

A Related Works

Neural Processes The original Conditional Neural Process (CNP) [17] introduced a simple encoder-decoder architecture based on multilayer perceptrons. Its extension, the Neural Process (NP) [18], incorporated a global latent variable to model epistemic uncertainty. In this paper, they first suggest the ELBO objective to train the NP models with a latent variable. This design was further improved by Attentive Neural Processes (ANP) [26], which integrated attention mechanisms into the encoder for more expressive context representations. On the other hand, Gordon et al. [20] and Foong et al. [16] incorporate a convolutional deep set module to introduce translation equivariance into NP modules. However, since these models rely on convolutional computations, they are structurally limited and difficult to extend to data beyond three dimensions. Recently, Transformer Neural Processes (TNP) [46] replaced MLPs or attention layers with masked transformer layers to enhance scalability and long-range interaction modeling. Building on TNP, the Dimension-Agnostic Neural Process (DANP) [31] introduced the Dimension Aggregator Block (DAB), enabling a single model to handle tasks with varying input-output dimensions. Our proposed test-time scaling via SMC is developed on top of latent-path-based models such as NP and DANP.

Beyond these, many works have explored improved uncertainty quantification, expressiveness, and inference strategies. For instance, Functional Neural Processes (FNP) [36] used local latent variables to better capture function-specific variations. Bootstrapping Neural Processes (BNP) [32] introduced a residual bootstrapping mechanism to address model misspecification, while Martingale Posterior Neural Processes (MPNP) [30] proposed martingale posterior distribution [15] theory based Bayesian inference for more principled uncertainty estimation. Recent variants like Translation-Equivariant Transformer Neural Processes (TE-TNP) [4], Latent Bottlenecked Attentive Neural Processes (LBANP) [12], and Mixture-of-Experts Neural Processes (MoE-NP) [53] improve generalization and scalability through architectural innovations such as equivariant design, latent bottlenecks, and mixture modeling. Other contributions include Autoregressive Conditional Neural Processes (AR-CNP) [6] for sequential prediction, Self-normalized Importance-weighted Neural Processes (SI-NP) [54] for refined inference, Constant Memory Attentive Neural Processes (CMANP) [13] for memory-efficient modeling, and Gaussian Neural Processes (GNP) [37] for tractable predictive covariance modeling.

Sequential Monte Carlo Our work draws upon foundational concepts in SMC methods and their integration with learned stochastic dynamics. While SMC is designed to efficiently sample from sequential distributions, it has also been extensively explored for sampling complicated posterior distributions through sequential tempering [41], introducing learnable variational distributions [42, 38], or incorporating Langevin dynamics to replace sequential transitions [7]. In parallel, recent works have focused on inference-time scaling in foundation models such as LLMs and diffusion models, leveraging SMC to facilitate this scaling. Zhao et al. [56] treat the autoregressive prediction of LLMs as an SMC process and define a conditional target distribution with a learnable twisted function, enabling the guiding of pretrained LLMs to handle new tasks. In the diffusion models, Kim et al. [27] enhance sample efficiency by using a tempered SMC sampler that balances exploration and exploitation during the sampling process, achieving superior performance in tasks such as aesthetic reward optimization and multi-objective optimization.

B Details about SMCS and Modeling Transition kernels

B.1 Sequential Monte Carlo Samplers

In this section, we briefly review Sequential Monte Carlo Samplers (SMCS) [10], which form the foundational framework for our test-time scaling approach. For a more comprehensive overview, we refer the reader to Del Moral et al. [10], Dai et al. [9].

Let $\pi(x) := \gamma(x)/Z$ denote a target distribution, where the unnormalized density $\gamma(x)$ is known pointwise but the normalization constant Z is unknown. SMCS aims to approximate expectations with respect to π , such as $\mathbb{E}\pi[f]$ for a test function f , while simultaneously providing an estimate of Z . Rather than sampling directly from the target π , SMCS constructs a sequence of intermediate distributions $\{\pi_t\}_{t=0}^T$, beginning with a tractable initial distribution π_0 and gradually transporting it toward the target $\pi_T := \pi$. These intermediate distributions are designed to interpolate between π_0

Algorithm 1 Multinomial Resampling

Require: Particles $\{x_t^i\}_{i=1}^N$ at time step t and corresponding importance weights $\{w_t^i\}_{i=1}^N$
Ensure: Resampled particles $\{\tilde{x}_t^i\}_{i=1}^N$ and normalized importance weights $\{\tilde{w}_t^i\}_{i=1}^N$.

- 1: Compute the normalized importance weights: $\tilde{w}_t^i = \frac{w_t^i}{\sum_{i=1}^N w_t^i}$ for $i \in \{1, \dots, N\}$
- 2: Compute the Effective Sample Size using \tilde{w}_t^i s: $\text{ESS} = \frac{1}{\sum_{i=1}^N (\tilde{w}_t^i)^2}$
- 3: **if** $\text{ESS} < 0.3N$ **then**
- 4: **for** $i = 1$ to N **do**
- 5: Sample j from $\{1, \dots, N\}$ using multinomial distribution with probability $\{\tilde{w}_t^i\}_{i=1}^N$
- 6: Replace particle x_t^i with $\tilde{x}_t^i = x_t^j$
- 7: **end for**
- 8: Replace normalized importance weights \tilde{w}_t^i s with $\tilde{w}_t^i = \frac{1}{N}$ for $i \in \{1, \dots, N\}$
- 9: **else**
- 10: Keep particles $\{\tilde{x}_t^i\}_{i=1}^N$ and normalized importance weights $\{\tilde{w}_t^i\}_{i=1}^N$
- 11: **end if**

890 and π , typically becoming increasingly complex as t increases. Since the intermediate distributions
 891 are generally not analytically tractable, SMCS employs sequential importance sampling to adjust
 892 the samples accordingly.

893 In detail, let γ_t be the unnormalized density of π_t , i.e., $\pi_t(x) = \gamma_t(x)/Z_t$ for $t = 0, \dots, T$ (note
 894 that $Z_0 = 1$ as π_0 is assumed to be fully known). SMCS introduce *forward transition kernels*
 895 $\{F_t(x_t | x_{t-1})\}_{t=1}^T$ which propagates a sample x_{t-1} to x_t . Starting with $x_0 \sim \pi_0$, a sample path
 896 $x_{0:T} := (x_0, \dots, x_T)$ is generated from the joint proposal with density,

$$Q(x_{0:T}) := \pi_0(x_0) \prod_{t=1}^T F_t(x_t | x_{t-1}). \quad (18)$$

897 In principle, one would like to use the marginal $\int Q(x_{0:T}) dx_{0:T-1}$ as a proposal density for the
 898 target π , but this is generally intractable. Instead, SMCS augments the target to the path space $x_{0:T}$
 899 with a sequence of *backward transition kernels* $\{B_{t-1}(x_{t-1} | x_t)\}_{t=1}^T$ and defines the unnormalized
 900 path density

$$\Pi(x_{0:T}) := \frac{\Gamma(x_{0:T})}{Z} \text{ where } \Gamma(x_{0:T}) = \gamma(x_T) \prod_{t=1}^T B_{t-1}(x_{t-1} | x_t). \quad (19)$$

901 A sample path $x_{0:T}$ drawn from Q is then corrected with the importance weights,

$$w_T(x_{0:T}) := \frac{\Gamma(x_{0:T})}{Q(x_{0:T})} = \frac{\gamma(x_T) \prod_{t=1}^T B_{t-1}(x_{t-1} | x_t)}{\pi_0(x_0) \prod_{t=1}^T F_t(x_t | x_{t-1})}. \quad (20)$$

902 Given the forward and backward kernels, SMCS starts by drawing N i.i.d. *particles* $\{x_0^i\}_{i=1}^N$ from
 903 π_0 . At each iteration, the particles are sequentially extended by drawing samples from the forward
 904 kernel F_t , and the extended particles are re-weighted through the importance weights, which are
 905 computed in recursive fashion as follows:

$$w_0(x_0^i) = 1, \quad w_t(x_{0:t}^i) = w_{t-1}(x_{0:t-1}^i) \times \frac{\gamma_t(x_t^i) B_{t-1}(x_{t-1}^i | x_t^i)}{\gamma_{t-1}(x_{t-1}^i) F_t(x_t^i | x_{t-1}^i)} \text{ for } i = 1, \dots, N. \quad (21)$$

906 After completing the sequential update up to the step T , we have an estimator for the expectation
 907 $\mathbb{E}_\pi[f]$ and the normalization constant Z ,

$$\mathbb{E}_\pi[f] \approx \frac{1}{N} \sum_{i=1}^N \frac{w_T(x_{0:T}^i)}{\sum_{j=1}^N w_T(x_{0:T}^j)} f(x_T^i), \quad Z \approx \frac{1}{N} \sum_{i=1}^N w_T(x_{0:T}^i). \quad (22)$$

908 Vanilla SMCS can suffer from particle degeneracy: after several iterations, only a handful of par-
 909 ticles carry nearly all the weight while the rest contribute little. A standard remedy is resampling,
 910 in which a new particle set is drawn from the current importance-weighted empirical distribution.

911 In practice, resampling is triggered adaptively, for instance, whenever the effective sample size falls
 912 below a chosen threshold. When the resampling is applied, the recursive importance weight updates,
 913 and the estimators for expectations and the normalizing constant require minor adjustments. Refer
 914 to [Algorithm 1](#) to see how we conducted resampling in this paper. For a full treatment, see Del Moral
 915 et al. [10].

916 **Choice of intermediate distributions.** There are various options for designing intermediate dis-
 917 tributions; see Del Moral et al. [10], Dai et al. [9] for examples. A common choice is the geometric
 918 annealing path setting $\gamma_t(x) = \pi_0(x)^{1-\beta_t}\gamma(x)^{\beta_t}$ with coefficients $0 = \beta_0 < \beta_1 < \dots < \beta_T = 1$.

919 **Choice of transition kernels.** The design of forward and backward kernels is a critical component
 920 of SMCS. A widely used approach is to set F_t as a transition invariant to γ_t , such as Metropolis-
 921 Hastings steps, as proposed in Annealed Importance Sampling (AIS) [43]. While there exist optimal
 922 backward kernels for given forward kernels [10], they are often intractable in practice. Hence, back-
 923 ward kernels are often chosen for tractability. Notably, in AIS, the forward and backward kernels
 924 are coupled in a specific way, allowing simplified importance weight computation that does not even
 925 require evaluating the forward transition densities. Alternatively, one may consider proposals based
 926 on the Unadjusted Langevin Algorithm (ULA), where the forward transition kernels are defined via
 927 a discretization of Langevin diffusion [24, 55, 49],

$$F_t(x_t | x_{t-1}) = \mathcal{N}(x_t | x_{t-1} + h_t \nabla \log \pi_t(x_{t-1}), 2h_t I), \quad (23)$$

928 with h_t denoting the step size. When h_t is small, the forward proposal is approximately time-
 929 reversible, making the following a natural choice for the backward kernel:

$$B_{t-1}(x_{t-1} | x_t) = F_t(x_{t-1} | x_t) = \mathcal{N}(x_{t-1} | x_t + h_t \nabla \log \pi_t(x_t), 2h_t I). \quad (24)$$

930 One may also consider underdamped Langevin that incorporates auxiliary momentums [19].

931 **Learning for SMCS.** Since both the intermediate distributions and the forward/backward kernels
 932 are crucial design choices in SMCS, it is beneficial to learn them when possible. Learning for SMCS
 933 can be formulated as a variational inference problem, where the ELBO provides a lower bound on
 934 the marginal likelihood defined by the SMCS procedure. A common approach is to begin with un-
 935 adjusted Langevin diffusions and modify the drift term (involving the score functions) to maximize
 936 the ELBO [11, 19, 51, 7]. Among these, Chen et al. [7] introduced an optimal control framework
 937 for tuning a continuous-time extension of SMCS with resampling, which forms the foundation of
 938 our proposed algorithm.

939 B.2 Model structures

940 As discussed in § 3.2, our TTSNP framework models both the pseudo representation generator h
 941 and the forward/backward transitions u and v using neural transition kernels. In this section, we
 942 provide detailed descriptions of how these two neural modules are implemented.

943 We begin with the pseudo representation generator h . As noted in § 3.2, h is designed to be
 944 permutation-invariant to ensure exchangeability of the generated context representations. Specif-
 945 ically, it takes the context set representations R_c and a set of random noise variables $\epsilon \sim p(\epsilon)$ (sam-
 946 pled from a standard Gaussian distribution in our experiments) as input and produces randomized
 947 pseudo context representations.

Following Lee et al. [30], we implement h using the induced self-attention block (ISAB) [33], a
 permutation-invariant attention module. The pseudo representation R_p generated by ISAB is defined
 as:

$$\text{ISAB}(\epsilon) = \text{MAB}(\epsilon, \eta), \quad \text{where} \quad \eta = \text{MAB}(R_c, \epsilon),$$

948 where MAB denotes Multihead Attention Blocks [52], and R_c is the encoded context representation.

949 Next, we describe the structure of the drift functions u and v , which follow the formulation in [Eq. 15](#):

$$u(\mathbf{r}_t, \tau_t) := \frac{\sigma^2}{2} (\nabla \log \tilde{\pi}_t(\mathbf{r}_t) + (1 - \beta_t) \text{NN}(\mathbf{r}_t, t, \mathcal{D}_c, \mathcal{D}_p)), \quad (25)$$

$$v(\mathbf{r}_t, \tau_t) := -\frac{\sigma^2}{2} (\nabla \log \tilde{\pi}_t(\mathbf{r}_t) + (1 - \beta_t) \text{NN}(\mathbf{r}_t, t, \mathcal{D}_c, \mathcal{D}_p)), \quad (26)$$

where $\text{NN}(\cdot)$ is a neural network that takes as input the latent state \mathbf{r}_t , the time step t , and the combined representations of the context and pseudo context sets $\mathcal{D}_c \cup \mathcal{D}_p$, obtained via the pre-trained encoder. Including t as input to NN helps model the adaptive variance of the intermediate posterior $\tilde{q}(\mathbf{r}|\mathcal{D}_c \cup \mathcal{D}_p)$.

The structure of NN is implemented as a 3-layer Multi-Layer Perceptron (MLP), with the time step t embedded using a sinusoidal positional encoding (PE) as in Vaswani et al. [52]. The final form of the network is:

$$\text{NN}(\mathbf{r}_t, t, \mathcal{D}_c, \mathcal{D}_p) = \text{MLP}(\text{concat}(\mathbf{r}_t, R_c, R_p) + \text{PE}(t/T)),$$

where MLP refers to a 3-layer feedforward network, and $\text{PE}(\cdot)$ is the sinusoidal positional embedding.

B.3 Details on KL divergence loss

Path measure and Radon-Nikodym Derivative The KL divergence loss, $\mathcal{L}_{\text{path}}$, is formulated using two path measures, \mathbb{P}^u and \mathbb{P}^v , which correspond to the forward and backward SDEs, respectively. Intuitively, the forward path measure \mathbb{P}^u can be interpreted as the joint distribution $Q(\mathbf{r}_{0:T}^u)$ in the limit as $T \rightarrow \infty$ [5]. If this divergence is minimized to zero, it implies that the forward and backward transitions are perfectly time-reversible, thereby ensuring ideal sampling [7]. Consequently, training the transition kernels with this objective directly enhances the accuracy and quality of samples generated by the SMCS procedure. A key requirement for enabling principled training and inference is the ability to compute the likelihood ratio w between the forward and reverse-time processes, known as the Radon-Nikodym derivative (RND) [51, 7]. This quantity serves as the backbone for defining meaningful objectives and ensuring the correctness of sample-based approximations. Without a tractable form of the RND, learning reliable transition dynamics becomes fundamentally intractable. Fortunately, recent advances [51, 7] provide a computable expression for this crucial quantity as in the following lemma, which we leverage directly in our framework.

Lemma B.1 (Radon-Nikodym Derivative [51]). *Let $\mathbb{P}_{[a,b]}^u$ and $\mathbb{P}_{[a,b]}^v$ denote the path space measures of the solutions to the forward and backward SDEs defined in Eq. 11 and Eq. 12, respectively, over the time interval $[a, b] \subset [0, 1]$, where the drift functions u and v satisfy Nelson’s identity. Assume that the marginal distributions satisfy $\mathbf{r}_a^u \sim \pi_a$ and $\mathbf{s}_b^v \sim \pi_b$. Then, $\mathbb{P}_{[a,b]}^u$ -almost surely, the RND between these two measures can be computed as:*

$$\begin{aligned} \ln w_{[a,b]} = \ln \frac{d\mathbb{P}_{[a,b]}^v}{d\mathbb{P}_{[a,b]}^u}(\mathbf{r}) &= \ln \pi_b(\mathbf{r}_b) - \ln \pi_a(\mathbf{r}_a) + \int_a^b \frac{\|u\|^2 - \|u - \sigma^2 \nabla \log \pi\|^2}{2\sigma^2}(\mathbf{r}_t, t) dt \\ &\quad + \int_a^b \frac{u - \sigma^2 \nabla \log \pi}{\sigma^2}(\mathbf{r}_t, t) \cdot d\mathbf{r}_t^v - \int_a^b \frac{u}{\sigma^2}(\mathbf{r}_t, t) \cdot d\mathbf{r}_t^u. \end{aligned}$$

As we can see in Lemma B.1, we can compute w by divide trajectories \mathbf{r}^u and \mathbf{s}^v into subtrajectories using multiple chunks as follows:

$$\ln w = \ln \frac{\mathbb{P}^v}{\mathbb{P}^u} = \ln \frac{\mathbb{P}_{[\tau_0, \tau_1]}^v}{\mathbb{P}_{[\tau_0, \tau_1]}^u} + \dots + \ln \frac{\mathbb{P}_{[\tau_{T-1}, \tau_T]}^v}{\mathbb{P}_{[\tau_{T-1}, \tau_T]}^u} = \ln w_{[\tau_0, \tau_1]} + \dots + \ln w_{[\tau_{T-1}, \tau_T]}. \quad (27)$$

Following Vargas et al. [51] and Chen et al. [7], we can further simplify the continuous-time RND expression from Lemma B.1 by applying a discretization approximation. This yields a practical discrete-time formulation suitable for implementation:

$$\ln w_{[\tau_{n-1}, \tau_n]}(\mathbf{r}) = \ln \frac{d\mathbb{P}_{[\tau_{n-1}, \tau_n]}^v}{d\mathbb{P}_{[\tau_{n-1}, \tau_n]}^u} \quad (28)$$

$$\approx \ln \pi_{\tau_n}(\mathbf{r}_{\tau_n}) - \ln \pi_{\tau_{n-1}}(\mathbf{r}_{\tau_{n-1}}) + \sum_{i=1}^L \frac{\ln p_{(n-1)L+i-1}^v(\mathbf{r}_{((n-1)L+i-1)h} | \mathbf{r}_{((n-1)L+i)h})}{\ln p_{(n-1)L+i}^u(\mathbf{r}_{((n-1)L+i)h} | \mathbf{r}_{((n-1)L+i-1)h})}, \quad (29)$$

where L is the number of steps per subtrajectory and h is the step size. Here, by the Euler-Maruyama discretization, forward and backward transition densities are computed as follows:

$$p_{(n-1)L+i-1}^u(\mathbf{r}_{\text{new}} | \mathbf{r}_{\text{pre}}) = \mathcal{N}(\mathbf{r}_{\text{new}} | \mathbf{r}_{\text{pre}} + hu(\mathbf{r}_{\text{pre}}, a'), h\sigma^2(a')) \quad (30)$$

$$p_{(n-1)L+i}^v(\mathbf{r}_{\text{pre}} | \mathbf{r}_{\text{new}}) = \mathcal{N}(\mathbf{r}_{\text{pre}} | \mathbf{r}_{\text{new}} + h(\sigma^2 \nabla \log \pi - u)(\mathbf{r}_{\text{new}}, b'), h\sigma^2(b')) \quad (31)$$

where a' and b' are $((n-1)L+i-1)h$ and $((n-1)L+i)h$, respectively.

Algorithm 2 Overall TTSNP inference algorithm

Require: Context dataset \mathcal{D}_c , trained pseudo context generator h , and learned forward and backward transitions u and v .

Ensure: Updated latent particles $\{\mathbf{r}_T^i\}_{i=1}^N$ and corresponding normalized importance weights $\{\tilde{w}_T^i\}_{i=1}^N$.

- 1: **First Stage: Generate pseudo Representation**
 - 2: Sample noise $\epsilon \sim p(\epsilon)$
 - 3: Generate pseudo representations R_p using $h(\mathcal{D}_c, \epsilon)$
 - 4: **Second Stage: Generate initial latent particles and importance weights**
 - 5: Sample N number of latent particles $\{\mathbf{r}_0^i\}_{i=1}^N$ from variational latent posterior $q(\mathbf{r}|\mathcal{D}_c)$
 - 6: Set corresponding initial importance weights $\{w_0^i\}_{i=1}^N$ where $w_0^i = \frac{1}{N}$ for $i \in \{1, \dots, N\}$
 - 7: **Third Stage: SMCS procedure**
 - 8: **for** $t = 1$ to T **do**
 - 9: **for** i to N **do**
 - 10: Sample $\mathbf{r}_t^i \sim \mathcal{N}(\mathbf{r}_t^i | \mathbf{r}_{t-1}^i + h_t u(\mathbf{r}_{t-1}^i, \tau_{t-1}), 2h_t I)$
 - 11: **end for**
 - 12: Update importance weight based on Eq. 8 and Eq. 29
 - 13: Resample the latent variables \mathbf{r}_t^i and update \tilde{w}_t^i following Algorithm 1
 - 14: **end for**
 - 15: **Forth Stage: Compute predictive distribution**
 - 16: With final $\{\mathbf{r}_T^i\}_{i=1}^N$ and $\{\tilde{w}_T^i\}_{i=1}^N$ compute the predictive distribution similar to Eq. 38
-

983 **KL divergence loss** Then, given the sequential nature of our approach, the KL divergence is
984 evaluated independently on each subinterval $[\tau_{t-1}, \tau_t]$, yielding:

$$\mathcal{L}_{\text{KL}} = \mathbb{E}_{\mathcal{D}_p} \left[\sum_{t=1}^T D_{\text{KL}} \left[\mathbb{P}_{[\tau_{t-1}, \tau_t]}^u || \mathbb{P}_{[\tau_{t-1}, \tau_t]}^v \right] \right], \quad (32)$$

985 where KL divergence is averaged by pseudo context due to the randomness \mathcal{D}_p . This formulation can
986 be rewritten as a simplified objective using importance weights, following Chen et al. [7]. Following
987 the approaches proposed in Matthews et al. [39] and Chen et al. [7], the KL divergence on each
988 interval can be calculated as:

$$D_{\text{KL}} \left[\mathbb{P}_{[\tau_{t-1}, \tau_t]}^u || \mathbb{P}_{[\tau_{t-1}, \tau_t]}^v \right] = -\mathbb{E}_{\mathbf{r}^u \sim \mathbb{P}_{[\tau_k, \tau_t]}^u} \left[\log w_{[\tau_{t-1}, \tau_t]}(\mathbf{r}^u) \cdot w_{[\tau_k, \tau_t]}(\mathbf{r}^u) \right], \quad (33)$$

989 where $\tau_k < \tau_{t-1}$ refers to the most recent resampling time prior to τ_{t-1} . Finally, the KL loss \mathcal{L}_{KL}
990 can be expressed in practice using importance weights as:

$$\mathcal{L}_{\text{KL}} = \mathbb{E}_{\mathcal{D}_p} \left[\sum_{t=1}^T \frac{1}{N} \sum_{i=1}^N \text{detach}(w_{\tau_k}^{(i)}) \log w_{[\tau_{t-1}, \tau_t]}^{(i)} \right], \quad (34)$$

991 where $\text{detach}(\cdot)$ indicates that gradients are not propagated through the resampled weights $w_{n-1}^{(i)}$. In
992 our experiment, for efficient training, we trained the model using a single set of \mathcal{D}_p .

993 B.4 Overall TTSNP latent variable sampling algorithm

994 In this section, we present the full inference procedure of TTSNP as an algorithm that uses an SMCS
995 procedure to refine latent samples toward the true posterior distribution. As outlined in Algorithm 2,
996 the overall method can be divided into four main stages: (1) ‘Generate pseudo representation’, (2)
997 ‘Generate initial latent particles and importance weights’, (3) ‘SMCS procedure’, and (4) Compute
998 predictive distribution’.

999 In the first stage, ‘Generate pseudo representation’, we use the pseudo context generator h to create
1000 pseudo representations conditioned on the context set. Next, in ‘Generate initial latent particles
1001 and importance weights’, we sample initial latent variables from the variational posterior $p(\mathbf{r} | \mathcal{D}_c)$
1002 and assign initial weights accordingly. These latent particles are then used to initialize the SMCS
1003 procedure.

Table 4: Model structure details of NP

CATEGORY	DETAILS
MODEL SPECIFICATIONS	
HIDDEN DIMENSION FOR DETERMINISTIC PATH	128
HIDDEN DIMENSION FOR LATENT PATH	128
MLP DEPTH FOR DETERMINISTIC PATH	4
MLP DEPTH FOR LATENT ENCODER PRE-MODULE	4
MLP DEPTH FOR LATENT ENCODER POST-MODULE	2
DECODER DEPTH	3
NUMBER OF PARAMETERS FOR 1D GP REGRESSION	232194

The third stage, ‘SMCS procedure’, iteratively updates the latent particles using the learned intermediate distributions—constructed using both the context and pseudo representations—and the learned forward and backward transition kernels u and v . Finally, in ‘Compute predictive distribution’, we use the updated latent particles and their associated importance weights to compute the final predictive distribution over target outputs.

C Experimental Details

To support reproducibility, we provide our full experimental code as part of the supplementary material. Our implementation is based on the official codebase¹ of DANP [31], and all experiments were performed using PyTorch [3]. Training and evaluation were carried out on either an NVIDIA GeForce RTX 3090 or an RTX A6000 GPU. We optimized all models using the Adam optimizer [28], combined with a cosine annealing schedule for the learning rate.

Unless stated otherwise, we selected hyperparameters based on validation log-likelihood across tasks, using the following search spaces: learning rates from $\{5 \times 10^{-5}, 7 \times 10^{-5}, 9 \times 10^{-5}, 1 \times 10^{-4}, 3 \times 10^{-4}, 5 \times 10^{-4}\}$, weight decay values of $\{0, 1 \times 10^{-5}\}$, and batch sizes of $\{16, 32\}$.

C.1 Model structural details

In this section, we summarize the architectural details of NP, DANP in Table 4 and Table 6, and their respective variants when integrated with the TTSNP method in Table 5 and Table 7. For the NP and NP+TTSNP models, the description is based on the 1D GP regression task. In contrast, both DANP and DANP+TTSNP are designed to maintain a consistent structure and parameter count, regardless of changes in input or output dimensionality.

A key aspect to note is that TTSNP extends the base latent NP architecture by introducing two additional components: a permutation-invariant pseudo representation generator using the ISAB module [33] and a gradient estimator using a Multi-Layer Perceptron (MLP) structure that approximates the score function corresponding to the variational posterior constructed from the pseudo representations.

¹<https://openreview.net/forum?id=uGJxl2odR0>

Table 5: Model structure details of NP with TTSNP

CATEGORY	DETAILS
MODEL SPECIFICATIONS	
HIDDEN DIMENSION FOR DETERMINISTIC PATH	128
HIDDEN DIMENSION FOR LATENT PATH	128
MLP DEPTH FOR DETERMINISTIC PATH	4
MLP DEPTH FOR LATENT ENCODER PRE-MODULE	4
MLP DEPTH FOR LATENT ENCODER POST-MODULE	2
DECODER DEPTH	3
NUMBER OF MULTIHEAD ATTENTION BLOCKS IN ISAB	2
OUTPUT DIMENSION FOR EACH BLOCK	128
HIDDEN DIMENSION FOR GRADIENT ESTIMATOR	256
MLP LAYER FOR GRADIENT ESTIMATOR	3
NUMBER OF PARAMETERS FOR 1D GP REGRESSION	562818

1029 C.2 Evaluation Metric for the tasks

1030 Following the approach of Le et al. [29], we evaluate baseline models, such as NP and DANP, using
 1031 the normalized predictive log-likelihood, which is estimated through Monte Carlo sampling as:

$$\frac{1}{|t|} \sum_{k \in t} \log p(y_{j,k} | x_{j,k}, \mathcal{D}_{j,c}) \approx \frac{1}{|t|} \sum_{k \in t} \log \left(\frac{1}{K} \sum_{k=1}^K p(y_{j,k} | x_{j,k}, \mathbf{r}_j^{(k)}) \right), \quad (35)$$

$$\frac{1}{|c|} \sum_{k \in c} \log p(y_{j,k} | x_{j,k}, \mathcal{D}_{j,c}) \approx \frac{1}{|c|} \sum_{k \in c} \log \left(\frac{1}{K} \sum_{k=1}^K p(y_{j,k} | x_{j,k}, \mathbf{r}_j^{(k)}) \right), \quad (36)$$

1032 where $\mathbf{r}_j^{(k)}$ are sampled from the variational posterior $q(\theta_j | \mathcal{D}_{j,c})$, and c and t denote the context and
 1033 target points, respectively. However, for the SMC and TTSNP, we utilize our importance weight w s
 1034 when computing the predictive log-likelihood as follows:

$$\frac{1}{|t|} \sum_{k \in t} \log p(y_{j,k} | x_{j,k}, \mathcal{D}_{j,c}) \approx \frac{1}{|t|} \sum_{k \in t} \log \left(\sum_{k=1}^K \bar{w}_j^{(k)} p(y_{j,k} | x_{j,k}, \mathbf{r}_j^{(k)}) \right), \quad (37)$$

$$\frac{1}{|c|} \sum_{k \in c} \log p(y_{j,k} | x_{j,k}, \mathcal{D}_{j,c}) \approx \frac{1}{|c|} \sum_{k \in c} \log \left(\sum_{k=1}^K \bar{w}_j^{(k)} p(y_{j,k} | x_{j,k}, \mathbf{r}_j^{(k)}) \right), \quad (38)$$

1035 where $\bar{w}_j^{(k)}$ are the normalized importance weight for each sample $\mathbf{r}_j^{(k)}$ after the SMC procedures.

1036 C.3 Dataset details of n-dimensional GP Regression task

1037 To construct GP tasks for our experiments, we generate synthetic datasets using GPs equipped with
 1038 one of three commonly used kernels: the RBF kernel, the Matern 5/2 kernel, and the RQ kernel.
 1039 Each kernel introduces different inductive biases, allowing us to evaluate model robustness across a
 1040 diverse range of smoothness conditions and correlation structures.

The RBF kernel is defined as

$$k(\mathbf{x}, \mathbf{x}') = s^2 \exp \left(-\frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\ell^2} \right),$$

Table 6: Model structure details of DANP

CATEGORY	DETAILS
MODEL SPECIFICATIONS	
HIDDEN DIMENSION FOR LINEAR PROJECTION IN DIMENSION AGNOSTIC BLOCK	32
HIDDEN DIMENSION FOR SELF-ATTENTION IN DIMENSION AGNOSTIC BLOCK	32
HIDDEN DIMENSION FOR LATENT PATH TRANSFORMER	64
NUMBER OF LAYERS FOR LATENT PATH TRANSFORMER	2
HIDDEN DIMENSION FOR LATENT PATH SELF-ATTENTION	64
HIDDEN DIMENSION FOR LATENT PATH MLP LAYERS	128
NUMBER OF LAYERS FOR LATENT PATH MLP LAYERS	2
HIDDEN DIMENSION FOR DETERMINISTIC PATH TRANSFORMER	128
NUMBER OF LAYERS FOR DETERMINISTIC PATH TRANSFORMER LAYERS	6
NUMBER OF HEADS FOR DETERMINISTIC TRANSFORMER LAYERS	4
DECODER DEPTH	2
NUMBER OF PARAMETERS	334562

while the Matern 5/2 kernel is given by

$$k(\mathbf{x}, \mathbf{x}') = s^2 \left(1 + \frac{\sqrt{5}d}{\ell} + \frac{5d^2}{3\ell^2} \right), \quad \text{where } d = \|\mathbf{x} - \mathbf{x}'\|.$$

Both use randomly sampled output scales $s \sim \text{Unif}(0.1, 1.0)$ and lengthscales $\ell \sim \text{Unif}(0.1, 0.6)$, and additive Gaussian noise drawn from $p \sim \text{Unif}(0.1, 0.5)$ is applied to the outputs for numerical stability.

The Rational Quadratic kernel, which generalizes the RBF by integrating over a distribution of lengthscales, is defined as

$$k(\mathbf{x}, \mathbf{x}') = s^2 \left(1 + \frac{\|\mathbf{x} - \mathbf{x}'\|^2}{2\alpha\ell^2} \right)^{-\alpha},$$

where we fix the mixture parameter $\alpha = 1.0$. For its implementation, the lengthscale and output scale are sampled similarly from uniform distributions, specifically:

$$\ell = 0.1 + (0.6 - 0.1) \cdot \text{Uniform}(0, 1), \quad s = 0.1 + (1.0 - 0.1) \cdot \text{Uniform}(0, 1).$$

Given a batch of inputs x , the pairwise normalized distances are computed and scaled accordingly, and a small diagonal term $\sigma_\epsilon^2 I$ is added to the covariance matrix to ensure numerical stability.

In our n -dimensional GP regression setup, we design the data generation process to scale with input dimensionality, ensuring meaningful predictive inference across varying dimensions.

And for the number of context points $|c|$, we used the sampled number from the range $\text{Unif}(5n^2, 50n^2 - |c|)$ where n indicates the x dimension for the GP task. This quadratic scaling with n reflects the increased data requirements for higher-dimensional input spaces. Similarly, the number of target points $|t|$ is sampled from $\text{Unif}(5n^2, 50n^2 - |c|)$, maintaining a fixed upper limit on the total number of points per task. And, inputs $\mathbf{x} \in \mathbb{R}^n$ for both context and target sets are drawn independently from a uniform distribution over $[-2, 2]^n$.

This flexible GP data generation process allows us to simulate tasks with varying degrees of smoothness, structure, and input dimensionality, providing a rigorous testbed for evaluating uncertainty-aware inference methods.

Table 7: Model structure details of DANP with TTSNP

CATEGORY	DETAILS
MODEL SPECIFICATIONS	
HIDDEN DIMENSION FOR LINEAR PROJECTION IN DIMENSION AGNOSTIC BLOCK	32
HIDDEN DIMENSION FOR SELF-ATTENTION IN DIMENSION AGNOSTIC BLOCK	32
HIDDEN DIMENSION FOR LATENT PATH TRANSFORMER	64
NUMBER OF LAYERS FOR LATENT PATH TRANSFORMER	2
HIDDEN DIMENSION FOR LATENT PATH SELF-ATTENTION	64
HIDDEN DIMENSION FOR LATENT PATH MLP LAYERS	128
NUMBER OF LAYERS FOR LATENT PATH MLP LAYERS	2
HIDDEN DIMENSION FOR DETERMINISTIC PATH TRANSFORMER	128
NUMBER OF LAYERS FOR DETERMINISTIC PATH TRANSFORMER LAYERS	6
NUMBER OF HEADS FOR DETERMINISTIC TRANSFORMER LAYERS	4
DECODER DEPTH	2
NUMBER OF MULTIHEAD ATTENTION BLOCKS IN ISAB	2
OUTPUT DIMENSION FOR EACH BLOCK	64
HIDDEN DIMENSION FOR GRADIENT ESTIMATOR	192
MLP LAYER FOR GRADIENT ESTIMATOR	3
NUMBER OF PARAMETERS FOR 1D GP REGRESSION	475490

Input range shift As mentioned earlier, we conducted experiments using inputs x sampled from the range $[-2, 2]$. To evaluate how well TTSNP and baseline methods generalize under covariate shift, we introduced an input range shift scenario. Specifically, we constructed a validation set by modifying only the input domain while keeping all other GP generation settings identical to those used with the RBF kernel. In this shifted setting, inputs x were sampled from $[-3, 1]$, creating a distributional mismatch between training and evaluation. We then assessed the inference performance of each method on this shifted validation set to examine their robustness to covariate shifts in the input space.

Hyperparameter range shift In addition to the input range shift, we also evaluated the generalization ability of TTSNP and baseline methods under a kernel hyperparameter shift, another form of covariate shift where the statistical properties of the data generation process change between training and inference. Specifically, we designed a validation scenario in which only the range of hyperparameters used in the RBF kernel was altered, while keeping other settings—such as the number of data points and the input range—identical to the original training setup.

During training, RBF kernel parameters were sampled from $s \sim \text{Unif}(0.1, 1.0)$ and $\ell \sim \text{Unif}(0.1, 0.6)$. For the shifted validation set, we extended these ranges to $s \sim \text{Unif}(0.1, 2.0)$ and $\ell \sim \text{Unif}(0.1, 1.0)$, effectively increasing the variance and smoothness of the generated functions. We then performed inference on this new validation data to assess how well each method adapts when faced with unseen kernel configurations, thus testing robustness under structural distribution shifts.

1077 C.4 EMNIST Dataset

1078 For our experiments, we constructed a modified version of the EMNIST Balanced dataset ² [8], a
1079 widely used benchmark derived from the original NIST Special Database. The full dataset includes
1080 47 alphanumeric character classes, but we curated a 10-class subset for focused evaluation. From
1081 this selection, we sampled 24,000 images for training and 4,000 for testing.

1082 Each image is a grayscale digit or letter rendered in a 28×28 resolution, represented as a single-
1083 channel input. Pixel locations were linearly scaled to lie within the range $[-0.5, 0.5]$, and intensity
1084 values were normalized to the same interval. During training, for each episode, we randomly chose
1085 the number of context points $|c|$ from a uniform distribution over $[5, 45]$, and the number of target
1086 points $|t|$ was drawn from $\text{Unif}(5, 50 - |c|)$, ensuring a total of at most 50 points per task.

1087 **Corrupted EMNIST Dataset** For the Corrupted EMNIST dataset, we evaluated model robustness
1088 by applying three types of image corruptions to the EMNIST evaluation set: (1) *snow*, (2) *flip*, and
1089 (3) *brightness*.

1090 In the *snow* corruption, we randomly added white noise to the image by setting a subset of pix-
1091 els to the maximum intensity. Specifically, each pixel was independently selected to be corrupted
1092 with probability proportional to the corruption intensity. Formally, a binary mask was generated
1093 where each pixel had a probability $0.1 \times \text{intensity}$ of being set to 1, and the corrupted image was
1094 obtained by replacing the masked pixels with maximum brightness.

In the *brightness* corruption setting, we simulated intensity-based corruption by amplifying the
pixel values and clipping them to stay within valid bounds. Specifically, we applied the transforma-
tion

$$\text{data} = \text{clamp}(\text{data} \times (1 + \text{intensity}), 0, 1),$$

1095 which increases overall brightness proportionally to a given intensity factor, followed by clipping to
1096 ensure pixel values remain in the $[0, 1]$ range.

1097 For the *flip* corruption, to simulate geometric distortions, we randomly flipped the image along
1098 spatial dimensions. Each image has a 5% chance of being flipped horizontally and another indepen-
1099 dent 50% chance of being flipped vertically. This introduces structural variations while preserving
1100 pixel intensity distributions.

1101 C.5 CelebA Dataset

1102 For experiments involving natural image completion, we used the CelebA dataset ³ [35], a large-
1103 scale face dataset commonly used in generative modeling benchmarks. The dataset consists of
1104 162,770 training images, 19,867 for validation, and 19,962 for testing. All images were center-
1105 cropped and downsampled to 32×32 resolution with 3 RGB channels to reduce computational
1106 complexity while preserving key facial features.

1107 As in our EMNIST experiments, we transformed each image into a pixel-level function over spatial
1108 coordinates. Specifically, pixel coordinates were scaled to the range $[-0.5, 0.5]$ along both axes,
1109 and pixel intensity values in each RGB channel were normalized to the interval $[-0.5, 0.5]$. This
1110 setup enables the image completion task to be framed as a conditional regression problem: the
1111 model is provided with a subset of known pixel values (the context) and is tasked with predicting
1112 the remaining pixels (the targets).

1113 To simulate diverse conditioning scenarios, we sampled the number of context points $|c| \sim$
1114 $\text{Unif}(5, 45)$, and drew the number of target points $|t| \sim \text{Unif}(5, 50 - |c|)$, ensuring that each training
1115 episode contains a variable and realistic number of observed and queried pixels.

1116 D Additional experiments

1117 D.1 Ablation study

²<https://www.nist.gov/itl/products-and-services/emnist-dataset>

³<https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html>

Number of training data In this section, we evaluate the learning efficiency and convergence speed of TTSNP compared to the Fine-tune baseline by analyzing their target log-likelihood performance. Both methods start from the same pre-trained NP model trained on 1D GP data generated with an RBF kernel. We vary the number of training data batches from 10 to 1000 and measure performance using the mean and standard deviation of the target log-likelihood across five random seeds. As shown in Fig. 6, TTSNP achieves significantly faster convergence and consistently outperforms the Fine-tune method, even with substantially fewer training samples. This result shows that our method not only samples more improved latent variables but is also more efficient to train compared to the Fine-tune.

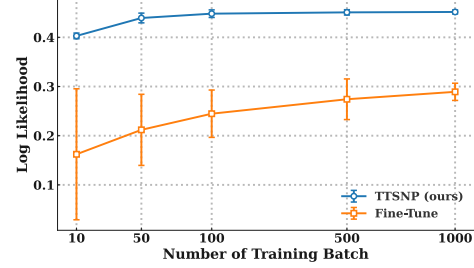


Figure 6: Comparison of target log-likelihood between TTSNP and the baseline ‘Fine-tune’ under varying amounts of training data, using the same number of samples.

Ablation on training objective In this section, we conduct an ablation study on the training objectives used in our method. As described in § 3.3, our full training loss consists of two components: (1) the KL divergence between two path measures, denoted as \mathcal{L}_{KL} , and (2) the log-likelihood maximization loss, denoted as \mathcal{L}_{LL} . To understand the individual contribution of each objective, we evaluate models trained with only one of the two loss terms.

As shown in Table 8, both \mathcal{L}_{KL} and \mathcal{L}_{LL} improve sample quality over the baseline variational posterior from the pre-trained model, highlighting their effectiveness in enhancing the efficiency and diversity of particles in the learned SMCS procedure. However, the best performance is achieved when both objectives are combined, as in TTSNP. This demonstrates that \mathcal{L}_{KL} and \mathcal{L}_{LL} are complementary, working together to improve posterior approximation and overall inference quality.

Table 8: Log-likelihood results for the ablation study on training objectives. In the second and third rows, the notation $-A$ indicates that the objective A was excluded from the training loss during model training.

Model	RBF	
	context	target
TTSNP (ours)	0.893 \pm 0.001	0.430 \pm 0.001
- \mathcal{L}_{KL}	0.880 \pm 0.001	0.420 \pm 0.001
- \mathcal{L}_{LL}	0.872 \pm 0.001	0.415 \pm 0.001
SMCS	0.868 \pm 0.001	0.405 \pm 0.001

Number of samples In this subsection, following the setup in § 3.1, we perform an ablation study on the number of samples used during inference. As in other experiments described in § 4, we include Pre-trained, Fine-tune, and SMCS as baselines for comparison. Both TTSNP and SMCS are evaluated with $T = 10$ inference steps.

The experiment is conducted using a pre-trained NP model trained on 1D Gaussian Process data with an RBF kernel. For SMCS, we apply the Unadjusted Langevin Algorithm as the transition kernel to sample from the posterior during inference. The results in Fig. 7 show that TTSNP achieves faster convergence compared to other baselines as the number of samples increases. This demonstrates that the learned intermediate distributions—constructed using pseudo context representations—along with the learned forward and backward transitions, effectively guide latent samples toward the true posterior while capturing diverse modes, resulting in improved sample efficiency.

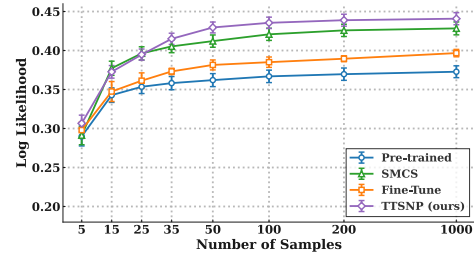


Figure 7: Comparison of target log-likelihood between TTSNP and the baselines ‘Pre-trained’, ‘SMCS’, and ‘Fine-tune’ using NP model trained with GP data generated by RBF kernel. Here, we compare the log-likelihood under the same varying amounts of latent samples.

Ablation on the Model structure In this section, we conduct an ablation study on the architectural design of the drift functions u and v . As detailed in [Appendix B.2](#), our default implementation defines the drift functions as follows:

$$u(\mathbf{r}_t, \tau_t) := \frac{\sigma^2}{2} (\nabla \log \tilde{\pi}_t(\mathbf{r}_t) + (1 - \beta_t) \text{NN}(\mathbf{r}_t, t, \mathcal{D}_c, \mathcal{D}_p)), \quad (39)$$

$$v(\mathbf{r}_t, \tau_t) := -\frac{\sigma^2}{2} (\nabla \log \tilde{\pi}_t(\mathbf{r}_t) + (1 - \beta_t) \text{NN}(\mathbf{r}_t, t, \mathcal{D}_c, \mathcal{D}_p)), \quad (40)$$

where $\text{NN}(\cdot)$ is implemented as a 3-layer MLP and is designed to approximate the score function $\nabla \log \tilde{q}(\mathbf{r} \mid \mathcal{D}_c \cup \mathcal{D}_p)$ of the intermediate distribution constructed from the context and pseudo context representations.

While this approach relies on a neural network to approximate the full score function, an alternative formulation can be derived by analytically decomposing the intermediate distribution using the Gaussian score form. In particular, we consider the following decomposition:

$$\nabla \log \pi_t(\mathbf{r}) := \beta_t \nabla \log p(\mathbf{r} \mid \mathcal{D}_c) + (1 - \beta_t) \nabla \log \tilde{q}(\mathbf{r} \mid \mathcal{D}_c) \quad (41)$$

$$+ (1 - \beta_t) \nabla \log \tilde{q}(\mathbf{r} \mid h(\mathcal{D}_c, t, \varepsilon_t) \cup \mathcal{D}_p), \quad (42)$$

$$= \nabla \log \tilde{\pi}_t(\mathbf{r}) + (1 - \beta_t) \frac{\mathbf{r} - \mu_{\text{NN}}(\mathcal{D}_c, \mathcal{D}_p, t)}{\sigma_{\text{NN}}^2(\mathcal{D}_c, \mathcal{D}_p, t)} \quad (43)$$

where $\tilde{\pi}_t(\mathbf{r})$ denotes the part of the intermediate distribution excluding contributions from the pseudo context, and the neural network models the additional gradient signal induced by the pseudo context generator h by modeling the mean μ_{NN} and the covariance σ_{NN}^2 of the Gaussian score. Also, more simply, we can design as follows:

$$\nabla \log \pi_t(\mathbf{r}) = \nabla \log \tilde{\pi}_t(\mathbf{r}) + (1 - \beta_t) \frac{\mathbf{r} - \mu_{\text{NN}}(\mathcal{D}_c, \mathcal{D}_p, t)}{\sigma^2}, \quad (44)$$

where σ^2 is the fixed variance, which we used for calibrating variational posteriors.

In this section, we compare the log-likelihood performance of TTSNP against the alternative drift modeling strategies described above, using 1D GP data generated with an RBF kernel. All methods share the same pre-trained NP base model, allowing for a fair evaluation of how different formulations of the drift function impact inference quality. We ensure a fair comparison by using the same training objectives, number of training samples, and number of inference steps across TTSNP and all other model variants—making the drift model structure the only varying factor. As shown in [Table 9](#), while alternative neural network designs also improve sample quality and log-likelihood performance compared to SMCS, TTSNP achieves the most significant improvement, highlighting the effectiveness of its architecture in modeling the score function and guiding posterior refinement.

D.2 Additional GP regression experiments

In [Table 10](#), we extend the GP regression experiments beyond those presented in [§ 4.1](#) and [§ 4.2](#) by exploring additional settings. Specifically, we first replicate the experiments originally conducted on 1D GP data in [§ 4.1](#), but now using 2D GP data. This allows us to empirically demonstrate that our method remains effective and directly applicable even when the input dimensionality increases. We first extend the **Sample Quality** experiment described in [§ 4.1](#) to a more challenging setting using 2D GP data generated with an RBF kernel. The results are reported in the left table of [Table 10](#). While the overall log-likelihood values are lower compared to the 1D case—due to the increased complexity of the 2D input space—the performance trends remain consistent. Specifically, TTSNP

Table 9: Log-likelihood results for the ablation study on drift model structures. In the second and third rows, the notations ‘Mean and Variance’ and ‘Mean’ indicate that the neural network is used to model both the mean and variance of the Gaussian score function, and only the mean, respectively.

Model	RBF	
	context	target
TTSNP (ours)	0.893 ± 0.001	0.430 ± 0.001
Mean and Variance	0.892 ± 0.002	0.420 ± 0.002
Mean	0.886 ± 0.002	0.421 ± 0.002
SMCS	0.868 ± 0.001	0.405 ± 0.001

Table 10: Log-likelihood results for Sample Quality (left) and Matern to RQ (right) scenarios.

Model	RBF		Model	Matern		RQ	
	context	target		context	target	context	target
Pre-train	-0.162 \pm 0.008	-0.392 \pm 0.010	Pre-train	-0.211 \pm 0.002	-0.462 \pm 0.003	0.299 \pm 0.004	0.084 \pm 0.005
Fine-tune	-0.156 \pm 0.005	-0.384 \pm 0.007	Fine-tune	-0.202 \pm 0.003	-0.440 \pm 0.005	0.310 \pm 0.004	0.092 \pm 0.005
SMCS	-0.154 \pm 0.004	-0.367 \pm 0.004	SMCS	-0.190 \pm 0.002	-0.432 \pm 0.002	0.318 \pm 0.003	0.104 \pm 0.004
TTSNP (ours)	-0.156 \pm 0.004	-0.351 \pm 0.005	TTSNP (ours)	-0.190 \pm 0.002	-0.420 \pm 0.001	0.316 \pm 0.003	0.110 \pm 0.003

continues to improve sample quality and enables efficient inference, confirming its effectiveness beyond simple 1D tasks and into higher-dimensional scenarios.

We also replicate the **Matern to RQ** experiment from § 4.1 in the 2D setting. The corresponding results are shown in the right table of Table 10. Again, we observe similar performance patterns to those seen in the 1D experiments, demonstrating that TTSNP enhances particle quality even in few-shot adaptation settings and generalizes effectively to unseen tasks. These results highlight the robustness of the learned intermediate distributions and transition kernels, even under high-dimensional covariate shift conditions.

E Broader Impact

This paper identifies a key limitation in latent Neural Process models: after pre-training, the global latent variables tend to be miscalibrated, resulting in low-quality samples from the variational posterior. To address this, we propose a method based on a Sequential Monte Carlo sampler that refines these latent samples to better match the true posterior distribution. While this approach introduces additional computational cost at test time compared to standard inference with pre-trained models, the cost is adjustable based on user preferences and is justified by the significant improvement in probabilistic inference quality. Aside from this computational trade-off, the proposed method poses no known negative societal impacts.