_\theta(y_T|x_T, z_{1:K}, e)$;
else
  Compute the distribution parameters α of assignment latent variables via Eq. (11);
  Output the predictive distribution $p_\theta(y_T|x_T, \mathcal{D}_\tau^C)$ via Eq. (9);
end

Algorithm 3: MoE-NPs for Meta RL.

Input : MDP distribution $p(\mathcal{T})$; Batch size of tasks $\mathcal{B}$; Training steps m; Learning rates λ_1, λ_2 and λ_3.

Output : Meta-trained parameters ϕ, θ and φ.

Initialize parameters ϕ, θ, φ and replay buffer$\{\mathcal{M}_\tau^C\}^{\mathcal{B}}$;
while *Meta-Training not Completed* **do**
  Sample a batch of tasks $\{\tau\}^{\mathcal{B}} \sim p(\mathcal{T})$;
  `// collect context transitions`
  for *each* $\tau \in \{\tau\}^{\mathcal{B}}$ **do**
    Initialize the context $\mathcal{D}_\tau^C = \{\}$;
    Execute Algorithm (4) in Appendix
    to update $\mathcal{D}_\tau^C$
  end
  `// actor critic learning in batchs`
  for $i = 1$ ***to*** m **do**
    for *each* $\tau \in \{\tau\}^{\mathcal{B}}$ **do**
      Sample context points $\mathcal{D}_\tau^C \sim \mathcal{S}_c(\mathcal{M}_\tau^C)$
      & batch of transitions $b_\tau \sim \mathcal{M}_\tau$;
      Sample $z_{1:K} \sim q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^C)$;
      for *each* $s \in b_\tau$ **do**
        Sample $e \sim q_{\phi_2}(e|z_{1:K}, s)$ to select z
        and augment the state as $[s, z] \in b_\tau$;
      end
      `// run forward propagation`
      $\mathcal{L}_A^\tau = \mathcal{L}_A^\tau(b_\tau)$ in Eq. (12);
      $\mathcal{L}_C^\tau = \mathcal{L}_C^\tau(b_\tau)$ in Eq. (13);
      $\mathcal{L}_{KL}^\tau = \mathcal{L}_{KL}^\tau(\mathcal{D}_\tau^C, b_\tau)$ in Eq. (14)
    end
    `// run back propagation`
    $\phi \leftarrow \phi - \lambda_1 \nabla_\phi \sum_\tau (\mathcal{L}_C^\tau + \mathcal{L}_{KL}^\tau)$ in Eq. (13/14);
    $\varphi \leftarrow \varphi - \lambda_2 \nabla_\varphi \sum_\tau \mathcal{L}_A^\tau$ in Eq. (12);
    $\theta \leftarrow \theta - \lambda_3 \nabla_\theta \sum_\tau \mathcal{L}_C^\tau$ in Eq. (13);
  end
end

Algorithm 4: MoE-NPs for Meta RL (Meta-Testing Phases).

Input : MDP distribution $p(\mathcal{T})$; meta-trained parameters ϕ, θ.

Output : Cumulative rewards.

Sample a test task $\tau \sim p(\mathcal{T})$;
Initialize parameters ϕ, θ, φ and replay buffer $\mathcal{M}_\tau^C$;
`// collect transitions for memory buffers`
Initialize the context $\mathcal{D}_\tau^C = \{\}$;
for $k = 1, 2, \ldots, K$ **do**
  Sample $z_{1:K} \sim q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^C)$;
  for *state* s *of each time step* **do**
    Sample $e \sim q_{\phi_2}(e|s, z_{1:K})$;
    Gather data from $\pi_\varphi(a|[s, z_{1:K}, e])$ to update $\mathcal{M}_\tau^C$;
    Update $\mathcal{D}_\tau^C = \{(s_j, a_j, s'_j, r_j)\}_{j=1}^N \sim \mathcal{M}_\tau^C$;
  end
end

B Frequently Asked Questions

Here we collect some frequently asked questions from reviewers and other literature researchers. We thank these reviewers for these precious questions and add more explanation.

Selection of Benchmarks. Admittedly, NP variants can be applied a series of downstream tasks. Our selection of benchmark missions is based on existing literature for NP models. The system identification task was previously investigated with NP variants in work [43; 47], in which learning transferable physics dynamics with NPs is a crucial application. The image completion task is more commonly used in work [9; 2; 1]. The meta reinforcement learning task can also be studied within NP framework [3; 48].

NP Family in Classification Tasks. We have tried to search neural architectures for few-shot image classification tasks, but the performance is not ideal in comparison to other metrics based methods. Meanwhile, we have gone through most NP related work, and it is challenging to achieve SOTA few-shot image classification results with standard neural architectures in NPs, *e.g.* multi-layer perceptrons (MLPs). Maybe this is due to the nature of stochastic processes, which can address regression problems more efficiently. Unless specialize modules instead of MLPs are used, we do not expect NP variants with MLPs can achieve SOTA performance. The aim of this paper focuses more on mixture expert inductive biases and place less attention on neural architecture search. So MLPs are shared across all baselines to enable fair comparison.

Expressiveness of Mixture of Expert Inductive Biases. A natural question about learning diverse functional representation is whether these multiple expert latent variables will collapse into one. We refer the collapsed representation to vanilla (C)NPs, and the previous empirical results show the collapsed ones work poorer than mixture of expert ones in both few-shot regression tasks and meta reinforcement learning tasks. Also, from the multimodal simulation result, we discover both latent assignment variables and expert variables are meaningful, which reflects the effectiveness of mixture expert inductive biases. That means, we have not encountered meta representation collapse issues in experiments.

Extension with other Mixture of Experts Models. Our work is the first time to examine MoEs inductive biases in NPs family, and the used MoE module is an amortized inference one. We have not found a trivial implementation of MoEs in meta learning domain. But in MoEs literature, there exist other more effective MoE models, which can better trade off communication/memory and performance. So NPs family can also be combined with these models, such as GShard [49], Deepsepeed MoEs [50] and etc.

Potential Applications in Industry. Here we provide two available applications with MoE-NPs in the industry. One is in multilingual machine translation or multilingual language auto-completion. In this case, a mixture of experts corresponds to multilingual functional priors for multi-modal signals [51] and enables the prediction with partial observations. Another application lies in modelling irregular time series [52; 53]. In this case, diverse experts can handle discontinuous components in a rich family of stochastic functions. Meanwhile, the entropy of learned assignment latent variables can tell us the regions likely to be discontinuous, which is quite helpful in anomaly detection in a black-box system.

C Probabilistic Graphs in Meta Training/Testing

As exhibited in Fig. (10)/(11), shown are computational diagrams when implementing MoE-NPs in meta learning tasks.

Variational Posteriors. Since the real posteriors for both $\{z_k\}_{k=1}^K$ and e_T are computationally intractable, the approximate ones are used in practice. These are called variational posteriors, e.g. $q_{\phi_{1,k}}$ for an expert latent variable z_k and $q_{\phi_{2,1}}(e|x, y, z_{1:K})$ for assignment latent variables e. For the sake of simplicity, we denote $\{q_{\phi_{1,k}}\}_{k=1}^K$ by q_{ϕ_1} in some time. Importantly, the Gumbel-softmax trick [54] is used to sample assignment latent variable e from categorical approximate distributions.

Variational Priors. In some cases, the prior distributions for $\{z_k\}_{k=1}^K$ and e_T are set to constrain the scope of prior distributions. For example, in few-shot supervised learning, since the context and the target have the same form, the variational prior is selected to be q_{ϕ_1} as well to ensure the consistency

and this works in meta testing phases. For the assignment latent variable e, this uses the same form in conditional VAE [55] as $p_{\phi_{2,2}}$.

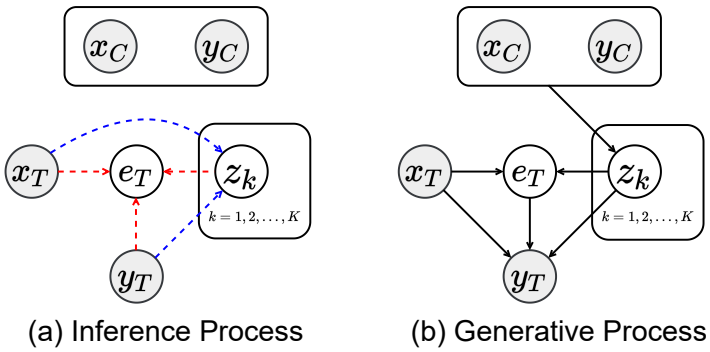

Figure 10: Computational Diagram in Few-shot Supervised Learning. Blue dotted lines are for expert latent variables while red dotted lines are for assignment latent variables in inference.

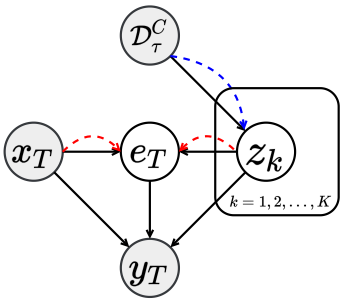

Figure 11: Computational Diagram in Meta Reinforcement Learning. This is in an information bottleneck form [6; 3]. The variational posteriors are $p_{\phi,k} = q_{\phi_{1,k}}(z|\mathcal{D}_\tau^C)$ and $q_{\phi_2} = q_{\phi_{1,k}}(e|s, z_{1:K})$. As for the variational prior, we use the same strategy as that in [6]. They are respectively the fixed normal $p(z) = \mathcal{N}(0, I)$ and the categorical $p(e) = \texttt{Cat}(e; K, [\frac{1}{K}, \frac{1}{K}, \ldots, \frac{1}{K}])$.

D More Descriptions of NP Family Models and Meta RL

In the main paper, we unify the description of NP family models in both few-shot supervised learning and meta reinforcement learning. This is the same with that in FCRL [3]. Meta learning in NP related models is to learn functional representations of different tasks and formulate the fast adaptation via inferring the task specific conditional distribution $p(\mathcal{D}_\tau^T|\mathcal{D}_\tau^C) = \int p(\mathcal{D}_\tau^T|z)p(z|\mathcal{D}_\tau^C)dz$ (equivalent to Eq. (1)).

To make the downstream reinforcement learning task using NP family models clearer, we add the following explanations. In few-shot supervised learning, $\mathcal{D}_\tau^C$ and $\mathcal{D}_\tau^T$ are of the same form. However, in context-based meta reinforcement learning, $\mathcal{D}_\tau^C$ is a set of task specific transitions and $\mathcal{D}_\tau^T$ is a set of state (action) values. As the result, the approximate posteriors and the selected priors to resolve Eq. (1) are distinguished in separate meta learning cases.

In context-based meta reinforcement learning, we can translate our problem into finding the distribution of optimal value functions in Eq. (5), this corresponds to learning meta critic modules with NP family models. Given a transition sample $[s, a, r(s, a), s']$, the target input is the state $x_T = s$ and the target output is the temporal difference target $y_T = \hat{Q}(s, a) = r(s, a) + \gamma V([s', z'])$. The standard Gaussian distribution is used as the prior $p(z_k) = \mathcal{N}(0, I)$ in Eq. (14), while the approximate posterior is learned from $\mathcal{D}_\tau^C$ with permutation invariant functions. In total, sampling from the state dependent approximate posterior $z \sim q_\phi(z|s, \mathcal{D}_\tau^C)$ corresponds to Eq. (15), where the operator $\odot$ denotes the selection process with help of Hadamard products.

$$z_{1:K} \sim q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^C),\ e \sim q_{\phi_2}(e|s, z_{1:K}),\ z = z_{1:K} \odot e \tag{15}$$

E MoE-NPs as Exchangeable $\mathcal{SP}$s

E.1 Generative Processes

To better understand our developed model in meta learning scenarios, we translate Eq. (6) into a step-wise generative process. The same with that in the main paper, the task distribution is denoted by $p(\mathcal{T})$ and we presume K-experts to summarize the stochastic traits of a task.

$$\tau \sim p(\mathcal{T}), \quad z_k \sim p(z_k|\mathcal{T}) \; \forall k \in \{1, 2, \dots, K\} \tag{16a}$$

$$x \sim p(x), \quad e \sim \prod_{k=1}^{K} \alpha_k(x, z_{1:K})^{\mathbb{I}[e_k=1]}, \quad z = [z_1, z_2, \dots, z_K]^T \odot e \tag{16b}$$

$$[\mu_x, \Sigma_x] = g_\theta(x, z), \quad y \sim \mathcal{N}(\mu_x, \Sigma_x + \epsilon^2 I) \tag{16c}$$

Here a MoE-NP for the task τ is specified with K-expert latent variables $z_{1:K}$ in Eq. (16.a). The probability mass function for a data point related categorical distribution $\texttt{Cat}(K, \alpha(x, z_{1:K}))$ is denoted by $p(e|x, z_{1:K}) = \prod_{k=1}^{K} \alpha_k(x, z_{1:K})^{\mathbb{I}[e_k=1]}$ in Eq. (16.b), and e is an assignment latent variable to select an expert for the generative process. After that, the distributional parameters for the output of a data point are learned via a function g_θ in Eq. (16.c), followed by the output distribution $\mathcal{N}(\mu_x, \Sigma_x + \epsilon^2 I)$.

Note that a collection of sampled functional experts are represented in a vector of variables $z_{1:K} = [z_1, z_2, \dots, z_K]^T$ and $e = [0, \cdots, \underbrace{1}_{k\text{-th position}}, \cdots, 0]^T \Leftrightarrow e_k = 1$ is a one-hot vector in Eq. (16.b). In Eq. (16.c), the expert is selected in a way $z = z_{1:K} \odot e$. For the sake of generality, irreducible noise $\mathcal{N}(0, \epsilon^2 I)$ is injected in the output. In experiments, K-expert latent variables $z_{1:K}$ as well as discrete assignment latent variables e are non-observable.

E.2 Consistency Properties

Definition 1. (Exchangeable Stochastic Process) Given a probability space be $(\Omega, \mathcal{F}, \mathbb{P})$, let $\mu_{x_1,\dots,x_N}$ be a probability measure on $\mathbb{R}^d$ with $\{x_1, \dots, x_N\}$ as a finite index set. The defined process is called an exchangeable stochastic process ($\mathcal{SP}$), $\mathcal{S} : X \times \Omega \to \mathbb{R}^d$ such that $\mu_{x_1,\dots,x_N}(F_1 \times \cdots \times F_N) = \mathbb{P}(\mathcal{S}_{x_1} \in F_1, \dots, \mathcal{S}_{x_N} \in F_N)$ when it satisfies the exchangeable consistency and marginalization consistency.

Remember that the generative model is induced in the main paper as follows. And we claim that *our designed generative model MoE-NP formulates a family of exchangeable $\mathcal{SP}$* in **Definition 1**.

$$\rho_{x_{1:N}}(y_{1:N}) = \int \prod_{k=1}^{K} p(z_k) \prod_{i=1}^{N} \left[\sum_{k=1}^{K} p(y_i|x_i, z_{1:K}, e_k = 1) p(e_k = 1|x_i, z_{1:K}) \right] dz_{1:K} \tag{17}$$

So it is necessary to verify two formerly mentioned consistencies according to Kolmogorov Extension Theorem [35] and de Finetti's Theorem [56]. This is to show the existence of $\mathcal{SP}$s in Eq. (17).

Exchangeability Consistency. For N data points from Eq. (17), we impose any permutation operation σ over their indices, and this results in $\sigma : [1, 2, \dots, N] \to [\sigma_1, \sigma_2, \dots, \sigma_N]$. Then we can check the following equation is satisfied since the element-wise product of probabilities can be swapped.

$$
\begin{aligned}
\rho_{x_{1:N}}(y_{1:N}) &= \int \left[\prod_{k=1}^{K} p(z_k)\right] dz_{1:K} \prod_{i=1}^{N} \left[\sum_{k=1}^{K} p(y_{\sigma_i}|x_{\sigma_i}, z_{1:K}, e_k = 1) p(e_k = 1|x_{\sigma_i}, z_{1:K})\right] \\
&= \int \left[\prod_{k=1}^{K} p(z_k)\right] \prod_{i=1}^{N} \left[\sum_{k=1}^{K} p(y_{\sigma_i}|x_{\sigma_i}, z_{1:K}, e_k = 1) p(e_k = 1|x_{\sigma_i}, z_{1:K})\right] dz_{1:K} \\
&= \rho_{x_{\sigma(1:N)}}(y_{\sigma(1:N)}) \quad \square
\end{aligned} \tag{18}
$$

Marginalization Consistency. Given the assumption that the integral in Eq. (17) is finite, we pick up a subset of indices $[M+1, M+2, \ldots, N]$ and make $M < N$ without difference in orders. And the result after marginalization over y-variable in the selected indices can be verified based on the following equation.

$$
\begin{aligned}
\int \rho_{x_{1:N}}(y_{1:N}) dy_{M+1:N} &= \int \left[\prod_{k=1}^{K} p(z_k)\right] dz_{1:K} \left[\int \prod_{i=1}^{N} p(y_i|x_i, z_{1:K}) dy_{M+1:N}\right] \\
&= \iint \prod_{i=1}^{M} p(y_i|x_i, z_{1:K}) \prod_{i=M+1}^{N} (p(y_i|x_i, z_{1:K}) \prod_{k=1}^{K} p(z_k) dz_{1:K} dy_{M+1:N} \\
= \int \left[\prod_{k=1}^{K} p(z_k)\right] dz_{1:K} \prod_{i=1}^{M} \left[\sum_{k=1}^{K} p(y_i|x_i, z_{1:K}, e_k = 1) p(e_k = 1|x_i, z_{1:K})\right] &= \rho_{x_{1:M}}(y_{1:M}) \quad \square
\end{aligned} \tag{19}
$$

Built on these two sufficient conditions, our developed MoE-NP is a well defined exchangeable $\mathcal{SP}$.

F Summary of Existing NP Related Models

F.1 Comparison in Technical Details

Here we give a brief summary on difference between MoE-NPs and existing typical Neural Process models in Table (2). Some crucial traits include forms of encoders and decoders structures, types of latent variable and inductive biases injected in modelling. Especially, the inductive bias for MoE-NP is reduced to be multiple functional priors, which means a collection of expert neural processes to induce the generated dataset. Since a general inductive bias behind NPs related models is the modelling of exchangeable stochastic processes with cheap computations. This corresponds to a distribution of functions, termed as `functional` in the Table. Note that the recognition model of NP models in Meta Training Scenarios is replaced with $q_\phi(z|[x_T, y_T])$ since all target points can be available, but in Meta Testing Scenarios, only $[x_C, y_C]$ are accessible.

F.2 Time Complexity

As for running time complexity, the vanilla NPs and CNPs are with $\mathcal{O}(N+M)$, while MoE-NPs are with $\mathcal{O}(K*(N+M))$ (making M predictions with N observations). In practice, the number of experts is small, so the increase of running time complexity can be ignored in practice. In contrast, traditional Gaussian processes are $\mathcal{O}((N+M)^3)$ in terms of running time complexity.

F.3 Additional Literature Review

Due to page limit in the main paper, we include other related works in this subsection. In unsupervised learning, the Neural Statistician Model [57] is introduced to compute summary statistics inside the dataset. The Generative Query Network [48], a variant of NPs for visual sensory dataset, makes use of a latent variable to abstract scenes in high dimensions. To capture heteroscedastic noise inside the stochastic process, DSVNP [58] induces latent variables at different levels. The functional neural

Table 2: Summary of Typical Neural Process Related Models (Meta-Testing Scenarios). The recognition model and the generative model respectively correspond to the encoder and the decoder in the family of neural processes.

Models	Recognition Model	Generative Model	Latent Variable	Inductive Bias
CNP [2]	$z = f_\phi(x_C, y_C)$	$p_\theta(y\|[x, z])$	continuous	conditional functional
NP [1]	$q_\phi(z\|[x_C, y_C])$	$p_\theta(y\|[x, z])$	continuous	global functional
ANP [14; 15]	$q_{\phi_1}(z\|[x_C, y_C])$ $f_{\phi_2}(z_*\|[x_C, y_C], x_*)$	$p_\theta(y\|[x, z, z_*])$	continuous continuous	global functional local embedding
FCRL [3]	$f_\phi(z\|[x_C, y_C])$	$p_\theta(y\|[x, z])$	continuous	contrastive functional
ConvNP [16]	$p_\phi(z\|[x_C, y_C])$	$p_\theta(y\|[x, z])$	continuous	convolutional functional
Conv-CNP [17]	$f_\phi(z_*\|[x_C, y_C], x_*)$	$p_\theta(y\|[x, z_*])$	continuous	convolutional functional
MoE-NP (Ours)	$q_{\phi_1}(z_{1:K}\|[x_C, y_C])$ $q_{\phi_{2,1}}(e\|z_{1:K}, x, y)$	$p_\theta(y\|[x, z_{1:K}, e])$ $p_{\phi_{2,2}}(e\|z_{1:K}, x)$	continuous categorical	multiple functional

processes infer the directed acyclic graph in the latent space and formulate flexible exchangeable stochastic processes for single task problems [59]. Inspired by self-supervised learning, [3; 60] propose to augment the neural process with contrastive losses. [61] combines context memories and recurrent memories to formulate sequential neural processes (SNPs). Though there exist a number of NP variants, none of them consider to inject multiple functional inductive bias in modeling.

G Formulation of Evidence Lower Bounds

Since functional priors reflected in the K-expert latent variables $z_{1:K}$ are learned via approximate distributions, this can be directly optimized within the framework of variational inference. So we leave these out in this discussion. The difficulty of optimization principally comes from the involvement of discrete latent variables. We therefore discuss chance of using another traditional optimization algorithm, called Expectation Maximization (EM) [62], in our settings. Omitting the K-expert latent variables $z_{1:K}$ and corresponding variational distributions, we take a closer look at the assignment latent variable e in the logarithm likelihood as $\ln\left(\sum_{k=1}^{K} p(y|x, z_{1:K}, e_k = 1)p(e_k = 1|x, z_{1:K})\right)$ and derive the corresponding EM algorithm.

Expectation(E)-Step: Note that the assignment variable e is discrete with the categorical probability function $p(e|x, z_{1:K}) = \texttt{Cat}(e; K, \alpha(x, z_{1:K}))$. This step is to update the posterior of the proportional coefficients $\alpha(x, z_{1:K})$ based on the last time step model parameters $\theta^{(t)}$.

$$\alpha_k^{(t+1)} = p(e_k = 1|x, z_{1:K}, y) \propto p(e_k = 1)p_{\theta^{(t)}}(y|x, z_{1:K}, e_k = 1) \tag{20}$$

Here the prior distribution $p(e)$ can be a commonly used one $\texttt{Cat}(K, [\frac{1}{K}, \frac{1}{K}, \ldots, \frac{1}{K}])$ or the last time updated one $p^{(t)}(e)$. As a result, updated categorical distribution parameters are:

$$\alpha^{(t+1)} = \begin{bmatrix} \alpha_1^{(t+1)} \\ \alpha_2^{(t+1)} \\ \vdots \\ \alpha_K^{(t+1)} \end{bmatrix} = \begin{bmatrix} \frac{\exp((\ln(p_{\theta^{(t)}}(y|x, z_{1:K}, e_1=1)))/\tau)}{\sum_{k=1}^{K}\exp((\ln(p_{\theta^{(t)}}(y|x, z_{1:K}, e_k=1)))/\tau)} \\ \frac{\exp((\ln(p_{\theta^{(t)}}(y|x, z_{1:K}, e_2=1)))/\tau)}{\sum_{k=1}^{K}\exp((\ln(p_{\theta^{(t)}}(y|x, z_{1:K}, e_k=1)))/\tau)} \\ \vdots \\ \frac{\exp((\ln(p_{\theta^{(t)}}(y|x, z_{1:K}, e_K=1)))/\tau)}{\sum_{k=1}^{K}\exp((\ln(p_{\theta^{(t)}}(y|x, z_{1:K}, e_k=1)))/\tau)} \end{bmatrix} \tag{21}$$

where τ is the temperature parameter.

Maximization(M)-Step: Once the distributional parameter of assignment latent variables are updated, the next step is to maximize the logarithm likelihood as $\theta^{(t+1)} =$

$\arg\max_\theta \sum_{(x,y)\in\mathcal{D}} \ln\left[p_{\theta^{(t)}}(y|x, z_{1:K}, e)\right]$ given the last time updated model parameter $\theta^{(t)}$. With help of gradient ascent, this can be written as follows,

$$\theta^{(t+1)} \leftarrow \theta^{(t)} + \lambda \sum_{(x,y)\in\mathcal{D}} \nabla_\theta \ln\left[p_{\theta^{(t)}}(y|x, z_{1:K}, e)\right],\ e = \texttt{one_hot}[\arg\max_k \alpha^{(t+1)}]\ \forall (x,y)\in\mathcal{D} \tag{22}$$

where λ is the learning rate.

Note that the coefficient α is data point dependent and the derivation of EM algorithms considers a subset of data points $\mathcal{D}$. However, in meta learning scenarios, we handle large-scale dataset and the above-mentioned EM framework is computationally expensive and impractical. Due to these considerations, *we do not apply EM algorithms to estimate the discrete distribution* and instead variational inference is employed for the assignment latent variable in optimization.

G.1 Variational Distributions

For continuous latent variables, diagonal Gaussians are commonly used as variational distributions. With Gaussian variational posteriors $\mathcal{N}(z;\mu,\Sigma)$ and corresponding priors $\mathcal{N}(z;\mu_p,\Sigma_p)$, the Kullback–Leibler Divergence can be analytically computed as follows.

$$D_{KL}[\mathcal{N}(z;\mu,\Sigma) \parallel \mathcal{N}(z;\mu_p,\Sigma_p)] = \frac{1}{2}[\ln\frac{|\Sigma_p|}{|\Sigma|} - d + (\mu-\mu_p)^T\Sigma_p^{-1}(\mu-\mu_p) + \texttt{Tr}\{\Sigma_p^{-1}\Sigma\}] \tag{23}$$

Meanwhile, when it comes to categorical distributions, the corresponding prior distribution is selected as $\texttt{Cat}(K,\alpha_0)$ with distribution parameters $\alpha_0 = [\alpha_{0,1}, \alpha_{0,2}, \ldots, \alpha_{0,K}]$. And the Kullback–Leibler Divergence is computed as follows.

$$D_{KL}[\texttt{Cat}(K,\alpha_*) \parallel \texttt{Cat}(K,\alpha_0)] = \sum_{k=1}^{K} \alpha_{*,k} \ln\left[\frac{\alpha_{*,k}}{\alpha_{0,k}}\right] \tag{24}$$

When $\alpha_0 = [\frac{1}{K}, \frac{1}{K}, \ldots, \frac{1}{K}]$, the divergence is further simplified as follows.

$$D_{KL}[\texttt{Cat}(K,\alpha_*) \parallel \texttt{Cat}(K,\alpha_0)] = \sum_{k=1}^{K} \alpha_{*,k} \ln\left[\frac{\alpha_{*,k}}{1/K}\right] = \sum_{k=1}^{K} \ln\alpha_{*,k} + \ln K \tag{25}$$

G.2 Lower Bound on the Evidence for Few-Shot Supervised Learning

Since K-expert latent variables are independent in settings, we denote the corresponding variational parameters by $q_{\phi_1} = \{q_{\phi_{1,1}}, q_{\phi_{1,2}}, \ldots, q_{\phi_{1,K}}\}$. $\phi_{1,k}$ denotes parameters of encoders for k-th expert model. Hence, the distribution follows that $q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^C) = \prod_{k=1}^{K} q_{\phi_{1,k}}(z_k|\mathcal{D}_\tau^C)$ and $q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^T) = \prod_{k=1}^{K} q_{\phi_{1,k}}(z_k|\mathcal{D}_\tau^T)$.

Note that $(x,y) \in \mathcal{D}_\tau^T$ and the variational posterior for expert latent variables are $q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^T)$ in the general NPs. As for the variational posterior for assignment latent variables, we choose $q_{\phi_{2,1}}(e|x,y,z_{1:K})$ as default. We will use these notations to formulate the evidence lower bound (ELBO) as follows.

$$\ln p(y|x, \mathcal{D}_\tau^C) = \ln \int p(y|x, z_{1:K}) p(z_{1:K}|\mathcal{D}_\tau^C) dz_{1:K} \tag{26a}$$

$$\geq \mathbb{E}_{q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^T)} [\ln p(y|x, z_{1:K})] - D_{KL}[q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^T) \parallel p(z_{1:K}|\mathcal{D}_\tau^C)] \tag{26b}$$

$$= \mathbb{E}_{q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^T)} \left[\ln \sum_{k=1}^{K} p(y, e_k = 1|x, z_{1:K}) \right] - D_{KL}[q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^T) \parallel p(z_{1:K}|\mathcal{D}_\tau^C)] \tag{26c}$$

$$= \mathbb{E}_{q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^T)} \left[\ln \sum_{k=1}^{K} p(y|x, z_k) p(e_k = 1|x, z_{1:K}) \right] - D_{KL}[q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^T) \parallel p(z_{1:K}|\mathcal{D}_\tau^C)] \tag{26d}$$

$$\geq \mathbb{E}_{q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^T)} \left[\mathbb{E}_{q_{\phi_{2,1}}(e|x,y,z_{1:K})} [\ln p_\theta(y|x, z_{1:K}, e)] \right] \tag{26e}$$

$$- \mathbb{E}_{q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^T)} \left[D_{KL}[\underbrace{q_{\phi_{2,1}}(e|x, y, z_{1:K})}_{\text{variational discrete posteriors}} \parallel p(e|x, z_{1:K})] \right] \tag{26f}$$

$$- \sum_{k=1}^{K} D_{KL}[\underbrace{q_{\phi_{1,k}}(z_k|\mathcal{D}_\tau^T)}_{K \text{ functional experts}} \parallel p(z_k|\mathcal{D}_\tau^C)] \tag{26g}$$

$$\approx \mathbb{E}_{q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^T)} \left[\mathbb{E}_{q_{\phi_{2,1}}(e|x,y,z_{1:K})} [\ln p_\theta(y|x, z_{1:K}, e)] \right] \tag{26h}$$

$$- \mathbb{E}_{q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^T)} \left[D_{KL}[\underbrace{q_{\phi_{2,1}}(e|x, y, z_{1:K})}_{\text{variational discrete posteriors}} \parallel \underbrace{p_{\phi_{2,2}}(e|x, z_{1:K})}_{\text{variational discrete priors}}] \right] \tag{26i}$$

$$- \sum_{k=1}^{K} D_{KL}[\underbrace{q_{\phi_{1,k}}(z_k|\mathcal{D}_\tau^T)}_{K \text{ functional experts}} \parallel q_{\phi_{1,k}}(z_k|\mathcal{D}_\tau^C)] = -\mathcal{L}(\theta, \phi_1, \phi_2) \quad \square \tag{26j}$$

By introducing the variational distribution q_{ϕ_2} for the discrete assignment latent variable e, Eq.(26.d) is further bounded by Eq. (26.e-g). Recall that when vanilla NP modules are used here, the approximate posterior in Eq. (26) in meta training should be substituted with $q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^T)$ with the corresponding approximate prior $p(z_k) = q_{\phi_{1,k}}(z_k|\mathcal{D}_\tau^C)$. And this matches the general form in the main paper for $-\mathcal{L}(\theta, \phi_1, \phi_2)$ in Eq. (7). When Dirac delta distributions are used in MoE-NPs, the divergence term about the continuous latent variable is removed as default. Denoting the approximate posterior by $q_{\phi_{2,1}}(e|x, y, z_{1:K}) = \texttt{Cat}(e; [\alpha_1(x, y, z_{1:K}), \alpha_2(x, y, z_{1:K}), \ldots, \alpha_K(x, y, z_{1:K})])$, we rewrite the log-likelihood inside the ELBO as Eq. (27).

$$\mathbb{E}_{q_{\phi_{2,1}}(e|x,y,z_{1:K})} [\ln p(y|x, z_{1:K}, e)] = \sum_{k=1}^{K} \alpha_k \ln p(y|x, z_k) \tag{27}$$

As for the approximate posterior of the assignment latent variable $q_{\phi_{2,1}}(e|x, y, z_{1:K})$, we provide two ways of implementations in our experiments: (i) use the target input y as the additional input to formulate $q_{\phi_{2,1}}(e|x, y, z_{1:K})$ (ii) use the same form as the conditional prior $q_{\phi_{2,1}}(e|x, y, z_{1:K}) = p_{\phi_{2,2}}(e|x, z_{1:K})$.

G.3 Selection of Categorical Approximate Posteriors/Priors

As previously observed in Acrobot system identification results, increasing the number of expert latent variables tends to weaken the generalization capability. This also happens in image completion, so we set the number of experts used is 2 in the task. We can attribute this to inference sub-optimality in categorical approximate posteriors/priors.

Remember that in image completion and Acrobot system identification, the used approximate posterior for the categorical latent variable is $q_{\phi_{2,1}}(e|x, y, z_{1:K})$ with the target information y for the input. Since the developed MoE-NP is a VAE-like models [30], the number of expert latent variables K decides the dimension of the assignment latent variable e. In auto-encoder models, when the dimension of latent variables in all hidden layers is higher than that of the input, the model tends to copy the input to the output and fails to learn effective representations. This is the direct source of overfitting and applies to conditional VAE methods [55]. For example, the dimension of output in Acrobot is 6, which implies the bottleneck constraint is weaker when the number of experts is greater than 6.

It is reasonable to alleviate such sub-optimality by directly using the conditional prior as the approximate posterior $q_{\phi_{2,1}}(e|x, y, z_{1:K}) = p_{\phi_{2,2}}(e|x, z_{1:K})$. You can find more clues from the following stochastic gradient estimates for the assignment latent variables in Eq. (31). Meanwhile, we report empirical results in image completion when $q_{\phi_{2,1}}(e|x, y, z_{1:K}) = p_{\phi_{2,2}}(e|x, z_{1:K})$ in Sec. (I.1). And you can see inference in this way does not suffer the overfitting issue caused by more experts.

G.4 Stochastic Gradient Estimates

Here the stochastic gradient estimates with respect to parameters in the negative ELBO $\mathcal{L}(\theta, \phi_1, \phi_2)$ in Eq. (7) are provided as follows.

$$\frac{\partial}{\partial\theta}\mathcal{L}(y; x, \theta, \phi_1, \phi_2) = \mathbb{E}_{q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^T)} \sum_{k=1}^{K} q_{\phi_{2,1}}(e_k = 1|x, y, z_{1:K}) \frac{\partial}{\partial\theta} \ln p_\theta(y|x, z_k) \tag{28}$$

$$\begin{aligned} \frac{\partial}{\partial\phi_{1,k}}\mathcal{L}(y; x, \theta, \phi_1, \phi_2) = &\int \left[\frac{\partial}{\partial\phi_{1,k}} q_{\phi_{1,k}}(z_k|\mathcal{D}_\tau^T)\right] \ln p_\theta(y|x, z_k) dz_k \\ &- \frac{\partial}{\partial\phi_{1,k}} D_{KL}[q_{\phi_{1,k}}(z_k|\mathcal{D}_\tau^T) \parallel q_{\phi_{1,k}}(z_k|\mathcal{D}_\tau^C)] \end{aligned} \tag{29}$$

$$\begin{aligned} \frac{\partial}{\partial\phi_2}\mathcal{L}(y; x, \theta, \phi_1, \phi_2) = &\mathbb{E}_{q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^T)} \left[\sum_{k=1}^{K} \left[\frac{\partial}{\partial\phi_2} q_{\phi_{2,1}}(e_k = 1|x, y, z_{1:K})\right] \ln p_\theta(y|x, z_k)\right] \\ &- \mathbb{E}_{q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^T)} \left[\frac{\partial}{\partial\phi_2} D_{KL}[q_{\phi_{2,1}}(e|x, y, z_{1:K}) \parallel p_{\phi_{2,2}}(e|x, z_{1:K})]\right] \end{aligned} \tag{30}$$

The reparameterization trick [30] is used to sample values from variational distributions of expert latent variables throughout the inference process and stochastic gradient estimates in Eq. (28)/(29)/(30). In prediction processes, the way to get values of assignment latent variables follows that in [63].

Besides, we provide the stochastic gradient estimate for another case when the variational posterior for the assignment latent variable is selected as the variational prior, which means $q_{\phi_{2,1}}(e|x, y, z_{1:K}) = p_{\phi_{2,2}}(e|x, z_{1:K})$. This case can drop off the divergence term for the discrete variable. Let the conditional prior for the discrete variable be $p_{\phi_{2,2}}(e|x, z_{1:K}) =$

$\texttt{Cat}(e; [\alpha_1(x, z_{1:K}), \alpha_2(x, z_{1:K}), \ldots, \alpha_K(x, z_{1:K})])$, we apply the log-derivative trick in a REINFORCE estimator [64] to Eq. (30) and can obtain the following equation as the gradient estimator[1].

$$\begin{aligned}
\frac{\partial}{\partial \phi_{2,2}} \mathcal{L}(y; x, \theta, \phi_1, \phi_2) &= \mathbb{E}_{q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^T)} \left[\sum_{k=1}^{K} \left[\frac{\partial}{\partial \phi_{2,2}} p_{\phi_{2,2}}(e_k = 1|x, z_{1:K}) \right] \ln p_\theta(y|x, z_k) \right] \\
&= \mathbb{E}_{q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^T)} \left[\sum_{k=1}^{K} \left[\underbrace{p_{\phi_{2,2}}(e_k = 1|x, z_{1:K}) \frac{\partial}{\partial \phi_{2,2}} \ln p_{\phi_{2,2}}(e_k = 1|x, z_{1:K})}_{\text{Score Function}} \right] \ln p_\theta(y|x, z_k) \right] \\
&= \mathbb{E}_{q_{\phi_1}(z_{1:K}|\mathcal{D}_\tau^T)} \left[\sum_{k=1}^{K} \alpha_k \frac{\partial}{\partial \phi_{2,2}} \ln p_{\phi_{2,2}}(e_k = 1|x, z_{1:K}) \ln p_\theta(y|x, z_k) \right]
\end{aligned} \tag{31}$$

As can be seen from Eq. (31)

G.5 Estimates of Statistics

Momentum. Given the pre-trained MoE-NPs, we can formulate the statistical momentum in predictive distributions. For the first order momentum, equivalently mean of the predictive distribution, we need to compute the conditional version $\mathbb{E}[Y|X = x, \mathcal{D}_\tau^C]$ in meta learning scenarios. Here the predictive distribution of one expert is parameterized in the form $p(y|x, z_k) = \mathcal{N}(y; m(x, z_k), \Sigma_k)$, where m is the learned mean function using a neural network and σ^2 is a variance parameter. Using a single stochastic forward pass in expert latent variable $z_{1:K}$, we can derive the estimate of the predictive mean $\hat{m} = \mathbb{E}[Y|X = x, \mathcal{D}_\tau^C]$.

$$\hat{m} = \sum_{k=1}^{K} \alpha_k \cdot m(x, z_k), \ \alpha_k = p_{\phi_{2,2}}(e_k = 1|z_{1:K}, x) \tag{32}$$

The second order moment can be estimated accordingly. Here we consider the case when the output is one dimensional and $\Sigma_k = \sigma_k^2$.

$$\mathbb{V}[Y|X = x, \mathcal{D}_\tau^C] = \mathbb{E}[Y^2] - \mathbb{E}[Y]^2 = \sum_{k=1}^{K} \alpha_k (\sigma_k^2 + m(x, z_k)^2) - \hat{m}^2 \tag{33}$$

Entropy. Note that our developed MoE-NPs can also be applied to out of detection (O.O.D) tasks. In this case, the entropy of predictive distribution plays a crucial role. Though the exact estimate of the predictive entropy for MoE-NPs is intractable due to the complexity inside the mixture components, we can measure the expected result of the entropy $\mathbb{E}[\mathcal{H}(Y)|X = x, \mathcal{D}_\tau^C]$ in prediction. We still use a single stochastic forward pass in expert latent variable $z_{1:K}$ in estimation. If $\mathcal{H}(Y_k|X = x, z_k) = -\int p(Y = y|X = x, z_k) \ln p(Y = y|X = x, z_k) dY$ is bounded $\forall k \in \{1, \ldots, K\}$, the estimate of entropy term is as follows.

$$\hat{\mathbb{E}}[\mathcal{H}(Y)|X = x, \mathcal{D}_\tau^C] = \sum_{k=1}^{K} \alpha_k \int p(Y = y|X = x, z_k) \mathcal{H}(Y = y) dy = \sum_{k=1}^{K} \alpha_k \mathbb{E}[\mathcal{H}(Y_k|X = x, z_k)] \tag{34}$$

H Experimental Settings and Neural Architectures

In this section, we provide with more experimental details. Importantly, neural modules of MoE-NPs in the PyTorch version are listed. For the few-shot regression, we provide an example of our implementation of MoE-NPs from the anonymous Github link

[1] For discrete latent variables, we can obtain the analytical form of the stochastic gradient.

https://github.com/codeanonymous233/ICMoENP. For the context-based meta RL algorithms, you can find the implementation of MoE-NPs from the anonymous Github link https://github.com/codeanonymous233/MoENP.

H.1 Dataset & Environments

H.1.1 Dataset in Few-shot Supervised Learning

System Identification. Note that in the used Acrobot simulator [2], the observation is the preprocessed state as a 6 dimensional vector $[\sin(\theta_1), \cos(\theta_1), \sin(\theta_2), \cos(\theta_2), \theta_1', \theta_2']$. The input of the Acrobot system is the concatenation of the observation and the executed action $[\sin(\theta_1), \cos(\theta_1), \sin(\theta_2), \cos(\theta_2), \theta_1', \theta_2', a]$. The output of Acrobot system is the predicted transited state. We generate 16 meta training tasks by varying the masses of two pendulums m_1 and m_2, which means the hyper-parameters of the system come from the Cartesian combination of the set $m_1 \in \{0.75, 0.85, 0.95, 1.15\}$ and $m_2 \in \{0.75, 0.85, 0.95, 1.15\}$. In meta training processes, a complete random policy interacts with batch of sampled MDPs to formulate transition dataset. As for meta testing tasks, we follow the same way to generate tasks by setting $m_1 \in \{0.85, 1.05, 1.25\}$ and $m_2 \in \{0.85, 1.05, 1.25\}$.

Image Completion. CIFAR10 dataset consists of 60000 32x32 color images in 10 categories. Among these images, 50000 are for meta training with the rest for meta testing as the default in image completion tasks. CIFAR10 images are processed via torchvision modules to normalize the pixel values between $[0, 1]$.

H.1.2 Environments in Meta Reinforcement Learning

Note that 2-D point robot navigation tasks, the distribution for meta training is a mixture of uniform distributions $[0, 2\pi/12] \cup [5\pi/12, 7\pi/12] \cup [10\pi/12, \pi]$. The rest of regions along the arc is for out of distribution tasks. The tasks in Mujoco [7] follows adaptations from [6; 65], where goals/velocities or multiple hyper parameters of simulation systems are sampled from mixture distributions. The horizon of an episode for mixture of point robots is 20, while that for Mujoco environments is 500. The required numbers of environment steps in meta training processes are respectively $2.5 * 1e6$ for point robots, $7.5 * 1e6$ for Half-Cheetah-CD and $6.5 * 1e6$ for Slim-Humanoid-CG. We leave more details and settings of each environment in the above github code link.

2-D Point Robot. 2-D robots attempt to reach goals located in specified regions of the arc.

Cheetah-Complex-Direction. 2-D Cheetah robots aim at running in given directions. The task includes multiple target directions and these change with split steps of episodes.

Humanoid-Complex-Goals. 3-D Humanoid robots aim at running towards goals. The task includes multiple goals and these change with split steps of episodes.

H.2 Implementations in Meta Learning Tasks

H.2.1 Toy Experiments

The input of functions is in a range $[-\pi, \pi] \cup [\pi, 2\pi]$. The general implementations of MoE-NPs are as follows. The x-domain for the function $f_1(x) = \sin(x) + \epsilon_1$ with $\epsilon_1 \sim \mathcal{N}(0, 0.03^2)$ is $[-\pi, \pi]$, while that for $f_2(x) = \cos(x) + \epsilon_2$ with $\epsilon_2 \sim \mathcal{N}(0, 0.01^2)$ is $[\pi, 2\pi]$. Sampling from these two components leads to a mixture dataset. The `Encoder` for all continuous latent variables is with two hidden layers (32 neuron units each). The Gaussian distribution is used for continuous latent variables in MoE-NPs. The `Encoder` for the discrete assignment latent variable in MoE-NPs corresponds to a softmax-output neural network with two hidden layers (16 neuron units each). The `Decoder` is with three hidden layers (128 neuron units each) as well. The number of data points in each sampled task is 100.

[2] https://github.com/openai/gym/blob/master/gym/envs/classic_control/acrobot.py

H.2.2 Few-Shot Supervised Learning

For gradient based methods, we use implementations of MAML[3] and CAVIA[4].

System Identification. The general implementations are as follows. For all NP related models, the dimension of a latent variable is set to be 16. The `Encoder` for all continuous latent variables is with two hidden layers (32 neuron units each). Note that Dirac delta distributions are used for continuous latent variables in MoE-NPs since this case works best in few-shot supervised learning. For MoE-NPs, we use three expert latent variables as the default, and the `Encoder` for the discrete assignment latent variable in MoE-NPs corresponds to a softmax-output neural network with two hidden layers (32 neuron units each). The `Decoder` is with four hidden layers (200 neuron units each) as well. The number of tasks in batch training is 4, batch size in training is 200 transition steps (for each task). The horizon for each episode of transitions is 200 time steps. In each iteration of meta training, 4 different environments are randomly selected, and the uniform random controller is used to interact for the collection of 5 episodes (for each task). In training dynamics systems, the training batch size in transition buffer is 200, the training epoch is 5, and the whole process iterates until convergence (the iteration number is 25). The learning rate for Adam optimizer is $1e-3$ as the default.

Image Completion. The general implementations follow that in [9; 2] and are applied to all baselines and MoE-NPs. The dimension of a latent variable is set to be 128. The `Encoder` for all NP variants is with three hidden layers (128 neuron units each). Note that Dirac delta distributions are used for continuous latent variables in MoE-NPs since this case works best in few-shot supervised learning. The `Decoder` is with five hidden layers (128 neuron units each) as well. For MoE-NPs, we use three expert latent variables as the default, and the `Encoder` for the discrete assignment latent variable corresponds to a softmax-output neural network with two hidden layers (32 neuron units each). Adam [30] is used as the optimizer, where the learning rate is set to be $5e-4$. The batch size in training is 8 images, and we meta train the model until convergence (the maximum epoch number is 50). Also note that the number of context pixels in CAVIA is 10 for fast adaptation in default implementations, and this leads to the best testing result of CAVIA in 10 pixel cases in Fig. (4). Note that, to train NP models, including CNP/NP/FCRL/MoE-NP, we set the form of negative log-likelihood objective consistent based on that in [66]. But in evaluation, to keep results of all methods consistent, we follow that in [2; 9; 3] and report the MSEs in Fig. (4) in the testing phase.

H.2.3 Meta Reinforcement Learning

In terms of implementations of baselines, we directly use the following open sourced code: PEARL[5], MAML[6] and CAVIA[7]. Note that the TRPO algorithm [67] is used for MAML/CAVIA as the default. We do not change too much except the replacement of our environments when running experiments.

Further we provide more details on how to modify MoE-NPs in meta RL domains. Notice that MoE-NP can also be seen as a latent variable model, and there exists a close relationship with PEARL algorithms [6] when it comes into meta RL. Implementations of MoE-NPs are the same as that in PEARL [6] except latent variable distributions and the inference way. Note that Soft Actor Critic (SAC) algorithm [39] is used in policy optimization, which requires parameterization of both actor and critic functions. As for the number of experts in MoE-NPs, we use 3 for all environments as default. You can find more details about neural architectures/optimizers for each modules from the above mentioned link.

As mentioned before, we use $p(z_k) = \mathcal{N}(0, I)$ as the prior distribution for expert latent variables. The approximate posterior is parameterized with a diagonal Gaussian distribution. The coefficient for KL divergence terms in Eq. (14) are $\beta_0 = 1.0, \beta_1 = 1.0$. The meta-training processes for reinforcement learning can be found in Algorithm (3) in the main paper. In terms of meta-testing processes, we report the pseudo code in Algorithm (4).

[3] `https://github.com/cbfinn/maml_rl`
[4] `https://github.com/lmzintgraf/cavia`
[5] `https://github.com/katerakelly/oyster`
[6] `https://github.com/cbfinn/maml_rl`
[7] `https://github.com/lmzintgraf/cavia`

H.3 Neural Architectures

To help readers better understand our models, we attach the python code of the `Encoder` for separate latent variables as follows. As for the `Decoder`, the structure is the same with that in a vanilla NP/CNP [1; 2].

```
import torch
import torch.nn as nn
import torch.nn.functional as F

#######################################################################
    # context encoders for expert latent variables
#######################################################################

class Context_Encoder(nn.Module):
    '''
    Encoder network for [x_c, y_c]
    '''
    def __init__(self,
                 input_size,
                 hidden_size,
                 act_type,
                 num_layers,
                 output_size):
        super(Context_Encoder, self).__init__()

        self.emb_c_modules = []
        self.emb_c_modules.append(nn.Linear(input_size,
                                            hidden_size))
        for i in range(num_layers):
            self.emb_c_modules.append(nn.ReLU())
            self.emb_c_modules.append(nn.Linear(hidden_size,
                                                hidden_size))
        self.emb_c_modules.append(nn.ReLU())
        self.context_net = nn.Sequential(*self.emb_c_modules)

        self.mu_net = nn.Linear(hidden_size,
                                output_size)
        self.logvar_net = nn.Linear(hidden_size,
                                    output_size)


    def forward(self, x, mean_dim=1):
        out = self.context_net(x)
        out = torch.mean(out,dim=mean_dim)

        mu, logvar = self.mu_net(out), self.logvar_net(out)

        return (mu,logvar)


#######################################################################
    # context encoders for discrete assignment latent variables
#######################################################################

class Softmax_Net(nn.Module):
    def __init__(self,
                 dim_xz,
                 experts_in_gates,
                 dim_logit_h,
                 num_logit_layers,
                 num_experts):
        super().__init__()
        self.dim_xz = dim_xz
```

```
        self.experts_in_gates = experts_in_gates
        self.dim_logit_h = dim_logit_h
        self.num_logit_layers = num_logit_layers
        self.num_experts = num_experts

        self.logit_modules = []
        if self.experts_in_gates:
            self.logit_modules.append(nn.Linear(self.dim_xz,
                                                self.dim_logit_h))
            for i in range(self.num_logit_layers):
                self.logit_modules.append(nn.ReLU())
                self.logit_modules.append(nn.Linear(self.dim_logit_h,
                                                    self.dim_logit_h))
            self.logit_modules.append(nn.ReLU())
            self.logit_modules.append(nn.Linear(self.dim_logit_h,
                                                1))
        else:
            self.logit_modules.append(nn.Linear(self.dim_xz,
                                                self.dim_logit_h))
            for i in range(self.num_logit_layers):
                self.logit_modules.append(nn.ReLU())
                self.logit_modules.append(nn.Linear(self.dim_logit_h,
                                                    self.dim_logit_h))
            self.logit_modules.append(nn.ReLU())
            self.logit_modules.append(nn.Linear(self.dim_logit_h,
                                                self.num_experts))
        self.logit_net=nn.Sequential(*self.logit_modules)


    def forward(self, x_z, temperature, gumbel_max=False):

        if self.experts_in_gates:
            logit_output = self.logit_net(x_z)
        else:
            x_z = torch.mean(x_z, dim=-2)
            logit_output = self.logit_net(x_z)

        if not self.experts_in_gates:
            logit_output = logit_output.unsqueeze(-1)

        if gumbel_max:
            logit_output = logit_output \
                + sample_gumbel(logit_output.size())

        softmax_y = F.softmax(logit_output/temperature, dim=-2)

        softmax_y = softmax_y.squeeze(-1)
        shape = softmax_y.size()
        _,ind = softmax_y.max(dim=-1)
        y_hard = torch.zeros_like(softmax_y).view(-1, shape[-1])
        y_hard.scatter_(1, ind.view(-1, 1), 1)
        y_hard = y_hard.view(*shape)

        y_hard = (y_hard-softmax_y).detach() \
            + softmax_y

        softmax_y, y_hard = softmax_y.unsqueeze(-1), y_hard.unsqueeze(-1)

        return softmax_y, y_hard
```

Putting the mentioned structures together, we include the MoE-NPs as follows.

```
#######################################################################
    # mixture of expert neural processes
#######################################################################

class MoE_NP(nn.Module):
    def __init__(self,args):
        super(MoE_NP,self).__init__()

        # extract parameters from args
        self.dim_x=args.dim_x
        self.dim_y=args.dim_y

        self.dim_h_lat=args.dim_h_lat
        self.num_h_lat=args.num_h_lat
        self.dim_lat=args.dim_lat
        self.num_lat=args.num_lat
        self.experts_in_gates=args.experts_in_gates
        self.num_logit_layers=args.num_logit_layers
        self.dim_logit_h=args.dim_logit_h
        self.temperature=args.temperature
        self.gumbel_max=args.gumbel_max
        self.info_bottleneck=args.info_bottleneck

        self.dim_h=args.dim_h
        self.num_h=args.num_h
        self.act_type=args.act_type
        self.amort_y=args.amort_y

        # encoding networks
        self.expert_modules=nn.ModuleList([Context_Encoder(self.dim_x+self.dim_y,
                                                           self.dim_h_lat,
                                                           self.act_type,
                                                           self.num_h_lat,
                                                           self.dim_lat).cuda()
                                          for i in range(self.num_lat)])

        if self.experts_in_gates:
            self.logit_net_post=Softmax_Net(self.dim_x+self.dim_y+self.dim_lat,
                                            self.experts_in_gates,
                                            self.dim_logit_h,
                                            self.num_logit_layers,
                                            self.num_lat)
            self.logit_net_prior=Softmax_Net(self.dim_x+self.dim_lat,
                                             self.experts_in_gates,
                                            self.dim_logit_h,
                                            self.num_logit_layers,
                                            self.num_lat)
        else:
            self.logit_net_post=Softmax_Net(self.dim_x+self.dim_y,
                                            self.experts_in_gates,
                                            self.dim_logit_h,
                                            self.num_logit_layers,
                                            self.num_lat)
            self.logit_net_prior=Softmax_Net(self.dim_x,
                                             self.experts_in_gates,
                                             self.dim_logit_h,
                                             self.num_logit_layers,
                                             self.num_lat)

        # decoding networks
        self.dec_modules=[]
        self.dec_modules.append(nn.Linear(self.dim_x+self.dim_lat,
                                          self.dim_h))
```

```
        for i in range(self.num_h):
            self.dec_modules.append(get_act(self.act_type))
            self.dec_modules.append(nn.Linear(self.dim_h,
                                              self.dim_h))
        if self.amort_y:
            self.dec_modules.append(get_act(self.act_type))
            self.dec_modules.append(nn.Linear(self.dim_h,
                                              2*self.dim_y))
        else:
            self.dec_modules.append(get_act(self.act_type))
            self.dec_modules.append(nn.Linear(self.dim_h,
                                              self.dim_y))
        self.dec_net=nn.Sequential(*self.dec_modules)



    def get_context_idx(self,M):
        # generate the indeces of the N context points from M points
        N = random.randint(1,M)
        idx = random.sample(range(0, M), N)
        idx = torch.tensor(idx).cuda()

        return idx


    def idx_to_data(self,data,sample_dim,idx):
        # get subset of an array
        ind_data= torch.index_select(data, dim=sample_dim, index=idx)

        return ind_data


    def reparameterization(self,mu,logvar):
        std=torch.exp(0.5*logvar)
        eps=torch.randn_like(std)

        return mu+eps*std


    def encoder(self,x_c,y_c,x_t,y_t):
        if self.training:
            memo_c,memo_t=torch.cat((x_c,y_c),dim=-1),torch.cat((x_t,y_t),dim=-1)
            emb_c_list_mu=torch.cat([expert_module(memo_c)[0].unsqueeze(0)
                                     for expert_module
                                     in self.expert_modules])
            emb_c_list_logvar=torch.cat([expert_module(memo_c)[1].unsqueeze(0)
                                         for expert_module
                                     in self.expert_modules])
            emb_t_list_mu=torch.cat([expert_module(memo_t)[0].unsqueeze(0)
                                     for expert_module
                                     in self.expert_modules])
            emb_t_list_logvar=torch.cat([expert_module(memo_t)[1].unsqueeze(0)
                                         for expert_module
                                     in self.expert_modules])

            emb_c_list_mu,emb_c_list_logvar=emb_c_list_mu.permute(1,0,2),\
                emb_c_list_logvar.permute(1,0,2)
            emb_t_list_mu,emb_t_list_logvar=emb_t_list_mu.permute(1,0,2),\
                emb_t_list_logvar.permute(1,0,2)

        else:
            memo_c=torch.cat((x_c,y_c),dim=-1)
            emb_c_list_mu=torch.cat([expert_module(memo_c)[0].unsqueeze(0)
                                     for expert_module
                                    in self.expert_modules])
```

```
        emb_c_list_logvar=torch.cat([expert_module(memo_c)[1].unsqueeze(0)
                                     for expert_module
                                    in self.expert_modules])
        emb_c_list_mu,emb_c_list_logvar=emb_c_list_mu.permute(1,0,2),\
            emb_c_list_logvar.permute(1,0,2)

        emb_t_list_mu,emb_t_list_logvar=0,0

    return emb_c_list_mu,emb_c_list_logvar,emb_t_list_mu,emb_t_list_logvar


def forward(self,x_c,y_c,x_t,y_t,x_pred,y_pred,
            whether_acrobot=False,whether_image=True):
    mu_c,logvar_c,mu_t,logvar_t=self.encoder(x_c, y_c, x_t, y_t)

    if self.training:
        if self.info_bottleneck:
            z_experts=self.reparameterization(mu_t,logvar_t)
        else:
            z_experts=mu_c
    else:
        assert y_pred==None
        if self.info_bottleneck:
            z_experts=self.reparameterization(mu_c,logvar_c)
        else:
            z_experts=mu_c

    z_experts_unsq=z_experts.unsqueeze(1).expand(-1,x_pred.size()[1],-1,-1)

    x_exp=x_pred.unsqueeze(2).expand(-1,-1,z_experts_unsq.size()[2],-1)

    if self.training:
        y_exp=y_pred.unsqueeze(2).expand(-1,-1,z_experts_unsq.size()[2],-1)
        if self.experts_in_gates:
            xz_exp=torch.cat((x_exp,z_experts_unsq),dim=-1)
            xy_exp=torch.cat((x_exp,y_exp),dim=-1)
            xyz_exp=torch.cat((xy_exp,z_experts_unsq),dim=-1)
            alpha_post,y_hard_post=self.logit_net_post(x_z=xyz_exp,
                                        temperature=self.temperature,
                                        gumbel_max=self.gumbel_max)
            alpha_prior,y_hard_prior=self.logit_net_prior(x_z=xz_exp,
                                        temperature=self.temperature,
                                        gumbel_max=self.gumbel_max)
        else:
            xy_exp=torch.cat((x_exp,y_exp),dim=-1)
            alpha_post,y_hard_post=self.logit_net_post(x_z=xy_exp,
                                        temperature=self.temperature,
                                        gumbel_max=self.gumbel_max)
            alpha_prior,y_hard_prior=self.logit_net_prior(x_z=x_exp,
                                        temperature=self.temperature,
                                        gumbel_max=self.gumbel_max)
    else:
        if self.experts_in_gates:
            xz_exp=torch.cat((x_exp,z_experts_unsq),dim=-1)
            alpha_post,y_hard_post=0,0
            alpha_prior,y_hard_prior=self.logit_net_prior(x_z=xz_exp,
                                        temperature=self.temperature,
                                        gumbel_max=self.gumbel_max)
        else:
            alpha_post,y_hard_post=0,0
            alpha_prior,y_hard_prior=self.logit_net_prior(x_z=x_exp,
                                        temperature=self.temperature,
                                        gumbel_max=self.gumbel_max)
    output=self.dec_net(torch.cat((x_exp,z_experts_unsq),dim=-1))
```

```
        if whether_acrobot:
            if self.amort_y:
                y_mean,y_std=output[...,:self.dim_y],\
                    F.softplus(output[...,self.dim_y:])
                return mu_c,logvar_c,mu_t,logvar_t,\
                       y_mean,y_std,alpha_post,alpha_prior
            else:
                y_pred=torch.cat((torch.cos(output[...,0:1]),
                                  torch.sin(output[...,0:1]),
                                  torch.cos(output[...,1:2]),
                                  torch.sin(output[...,1:2]),
                                  4*pi*torch.tanh(output[...,2:3]),
                                  9*pi*torch.tanh(output[...,3:4])),axis=-1)
                return mu_c,logvar_c,mu_t,logvar_t,\
                       y_pred,alpha_post,alpha_prior
        elif whether_image:
            if self.amort_y:
                y_mean,y_std=F.sigmoid(output[...,:self.dim_y]),\
                    F.softplus(output[...,self.dim_y:])
                return mu_c,logvar_c,mu_t,logvar_t,\
                       y_mean,y_std,alpha_post,alpha_prior
            else:
                y_pred=F.sigmoid(output)
                return mu_c,logvar_c,mu_t,logvar_t,\
                       y_pred,alpha_post,alpha_prior
        else:
            if self.amort_y:
                y_mean,y_std=output[...,:self.dim_y],\
                    F.softplus(output[...,self.dim_y:])
                return mu_c,logvar_c,mu_t,logvar_t,\
                       y_mean,y_std,alpha_post,alpha_prior
            else:
                y_pred=output
                return mu_c,logvar_c,mu_t,logvar_t,\
                       y_pred,alpha_post,alpha_prior
```

I Additional Experimental Results

I.1 Additional Analysis of Learned Latent Variables

Here we give more detailed analysis *w.r.t.* learned latent variables in MoE-NPs.

Entropy of Assignment Latent Variables. Still we take the 1-dimensional toy stochastic function as example because it is intuitive to understand the latent variable meanings of different levels. Remember that the role of the discrete latent variable e is to assign the diverse functional prior $z_{1:K}$ to a given data point. With the learned conditional prior $p_{\phi_{2,2}}(e|z_{1:K}, x_i)$ for a data point x_i, we can quantify the uncertainty of assignment via the entropy of such a Bernoulli latent variable $\mathcal{H}[e]$.

$$\mathcal{H}[e] = \sum_{k=1}^{K} -p_{\phi_{2,2}}(e_k = 1|z_{1:K}, x_i) \ln p_{\phi_{2,2}}(e_k = 1|z_{1:K}, x_i) = \sum_{k=1}^{K} -\alpha_k \ln \alpha_k \tag{35}$$

This has a practical significance in discontinuous functions. For example, in regions close to demarcation points, it should be difficult to judge the best expert $z_{1:K}$ to handle these data points, which means the set of $\mathcal{H}[e]$ theoretically exhibits higher uncertainty. Similarly, in regions without context points, it is hard to determine the function as well.

Interestingly, we observe that MoE-NPs are able to exhibit the above effect on the right side of Fig. (12). The sampled function consists of two components respectively in the interval $[-\pi, \pi]$ and $[\pi, 3\pi]$. The entropy values of our interest are computed via Eq. (35). Here $K = 2$ and the learned conditional prior $p_{\phi_{2,2}}(e|z_{1:K}, x_i)$ has highest entropy around the demarcation data point π and the

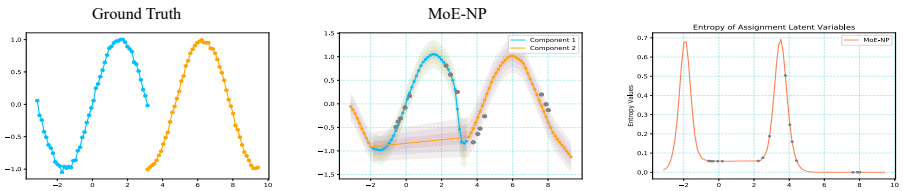

Figure 12: Entropy of Assignment Latent Variables in MoE-NPs. From left to right are respectively the sampled ground truth function, MoE-NP fitting results and the entropy value of discrete latent variable for each data point $\mathcal{H}[p_{\phi_{2,2}}(e|z_{1:K}, x_i)]$.

data point -2.0 with no context points nearby. This finding further verifies the role of the assignment latent variable in MoE-NPs.

Number of Expert Latent Variables. By setting the approximate posterior of assignment latent variables as $q_{\phi_{2,1}}(e|x, y, z_{1:K}) = p_{\phi_{2,2}}(e|x, z_{1:K})$, we further investigate the scalability issue of MoE-NPs in CIFAR10 image completion. As reported in Table (3), we can find with more experts ($>= 3$), the performance can be further improved and no overfitting issue occurs. It can also be inferred when the number of experts reaches a certain level, the improvement from the increase of expert numbers is quite limited. So in general, when the output dimension is lower, the best choice of the approximate posterior for assignment latent variables is in a form without y as the input.

Table 3: Pixel-wise mean squared errors (MSEs) with varying number of experts in CIFAR10 image completion. The number of random context points is varied in a range $(10, 200, 500, 800, 1000)$ to test performance at different levels.

#	10	200	500	800	1000
MoE-NPs (3 experts)	0.0482	0.0183	0.0170	0.0166	0.0165
MoE-NPs (5 experts)	0.0362	0.0103	0.0071	0.0061	0.0057
MoE-NPs (7 experts)	0.0359	0.0095	0.0062	0.0052	0.0048

I.2 Additional Results of NP Variants in Toy Regression

Note that the variation of tasks in the previous toy regression is quite limited and its goal is to show the role of latent variables. To further examine the performance, we construct the mixture of sinusoidal functions by varying the amplitude and the phase as follows.

The learning data points are randomly sampled in x-domain and merged from a mixture of randomized functions $f_1(x) = A\sin(x - B) + \epsilon$ in x-domain $[-\pi, \pi]$ and $f_2(x) = A\cos(x - B) + \epsilon$ in x-domain $[\pi, 3\pi]$ with equal probability, where $\epsilon \sim \mathcal{N}(0, 0.03^2)$. The range of the amplitude is $A \in [0.1, 5.0]$ while that of the phase is $B \in [0, \pi]$.

In each training iteration, we sample a batch of data points and randomly partition context points and target points for learning. Each task consists of 100 randomly sampled data points from the mixture of sinusoidal functions with the random number of context points between $[5, 50]$. The default number of tasks in a meta batch is 25 and we set the number of iteration steps at most 50000. In meta testing phase, we draw up 500 tasks with 15 random data points selected as the context. As for neural architectures of all baselines, we retain that in the previous toy regression in Sec. (H.2.1). Still we use two experts for MoE-NPs as default to fit mixture sinusoidal functions.

Table 4: Test Performance in Mixture Sinusoidal Functions. Shown are mean square errors and standard deviations in fitting 500 sampled tasks. The best results in 5 runs are in bold with standard deviations in bracket.

CNP	NP	FCRL	ANP	MoE-NP
0.053(±1E-4)	0.070(±3E-4)	0.040(±0.0)	**0.027(±2E-4)**	0.032(1E-4)

As shown in Table (4), ANP achieves best performance in meta testing, followed by MoE-NP.

I.3 Additional Results of NP Variants in System Identification

To understand how performance evolves with more context transitions in Acrobot system, we extend the result of 50 context points in the main paper to Fig. (13). As can be seen, MoE-NP still outperforms all NP baselines in all cases. And with the increase of context transitions, we can find the predictive MSEs degrade accordingly.

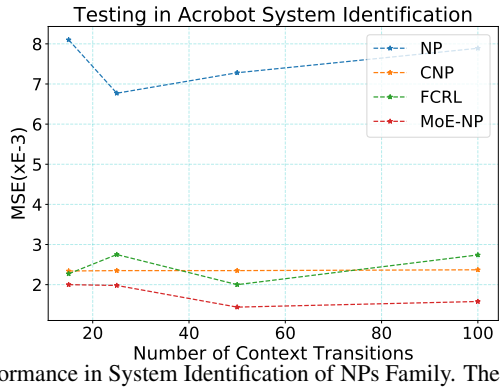

Figure 13: Asymptotic Performance in System Identification of NPs Family. The numbers of random transitions as the context are respectively $\{15, 25, 50, 100\}$.

I.4 Comparison with Attentive Neural Processes

Since neural architectures for attentive neural processes (ANPs) [14] are bit different from used baselines and cannot be trivially modified to meta RL cases, we report additional results in this subsection.

We implement ANPs with dot attention networks (since ANPs have more model complexity and can easily lead to cuda out of memory in practice, we choose the basic version of ANPs), the input embedding dimension of to compute attention weights is 32 and 4 layers are used to transform the deterministic embedding z_{attn}. The local embedding is concatenated with the input x and the global latent variable z for the decoder. We use three heads for system identification tasks and one head for image completion tasks. The related results are reported as follows. It can be seen in Table (5), MoE-NPs still outperform ANPs, while ANPs beat NPs a lot in predicting Acrobot system dynamics. As for CIFAR10 image completion, we can draw the same conclusion in Table (6) that Mixture Expert inductive biases are more effective than local latent variables embedded in attention modules.

Table 5: System identification in Acrobot. Meta testing results are reported. We use the number of random transitions as the context and test performance for ANPs to compare. Figures in the Table are scaled by multiplying `E-3` for means and standard deviations.

#	15	25	50	100
ANP	3.0($\pm$0.36)	2.8($\pm$0.41)	2.5($\pm$0.14)	2.8($\pm$0.19)
MoE-NP	**2.0($\pm$0.45)**	1.9($\pm$**0.28**)	**1.4($\pm$0.06)**	**1.5($\pm$0.06)**

Table 6: Pixel-wise mean squared errors (MSEs) in the image completion tasks on the CIFAR10 dataset. The number of random context points is varied in a range $(10, 200, 500, 800, 1000)$ to test performance at different levels.

#	10	200	500	800	1000
ANP	0.0377	0.0223	0.0217	0.0215	0.0215
MoE-NP	**0.0377**	**0.0142**	**0.0117**	**0.0110**	**0.0107**

I.5 Augmenting MoE-NPs with Convolutional Modules

In this section, we examine the chance of Since neural architectures of encoders are quite different between ConvCNPs [17] and previous mentioned NP baselines, we only report the results of NP related models with the same functional encoder structures in the main paper. Note that the translation equivariance is injected in ConvCNPs, which is a strong inductive bias for image dataset. Naturally, we also develop the convolutional version of MoE-NPs. And we report the separate results in image completion here. In our settings, MoE-ConvCNPs use 3 experts in convolutional modules. The implementation of ConvCNPs can be found in [17]. It can be seen that in Table. (7), in comparison to ConvCNP, the image completion performance is further improved with help of multiple experts.

Table 7: Pixel-wise mean squared errors (MSEs) in the image completion tasks on the CIFAR10 dataset. The number of random context points is varied in a range $(10, 200, 500, 800, 1000)$ to test performance at different levels.

#	10	200	500	800	1000
ConvCNPs	0.036	0.0062	0.002	0.0011	0.0019
MoE-ConvCNPs	**0.035**	**0.0057**	**0.0017**	**0.0007**	**0.0009**

I.6 More Visualization Results

Here we include more visualized CelebA images by varying the number of observed pixels. These are produced using CNN Augmented MoE-NPs (MoE-ConvCNPs). Fig.s (14)/(15)/(16)/(17) are image completion results given the fixed number of random context pixels. Fig.s (18)/(19)/(20)/(21) are image completion results given the fixed number of ordered context pixels.

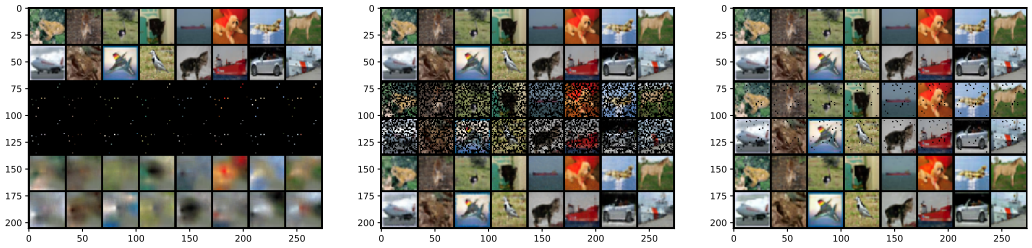

Figure 14: Image Completion Results using CNN Augmented MoE-NPs.

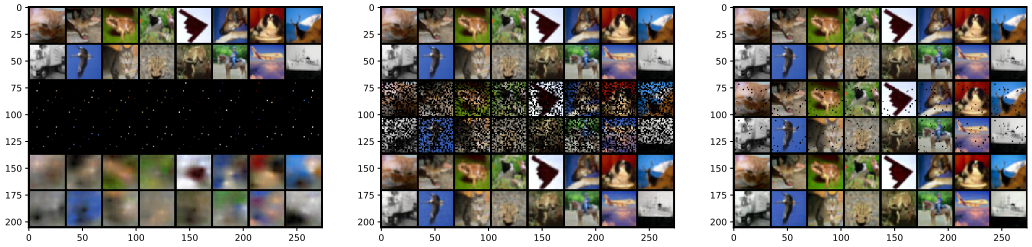

Figure 15: Image Completion Results using CNN Augmented MoE-NPs.

J Computational Devices

Throughout the research process, we use NVIDIA 1080-Ti GPUs and Pytorch is used as the deep learning toolkit.

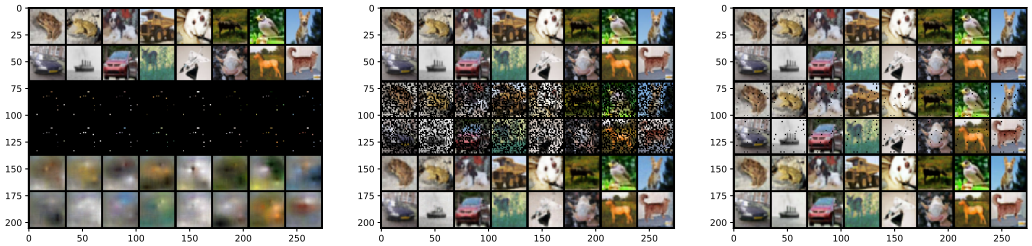

Figure 16: Image Completion Results using CNN Augmented MoE-NPs.

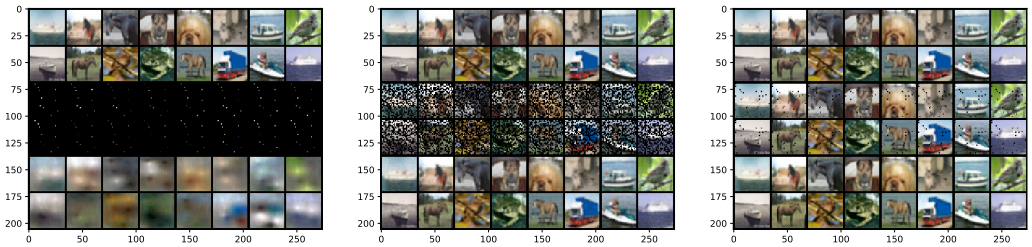

Figure 17: Image Completion Results using CNN Augmented MoE-NPs.

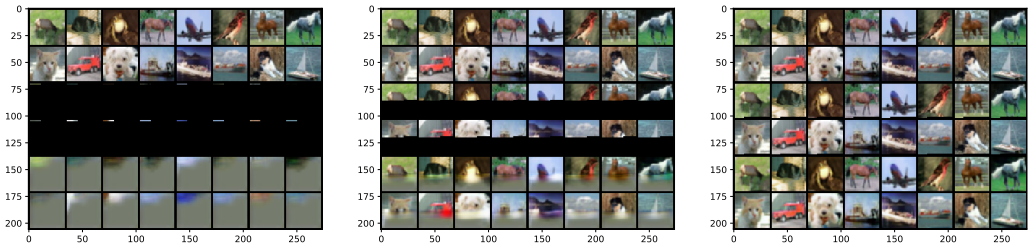

Figure 18: Image Completion Results using CNN Augmented MoE-NPs.

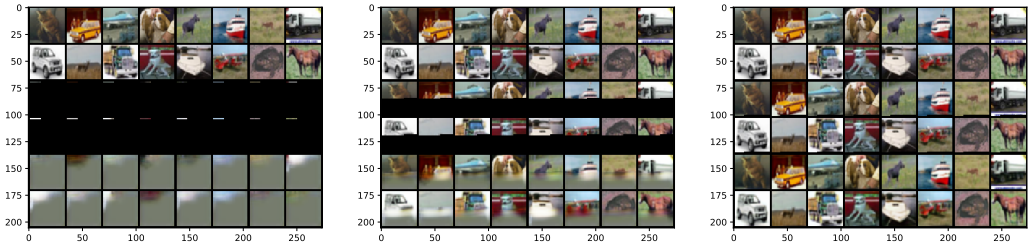

Figure 19: Image Completion Results using CNN Augmented MoE-NPs.

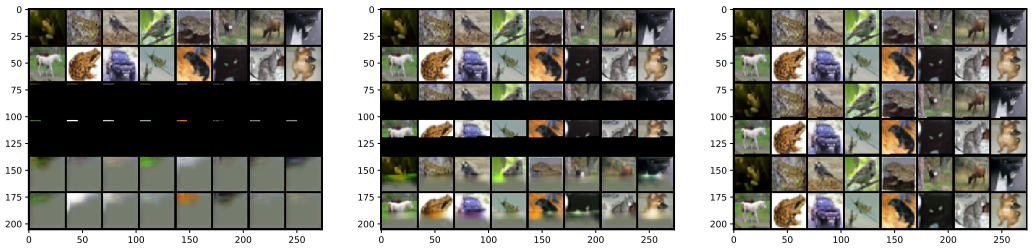

Figure 20: Image Completion Results using CNN Augmented MoE-NPs.

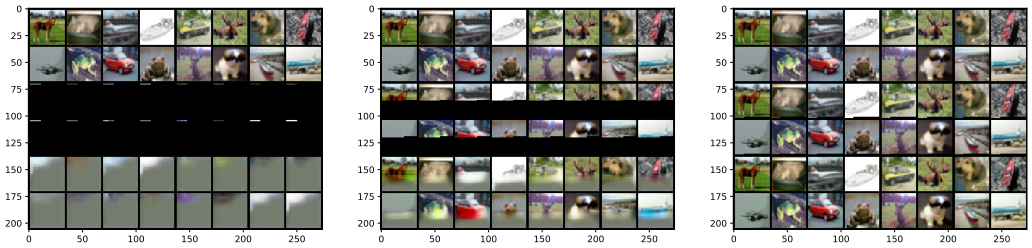

Figure 21: Image Completion Results using CNN Augmented MoE-NPs.