Predictive Deep Sets

Alex Hämäläinen
ELLIS Institute Finland;
Department of Computer Science,
Aalto University, Finland

Sammie Katt
ELLIS Institute Finland;
Department of Computer Science,
Aalto University, Finland

Samuel Kaski
ELLIS Institute Finland;
Department of Computer Science,
Aalto University, Finland;
Department of Computer Science,
University of Manchester

Abstract

Amortized meta-learning methods, such as neural processes, promise near-instantaneous inference on new labeled datasets encountered during downstream tasks. Recent adaptations of the transformer architecture have propelled these approaches to impressive performance in tasks like function estimation, parameter inference, and decision-making. Curiously, their success still primarily stems from the expressiveness of transformers, lacking a bias for modeling the functional structures between features and labels shared across datasets. We argue and show that this leads to training sample inefficiency and sub-optimal performance, and address this by introducing a novel set encoding technique called *Predictive Deep Sets*. Our approach exploits a strong bias towards functional structures by meta-learning an RKHS that captures domain-critical functional patterns, and by representing datasets as optimal fit functions within this space. Besides providing theoretical justification for this approach, we empirically demonstrate orders of magnitude increases in training data sample efficiency compared to strong baselines across various settings.

1 INTRODUCTION

Amortized meta-learning re-frames prediction as quick, task-specific inference. Instead of learning a model independently for each new task, the idea is to learn a map directly from labeled task-specific datasets to output predictions. Recently, this approach has demonstrated impressive capabilities in form of tabular foundation models (Hollmann et al., 2022, 2025), and in applications to decision-making (Galashov et al., 2019; Huang et al., 2024), data acquisition (Huang et al., 2025), and experimental design (Blau et al., 2023).

This approach can be studied under the framing of *prediction map* learning (Foong et al., 2020; Markou et al., 2022), which covers solutions like Neural Processes (NPs) (Jha et al., 2022) and Prior-Data Fitted Networks (PFN) (Müller et al., 2021), but also instantiations to parameter inference (Chang et al., 2024). Current state-of-the-art prediction maps, including Transformer Neural Processes (TNP) (Nguyen and Grover, 2022) Amortized Conditioning Engine (ACE) (Chang et al., 2024), and TabPFN (Hollmann et al., 2022) capitalize on the expressivity and universality of the popular transformer architecture (Vaswani, 2017). Although the approach is effective, transformers lack a structural sensitivity and inductive bias towards capturing the functional patterns and geometries shared across datasets' features and labels. We argue, and empirically show, that this leads to significant sample inefficiency during training and sub-optimal performance.

We address this by introducing a new encoding technique for prediction map learning, called *Predictive Deep Sets* (PDS). PDS represents labeled datasets as optimal fit functions within a deep Reproducing Kernel Hilbert Space (RKHS) (Berlinet and Thomas-Agnan, 2011; Wilson et al., 2016). By meta-learning the RKHS over datasets, we explicitly learn the basis of domain-critical function patterns, while also encoding features like smoothness and interpolation. Optimal fit functions from this space double as well-performing function estimators, and informative representations.

In our empirical evaluations[1], we instantiate PDS for

Proceedings of the 29th International Conference on Artificial Intelligence and Statistics (AISTATS) 2026, Tangier, Morocco. PMLR: Volume 300.

[1] Code is made available here.

function prediction and parameter estimation tasks. We find that PDS consistently outperforms strong transformer prediction map baselines by achieving orders of magnitude higher training data efficiency, while retaining similar computational scaling. In summary, our contributions are:

- We provide a theoretical justification for representing functions on labeled datasets using RKHS loss minimizers.
- Building on this result, we propose a new set encoding technique, Predictive Deep Sets (PDS), with realizations to both function prediction and parameter inference.
- We empirically study the benefits of PDS using benchmarks and several case-studies, including realistic challenges from cognitive science.

2 BACKGROUND

Before introducing our approach, we first review the relevant background on prediction maps (Section 2.1) and set function learning (Section 2.2).

2.1 Prediction Map Learning

Prediction map models (Foong et al., 2020; Markou et al., 2022; Chang et al., 2024) are a broad family of approaches that learn representations of functions based on labeled datasets. Prediction maps are were originally proposed as a framing for formalizing Neural Process (NP) problems (Foong et al., 2020; Markou et al., 2022) and have later extended for alternative problems; we begin by introducing them in the NP setting, and then follow-up with a more general definition.

NPs (Garnelo et al., 2018b; Jha et al., 2022) seek to quickly predict *target* values of a new function given some *context* dataset from the same function. Formally, we assume there is a distribution over functions $p(f)$, where each function maps inputs $x \in \mathcal{X}$ to outputs $y \in \mathcal{Y}$ with i.i.d. noise: $y = f(x) + \epsilon$. Based on datasets from numerous functions, the task is to learn a model to predict the distribution $p(T \mid Q, Z)$ of target labels $T = \{y_j\}_j$ at query inputs $Q = \{x_j\}_j$, conditioned on the context dataset $Z = (X, Y) = \{(x_i, y_i)\}_i$. The NP approximation $\pi(T; Q, Z)$ of this distribution is also called a *prediction map*.

While NPs strictly define the queries Q and targets T to match such function prediction tasks, in prediction maps one can more generally define Q and T also as other quantities depending on the downstream task. For instance, the prediction map formulation has been extended to parameter inference (Chang et al., 2024). Here, one might observe a dataset from a Gaussian Process and want to infer its kernel length- and output scales. For such problems, the target T equals the parameters of interest as $\theta \in \Theta$ and the query Q is fixed; as proposed by Chang et al. (2024), one can also introduce a $|\theta|$-dimensional learnable token as Q.

Like NPs, most prediction map approaches approximate the distribution over targets by meta-learning from multiple tuples (T, Q, Z), i.e., *tasks*, seen during training. Here one assumes that each task corresponds to an independent sample f (and θ) underlying the tuples from some $p(f)$ (or $p(f, \theta)$). Most concrete prediction map designs follow the decomposition

$$\pi\big(T; Q, Z\big) = q\big(T; Q, r(Z)\big) \quad \big(\approx p(T \mid Q, Z)\big), \quad (1)$$

where r constructs a fixed-size representation of the current task Z (and optionally also Q) to parametrize the distribution q. This formulation enables end-to-end learning with a straightforward maximum-likelihood objective by conditioning on datasets directly.

2.2 Learning Functions on Sets

One of the most critical design choices of a prediction map model is the observation set encoding $r(Z)$ in Eqn. 1. Today, the most prediction map architectures build this encoding by leveraging approaches to *set function learning* (Zaheer et al., 2017; Murphy et al., 2018; Wagstaff et al., 2019). Set functions are permutation-invariant functions that map finite, varying-sized observation sets[2] $S \in 2^{\mathcal{S}}$ into fixed size outputs $\mathbb{R}^d$. For prediction map learning, set functions analogously allow for dealing with varying sized labeled input sets $Z = (X, Y)$ while incorporating a critical inductive bias of invariance to the permutations of Z.

Advances in set function model development have quickly translated into new developments in prediction map models. The earliest instances, (Conditional) NPs (Garnelo et al., 2018a,b), were built by applying Deep Sets Zaheer et al. (2017), and recently followed by more powerful attention-based variants (Kim et al., 2019) and solutions leveraging various equivariances and symmetries (Gordon et al., 2019; Huang et al., 2023). Today, the most prominent prediction map models, such TNPs (Nguyen and Grover, 2022), are largely based on leveraging the expressivity of transformers (Vaswani, 2017).

3 PREDICTIVE DEEP SETS

Despite the signature assumption of prediction map learning — that the Z are labeled datasets from a

[2]We use S to denote any set and Z for labeled sets.

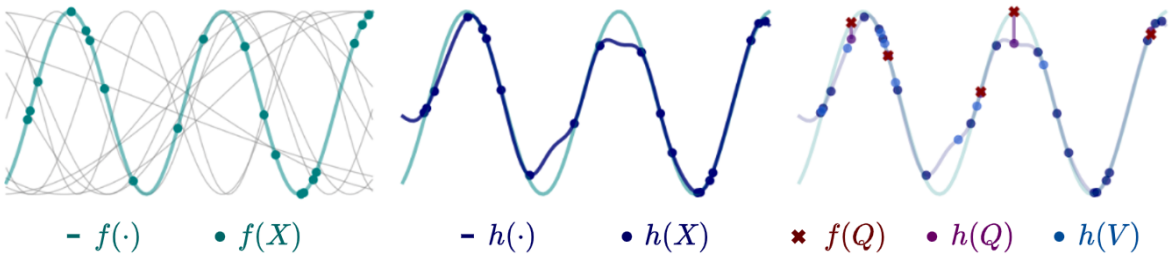


Figure 1: Simple conceptual illustration of PDS with sinusoidal functions. **Left:** In this example task, we observe a dataset $Z = (X, Y)$ from function f (teal) at a set of locations X. The gray lines illustrate alternative functions from the task distribution. **Middle:** Predictive set representation $h(\cdot)$ of the dataset is obtained by solving an optimization objective within a meta-learned RKHS. **Right:** The representation $h(\Xi)$ discretized along evaluation domain $\Xi = [X, V, Q]$ (detailed in Section 3.3) inherits the structural bias of the RKHS; learning to predict $f(\cdot)$ at target locations Q reduces to learning small corrections on $h(Q)$ conditioned on $h(\Xi)$.

distribution of functions — modern prediction maps lack any inductive bias that makes proper use of this assumption. We address this with an approach that exploits this assumption by learning a deep RKHS where each dataset are then represented as the optimal function. This brings a heavy inductive bias, allowing the data-driven learning focus on mainly fine-tuning these representations for prediction (see Fig. 1).

In Section 3.1, we discuss the principles of representing sets as optimal fit RKHS functions, and their practical and computationally affordable approximation via functional gradients. Sections 3.2 and 3.3 contain, respectively, our core contributions in the form of the theoretical justification of the approach, and its application to prediction map learning.

3.1 Predictive Set Representations

We aim to encode datasets as function representations that minimize an empirical loss functional; we call such encodings *predictive set representations*. Before formal definition, we voice our core assumptions about the datasets and their structure by defining functional sets.

Definition 1 (Functional sets.)**.** *Let $C_b(\mathcal{X}, \mathcal{Y}) \ni f : \mathcal{X} \to \mathcal{Y}$ denote a space of bounded and continuous functions f with a compact support on $\mathcal{X}$ and let ϵ_i be observation noise. We call a finite set $Z = [(x_i, y_i)]_i$ a functional set iff*

$$\exists f \in C_b(\mathcal{X}, \mathcal{Y}),\ \ s.t.\ \forall i : y_i = f(x_i) + \epsilon_i. \tag{2}$$

We then call the space $2^{\mathcal{Z}}$ of all such sets the functional set space of $C_b(\mathcal{X}, \mathcal{Y})$ iff it decomposes as a finite union of fixed-size components $\bigcup_{n=1}^{N} 2^{\mathcal{Z}}_n$ for some finite N, and that for each n, $2^{\mathcal{Z}}_n$ is topologically closed and closed under permutations.[3]

[3]In other words, we assume $2^{\mathcal{Z}}$ behaves nicely at limits.

We then define predictive set representations as follows:

Definition 2 (Predictive set representations.)**.** *Let $Z \in 2^{\mathcal{Z}}$ be a functional set, $\mathcal{L} : C_b(\mathcal{X}, \mathcal{Y}) \times 2^{\mathcal{Z}} \to \mathbb{R}_+$ a loss functional*[4]*, and $g : \mathbb{R}_+ \to \mathbb{R}$ a regularizer. Call $h^\star \in C_b(\mathcal{X}, \mathcal{Y})$ a predictive set representation of Z if it minimizes the regularized loss:*

$$h^\star = \arg\min_{h \in C_b(\mathcal{X}, \mathcal{Y})} \mathcal{L}(h, Z) + g(\|h\|_{C_b(\mathcal{X}, \mathcal{Y})}). \tag{3}$$

Our goal is to build an encoding that maps any functional set Z to its predictive set representation $h^\star$. The optimization problem of obtaining $h^\star$ can be made more tangible by working with functions $h \in \mathcal{H}_k$ belonging to an RKHS $\mathcal{H}_k \subseteq C_b(\mathcal{X}, \mathcal{Y})$ with a reproducing kernel k. In particular, this RKHS assumption allows for estimate $h^\star$ with functional gradient descent (Mason et al., 1999) on the RKHS $\mathcal{H}_k$ (see the supplement of Xu et al., 2020 for an excellent overview on the topic). In summary, given the definition of a functional RKHS gradient $\nabla_{\mathcal{H}_k} \hat{\mathcal{L}}(h, Z) \coloneqq K(\cdot, X) \partial \hat{\mathcal{L}}(h, Z) / \partial h$ w.r.t. a regularized loss functional $\hat{\mathcal{L}}(h, Z) = \mathcal{L}(h, Z) + g(\|h\|_{\mathcal{H}_k})$, $h^\star$ can be obtained through gradient descent on $\mathcal{H}_k$. For this, one can derive the following update rule:

$$h^{(m+1)} = h^{(m)} - \eta \nabla_{\mathcal{H}_k} \hat{\mathcal{L}}\big(h^{(m)}, Z\big), \tag{4}$$

where η is the step size, i.e., equivalent of learning rate in standard parametric gradient descent. The initial $h^{(0)} \in \mathcal{H}_k$ can be e.g. learned or fixed to a constant.

Assuming that the gradient $\nabla_{\mathcal{H}_k} \hat{\mathcal{L}}(h, Z)$ can be computed, i.e., that $\hat{\mathcal{L}}$ is differentiable, the functional gradient descent scheme of Eqn. 4 is guaranteed to converge to $h^\star$ at the limit $\lim_{M \to \infty} h^{(M)} = h^\star$ for convex $\hat{\mathcal{L}}$, with early stopping emulating Tikhonov-style regularization (Raskutti et al., 2014; Ustimenko et al., 2022).

For our purposes, this means that the RKHS gradient descent scheme yields well-behaving estimates for

[4]We use the shorthand $\mathcal{L}(h, Z)$ for $\mathcal{L}\big(h(X), Y\big)$.

the predictive set representation $h^\star$ without the need for closed-form solutions (which are computationally expensive here). In particular, obtaining $h^{(M)} \approx h^\star$ comes with a reasonable computational complexity of $\mathcal{O}(M|Z|^2 d_y)$, which matches that of transformer-based prediction maps ($= \mathcal{O}(l|Z|^2(d_x + d_y))$ with l layers).

3.2 Predictive Deep Sets

As discussed in Section 2, prediction map learning seeks to estimate mappings π from query sets $Q \in 2^{\mathcal{Q}}$ to (distributions over) target sets $T \in 2^{\mathcal{T}}$, by conditioning on sets $Z = (X, Y) \in 2^{\mathcal{Z}}$. We study the construction of π (Eqn. 1) using predictive set representations.

Formally, let $r_{\mathcal{H}_k} : 2^{\mathcal{Z}} \to \mathcal{H}_k$ denote an encoder that maps functional sets to their respective predictive set representation approximations $h^{(M)} \in \mathcal{H}_k$ with M gradient steps in RKHS $\mathcal{H}_k$. Let $\rho_{\mathcal{H}_k} : 2^{\mathcal{Q}} \times 2^{\mathcal{X}} \times \mathcal{H}_k \to \mathcal{P}(2^{\mathcal{T}})$ denote a flexible, parametric decoder that maps the queries into a distribution over the targets by conditioning on predictive set representations $h^{(M)}$ at locations $X \in 2^{\mathcal{X}}$ while being invariant to X permutations. Given $r_{\mathcal{H}_k}$ and $\rho_{\mathcal{H}_k}$, we define Predictive Deep Sets (PDS) as prediction maps $\Phi_{\mathcal{H}_k}$ of form

$$\Phi_{\mathcal{H}_k}(T; Q, Z) = \rho_{\mathcal{H}_k}(T; Q, X, r_{\mathcal{H}_k}(Z)). \tag{5}$$

The following Proposition 3 shows that this design can approximate any prediction map.

Proposition 3 (Predictive deep sets)**.** *Let $Z = (X, Y) = [(x_i, y_i)] \in 2^{\mathcal{Z}}$ denote a functional set, $Q \in 2^{\mathcal{Q}}$ and $T \in 2^{\mathcal{T}}$ denote the query and the target, and k be a continuous, positive definite reproducing kernel of RKHS $\mathcal{H}_k$. Further, let $r_{\mathcal{H}_k} : 2^{\mathcal{Z}} \to \mathcal{H}_k$ denote a map from sets $Z \in 2^{\mathcal{Z}}$ to their approximate predictive set representations with $M < \infty$ RKHS gradient steps w.r.t. MSE-loss. Assume sufficiently small step-size η, $h^{(0)} \in \mathcal{H}_k$ independent of Y, and $\forall i \neq j : x_i \neq x_j$. Then, a continuous function $\pi : 2^{\mathcal{Z}} \times 2^{\mathcal{Q}} \to \mathcal{P}(2^{\mathcal{T}})$ is invariant to permutations of Z if and only if it permits a representation of form*

$$\pi(T; Q, Z) = \rho_{\mathcal{H}_k}\big(T; Q, X, r_{\mathcal{H}_k}(Z)\big), \tag{6}$$

with suitable $\rho_{\mathcal{H}_k} : 2^{\mathcal{Q}} \times 2^{\mathcal{X}} \times \mathcal{H}_k \to \mathcal{P}(2^{\mathcal{T}})$.

See Appendix A.1 for proof. In summary, the proof builds on an analogous strategy as the Theorem 1 of Gordon et al. (2019). Our unique contribution is to show that it is possible to homeomorphically embed fixed-sized functional sets Z into predictive set representation evaluations $r_{\mathcal{H}_k}(Z)(X)$. This result then extends to the full scope of Prop. 3 via applying analogous arguments as in Gordon et al. (2019). The *suitability* of $\rho_{\mathcal{H}_k}$ means that it must be a powerful and continuous function approximator that is invariant to X permutations.

Remark (Multiplicities)**.** *Proposition 3 assumes that the sets $Z \in 2^{\mathcal{Z}}$ have a multiplicity of one, i.e., they contain each unique x_i no more than once. We note that Gordon et al. (2019) show that it is possible to design injective encodings that can deal with higher multiplicities; complementing our $r_{\mathcal{H}_k}$ with such encodings allows for extending our result and approach to higher multiplicities as well.*

Remark (Alternative loss functions)**.** *Proposition 3 assumes MSE-loss mainly due to its prominence, and convenience for the proof. We however suspect that the result should extend to many smooth and strongly convex losses under mild conditions. We still advise confirming the exact conditions on a case-by-case basis.*

3.3 Practical implementation

This section describes the design for scaling the decomposition of Eqn. 6 into an effective prediction map model. Figure 2 gives and overview of the proposed PDS architecture. In the following, we introduce the design (and motivation) of each of these components, including an adaptive design for the RKHS $\mathcal{H}_k$.

Adaptive RKHS design. The utility of the predictive set representations depends on the RKHS $\mathcal{H}_k$ via the design of its reproducing kernel k. While some tasks might allow for tailoring k using domain-specific knowledge, this is still usually difficult in practice, and the RKHS might miss critical domain-specific patterns. To combat this, we take a more task-agnostic approach in the form of deep kernel transfer (Patacchiola et al., 2020; Tossou et al., 2019). To be precise, given a universal kernel k (Micchelli et al., 2006) with hyperparameters $\kappa \in \mathcal{K}$, we use a deep kernel of the form

$$\psi(k)(\cdot, \cdot) = k_{\phi_k(Z)}\big(\phi_x(\cdot), \phi_x(\cdot)\big), \tag{7}$$

where $\phi_x : \mathcal{X} \to \mathbb{R}^d$ denotes a neural network and $\phi_k : 2^{\mathcal{Z}} \to \mathcal{K}$ is a set function mapping datasets to the kernel hyperparameters. Meta-learning ϕ_x and ϕ_k end-to-end according to the downstream task enables the RKHS to adapt to domains' function patterns.

Learnable residual corrections. While the usage of transfer kernels allows for identifying domain-critical functional patterns across datasets, the networks ϕ_x and ϕ_k would need to capture these structures via a single forward pass. In order to increase model expressivity, we divide this workload across multiple steps via learnable residual corrections, an approach that is related to strategies explored in learnable optimizers (Andrychowicz et al., 2016; Wichrowska et al., 2017) and gradient preconditioning (Kang et al., 2023).

Specifically, we introduce learnable residual transfor-

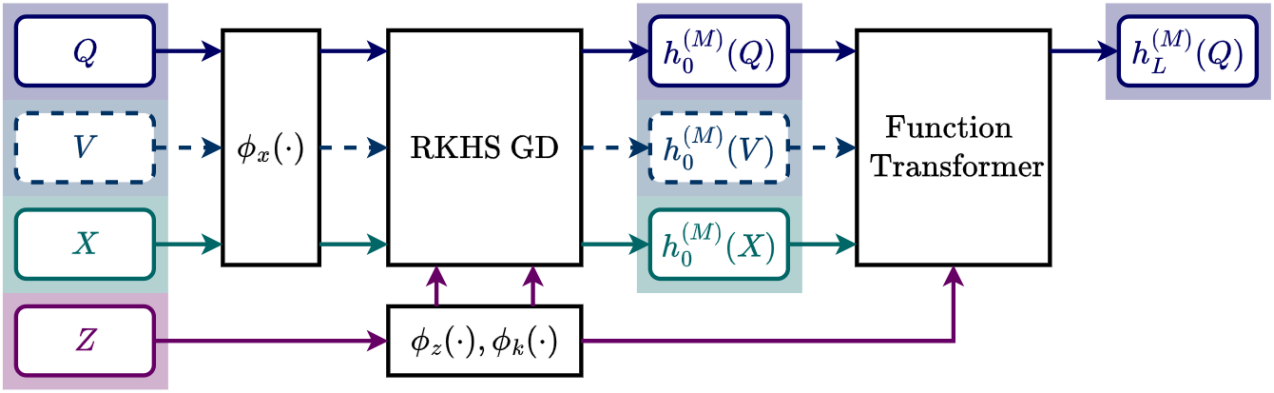


Figure 2: Overview illustration of the proposed PDS architecture. The model takes the dataset $Z = (X, Y)$ as an input, together with the learned support vectors V and the target queries Q. The RKHS GD scheme is then performed on deep embeddings of these inputs, and is followed by a function transformer, which maps discretizations of the predictive set representations into a representation that parametrizes the output distribution.

mations ω_1, ω_2 in the RKHS gradient updates (Eqn.4):

$$h^{(m+1)} = h^{(m)} - \eta\big(\delta^{(m)} + \omega_1(\delta^{(m)})\big)$$
$$\delta^{(m)} = \nabla_{\mathcal{H}_{\psi(k)}} \hat{\mathcal{L}}\big(h^{(m)} + \omega_2(h^{(m)}), (\phi_x(X), \phi_z(Z))\big). \tag{8}$$

Here ω_1 warps the gradient update $\delta^{(m)}$ while ω_2 can further tune the RKHS $\mathcal{H}_{\psi(k)}$ by re-embedding $h^{(m)}$ at each iteration. Meta-learning the mappings ω_1, ω_2 allows the predictive set representations to adapt to each dataset via a number of steps, increasing overall expressivity. For further increases, we additionally use the deep element-wise embeddings $(\phi_x(X), \phi_z(Z))$ instead of Z for computing $\hat{\mathcal{L}}$. Although ω_1 and ω_2 softly break the guarantee that Eqn. 8 descends $\hat{\mathcal{L}}$ in $\mathcal{H}_{\psi(k)}$, allowing for more flexible representations when needed, their residual nature helps keeping these updates still closely anchored to true descent by default.

Decoder design. The PDS decoder $\rho_{\mathcal{H}_k} : 2^{\mathcal{Q}} \times 2^{\mathcal{X}} \times \mathcal{H}_{\psi(k)} \to \mathcal{P}(2^{\mathcal{T}})$ transforms the queries $Q \in 2^{\mathcal{Q}}$ into a distribution over targets $T \in 2^{\mathcal{T}}$ by conditioning on the predictive set representations $h^{(M)}$ obtained with Eqn. 8 and input locations $X \in 2^{\mathcal{X}}$. Proposition 3 suggests that $\rho_{\mathcal{H}_k}$ should be a flexible and powerful approximator that is invariant to permutations of X. To match this, we implement $\rho_{\mathcal{H}_k}$ using the decomposition

$$\rho_{\mathcal{H}_k}(T; Q, X, h^{(M)}) = q\big(T \mid \mathcal{F}_{\mathcal{H}_k}(Q, X, h^{(M)})\big), \tag{9}$$

where q is the conditional output distribution over targets T, and $\mathcal{F}_{\mathcal{H}_k}$ is a learnable transformation. We call our design for $\mathcal{F}_{\mathcal{H}_k}$ a *function transformer*, which is an adaptation of transformer prediction maps tailored to learn a sequence of residual transformations on the functional representation $h^{(M)}$. To facilitate this, the functional transformer modifies the standard self-attention (Vaswani, 2017) as:

$$\mathcal{A}(h; \psi(k))(\cdot) = \mathrm{SM}\left(\frac{h(\cdot)h^T(\cdot)}{\sqrt{d_h}} + \log \psi(k)(\cdot, \cdot)\right) h(\cdot), \tag{10}$$

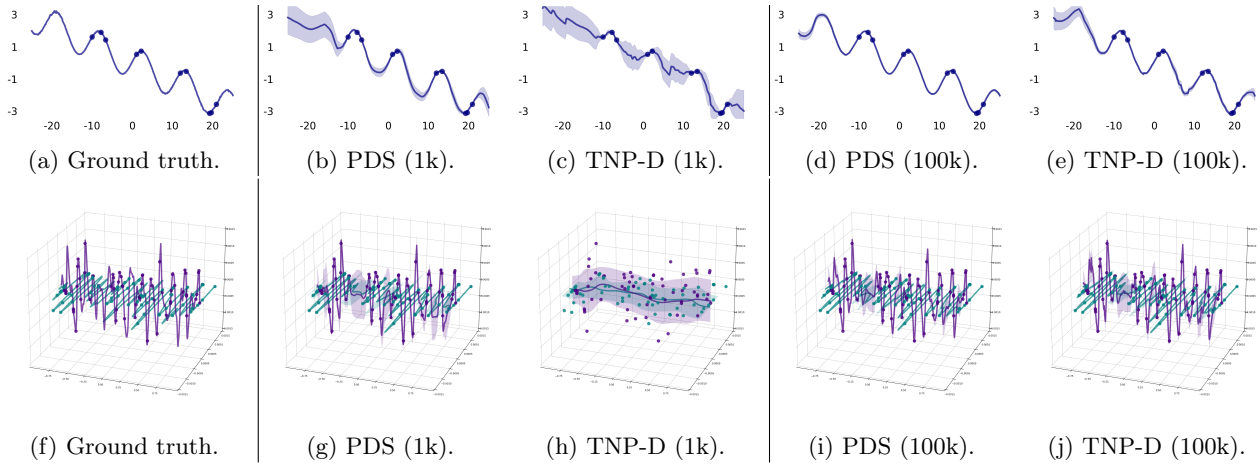
where $h \in \mathcal{H}_{\psi(k)}$ is the input function, $\psi(k)$ is the deep kernel of Eqn. 7, and SM is the softmax function.

Eqn. 10 is a variant of the standard attention mechanism that incorporates the kernel term $\log \psi(k)(\cdot, \cdot)$ to allow the attention weights to depend on geometric information over the domain $\mathrm{supp}(h)$. For practical computation, the functions h are discretized into manageable finite-dimensional vectors $h(\Xi)$ along with an evaluation domain $\Xi = \phi_x([X, V, Q]) \in \mathrm{supp}(h)$, where $V \in \mathcal{X}$ are meta-learnable support vectors.

The full function transformer model is a stack of L transformer encoder layers, each decorated with a multi-head structure over $\mathcal{A}$ (Eqn. 10) and skip-connections between the layers. The attention is masked similarly as in transformer prediction maps (Nguyen and Grover, 2022; Chang et al., 2024). The final output at Q parametrizes the output distribution in Eqn. 9.

Putting everything together. The concrete PDS design follows the decomposition of Eqn. 5 where $\rho_{\mathcal{H}_k}$ follows the definition of Eqn 9 and $r_{\mathcal{H}_k}(Z) = h^{(M)}$ follows Eqn. 8. Like previous prediction map models, PDS can be used for down-stream tasks by using an appropriate parametric output distribution q in ρ (Eqn 9). For instance, in our function prediction experiments the PDS outputs parametrize a Gaussian, while in parameter inference experiments we use Flow Matching (Wildberger et al., 2023) as the density estimator.

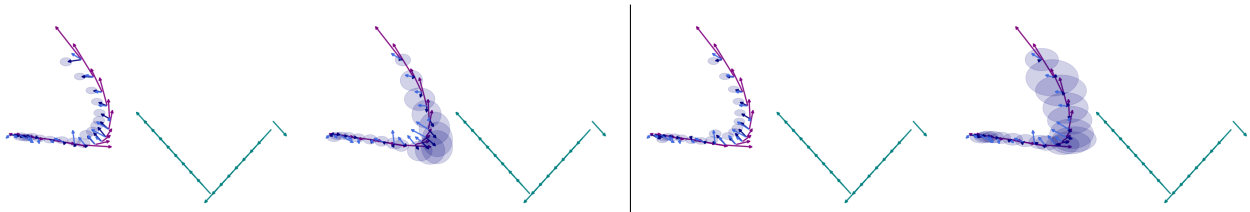
All the free parameters of PDS are learned end-to-end with a maximum likelihood objective appropriate for



(a) Ground truth. (b) PDS (1k). (c) TNP-D (1k). (d) PDS (100k). (e) TNP-D (100k).

(f) Ground truth. (g) PDS (1k). (h) TNP-D (1k). (i) PDS (100k). (j) TNP-D (100k).

Figure 3: Neural process prediction results with Sinusoid-Linear (a-e) and Turin (f-j) models for 1k and 100k training datasets. We see that our PDS captures the functional structures of the data in both domains with only 1k datasets, while TNP-D requires 100k datasets for similar results.



(a) PDS (1k). (b) TNP-D (1k). (c) PDS (100k). (d) TNP-D (100k).

Figure 4: Neural process prediction results with the Point-and-Click model for 1k and 100k training datasets. The teal and purple arrows depict the velocity vectors of the target and cursor at each timestep. The light blue arrows are the ground-truth cursor acceleration vectors caused by human moving the mouse device, and dark blue arrows and the transparent blue plates depict the predicted cursor accelerations with uncertainties. We see how PDS appropriately predicts, with only 1k datasets, the accelerations that lead to a cursor trajectory missing the target. TNP-D fails to capture this pattern even with 100k training datasets.

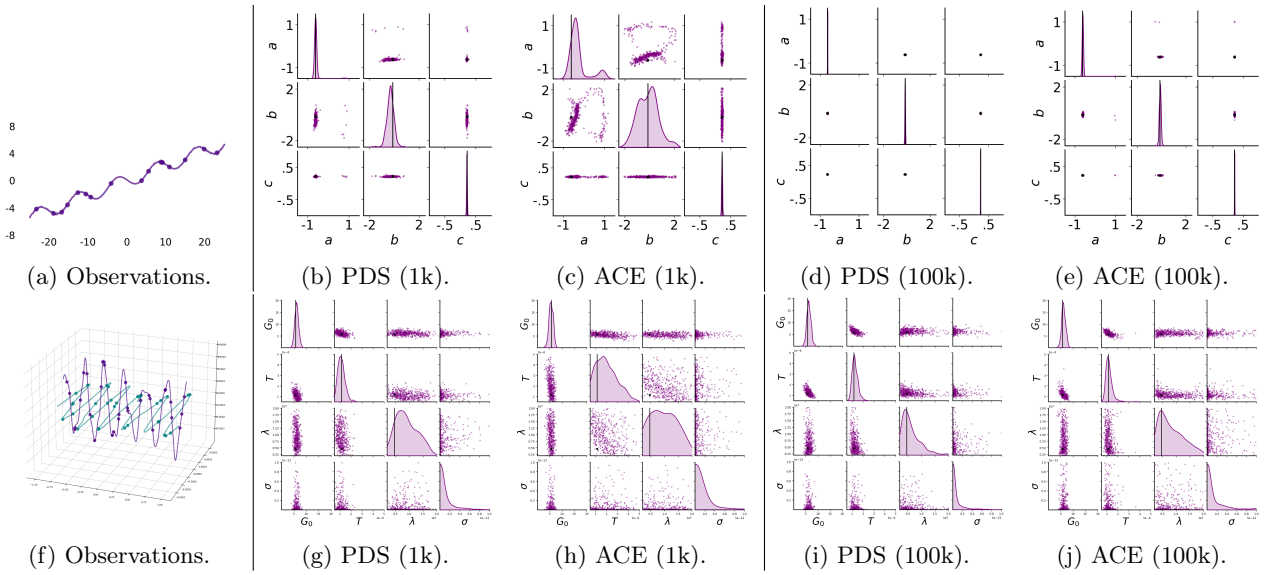
the downstream task. See Appendix B for details.

4 RELATED WORK

Neural processes. Among NPs, ConvCNPs (Gordon et al., 2019) and GNPs (Bruinsma et al., 2021) are probably the closest approaches to our solution. Our predictive set representations can be viewed as a special instance of the RKHS representations used there, and the theoretical justification of PDS is largely inspired by Gordon et al. (2019). The key difference is in how the RKHS representations are used: these models use straightforward embeddings into fixed RKHS, while we treat the RKHS itself as an element to be learned and design the embedding around loss-minimization with RKHS GD. On the other hand, we do not specifically consider translation equivariance, the signature feature of ConvCNPs. We speculate that PDS can be extended with this property, along with autoregressive modeling of output correlations as proposed by Nguyen and Grover (2022); Bruinsma et al. (2023).

TNP-D (Nguyen and Grover, 2022) and the related Prior-Data Fitted Networks (PFN) (Müller et al., 2021) are other related models; our function transformer component is inspired by them. However, in addition to lacking our RKHS optimization bias, these models also operate on learned joint element-wise embeddings. The function transformer acts on discretizations of optimal RKHS representations, while incorporating a transfer kernel weighted attention. Jenson et al. (2024) were the first to enrich TNP attention with kernels; their approach injects domain-specific knowledge via kernels, while we have a more domain-agnostic approach via learnable transfer kernels.

Gradient-based meta-learning. Approaches like MAML (Finn et al., 2017) adapt to each task via updating model parameters via gradients. Our use of functional gradients is related, yet distinct in the sense



(a) Observations. (b) PDS (1k). (c) ACE (1k). (d) PDS (100k). (e) ACE (100k).

(f) Observations. (g) PDS (1k). (h) ACE (1k). (i) PDS (100k). (j) ACE (100k).

Figure 5: Parameter inferences in Sinusoid-Linear (a-e) and Turin (f-j) models at 1k and 100k training datasets. Figures (a) and (f) show the observed datasets with the ground truth functions, (b-e) and (g-j) illustrate pairwise plots of estimated parameter posterior samples (purple) and the ground truth parameters (black). The posteriors of PDS (ours) are better concentrated around the true parameters than ACE in all scenarios.

that ours can be computed fully non-parametrically in RKHS, hence enjoying cheap memory and computation cost together with a guarantee of a convex loss landscape. More generally, prediction map learning amortizes the adaptation step in gradient-based meta-learning with a simple forward pass via set functions which is known to save memory and computation, while often achieving superior performance (Jha et al., 2022).

Xu et al. (2020) propose MetaFun, a meta-learning approach similarly inspired by functional gradients. MetaFun fully replaces the gradients with an end-to-end learnable update rule while we use gradients of a loss function with learnable residual corrections within a meta-learned RKHS. Although the free-form updates are expressive, there is no guarantee that there exists a loss function that these updates minimize. These result in MetaFun having a weaker inductive bias. We include comparisons with MetaFun in our experiments.

Amortized Conditioning Engine. Amortized Conditioning Engine (ACE) (Chang et al., 2024) is a recent model which extends TNPs to jointly amortize multiple inferences like function estimation and parameter inference at once, with an option to inject task-specific priors. While we also consider (individually) both function estimation and parameter inference, our contributions are complementary to ACE. We improve TNP-style encodings by introducing a strong inductive bias, and believe that PDS can be extended to improve sample efficiency also in ACE-style settings. We include (single-output inference) ACE in our comparisons.

5 EXPERIMENTS

Our experiments evaluate PDS in parameter inference and function estimation tasks in seven domains. We consider increasingly complex settings, culminating eventually into realistic and challenging case studies on computational models of cognition. We also include further ablation studies in the Appendix D.

All experiments involve a parametric data generating model $f(x;\theta)$; we generate labeled datasets $Z = (X, Y)$ by first sampling θ from a prior $p(\theta)$ and then evaluate f at randomly selected locations X. For function estimation, we split Z into context Z_c and target Z_t, and estimate $p(Y_t \mid X_t, Z_c)$. In parameter inference, the posterior $p(\theta \mid Z_c)$ is estimated instead. Our experiments implement PDS using MSE as the context-loss $\hat{\mathcal{L}}$, EQ kernel k for $\psi(k)$, and employ $M = 16$ RKHS gradient descent steps.

Evaluation metrics and baselines. We evaluate the methods with the usual negative log likelihood (NLL) metric in both tasks, in addition to qualitative assessment. To properly assess the impact of the amount of training data, we repeat these comparisons by training all the models with three different numbers of datasets in all experiments: $10^3, 10^4$, and 10^5, each repeated using 5 different seeds.

In the parameter inference tasks, we compare to DeepSets (DS) (Zaheer et al., 2017), Set Transformer (ST) (Lee et al., 2019), and ACE (Chang et al., 2024),

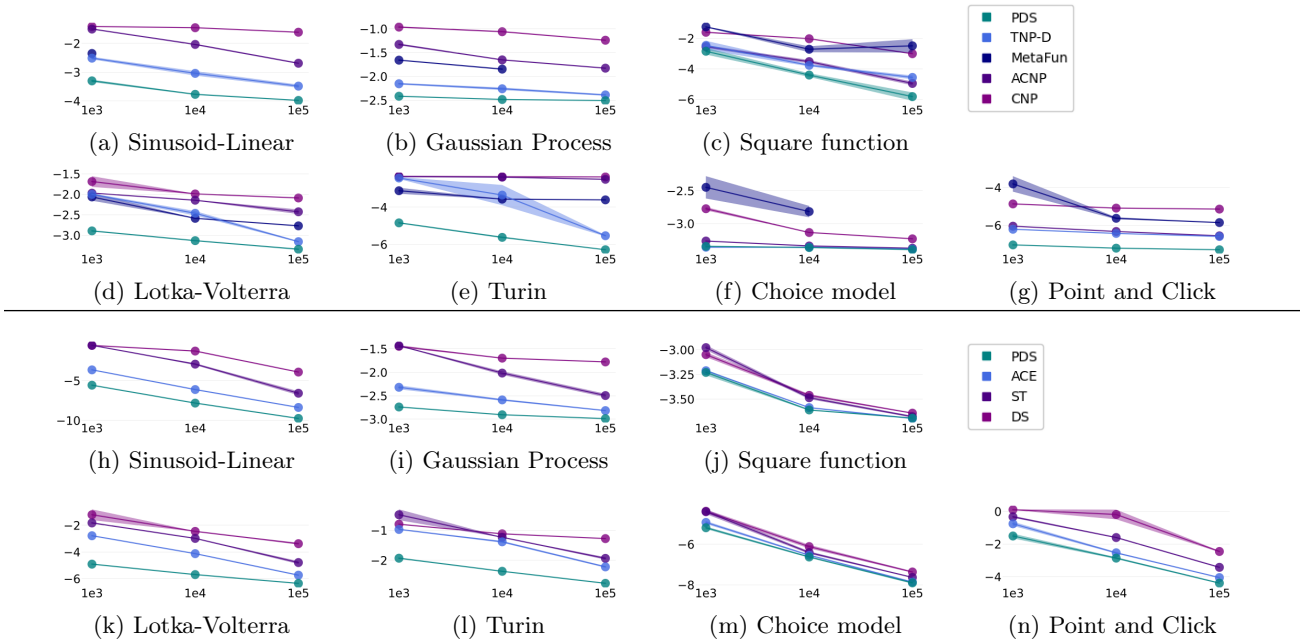


Figure 6: Negative log likelihoods (NLL) in function estimation (a-g) and parameter inference (h-n) as a function of the number of training datasets. The plots show the mean NLLs with standard errors over 5 different seeds. We see that PDS (ours) outperforms the baselines with a clear margin, demonstrating at least an order of magnitude improvement in sample efficiency in most cases.

each coupled with the same Flow Matching density estimator (Wildberger et al., 2023) as our method. The function estimation tasks have CNP (Garnelo et al., 2018a), ACNP (Kim et al., 2019), and TNP (Nguyen and Grover, 2022) as baselines. We also compare with MetaFun (Xu et al., 2020), although we were not able to do so in all instances, due to its instability during training. Each model is given the same hyperparameter tuning treatment, and all the transformer-based models (including ours) have roughly the same number of parameters. See the Appendix C.1 for more details).

5.1 Benchmarks

We begin our evaluation with simple benchmarks on 1-dimensional problems: (1) linear-sinusoidal process (2) Gaussian process with Matérn3/2 kernel, and (3) discrete square functions. Here (1) constructs datasets with smooth and predictable structure, (2) produces erratically behaving functions, and (3) produces highly non-smooth and non-continuous functions that we expect to be difficult to capture with our RKHS representations. The exact setting details are in Appendix C.2.

5.2 Case studies

We also conduct four case studies. The first two use the Lotka-Volterra model (Goel et al., 1971), describing the co-evolution of predator-prey populations, and the Turin model (Turin et al., 1972), a complex engineering model of multipath radio signal propagation.

The remaining experiments are on recent models from computational cognitive science. The first one is the model of computationally rational choice behavior (De Peuter et al., 2024), which models how latent biases affect humans' utility estimates. In this experiment, the functional structure in the datasets is governed by simple linear utility functions with highly complex internal noise model: this allows us to study PDS performance in a stochastic setting where there is little functional patterns to leverage.

The second model is the point-and-click model of Do et al. (2021). This highly complex model simulates how humans perceive the screen, plan motor movements, and use a mouse device when pointing and clicking a moving target with the cursor, and how individual cognitive features and constraints affect this behavior. This experiment poses our most challenging task, both in terms of complexity and volume, by incorporating 100k dense datasets with dimensions up to 300×11, each containing tens of simulated trials. The exact setting details can again be found in the Appendix C.2.

5.3 Results

The results for all function estimation and parameter inference tasks are summarized in Figure 6. Fig-

ures 3, 4 and 5 visualize individual runs. In summary, we observe that the proposed PDS encoding clearly surpasses the sample-efficiency of all baselines across all the task-domains, and that the difference generally grows larger in more complex tasks. In the most challenging Point-and-Click function estimation setting (Figure 4 and 6), none of the baselines got even close to PDS performance, even when given 100 times more training data. We also notice that the benefits of the PDS encoding are clearer in the NP-prediction tasks than in parameter inference; this is likely explained by the closer similarity between the NP-prediction task and the optimal RKHS fit representations of PDS.

Experiments with the Choice model are a clear outlier, in the sense that even the simpler models (Deep Sets and ACNP) can solve the problem with little data, and achieve similar performance to PDS (Figure 6). This is likely explained by the fact that the functional structures of the domain are simple and linear, and that here the performance "bottleneck" is the complex internal noise model. See Appendix E for more results.

6 DISCUSSION

Summary. This work introduced PDS, a new encoding method for sample efficient prediction map learning. Our experiments in function prediction and parameter inference tasks show evidence that PDS encoding can significantly increase training sample-efficiency by orders of magnitudes, compared to strong baselines, in complex settings. We also provided formal guarantees on the universality of PDS for estimating any prediction map.

Limitations. Despite sharing quadratic scaling w.r.t. cardinality of each dataset with SOTA models such as TNP and ACE, PDS may involve a heavier computational overhead if a large number of gradient steps is used to compute the RKHS representations. The number of steps can be tuned to balance accuracy and overhead depending on application constraints.

We also note that the formal guarantees we provided about PDS universality do not fully explain the sample-efficiency we observed in our experiments. Although the PDS approach of using deep, optimal fit RKHS representations is an attractive inductive bias, and demonstrably effective, we leave the theoretical analysis on the nature of this benefit for future work.

Impact. The core advantage of PDS, training sample efficiency, directly addresses a key bottleneck of modern amortized meta-learning methods. We expect our approach to enable the application of these methods in scenarios that were previously not possible due to data scarcity, particularly in those with complex functional patterns. Appealing future directions include extensions to decision-making domains, ACE-style multi-inference, and even TabPFN-like regimes.

Acknowledgements

This work was supported by EU funding EU ELLIOT 101214398, the Research Council of Finland Flagship programme: Finnish Center for Artificial Intelligence FCAI and decision 359207. SK was supported by the UKRI Turing AI World-Leading Researcher Fellowship (EP/W002973/1). We also acknowledge the computational resources provided by the Aalto Science-IT Project from Computer Science IT.

References

Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. *Advances in neural information processing systems*, 29, 2016.

Alain Berlinet and Christine Thomas-Agnan. *Reproducing kernel Hilbert spaces in probability and statistics*. Springer Science & Business Media, 2011.

Tom Blau, Iadine Chades, Amir Dezfouli, Daniel Steinberg, and Edwin V Bonilla. Statistically efficient bayesian sequential experiment design via reinforcement learning with cross-entropy estimators. *arXiv preprint arXiv:2305.18435*, 2023.

Wessel P Bruinsma, James Requeima, Andrew YK Foong, Jonathan Gordon, and Richard E Turner. The gaussian neural process. *arXiv preprint arXiv:2101.03606*, 2021.

Wessel P Bruinsma, Stratis Markou, James Requiema, Andrew YK Foong, Tom R Andersson, Anna Vaughan, Anthony Buonomo, J Scott Hosking, and Richard E Turner. Autoregressive conditional neural processes. *arXiv preprint arXiv:2303.14468*, 2023.

Paul E Chang, Nasrulloh Loka, Daolang Huang, Ulpu Remes, Samuel Kaski, and Luigi Acerbi. Amortized probabilistic conditioning for optimization, simulation and inference. *arXiv preprint arXiv:2410.15320*, 2024.

Sebastiaan De Peuter, Shibei Zhu, Yujia Guo, Andrew Howes, and Samuel Kaski. Preference learning of latent decision utilities with a human-like model of preferential choice. *Advances in Neural Information Processing Systems*, 37:123608–123636, 2024.

Seungwon Do, Minsuk Chang, and Byungjoo Lee. A simulation model of intermittently controlled point-and-click behaviour. In *Proceedings of the 2021 CHI*

Conference on Human Factors in Computing Systems, pages 1–17, 2021.

Tsvetomira Dumbalska, Vickie Li, Konstantinos Tsetsos, and Christopher Summerfield. A map of decoy influence in human multialternative choice. *Proceedings of the National Academy of Sciences*, 117(40): 25169–25178, 2020.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pages 1126–1135. PMLR, 2017.

Andrew Foong, Wessel Bruinsma, Jonathan Gordon, Yann Dubois, James Requeima, and Richard Turner. Meta-learning stationary stochastic process prediction with convolutional neural processes. *Advances in Neural Information Processing Systems*, 33:8284–8295, 2020.

Alexandre Galashov, Jonathan Schwarz, Hyunjik Kim, Marta Garnelo, David Saxton, Pushmeet Kohli, SM Eslami, and Yee Whye Teh. Meta-learning surrogate models for sequential decision making. *arXiv preprint arXiv:1903.11907*, 2019.

Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and SM Ali Eslami. Conditional neural processes. In *International Conference on Machine Learning*, pages 1704–1713. PMLR, 2018a.

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018b.

Narendra S Goel, Samaresh C Maitra, and Elliott W Montroll. On the volterra and other nonlinear models of interacting populations. *Reviews of modern physics*, 43(2):231, 1971.

Jonathan Gordon, Wessel P Bruinsma, Andrew YK Foong, James Requeima, Yann Dubois, and Richard E Turner. Convolutional conditional neural processes. *arXiv preprint arXiv:1910.13556*, 2019.

Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. Tabpfn: A transformer that solves small tabular classification problems in a second. *arXiv preprint arXiv:2207.01848*, 2022.

Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 637(8045):319–326, 2025.

Andrew Howes, Paul A Warren, George Farmer, Wael El-Deredy, and Richard L Lewis. Why contextual preference reversals maximize expected value. *Psychological review*, 123(4):368, 2016.

Daolang Huang, Manuel Haussmann, Ulpu Remes, ST John, Grégoire Clarté, Kevin Luck, Samuel Kaski, and Luigi Acerbi. Practical equivariances via relational conditional neural processes. *Advances in Neural Information Processing Systems*, 36:29201–29238, 2023.

Daolang Huang, Yujia Guo, Luigi Acerbi, and Samuel Kaski. Amortized bayesian experimental design for decision-making. *Advances in Neural Information Processing Systems*, 37:109460–109486, 2024.

Daolang Huang, Xinyi Wen, Ayush Bharti, Samuel Kaski, and Luigi Acerbi. Aline: Joint amortization for bayesian inference and active data acquisition. *arXiv preprint arXiv:2506.07259*, 2025.

Daniel Jenson, Jhonathan Navott, Mengyan Zhang, Makkunda Sharma, Elizaveta Semenova, and Seth Flaxman. Transformer neural processes–kernel regression. *arXiv preprint arXiv:2411.12502*, 2024.

Saurav Jha, Dong Gong, Xuesong Wang, Richard E Turner, and Lina Yao. The neural process family: Survey, applications and perspectives. *arXiv preprint arXiv:2209.00517*, 2022.

Suhyun Kang, Duhun Hwang, Moonjung Eo, Taesup Kim, and Wonjong Rhee. Meta-learning with a geometry-adaptive preconditioner. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16080–16090, 2023.

Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. *arXiv preprint arXiv:1901.05761*, 2019.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning*, pages 3744–3753. PMLR, 2019.

Richard L Lewis, Andrew Howes, and Satinder Singh. Computational rationality: Linking mechanism and behavior through bounded utility maximization. *Topics in cognitive science*, 6(2):279–311, 2014.

Stratis Markou, James Requeima, Wessel P Bruinsma, Anna Vaughan, and Richard E Turner. Practical conditional neural processes via tractable dependent predictions. *arXiv preprint arXiv:2203.08775*, 2022.

Llew Mason, Jonathan Baxter, Peter L Bartlett, Marcus Frean, et al. Functional gradient techniques for combining hypotheses. *Advances in Neural Information Processing Systems*, pages 221–246, 1999.

Charles A Micchelli, Yuesheng Xu, and Haizhang Zhang. Universal kernels. *Journal of Machine Learning Research*, 7(12), 2006.

Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do bayesian inference. *arXiv preprint arXiv:2112.10510*, 2021.

Ryan L Murphy, Balasubramaniam Srinivasan, Vinayak Rao, and Bruno Ribeiro. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. *arXiv preprint arXiv:1811.01900*, 2018.

Tung Nguyen and Aditya Grover. Transformer neural processes: Uncertainty-aware meta learning via sequence modeling. *arXiv preprint arXiv:2207.04179*, 2022.

Massimiliano Patacchiola, Jack Turner, Elliot J Crowley, Michael O'Boyle, and Amos J Storkey. Bayesian meta-learning for the few-shot setting via deep kernels. *Advances in Neural Information Processing Systems*, 33:16108–16118, 2020.

Garvesh Raskutti, Martin J Wainwright, and Bin Yu. Early stopping and non-parametric regression: an optimal data-dependent stopping rule. *The Journal of Machine Learning Research*, 15(1):335–366, 2014.

Adel AM Saleh and Reinaldo Valenzuela. A statistical model for indoor multipath propagation. *IEEE Journal on selected areas in communications*, 5(2): 128–137, 1987.

Satoshi Takabe and Tadashi Wadayama. Convergence acceleration via chebyshev step: Plausible interpretation of deep-unfolded gradient descent. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 105(8):1110–1120, 2022.

Prudencio Tossou, Basile Dura, Francois Laviolette, Mario Marchand, and Alexandre Lacoste. Adaptive deep kernel learning. *arXiv preprint arXiv:1905.12131*, 2019.

Konstantinos Tsetsos, Rani Moran, James Moreland, Nick Chater, Marius Usher, and Christopher Summerfield. Economic irrationality is optimal during noisy decision making. *Proceedings of the National Academy of Sciences*, 113(11):3102–3107, 2016.

George L Turin, Fred D Clapp, Tom L Johnston, Stephen B Fine, and Dan Lavry. A statistical model of urban multipath propagation. *IEEE transactions on vehicular technology*, 21(1):1–9, 1972.

Aleksei Ustimenko, Artem Beliakov, and Liudmila Prokhorenkova. Gradient boosting performs gaussian process inference. *arXiv preprint arXiv:2206.05608*, 2022.

A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

Edward Wagstaff, Fabian Fuchs, Martin Engelcke, Ingmar Posner, and Michael A Osborne. On the limitations of representing functions on sets. In *International Conference on Machine Learning*, pages 6487–6494. PMLR, 2019.

Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *International conference on machine learning*, pages 3751–3760. PMLR, 2017.

Jonas Wildberger, Maximilian Dax, Simon Buchholz, Stephen Green, Jakob H Macke, and Bernhard Schölkopf. Flow matching for scalable simulation-based inference. *Advances in Neural Information Processing Systems*, 36:16837–16864, 2023.

Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial intelligence and statistics*, pages 370–378. PMLR, 2016.

Jin Xu, Jean-Francois Ton, Hyunjik Kim, Adam Kosiorek, and Yee Whye Teh. Metafun: Meta-learning with iterative functional updates. In *International Conference on Machine Learning*, pages 10617–10627. PMLR, 2020.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Yes]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Yes]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes] The link to source code is provided.

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Yes]

(b) Complete proofs of all theoretical results. [Yes] Provided in the supplementary material.

(c) Clear explanations of any assumptions. [Yes]

3. For all figures and tables that present empirical results, check if you include:

(a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes] The link to source code is provided.

(b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes] Included in the supplementary material.

(c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]

(d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes] Included in the supplementary material.

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

(a) Citations of the creator If your work uses existing assets. [Yes]

(b) The license information of the assets, if applicable. [Not Applicable]

(c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]

(d) Information about consent from data providers/curators. [Not Applicable]

(e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:

(a) The full text of instructions given to participants and screenshots. [Not Applicable]

(b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]

(c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Predictive Deep Sets: Supplementary Materials

A MISSING PROOFS

A.1 Proof of Proposition 1

Proof. In summary, our proof builds on a similar strategy as Theorem 1 of Gordon et al. (2019). We first show that the mapping from fixed-sized functional sets $Z = (X, Y)$ into the evaluations $\big(X, h^{(M)}(X)\big)$ of predictive set representations $h^{(M)} \in \mathcal{H}_k$, induced by $r_{\mathcal{H}_k}$, is a homeomorphism onto its image. Then, by invoking an analogous "pasting" argument as in Lemma 3 in Gordon et al. (2019), this homeomorphism argument can be extended also for varying-sized sets. This then allows for proving the sufficiency and necessity of the decomposition

$$\pi(T; Q, Z) = \rho_{\mathcal{H}_k}(T; Q, X, r_{\mathcal{H}_k}(Z)). \tag{11}$$

Setting and useful derivations. Let $Z = (X, Y) \in 2^{\mathcal{Z}}$ denote a functional set, $Q \in 2^{\mathcal{Q}}$ and $T \in 2^{\mathcal{T}}$ denote the query and the target, and k be a continuous reproducing kernel of RKHS $\mathcal{H}_k$. We define $r_{\mathcal{H}_k} : 2^{\mathcal{Z}} \to \mathcal{H}_k$ as:

$$\begin{aligned} r_{\mathcal{H}_k}(Z) &= h^{(M)}, \text{ with} \\ h^{(m+1)} &= h^{(m)} - \eta \nabla_{\mathcal{H}_k} \hat{\mathcal{L}}\big(h^{(m)}, Z\big) \\ &= h^{(m)} - \eta \sum_{x,y \in Z} k(\cdot, x) \partial \hat{\mathcal{L}}\big(h^{(m)}(x), y\big) / \partial h^{(m)}. \end{aligned} \tag{12}$$

We assume that $M < \infty$ and $0 < \eta < n/\lambda_{\max}\big(K(X,X)\big)$, that $h^{(0)} \in \mathcal{H}_k$ is independent of Y, and use MSE-loss, i.e. $\hat{\mathcal{L}}\big(h^{(m)}, Z\big) = \frac{1}{2n} \sum_{x,y \in Z} \|h^{(m)}(x) - y\|^2$ (we divide by 2 for simplicity). Here $n = |Z|$ and $\lambda_{\max}\big(K(X,X)\big)$ is the largest eigenvalue of $K(X, X)$. Plugging the MSE-loss into Eqn. 12 yields

$$h^{(m+1)} = h^{(m)} - \frac{\eta}{n} \sum_{x,y \in Z} (h^{(m)}(x) - y) k(\cdot, x), \tag{13}$$

and evaluating $h^{(m+1)}$ on X yields

$$h^{(m+1)}(X) = \big(I - \frac{\eta}{n} K(X,X)\big) h^{(m)}(X) + \frac{\eta}{n} K(X,X) Y. \tag{14}$$

From Eqn. 14, it is straightforward to show by induction that:

$$\begin{aligned} h^{(m+1)}(X) &= A^{m+1} h^{(0)}(X) + (I - A^{m+1}) Y, \\ A &= I - \frac{\eta}{n} K(X,X). \end{aligned} \tag{15}$$

Homeomorphism: Our first step is to show that for Z with fixed size of n, the map $Z \to \big(X, r_{\mathcal{H}_k}(Z)(X)\big)$ is a homeomorphism onto its image; that it is injective and continuous with a continuous inverse. Call this map E_n.

We begin by showing the injectivity. Suppose we have $Z = (X, Y)$ and $Z' = (X', Y')$ with identical images under the map E_n, i.e.,

$$X, r_{\mathcal{H}_k}(Z; M)(X) = X', r_{\mathcal{H}_k}(Z')(X'). \tag{16}$$

We trivially have $X = X'$. Plugging in Eqn. 15 yields

$$A^M h^{(0)}(X) + (I - A^M) Y = A^M h^{(0)}(X) + (I - A^M) Y', \tag{17}$$

which simplifies into:

$$\big(I - A^M\big)(Y - Y') = 0 \tag{18}$$

Under the definition of functional sets, $K(X, X)$ is full rank and hence invertible. Combined with the assumed step-size constraint $0 < \eta < n/\lambda_{\max}\big(K(X, X)\big)$, this means that $I - A^M = I - \big(I - \frac{\eta}{n}K(X, X)\big)^M$ is invertible as well. It follows that Eqn. 18 has only one solution $Y = Y'$; this means that we have proven that the equivalence of images under the map E_n implies equivalence of preimages as well, i.e., that the map E_n is injective.

Regarding the continuity conditions, it is easy to see that the continuity of the map E_n follows directly from the continuity of the affine mapping of Eqn. 15 and of k. The continuity of the inverse map E_n^{-1} is analogously straightforward to confirm by solving for Y in Eqn. 15 through using the invertibility of $I - A^M$.

Having showed these properties for the map E_n over fixed-sized Z, we next extend this result to apply for E, the map $Z \to \big(X, r_{\mathcal{H}_k}(Z)(X)\big)$ over varying sized Z as well, assuming the maximum size of sets Z is bounded by some N. With the topology of $2^{\mathcal{Z}}$ assumed in Definition 1, this extension can be obtained through invoking an analogous argument as used in Lemma 3 in Gordon et al. (2019). In summary, the extension directly results from "pasting" the component-wise homeomorphisms over the finite union of sizes $n \in \{1, \dots, N\}$ together, as detailed in (Gordon et al., 2019).

We are now ready to proceed to proving the sufficiency and necessity of the decomposition.

Sufficiency: In Eqn. 13, the addition over $x, y \in Z$ is commutative and associative, guaranteeing permutation invariance of $r_{\mathcal{H}_k}$. Under the assumption that also $\rho_{\mathcal{H}_k}$ is invariant to permutations of X, the full prediction map π is guaranteed to be permutation invariant.

Necessity: Assume a permutation invariant and continuous prediction map $\pi : 2^{\mathcal{Z}} \times 2^{\mathcal{Q}} \to \mathcal{P}(2^{\mathcal{T}})$. The invertibility of E guarantees existence of E^{-1}. For finite $Z \in 2^{\mathcal{Z}}$ we hence have:

$$\pi(T; Q, Z) = \pi\big(T; Q, E^{-1}\big(E(Z)\big)\big) = \pi\big(T; Q, E^{-1}(X, r_{\mathcal{H}_k}(Z)(X))\big). \tag{19}$$

By defining $\rho_{\mathcal{H}_k}(T; Q, \cdot, g) = \pi\big(T; Q, E^{-1}(\cdot, g(\cdot))\big)$ for $\cdot \in 2^{\mathcal{X}}, g \in \mathcal{H}_k$ (again, the invertibility of E guarantees existence of such a $\rho_{\mathcal{H}_k}$), and plugging this to Eqn. 19, we obtain

$$\pi(T; Q, Z) = \rho_{\mathcal{H}_k}\big(T; Q, X, r_{\mathcal{H}_k}(Z)\big), \tag{20}$$

showing that there always exists a $\rho_{\mathcal{H}_k}$ for which any continuous and permutation invariant π permits the decomposition of Eqn. 6. We further verify that such $\rho_{\mathcal{H}_k}$ is continuous: we have previously proved continuity of $r_{\mathcal{H}_k}(Z)(X)$ and E^{-1}; having assumed that π is assumed to be continuous as well, $\rho_{\mathcal{H}_k}(T; Q, X, r_{\mathcal{H}_k}(Z))$ must be continuous by being a composition of continuous operations. This completes the proof. □

B PREDICTIVE DEEP SETS ARCHITECTURE DETAILS

This section lists further implementation details about the proposed PDS design.

B.1 RKHS gradient descent.

Initial function representation. The PDS RKHS gradient descent is implemented by iteratively applying the equations

$$\begin{aligned} h^{(m+1)} &= h^{(m)} - \eta\big(\delta^{(m)} + \omega_1(\delta^{(m)})\big) \\ \delta^{(m)} &= \nabla_{\mathcal{H}_{\psi(k)}}\hat{\mathcal{L}}\big(h^{(m)} + \omega_2(h^{(m)}), (\phi_x(X), \phi_z(Z))\big) \end{aligned} \tag{21}$$

over M steps. In practice, computing these iterations requires specifying the initial representation $h^{(0)}$. As we are within this context interested in evaluating any $h^{(m)}$ at the finite domain $\Xi \subseteq \operatorname{supp} h^{(m)}$ are only need to define $h^{(0)}$ at the domain Ξ:

$$h^{(0)}(\Xi) = \phi_y(\mathrm{cat}_f[\Xi, \mathrm{cat}_s[Y, \mathbf{0}, \mathbf{0}]]), \tag{22}$$

where cat_f and cat_s denote concatenations over feature and set dimensions respectively. This is equivalent to jointly embedding all the X and Y values while defaulting the Y values to zero unless they are observed. We

did not extensively explore how the choice of $h^{(0)}$ affects the performance; this definition was chosen due to its success on early preliminary tests. We expect more careful tuning of $h^{(0)}$ to yield performance increases.

We remark that, although Prop. 3 does assume that $h^{(0)}$ should be independent of Y, with the additional flexibility brought by the embedding ϕ_y, it is trivial to show that this choice does not restrict the overall capability of the model.

Multi-head structure. Our implementation used in experiments further enriches the Eqn. 21 by incorporating a similar multi-head structure as used in transformers (Vaswani, 2017). This is done by splitting the terms

$$K_\psi(\cdot, X) = \sum_{x \in X} \psi(k)(\cdot, x) \text{ and} \tag{23}$$

$$\partial \hat{\mathcal{L}}\big(h^{(m)} + \omega_2(h^{(m)}), (\phi_x(X), \phi_z(Z))\big)/\partial h^{(m)} \tag{24}$$

into multiple channels across feature dimension before multiplying them together — and then immediately concatenating the channels back together after multiplication — when computing the RKHS gradient in Eqn. 21. Similar channel split is done for $K_\psi(\cdot, X)$ when it is used for attention computation in the function transformer.

Accelerated descent. We additionally employ a straightforward two-step Chebyshev acceleration on the Eqn. 21 by replacing the update with:

$$h^{(m+1)} = \sigma(\eta)\big(\delta^{(m)} + \omega_1(\delta^{(m)})\big) - h^{(m)}, \tag{25}$$

which is known to accelerate convergence in related algorithms (Takabe and Wadayama, 2022). We did not extensively explore alternative acceleration schemes but found this useful for helping to balance the computational budget and model accuracy; this observation highlights the potential for further improvements through investigating alternative approaches to convergence speed acceleration.

Note that we here also wrap the step-size η inside the sigmoid function σ and treat it as a learnable parameter; this helps ensuring the step-size condition of Prop. 3.

Support vectors. The support vectors V are learned to enrich the function representations within the model. We initialize V randomly across the support domain of functions in each task; in our preliminary tests we found this to be important for model performance. In practice this is easy to achieve by initializing them e.g. within the interval $[-1, 1]$ if and when the input data is normalized appropriately as in our experiments.

B.2 Function Transformer

The function transformer is defined through the equations:

$$h_{l+1}^{(M)} = h_l^{(M)} + \mathrm{MLP}\big(h_l^{(M)} + \mathcal{A}(h_l^{(M)}; \psi(k))\big) \tag{26}$$

$$\mathcal{A}(h; \psi(k)) = \mathrm{Softmax}\left(\frac{h(\cdot)h(\cdot)^T}{\sqrt{d_h}} + \log \psi(k)(\cdot, \cdot) + \mathrm{M}\right) h(\cdot), \tag{27}$$

where each layer l updates the representation $h_{l-1}^{(M)}$ into $h_l^{(M)}$ via a residual term. We employ the usual multi-head structure here as well: the inputs $h(\cdot)$ to attention computation are projected into H heads via learned linear weights $W_j^{(1)}, W_j^{(2)}$, and then projected back with $W^{(3)}$ and summed with $h(\cdot)$ after the attention computation:

$$\text{MH-}\mathcal{A}(h; \psi(k)) = h(\cdot) + \mathrm{cat}_f[O_1, \ldots, O_H]W^{(3)}, \text{ with} \tag{28}$$

$$O_j = \mathrm{Softmax}\left(\frac{h(\cdot)W_j^{(1)} \times (h(\cdot)W_j^{(1)})^T}{\sqrt{d_h}} + \log \psi(k)(\cdot, \cdot) + \mathrm{M}\right) h(\cdot)W_j^{(2)}, \tag{29}$$

where $\times$ denotes matrix multiplication. We use MH-$\mathcal{A}(h; \psi(k))$ in place of $\mathcal{A}(h; \psi(k))$ in Eqn. 26.

The attention mask M is here implemented similarly to the transformer prediction maps Nguyen and Grover (2022) and Chang et al. (2024). Given the input $h_{l-1}^{(M)}$ of layer l, M lets $h_{l-1}^{(M)}(X_c)$ attend to itself, $h_{l-1}^{(M)}(V)$ to $h_{l-1}^{(M)}(X_c) \cup h_{l-1}^{(M)}(V)$, and $h_{l-1}^{(M)}(X_t)$ to full $h_{l-1}^{(M)}(\Xi)$.

B.3 Implementation details

Below in Table 1 we detail the exact design of the PDS components we use in our experiments.

Table 1: PDS architecture details.

Parameter name	Value
Embedding networks ϕ_x, ϕ_z	
Input size	$d_x / d_x + d_y$
Hidden size	128
Output size	128
Num layers	3
Activation function	ReLU
Kernel hyperparameter network ϕ_k	
Input size	128
Hidden size	16
Output size	2
Num layers	3
Activation function	ReLU
RKHS gradient descent	
Num steps M	16
Num heads	4
Initial step size η	M^{-2}
Residual corrections ω_1, ω_2	
Input size	128
Hidden size	128
Output size	128
Num layers	3
Activation function	ReLU
Function transformer	
Num layers	3
Num heads	4
Hidden size	128
Output size	128
Activation function	ReLU

C EXPERIMENT DETAILS

The code for reproducing the experiments is available here `https://github.com/epuriepu/PredictiveDeepSets`. The parameter inference tasks use the code of Wildberger et al. (2023) for implementing the Flow Matching density estimators.

Baselines. In summary, we implement most of the baseline models (such as TNP-D) using the code from the repositories corresponding to the original papers when applicable. ACE is normalized for our function estimation and parameter inference comparisons by using a standard TNP-D model and a TNP-D variant decorated with a learnable token together with the same Flow Matching density estimator (Wildberger et al., 2023) as our method respectively. This is in contrast to the original ACE model which uses Gaussian mixture outputs for both function estimation and parameter inference tasks. Note that one can similarly use Gaussian mixtures with our method if they are preferred; the choice of output distribution is orthogonal to our contributions.

C.1 Training details

The training setting, including training data, hyperparameter tuning, etc. are the same for all models. We train each model for 200, 100, or 50 epochs in the settings involving 1k, 10k, or 100k training datasets respectively, which was enough to guarantee convergence without overfitting in most cases. The only exception vas the Sinusoid-Linear model, where we doubled the amount of epochs due to slow convergence.

We generate the fixed numbers of datasets before training, and repeat the training over these datasets at each epoch by resampling the context-target data (see Section C.2); this allows us to study training sample efficiency of the methods. This is unlike in typical NP-literature, where new data is iteratively generated at each epoch.

We tune the model hyperparameters by repeating the training using four different learning rates $\lambda \in [0.001, 0.0005, 0.0002, 0.0001]$, which was found to be the most impactful hyperparameter for model performance in preliminary experiments. For each model in each setting, we select the hyperparameters with the best evaluation performance. We repeat the model training and evaluation using altogether 5 different random seeds in each case.

In all experiments, all models are evaluated using 1k datasets which are generated independently from the training data. The model training was divided across 8 NVIDIA RTX A2000 GPUs and Intel Core i5-12600K CPUs. The training time varied highly depending on the task, model, and the number of training datasets; our rough estimate for the total execution time for the experiments is around 1k GPU hours.

Table 2 below collects all the exact details shared across settings.

Table 2: Training details shared across all models. In the Point-and-Click experiment we use batch-size of 8 due to larger dataset size and double the number of batches per epoch to match the total number of training datasets with other experiments.

Parameter name	Value
Optimizer	Adam
Scheduler	Cosine annealing
Batch size	16
Datasets per epoch	1024/10240/102400
Batches per epoch	64/640/6400
Num epochs	200/100/50

C.2 Task details

We detail each experiment setting below. In each task, we also appropriately normalize the data before feeding it to the models.

Sinusoid-Linear. The sinusoidal-linear model defines a distribution over parameters $\theta = (\theta_1, \theta_2, \theta_3) \sim \mathcal{U}(-1, 1)\mathcal{U}(-0.2, 0.2)\mathcal{U}(-0.5, 0.5)$ of the sine function $f(x) = \sin(\theta_1 x - \theta_2) + \theta_3 x + \epsilon$, where ϵ is i.i.d. Gaussian noise. We generate datasets $Z = (X, Y)$ by first sampling 50 random i.i.d. evaluation locations according to $x \sim p(x) = \mathcal{U}(-8\pi, 8\pi)$, to evaluate $f(x)$. For model training, each dataset is split by randomly selecting $N_c \sim \mathcal{U}(1, 49)$ datapoints to act as the context dataset Z_c — the remaining points are used as the target Z_t.

Gaussian Process. This benchmark uses a GP with Matérn-3/2 kernel. The GP is here parametrized by the kernel lengthscale (θ_1) and outputscale (θ_2), together with the GP mean prior (θ_3). For each task, we sample $\log \theta_1 \sim \mathcal{U}(0.5, 5), \theta_2 \sim \mathcal{U}(0.5, 2)$, and $\theta_3 \sim \mathcal{U}(-4, 4)$. The datasets are created by evaluating random functions sampled from the GPs at 50 i.i.d sampled random locations $x \sim \mathcal{U}(-10, 10)$. The context-target splits are done similarly as in the Sinusoid-Linear model.

Square Function. The square function model is constructed using the partial definition:

$$f(x) = \begin{cases} 1 & \text{if } \theta_1 < x < \theta_2 \\ 0 & \text{otherwise.} \end{cases} \tag{30}$$

We sample $\theta_1 \sim \mathcal{U}(0.25, 0.50)$ and $\theta_2 \sim \mathcal{U}(0.50, 0.75)$. We evaluate $f(x)$ at 10 random locations $x \sim \mathcal{U}(0, 1)$. The context-target splits are done similarly as before with $N_c \sim \mathcal{U}(1, 9)$.

Lotka-Volterra. The Lotka-Volterra model (Goel et al., 1971) describes co-evolution of predator and prey populations over time using non-linear differential equations describing the growth rates of prey ($y^{(1)}$) and predator ($y^{(2)}$) populations:

$$\frac{dy^{(1)}}{dt} = \alpha y^{(1)} - \beta y^{(1)} y^{(2)} \tag{31}$$

$$\frac{dy^{(2)}}{dt} = -\gamma y^{(2)} + \delta y^{(1)} y^{(2)}. \tag{32}$$

Here α and β describe prey population growth rate and the effect the predator has on their death rate, while γ and δ describe predator population death rate and the prey's effect on growth rate.

We sample random initial populations $y_0^{(1)} \sim \mathcal{U}(10, 20)$ and $y_0^{(2)} \sim \mathcal{U}(5, 15)$ and parameters

$$\begin{aligned} \alpha &\sim \mathcal{U}(0.5, 1.5) \\ \beta &\sim \mathcal{U}(0.01, 0.1) \\ \gamma &\sim \mathcal{U}(0.5, 1.5) \\ \delta &\sim \mathcal{U}(0.01, 0.1). \end{aligned} \tag{33}$$

We then simulate the system using 5000 timesteps (corresponding to 20 years) and randomly pick 50 timesteps where either $y^{(1)}$ or $y^{(2)}$ is revealed. We construct Y by concatenating the sets of $y^{(1)}$ and $y^{(2)}$ values along the set dimension while adding a small noise. X is simply the 50 timesteps; we flip the sign of each timestep x to negative (i.e., use $-x$ instead) if the prey population $y^{(1)}$ was revealed at that particular time (this is to represent the partial observations without padding). We again do the context-target split similarly as before with $N_c \sim \mathcal{U}(1, 49)$.

Turin model. The Turin model (Turin et al., 1972) is a physically motivated channel model describing radio signal propagation in urban environments. Here we consider a particular modification by (Saleh and Valenzuela, 1987) which adds a light realism layer modeling how physical signal reflectors, such as buildings, tend to cluster signal paths together. We refer the reader to Turin et al. (1972) and Saleh and Valenzuela (1987) for detailed model specification. We additionally add slight variation to some of the model parameters (which are typically statistically fixed) to induce variation on the model behavior, together with exponential delay decays and Gaussian tap statistics rather than strict exponential inter-cluster processes of the original model. The latter changes have little to no effect on the functions produced by the model, but were found helpful to combat identifiability issues on the parameter inference task.

In concrete terms, the model generates complex-valued amplitude and phase signal responses (Y) as a function of frequencies (X), which are here used to construct the datasets $Z = (X, Y)$. The model is controlled by four parameters:

$$\text{Signal power: } G_0 \sim \log \mathcal{N}(0, 1) \tag{34}$$

$$\text{Decay speed: } T \sim \mathcal{U}(5 \cdot 10^{-5}, 5 \cdot 10^{-6}) \tag{35}$$

$$\text{Ray intensity: } \lambda \sim \mathcal{U}(2 \cdot 10^{6}, 2 \cdot 10^{7}) \tag{36}$$

$$\text{Diffuse tap noise: } s^2 \sim \log \mathcal{U}(-16, -12) \tag{37}$$

we sample independently for each dataset. Again, we sample 100 random frequencies and perform the context-target split with $N_c \sim \mathcal{U}(1, 99)$.

Choice model. This case study builds on a recent model (De Peuter et al., 2024) of computationally rational choice behavior (Howes et al., 2016). This model leverages the principles of computational rationality (Lewis et al., 2014) to capture how human choice behavior emerges as a result of bounded optimization, including various apparent biases governing human choices, such as contextual choice effects (Tsetsos et al., 2016). We consider this cognitive model in the context of a property task (Dumbalska et al., 2020), where the human is asked to rank

properties based on rental costs and perceived value. We use a utility function that is linear w.r.t. the choice attributes, similarly as done in De Peuter et al. (2024).

Formally, the model describes how humans make choices y from sets of options x according to their latent utility function u (we use choice option sets of size 3). The utility function u is parametrized by a list of preference weights w w.r.t. choice attributes and a set of other parameters (Howes et al., 2016) collected in θ. See De Peuter et al. (2024) for further details about the model.

Per usual, we generate datasets by randomly sampling 50 option lists and collect the human choices predicted by the choice model. We use $N_c \sim \mathcal{U}(1, 49)$ for obtaining the context-target split.

Point-and-Click. Our final case study considers the point-and-click model of Do et al. (2021). The point-and-click simulator is a complex model that modularly captures how humans perceive the screen, plan motor movements, and use a mouse device when pointing and clicking a moving target with their cursor. This model too is based on the computational rationality principle, which states that humans are likely to behave task-optimally, but within their individual constrains, which here arise from visual perception, planning capability, arm motor movement and precision mouse handling. This model encodes these constraints and approximates the human behaviors via reinforcement learning. The model captures cognitive bounds with parameters θ and represent humans' behavior with policy that maps the humans' observation of the current environment state (comprises of positions and velocities of the cursor and target, target size, and hand position) to their action (mouse usage).

In this case study, we generate datasets $Z = (X, Y)$ by sampling the parameters θ: *motor noise constant* n_v, *precision of one's internal clock* σ_v, *visual perception noise* c_σ, and *planning horizon* T_{max} which are a subset of the full model parameters (we refer the reader to Do et al. (2021) for details on their effects). For each parameter sample θ, we simulate up to 32 trials with random initial cursor/target locations and target velocity, and concatenate the horizontal/vertical cursor and target positions and velocities at each timestep (used as X) and the horizontal/vertical mouse accelerations and click decisions (used as Y) from each trial to construct the datasets. We pad each dataset to match the maximum dataset size (typically above 300 elements) within its batch. Again, the context-target split is done with $N_c \sim \mathcal{U}(1, 299)$.

D ABLATION STUDIES

We perform a series of ablation studies to better pinpoint how different aspects and design choices of our method contribute to the overall performance. The ablations are carried out in the linear-sinusoid NP prediction task using 1k datasets, and study the impact of (1) the number of GD steps, (2) using RKHS GD over alternatives, (3) the decoder design, and (4) the kernel design. The details and results are presented in the following subsections; we also briefly summarize the ablation results below.

Summary on ablation results. The ablations indicate that the proposed usage of RKHS GD is the single most impactful factor for the strong PDS performance, and that the residual corrections are crucial for enabling its full effect in practice. The benefits of the RKHS GD are still clearly dependent on the design of its reproducing kernel, and data-driven deep kernels seem to have practical advantage here. The number of RKHS GD steps is also relatively impactful, and it can be tuned to balance between the performance and computational cost. Lastly, the decoder seems to be the least impactful factor for PDS performance: the proposed function transformer does still bring a small performance benefit while requiring slightly less parameters than conventional designs.

D.1 Gradient descent steps ablation

We measure the impact of the number of RKHS GD steps M by repeating the experiment with values $[0, 1, 2, 4, 8, 16, 32, 64]$ (at 0 steps the RKHS GD is entirely skipped and we feed the inputs directly to the function transformer). The results reported in the main paper experiments use 16 steps.

In Table 3, we see that already one step brings significant performance benefits in contrast to the 0-step case. As a rule of thumb, more steps seem to equal better performance, but with diminishing returns at high numbers. This means that the number of gradient steps can be used to balance between the model performance, and computational load. Surprisingly, the effect seems to be slightly reversed between steps 1-4, although we suspect this to be a quirk caused by the step size η being initialized as a function of the number of GD steps.

Table 3: Negative Log-Likelihoods for different numbers of GD steps.

TNP	PDS (0)	PDS (1)	PDS (2)	PDS (4)
-2.51 ± 0.04	-2.62 ± 0.02	-2.98 ± 0.03	-2.94 ± 0.03	-2.90 ± 0.03
	PDS (8)	PDS (16)	PDS (32)	PDS (64)
	-3.14 ± 0.07	-3.30 ± 0.04	-3.37 ± 0.02	-3.39 ± 0.02

Comparison between PDS (0) and the TNP baseline implies that the benefits of the architectural changes our function transformer introduces over a standard transformer are small compared to the benefits of using the RKHS GD scheme.

D.2 RKHS gradient descent ablation

Our second ablation studies (1) the impact of building the function representations h specifically via RKHS GD by constructing them with a Nadaraya Watson estimator (NWE) instead, and (2) the impact of the learned residual corrections by removing them entirely.

These result in (1) a NLL of (-2.47 ± 0.02) and (2) a NLL of (-2.83 ± 0.01).

We see that use of NWE seems to collapse the performance near the TNP ballpark (-2.51 ± 0.04), and that RKHS GD clearly beats both TNP and the NWE-version even without the residual corrections. While RKHS GD is a more powerful method than NWE, providing also the benefit of regularization, it is still interesting to see such a dramatic difference between (1) and (2). We still see, however, that the removal of the residual corrections results in a large performance drop as even the PDS (1) (-2.98 ± 0.03) from the first ablation study is able to outperform this ablation, despite it using 16 steps.

D.3 Decoder ablation

Here we measure the impact of the decoder design $\rho_{\mathcal{H}_k}$ by omitting it entirely. We instead map the discretized functions from RKHS GD directly to the output distribution via a feedforward network.

We obtain a NLL of (-3.14 ± 0.04), which still clearly beats the baselines and is only slightly worse than the full model while using only roughly one third of its parameters. This implies that the impact of having an attention-based decoder at all is relatively small, although this effect may also be downplayed by the simplicity of the linear-sinusoid task.

While parameter efficiency is not among the focus points of our work, this result is encouraging in terms of our method's ability to also bring significant increases to parameter efficiency of prediction map models. More thorough investigation on this is still required, and we leave this for future work.

D.4 Kernel design ablation

Finally, we investigate the effect of the kernel design by repeating the experiment with (1) an EQ-kernel with fixed hyperparameters, (2) an EQ-kernel with meta-learned hyperparameters, and (3) a deep (EQ) kernel with meta-learned hyperparameters.

Table 4: Negative Log-Likelihoods from the kernel design ablation.

PDS (original)	PDS (1)	PDS (2)	PDS (3)
-3.30 ± 0.04	-2.85 ± 0.05	-3.01 ± 0.03	-3.26 ± 0.05

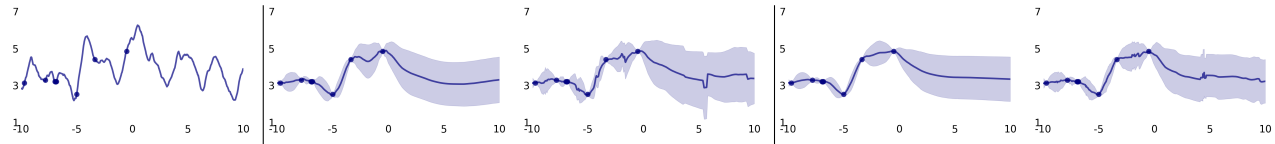
The results shown in Table 4 indicate that constructing the RKHS in a data-driven way leads to significant increases in model performance; using a deep RKHS is more impactful than simply meta-learning hyperparameters of an EQ-kernel.

We also see that the proposed use of deep transfer kernels brings only a small improvement over standard deep kernels. This is perhaps not that surprising given that the optimization via RKHS GD effectively already handles

adapting the function representations to individual datasets, overlapping with the purpose of transfer kernels. The more important factor is that the RKHS is expressively fitted to the particular task domain.

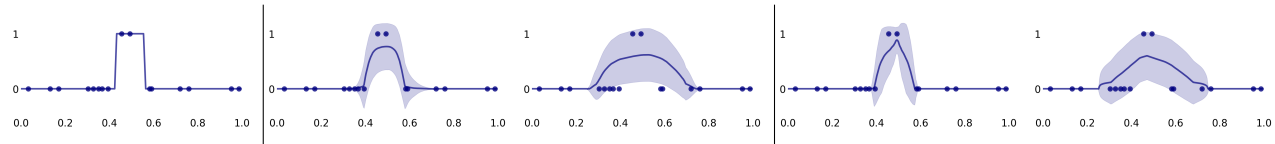
E ADDITIONAL RESULTS

Here we gather additional qualitative results. Figures 7, 8, and 9 illustrate results for the function estimation tasks, and Figures 10, 11, and 12 for the parameter inference tasks. Similarly as with the results reported in Section 5, the benefits of the PDS against the baselines are generally clearer in function estimation tasks.



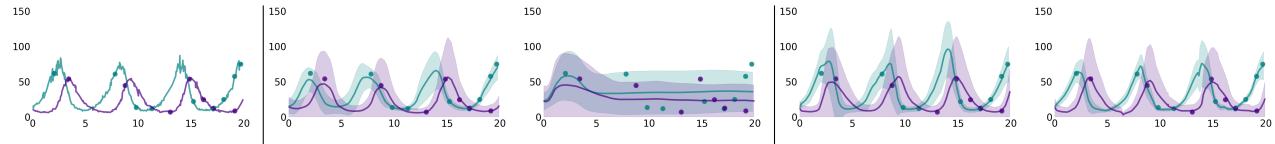
(a) Ground truth. (b) PDS-NP (1k). (c) TNP-D (1k). (d) PDS-NP (100k). (e) TNP-D (100k).

Figure 7: Illustration of the performance of the proposed redictive Deep Sets (PDS) at a neural process prediction task with the Gaussian Process model for different amounts of training datasets (1k and 100k). Both models correctly represent the uncertainties in their predictions, given that the number of context points is here too small to identify the function due to the erratic nature of a GP with Matérn kernel. PDS seems to overall achieve smoother predictions, while TNP-D frequently produces jittery artifacts.



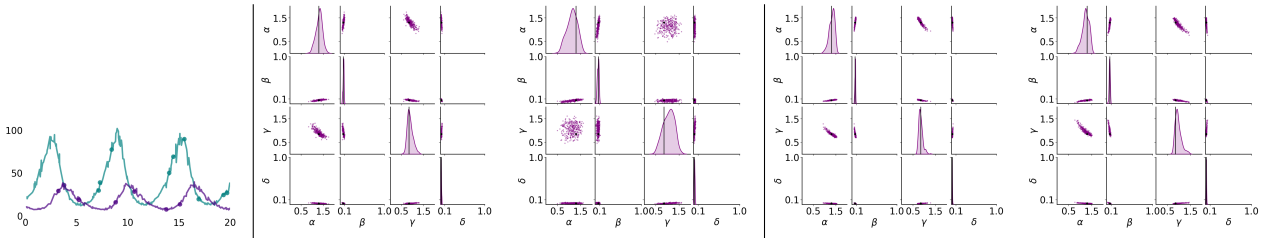
(a) Ground truth. (b) PDS-NP (1k). (c) TNP-D (1k). (d) PDS-NP (100k). (e) TNP-D (100k).

Figure 8: Illustration of the performance of the proposed Predictive Deep Sets (PDS) at a neural process prediction task with the square function model for different amounts of training datasets (1k and 100k). Independently of the amount of data, we see how PDS is able to better capture the sharp-edged structure than TNP-D.



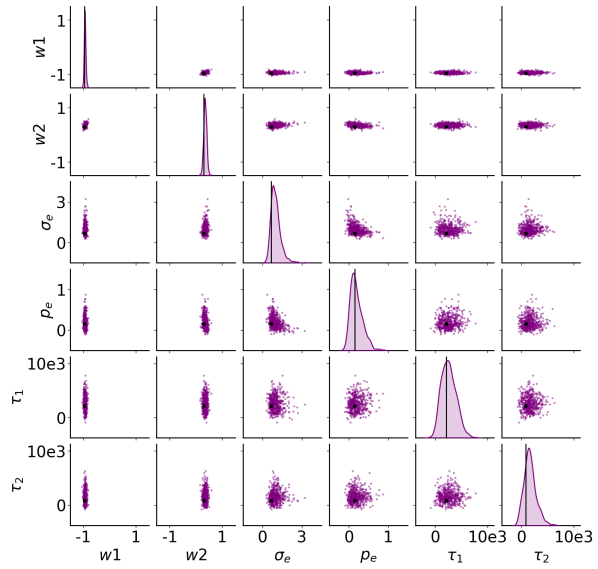
(a) Ground truth. (b) PDS-NP (1k). (c) TNP-D (1k). (d) PDS-NP (100k). (e) TNP-D (100k).

Figure 9: Illustration of the proposed Predictive Deep Sets (PDS) at one neural process prediction task with the Lotka-Volterra model for different amounts of training datasets (1k and 100k). The plots depict the evolution of prey (teal) and predator (purple) populations as a function of time (years). With mere 1k simulations, we see that PDS is capable of faithfully capturing the functional behaviour of the data while D-TNP completely misses this structure. TNP-D needs nearly 100k simulations to model this functional behavior and to obtain performance similar to PDS-NP (1k).

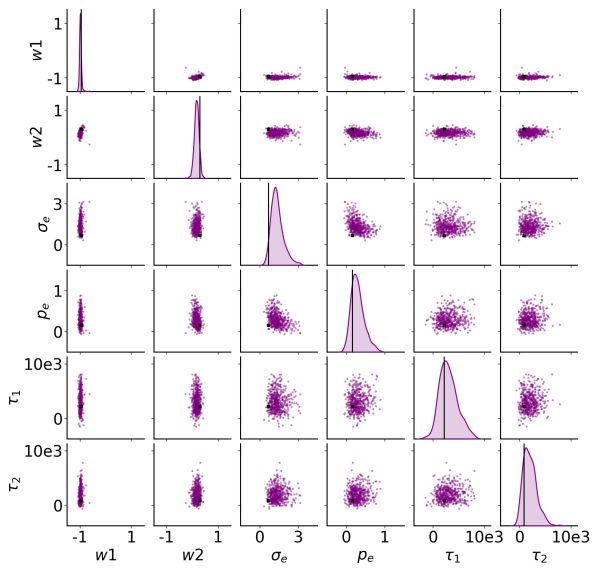


(a) Observations. (b) PDS (1k). (c) ACE (1k). (d) PDS (100k). (e) ACE (100k).

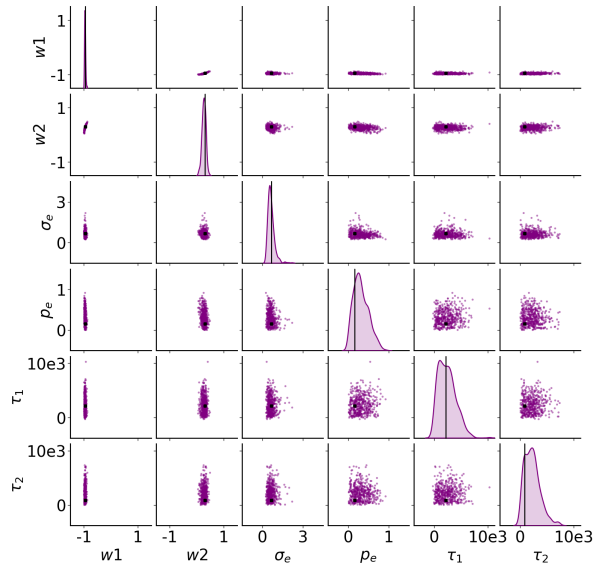
Figure 10: Parameter inferences in the Lotka-Volterra task at 1k and 100k training datasets. The figure (a) shows the ground truth function and the observed datapoints. Figures (b-e) illustrate pairwise plots of parameter posterior samples (purple) estimated from 16 choices, and the ground truth parameters (black). The posteriors of PDS (ours) are better concentrated around the true parameters with 1k datasets than ACE. The difference gets smaller when the models are trained with 100k datasets.



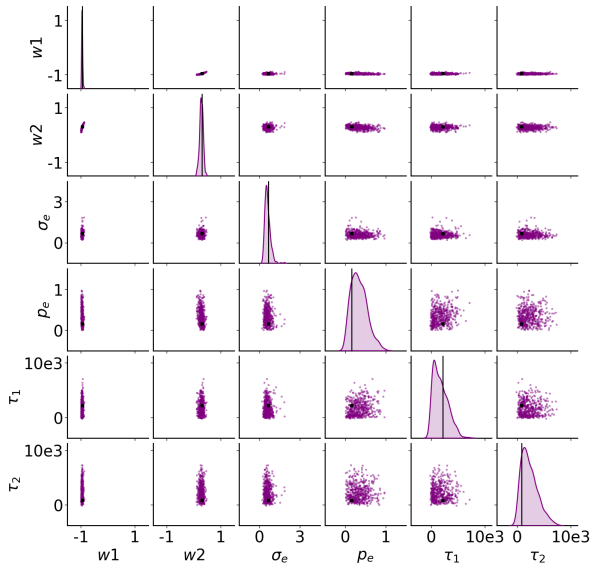
(a) PDS (1k).



(b) ACE (1k).

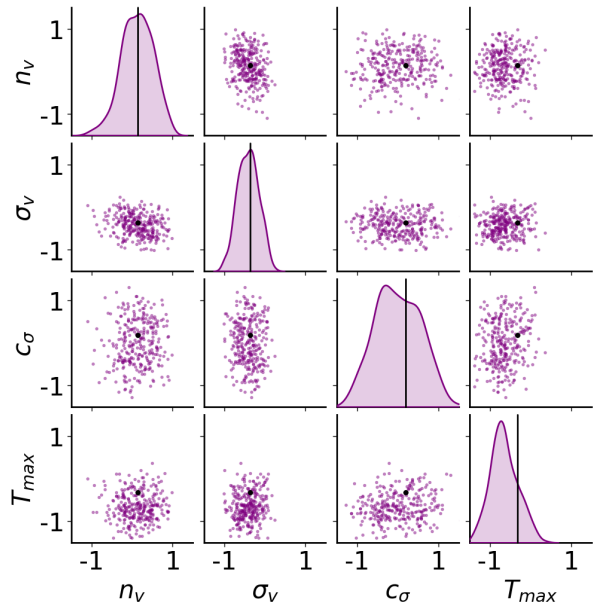


(c) PDS (100k).

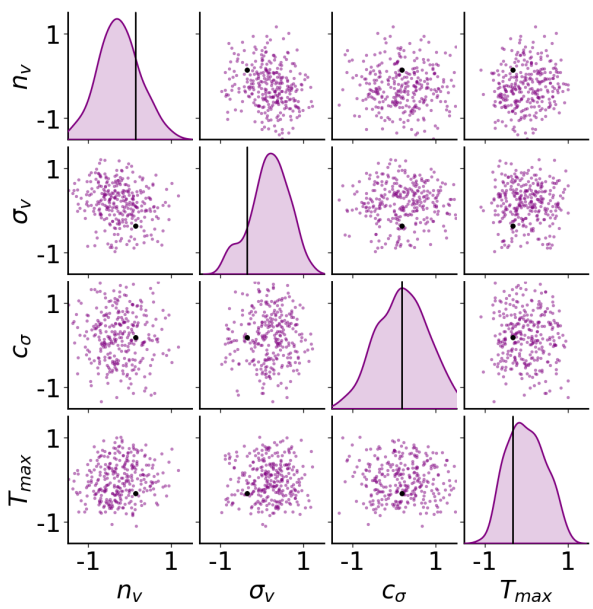


(d) ACE (100k).

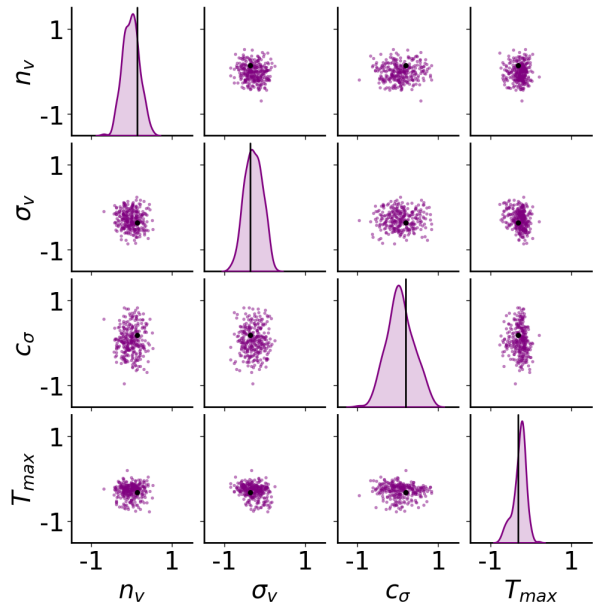
Figure 11: Parameter inferences in the Choice model task at 1k and 100k training datasets. The figures illustrate pairwise plots of parameter posterior samples (purple) estimated from 16 choices, and the ground truth parameters (black). The posteriors of PDS (ours) are slightly better concentrated around the true parameters with 1k datasets than ACE. The difference gets minimal when the models are trained with 100k datasets.



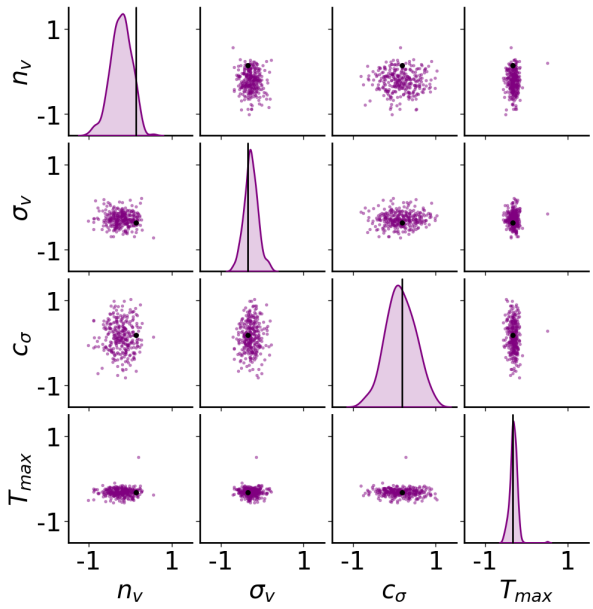
(a) PDS (1k).



(b) ACE (1k).



(c) PDS (100k).



(d) ACE (100k).

Figure 12: Parameter inferences in the Point-and-Click task at 1k and 100k training datasets. The figures illustrate pairwise plots of parameter posterior samples (purple) estimated from 128 timesteps over multiple trials, and the ground truth parameters (black). The posteriors of PDS (ours) are clearly better concentrated around the true parameters with 1k datasets than ACE. The difference gets persists but gets smaller when the models are trained with 100k datasets.