

# Amortized Probabilistic Conditioning for Optimization, Simulation and Inference

Paul E. Chang<sup>\*1</sup> Nasrulloh Loka<sup>\*1</sup> Daolang Huang<sup>\*2</sup> Ulpu Remes<sup>3</sup>  
Samuel Kaski<sup>2,4</sup> Luigi Acerbi<sup>1</sup>

<sup>1</sup>Department of Computer Science, University of Helsinki, Helsinki, Finland

<sup>2</sup>Department of Computer Science, Aalto University, Espoo, Finland

<sup>3</sup>Department of Mathematics and Statistics, University of Helsinki, Helsinki, Finland

<sup>4</sup>Department of Computer Science, University of Manchester, Manchester, United Kingdom

## Abstract

Amortized meta-learning methods based on pre-training have propelled fields like natural language processing and vision. Transformer-based neural processes and their variants are leading models for probabilistic meta-learning with a tractable objective. Often trained on synthetic data, these models implicitly capture essential latent information in the data-generation process. However, existing methods do not allow users to flexibly *inject* (condition on) and *extract* (predict) this probabilistic latent information at runtime, which is key to many tasks. We introduce the Amortized Conditioning Engine (ACE), a new transformer-based meta-learning model that explicitly represents latent variables of interest. ACE affords conditioning on both observed data and interpretable latent variables, the inclusion of priors at runtime, and outputs predictive distributions for discrete and continuous data and latents. We show ACE’s practical utility across diverse tasks such as image completion and classification, Bayesian optimization, and simulation-based inference, demonstrating how a general conditioning framework can replace task-specific solutions.

## 1 INTRODUCTION

Amortization, or pre-training, is a crucial technique for improving computational efficiency and generalization

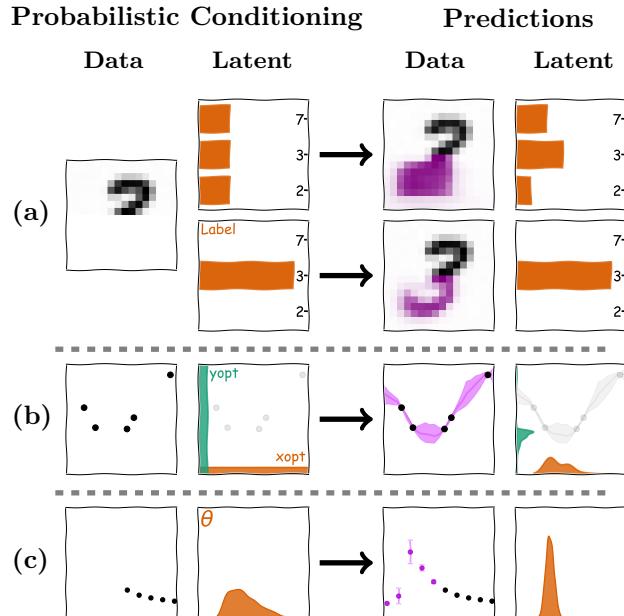


Figure 1: **Probabilistic conditioning and prediction.** Many tasks reduce to *probabilistic conditioning* on data and key latent variables (left) and then *predicting* data and latents (right). (a) Image completion and classification (data: pixels; latents: classes). *Top:* Class prediction. *Bottom:* Conditional generation. (b) Bayesian optimization (data: function values; latents: optimum location  $x_{\text{opt}}$  and value  $y_{\text{opt}}$ ). We predict both the function values and  $x_{\text{opt}}, y_{\text{opt}}$  given function observations and a prior over  $x_{\text{opt}}, y_{\text{opt}}$  (here flat). (c) Simulator-based inference (data: observations; latents: model parameter  $\theta$ ). Given data and a prior over  $\theta$ , we can compute both the posterior over  $\theta$  and predictive distribution over unseen data. Our method fully amortizes probabilistic conditioning and prediction.

\*Equal contribution.

and simulation-based inference (Cranmer et al., 2020). By training a deep neural network on a large dataset of related problems and solutions, amortization can achieve both fast inference time, solving problems with a single forward pass, and meta-learning, better adapting to new problems by capturing high-level statistical relations (Brown et al., 2020). Probabilistic meta-learning models based on the transformer architecture (Vaswani et al., 2017) are the state-of-the-art for amortizing complex predictive data distributions (Nguyen and Grover, 2022; Müller et al., 2022).

This paper capitalizes on the fact that many machine learning problems reduce to *predicting* data and task-relevant latent variables after *conditioning* on other data and latents (Ghahramani, 2015); see Fig. 1. Moreover, in many scenarios the user has exact or probabilistic information (priors) about task-relevant variables that they would like to leverage, but incorporating such prior knowledge is challenging, requiring dedicated, expensive solutions.

For instance, in Bayesian optimization (Garnett, 2023), the goal is to find the location  $\mathbf{x}_{\text{opt}}$  and value  $y_{\text{opt}}$  of the global minimum of a function (Fig. 1b). These are *latent variables*, distinct from the data  $\mathcal{D}_N = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$  consisting of observed function location and values. Following information-theoretical principles, we should query points that would reduce uncertainty (entropy) about the latent optimum’s location or value. However, predictive distributions over  $\mathbf{x}_{\text{opt}}$  and  $y_{\text{opt}}$  are intractable, leading to complex approximation techniques (Hennig and Schuler, 2012; Hernández-Lobato et al., 2014; Wang and Jegelka, 2017). Another case of interest is when prior information is available, such as knowing  $y_{\text{opt}}$  due to the problem formulation (e.g.,  $y_{\text{opt}} = 0$  for some theoretical reason), or expert knowledge of more likely locations for  $\mathbf{x}_{\text{opt}}$  – but injecting such information in current methods is highly nontrivial (Nguyen and Osborne, 2020; Souza et al., 2021; Hvarfner et al., 2022). Crucially, if we had access to  $p(\mathbf{x}_{\text{opt}}, y_{\text{opt}} | \mathcal{D}_N)$ , and we could likewise condition on  $\mathbf{x}_{\text{opt}}$  or  $y_{\text{opt}}$  or set priors over them, many challenging tasks that so far have required dedicated heuristics or computationally expensive solutions would become straightforward. Similar challenges extend to many machine learning tasks, including regression and classification (Fig. 1a), and simulation-based inference (Fig. 1c), all involving predicting, sampling, and probabilistic conditioning on either exact values or distributions (priors) at runtime.

In this work, we address the desiderata above by introducing the *Amortized Conditioning Engine* (ACE),

a general amortization framework which extends transformer-based meta-learning architectures (Nguyen and Grover, 2022; Müller et al., 2022) with explicit and flexible probabilistic modeling of task-relevant latent variables. Our main goal with ACE is to develop a method capable of addressing a variety of tasks that would otherwise require bespoke solutions and approximations. Through the lens of amortized probabilistic conditioning and prediction, we provide a unifying methodological bridge across multiple fields.

**Contributions.** Our contributions include:

- We propose ACE, a transformer-based architecture that simultaneously affords, at inference time, conditioning and autoregressive probabilistic prediction for arbitrary combinations of data and latent variables, both continuous and discrete.
- We introduce a new technique for allowing the user to provide probabilistic information (priors) over each latent variable at inference time.
- We substantiate the generality of our framework through a series of tasks from different fields, including image completion and classification, Bayesian optimization, and simulation-based inference, on both synthetic and real data.

ACE requires availability of predefined, interpretable latent variables during training (e.g.,  $\mathbf{x}_{\text{opt}}, y_{\text{opt}}$ ). For many tasks, this can be achieved through explicit construction of the generative model, as shown in Section 4.

## 2 PRELIMINARIES

In this section, we review previous work on transformer-based probabilistic meta-learning models within the framework of prediction maps (Foong et al., 2020; Markou et al., 2022) and Conditional Neural Processes (CNPs; Garnelo et al., 2018b). We denote with  $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^D$  input vectors (covariates) and  $y \in \mathcal{Y} \subseteq \mathbb{R}$  scalar output vectors (values). Table S1 in Appendix A summarizes key acronyms used in the paper.

**Prediction maps.** A *prediction map*  $\pi$  is a function that maps (1) a *context set* of input/output pairs  $\mathcal{D}_N = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_N, y_N)\}$  and (2) a collection of *target inputs*  $\mathbf{x}_{1:M}^* \equiv (\mathbf{x}_1^*, \dots, \mathbf{x}_M^*)$  to a distribution over the corresponding *target outputs*  $y_{1:M}^* \equiv (y_1^*, \dots, y_M^*)$ :

$$\pi(y_{1:M}^* | \mathbf{x}_{1:M}^*; \mathcal{D}_N) = p(y_{1:M}^* | \mathbf{r}(\mathbf{x}_{1:M}^*, \mathcal{D}_N)), \quad (1)$$

where  $\mathbf{r}$  is a *representation vector* of the context and target sets that parameterizes the predictive distribution. Such map should be invariant with respect to permutations of the context set and, separately, of the

targets (Foong et al., 2020). The Bayesian posterior is a prediction map, with the Gaussian Process (GP; Rasmussen and Williams, 2006) posterior a special case.

**Diagonal prediction maps.** We call a prediction map *diagonal* if it represents and predicts each target independently:

$$\pi(y_{1:M}^* | \mathbf{x}_{1:M}^*; \mathcal{D}_N) = \prod_{m=1}^M p(y_m^* | \mathbf{r}(\mathbf{x}_m^*, \mathbf{r}_D(\mathcal{D}_N))) \quad (2)$$

where  $\mathbf{r}_D$  denotes a representation of the context alone. Diagonal outputs ensure that a prediction map is permutation and marginalization consistent with respect to the targets for a fixed context, necessary conditions for a valid stochastic process (Markou et al., 2022). Importantly, while diagonal prediction maps directly model conditional 1D marginals, they can represent *any* conditional joint distribution autoregressively (Bruinsma et al., 2023). CNPs are diagonal prediction maps parameterized by deep neural networks (Garnelo et al., 2018b). CNPs encode the context set to a *fixed-dimension* vector  $\mathbf{r}_D$  via a DeepSet (Zaheer et al., 2017) to ensure permutation invariance of the context, and each target predictive distribution is a Gaussian whose mean and variance are decoded by a multi-layer perceptron (MLP). Given the likelihood in Eq. (2), CNPs are easily trainable via maximum-likelihood optimization of parameters of encoder and decoder networks by sampling batches of context and target sets.

**Transformers.** Transformers (Vaswani et al., 2017) are deep neural networks based on the attention mechanism, which computes a weighted combination of hidden states of dimension  $D_{\text{emb}}$  through three learnable linear projections: query ( $Q$ ), key ( $K$ ), and value ( $V$ ). The attention operation,  $\text{softmax}(QK^T / \sqrt{D_{\text{emb}}})V$ , captures complex relationships between inputs. *Self-attention* computes  $Q$ ,  $K$ , and  $V$  from the same input set, whereas *cross-attention* uses two sets, one for computing the queries and another for keys and values. A standard transformer architecture consists of multiple stacked self-attention and MLP layers with residual connections and layer normalization (Vaswani et al., 2017). Without specifically injecting positional information, transformers process inputs in a permutation-equivariant manner.

**Transformer diagonal prediction maps.** We define here a general *transformer prediction map* model family, focusing on its *diagonal* variant (TPM-D), which includes the TNP-D model from Nguyen and Grover (2022) and *prior-fitted networks* (PFNs; Müller et al., 2022). TPM-Ds are not strictly CNPs because the context set is encoded by a *variable-size* representation, but they otherwise share many similarities. In

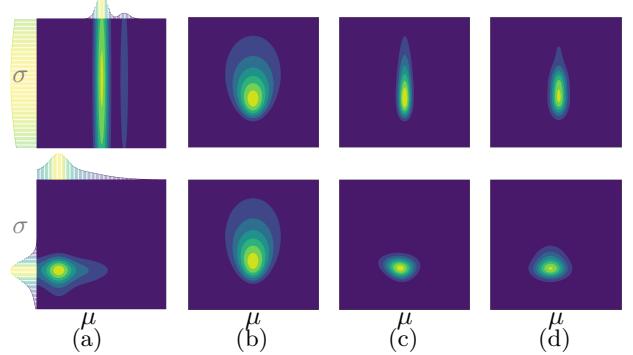


Figure 2: **Prior amortization.** Two example posterior distributions for the mean  $\mu$  and standard deviation  $\sigma$  of a 1D Gaussian. (a) Prior distribution over  $\theta = (\mu, \sigma)$  set at runtime. (b) Likelihood for the observed data. (c) Ground-truth Bayesian posterior. (d) ACE’s predicted posterior approximates well the true posterior.

a TPM-D, context data  $(\mathbf{x}_n, y_n)_{n=1}^N$  and target inputs  $(\mathbf{x}_m^*)_{m=1}^M$  are first individually mapped to vector embeddings of size  $D_{\text{emb}}$  via an embedder  $f_{\text{emb}}$ , often a linear map or an MLP. The embedded context points are processed together via a series of  $B - 1$  transformer layers implementing *self-attention* within the context set. We denote by  $\mathbf{E}^{(b)} = (\mathbf{e}_1^{(b)}, \dots, \mathbf{e}_N^{(b)})$  the matrix of output embeddings of the  $b$ -th transformer layer, with  $b = 0$  the embedding layer. The encoded context representation is the stacked output of all layers, i.e.  $\mathbf{r}_D = (\mathbf{E}^{(0)}, \dots, \mathbf{E}^{(B-1)})$ , whose size is *linear* in the context size  $N$ . The decoder is represented by a series of  $B$  transformer layers that apply *cross-attention* from the embedded target points to the context set layer-wise, with the  $b$ -th target transformer layer attending the output  $\mathbf{E}^{(b-1)}$  of the previous context transformer layer. The decoder transformer layers operate *in parallel* on each target point. The  $M$  outputs of the  $B$ -th decoder block are fed in parallel to an output head yielding the predictive distribution, Eq. (2). This shows that indeed TPM-Ds are diagonal prediction maps. The predictive distribution is a single Gaussian in TNP-D (Nguyen and Grover, 2022) and a ‘Riemannian distribution’ (a mixture of uniform distributions with fixed bin edges and half-Gaussian tails on the sides) in PFNs (Müller et al., 2022). While in TPM-Ds encoding is mathematically decoupled from decoding, in practice encoding and decoding are commonly implemented in parallel within a single transformer layer via masking (Nguyen and Grover, 2022; Müller et al., 2022).

### 3 AMORTIZED CONDITIONING ENGINE

We describe now our proposed Amortized Conditioning Engine (ACE) architecture, which affords arbitrary probabilistic conditioning and predictions. We assume

the problem has  $L$  task-relevant latent variables of interest  $\boldsymbol{\theta} = (\theta_1, \dots, \theta_L)$ . ACE amortizes arbitrary conditioning over latents (in context) and data to predict arbitrary combinations of latents (in target) and data. ACE also amortizes conditioning on probabilistic information about unobserved latents, represented by an approximate prior distribution  $p(\theta_l)$  for  $l \in \{1, \dots, L\}$ ; see Fig. 2 for an example (details in Appendix B.1).

### 3.1 ACE encodes latents and priors

We demonstrate here that ACE is a new member of the TPM-D family, by extending the prediction map formalism to explicitly accommodate latent variables. In ACE, we aim to seamlessly manipulate variables that could be either data points  $(\mathbf{x}, y)$  or latent variables  $\theta_l$ , for a finite set of continuous or discrete-valued latents  $1 \leq l \leq L$ . We redefine inputs as  $\xi \in \mathcal{X} \cup \{\ell_1, \dots, \ell_L\}$  where  $\mathcal{X} \subseteq \mathbb{R}^D$  denotes the data input space (covariates) and  $\ell_l$  is a marker for the  $l$ -th latent. We also redefine the values as  $z \in \mathcal{Z} \subseteq \mathbb{R}$  where  $\mathcal{Z}$  can be continuous or a finite set of integers for discrete-valued output. Thus,  $(\xi, z)$  could denote either a (input, output) data pair or a (index, value) latent pair with either continuous or discrete values. With these new flexible definitions, ACE is indeed a transformer diagonal prediction map (TPM-D). In particular, we can predict any combination of target variables (data or latents) conditioning on any other combination of context data and latents,  $\mathfrak{D}_N = \{(\xi_1, z_1), \dots, (\xi_N, z_N)\}$ :

$$\pi(z_{1:M}^* | \xi_{1:M}^*; \mathfrak{D}_N) = \prod_{m=1}^M p(z_m^* | \mathbf{r}(\xi_m^*, \mathbf{r}_{\mathcal{D}}(\mathfrak{D}_N))). \quad (3)$$

**Prior encoding.** ACE also allows the user to express probabilistic information over latent variables as prior probability distributions at runtime. Our method affords prior specification separately for each latent, corresponding to a factorized prior  $p(\boldsymbol{\theta}) = \prod_{l=1}^L p(\theta_l)$ . To flexibly approximate a broad class of distributions, we convert each one-dimensional probability density function  $p(\theta_l)$  to a normalized histogram of probabilities  $\mathbf{p}_l \in [0, 1]^{N_{\text{grid}}}$  over a predefined grid  $\mathcal{G}$  of  $N_{\text{bins}}$  bins uniformly covering the range of values. We can represent this probabilistic conditioning information within the prediction map formalism by extending the context output representation to  $z \in \{\mathcal{Z} \cup [0, 1]^{N_{\text{bins}}}\}$ , meaning that a context point either takes a specific value or a prior defined on  $\mathcal{G}$  (see Appendix B.1).

### 3.2 ACE architecture

We detail below how ACE extends the general TPM-D architecture presented in Section 2 to implement latent and prior encoding, enabling flexible probabilistic conditioning and prediction. We introduce a novel

embedding layer for latents and priors, adopt an efficient transformer layer implementation, and provide an output represented by a flexible Gaussian mixture or categorical distribution. See Appendix B.2 for an illustration and comparison to the TNP-D architecture.

**Embedding layer.** In ACE, the embedders map context and target data points and latents to the same embedding space of dimension  $D_{\text{emb}}$ . The ACE embedders handle discrete and continuous inputs without the need of tokenization. For the context set, we embed an observed data point  $(\mathbf{x}_n, y_n)$  as  $f_{\mathbf{x}}(\mathbf{x}_n) + f_{\text{val}}(y_n) + \mathbf{e}_{\text{data}}$ , while we embed an observed latent variable  $\theta_l$  as  $f_{\text{val}}(\theta_l) + \mathbf{e}_l$ , where  $\mathbf{e}_{\text{data}}$  and  $\mathbf{e}_l$  for  $1 \leq l \leq L$  are learnt vector embeddings, and  $f_{\mathbf{x}}$  and  $f_{\text{val}}$  are learnt nonlinear embedders (MLPs) for the covariates and values, respectively. For discrete-valued variables (data or latents),  $f_{\text{val}}$  is replaced by a vector embedding matrix  $\mathbf{E}_{\text{val}}$  with a separate row for each discrete value. Latent variables with a prior  $\mathbf{p}_l$  are mapped to the context set as  $f_{\text{prob}}(\mathbf{p}_l) + \mathbf{e}_l$ , where  $f_{\text{prob}}$  is a learnt MLP. In the target set, the *value* of a variable is unknown and needs to be predicted, so we replace the value embedders above with a learnt ‘unknown’ embedding  $\mathbf{e}_?$ , i.e.  $f_{\mathbf{x}}(\mathbf{x}_n) + \mathbf{e}_? + \mathbf{e}_{\text{data}}$  for data and  $\mathbf{e}_? + \mathbf{e}_l$  for latents.

**Transformer layers.** The embedding layer is followed by  $B$  stacked transformer layers, each with a multi-head attention block followed by a MLP (Vaswani et al., 2017). Both the attention and MLP blocks are followed by a normalization layer and include skip connections. The attention block combines encoder and decoder in the same step, with self-attention on the context points (encoding) and cross-attention from the target points to the context (decoding). Computing a single masked context + target attention matrix would incur  $O((N + M)^2)$  cost (Nguyen and Grover, 2022; Müller et al., 2022). Instead, by separating the context self-attention and target cross-attention matrices we incur a  $O(N^2 + NM)$  cost (Feng et al., 2023).

**Output heads.** A prediction output head is applied in parallel to all target points after the last transformer layer. For a continuous-valued variable, the output head is a Gaussian mixture, consisting of  $K$  MLPs that separately output the parameters of  $K$  1D Gaussians, i.e., ‘raw’ weight, mean, standard deviation for each mixture component (Uria et al., 2016). A learnt global raw bias term ( $3 \times K$  parameters) is added to each raw output, helping the network learn deviations from the global distribution of values. Then weights, means and standard deviations for each Gaussian are obtained through appropriate transformations (softmax, identity, and softplus, respectively). For discrete-valued variables, the output head is a MLP that outputs a softmax categorical distribution over the discrete values.

### 3.3 Training and prediction

ACE is trained via maximum-likelihood on synthetic data consisting of batches of context and target sets, using the Adam optimizer (details in Appendix B.3).

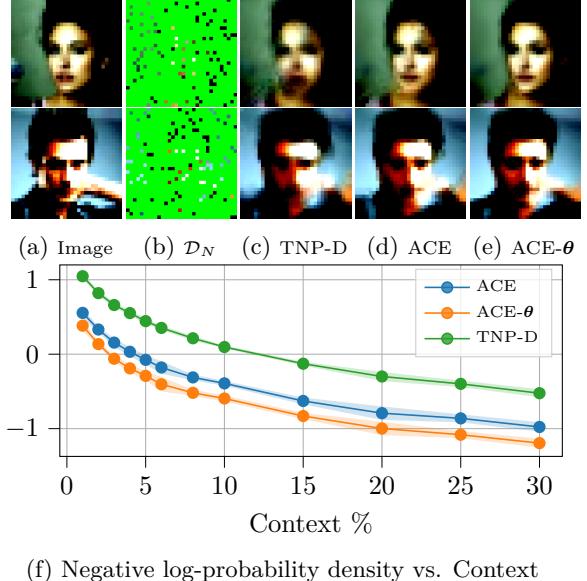
**Training.** We generate each problem instance hierarchically by first sampling the latent variables  $\theta$ , and then data points  $(\mathbf{X}, \mathbf{y})$  according to the generative model of the task. For example,  $\theta$  could be length scale and output scale of a 1D Gaussian process with a given kernel, and  $(\mathbf{X}, \mathbf{y})$  input locations and function values. Data and latents are randomly split between context and target. For training with probabilistic information  $\mathbf{p}_l$ , we first sample the priors for each latent variable from a hierarchical model  $\mathcal{P}$  which includes mixtures of Gaussians and Uniform distributions (see Appendix B.1) and then sample the value of the latent from the chosen prior. During training, we minimize the expected negative log-likelihood of the target set conditioned on the context,  $\mathcal{L}(\mathbf{w})$ :

$$\mathbb{E}_{\mathbf{p} \sim \mathcal{P}} \left[ \mathbb{E}_{\mathfrak{D}_N, \xi_{1:M}, \mathbf{z}_{1:M} \sim \mathbf{p}} \left[ - \sum_{m=1}^M \log q(z_m^* | \mathbf{r}_w(\xi_m^*, \mathfrak{D}_N)) \right] \right] \quad (4)$$

where  $q$  is our model’s prediction (a mixture of Gaussians or categorical), and  $\mathbf{w}$  are the model parameters. Minimizing Eq. (4) is equivalent to minimizing the Kullback-Leibler (KL) divergence between the data sampled from the generative process and the model. Since the generative process is consistent with the provided contextual prior information, training will aim to converge (KL-wise) as close as possible, for the model capacity, to the correct Bayesian posteriors and predictive distributions for the specified generative model and priors (Müller et al., 2022; Elsemüller et al., 2024).

**Prediction.** ACE is trained via *independent* predictions of target data and latents, Eq. (4). Given the closed-form likelihood (mixture of Gaussians or categorical), we can easily evaluate or sample from the predictive distribution at any desired target point (data or latent) in parallel, conditioned on the context. Moreover, we can predict *non-independent* joint distributions autoregressively (Nguyen and Grover, 2022; Bruinsma et al., 2023); see Appendix B.4 for details.

**Task-specific contributions.** The availability of predictive distributions in closed form allows ACE to simplify tasks or perform new ones. We give an example in Section 4.2, where ACE facilitates the computation of acquisition functions in Bayesian optimization.



**Figure 3: Image completion.** Image (a) serves as the reference for the problem, where 10% of the pixels are observed (b). Figures (c) through (e) display different models’ prediction conditioned on the observed pixels (b). In addition, (e) incorporates latent variable  $\theta$  information for the ACE model. Figure (f) illustrates the different models’ performance across varying levels of context.

## 4 EXPERIMENTS

The following section showcases ACE’s capabilities as a general framework applicable to diverse machine learning and modeling tasks.<sup>1</sup>

Firstly, Section 4.1 demonstrates how ACE complements transformer-based meta-learning in image completion and classification. In Section 4.2, we show how ACE can be applied to Bayesian optimization (BO) by treating the location and value of the global optimum as latent variables. We then move to simulation-based inference (SBI) in Section 4.3, where ACE unifies the SBI problem into a single framework, treating parameters as latent variables and affording both forward and inverse modelling. Notably, SBI and BO users may have information about the simulator or target function. ACE affords incorporation of informative priors about latent variables at runtime, as detailed in Section 3.1, a variant we call ACEP in these experiments. Finally, in Appendix C.1 we provide extra experimental results on Gaussian Processes (GPs) where ACE can accurately predict the kernel, *i.e.* model selection (Fig. S6), while at the same time learn the hyperparameters, in addition to the common data prediction task.

<sup>1</sup>The code implementation of ACE is available at [github.com/acerbilab/amortized-conditioning-engine/](https://github.com/acerbilab/amortized-conditioning-engine/).

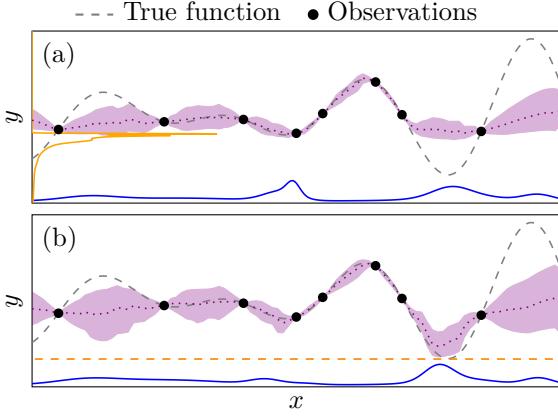


Figure 4: Bayesian Optimization example. (a) ACE predicts function values ( $\dots p(y|x, \mathcal{D}_N)$ ) as well as latents: optimum location ( $\textcolor{blue}{—} p(x_{\text{opt}}|\mathcal{D}_N)$ ) and optimum value ( $\textcolor{orange}{—} p(y_{\text{opt}}|\mathcal{D}_N)$ ). (b) Further conditioning on  $\textcolor{orange}{---} y_{\text{opt}}$  (here the true minimum value) leads to updated predictions.

#### 4.1 Image completion and classification

We treat image completion as a regression task (Garnelo et al., 2018a), where the goal is given some limited  $\mathcal{D}_N$  of image coordinates and corresponding pixel values to predict the complete image. For the MNIST (Deng, 2012) task, we downsize the images to  $16 \times 16$  and likewise for CELEBA to  $32 \times 32$  (Liu et al., 2015). We turn the class label into a single discrete latent for MNIST while for CELEBA, we feed the full 40 corresponding binary features (e.g., BROWNAIR, MAN, SMILING). The latents are sampled using the procedure outlined in Appendix B.3 and more experimental details of the image completion task can be found in Appendix C.2. Notably, ACE affords conditional image generation, i.e., predictions of pixels based on latent variables ( $\boldsymbol{\theta}$ ) – such as class labels in MNIST and appearance features in CELEBA – as well as image classification, i.e. the prediction of these latent variables themselves from pixel values.

**Results.** In Fig. 3 we present a snapshot of the results for CELEBA and in Appendix C.2 we present the same for MNIST. The more complex output distribution allows ACE to outperform other Transformer NPs convincingly, and the integration of latent information shows a clear improvement. In Appendix C.2, we present our full results, including predicting  $\boldsymbol{\theta}$ .

#### 4.2 Bayesian optimization (BO)

BO aims to find the global minimum  $y_{\text{opt}} = f(\mathbf{x}_{\text{opt}})$  of a black-box function. This is typically achieved iteratively by building a *surrogate model* that approximates the target and optimizing an *acquisition function*  $\alpha(\mathbf{x})$

to determine the next query point. ACE provides additional modeling options for the BO loop by affording direct conditioning on, and prediction of, key latents  $\mathbf{x}_{\text{opt}}$  and  $y_{\text{opt}}$ , yielding closed-form predictive distributions and samples for  $p(\mathbf{x}_{\text{opt}}|\mathcal{D}_N)$ ,  $p(y_{\text{opt}}|\mathcal{D}_N)$ , and  $p(\mathbf{x}_{\text{opt}}|\mathcal{D}_N, y_{\text{opt}})$ ; see Fig. 4.

For the BO task, we trained ACE on synthetic functions generated from GP samples with RBF and Matérn-( $1/2$ ,  $3/2$ ,  $5/2$ ) kernels and a random global optimum  $(\mathbf{x}_{\text{opt}}, y_{\text{opt}})$  within the function domain; see Appendix C.3 for details. We leverage ACE’s explicit modeling of latents in multiple ways.

**Acquisition functions with ACE.** ACE affords a straightforward implementation of a variant of **Thompson Sampling** (TS) (Dutordoir et al., 2023; Liu et al., 2024). First, we sample a candidate optimum value,  $y_{\text{opt}}^*$ , conditioning on it being below a threshold  $\tau$ , from the truncated predictive distribution  $y_{\text{opt}}^* \sim p(y_{\text{opt}}|\mathcal{D}_N, y_{\text{opt}} < \tau)$ ; see Fig. 4a. Given  $y_{\text{opt}}^*$ , we then sample the query point  $\mathbf{x}^* \sim p(\mathbf{x}_{\text{opt}}|\mathcal{D}_N, y_{\text{opt}}^*)$ ; Fig. 4b.<sup>2</sup> This process is repeated iteratively within the BO loop (see Fig. S14). For higher input dimensions ( $D > 1$ ), we sample  $\mathbf{x}^*$  autoregressively; see Appendix C.3.2. Crucially, ACE’s capability of directly modeling  $\mathbf{x}_{\text{opt}}$  and  $y_{\text{opt}}$  bypasses the need for surrogate optimization or grid searches typical of standard TS implementations (e.g., GP-TS or TNP-D based TS).

ACE also easily supports advanced acquisition functions used in BO, such as **Max-Value Entropy Search** (MES; Wang and Jegelka, 2017). For a candidate point  $\mathbf{x}^*$ , MES evaluates the expected gain in mutual information between  $y_{\text{opt}}$  and  $\mathbf{x}^*$ :

$$\begin{aligned} \alpha_{\text{MES}}(\mathbf{x}^*) &= H[p(y^*|\mathbf{x}^*, \mathcal{D}_N)] \\ &\quad - \mathbb{E}_{p(y_{\text{opt}}|\mathcal{D}_N)} [H[p(y^*|\mathbf{x}^*, \mathcal{D}_N, y_{\text{opt}})]] . \end{aligned} \quad (5)$$

With all predictive distributions available in closed form, ACE can readily calculate the expectation and entropies in Eq. (5) via Monte Carlo sampling and fast 1D numerical integration, unlike other methods that require more laborious approximations. For maximizing  $\alpha(\mathbf{x}^*)$ , we obtain a good set of candidate points via Thompson Sampling (see Appendix C.3.2 for details).

**Results.** We compare ACE with gold-standard Gaussian processes (GPs) and the state-of-the-art TNP-D model (Fig. 5). Additionally, we test a setup where prior information is provided about the location of the optimum (Souza et al., 2021; Hvarfner et al., 2022;

<sup>2</sup>Why not sampling directly from  $p(\mathbf{x}_{\text{opt}}|\mathcal{D}_N)$ ? The issue is that  $p(\mathbf{x}_{\text{opt}}|\mathcal{D}_N)$  may reasonably include substantial probability mass at the current optimum, which would curb exploration. The constraint  $y_{\text{opt}} < \tau$ , with  $\tau$  (just) below the current optimum value, ensures continual exploration.

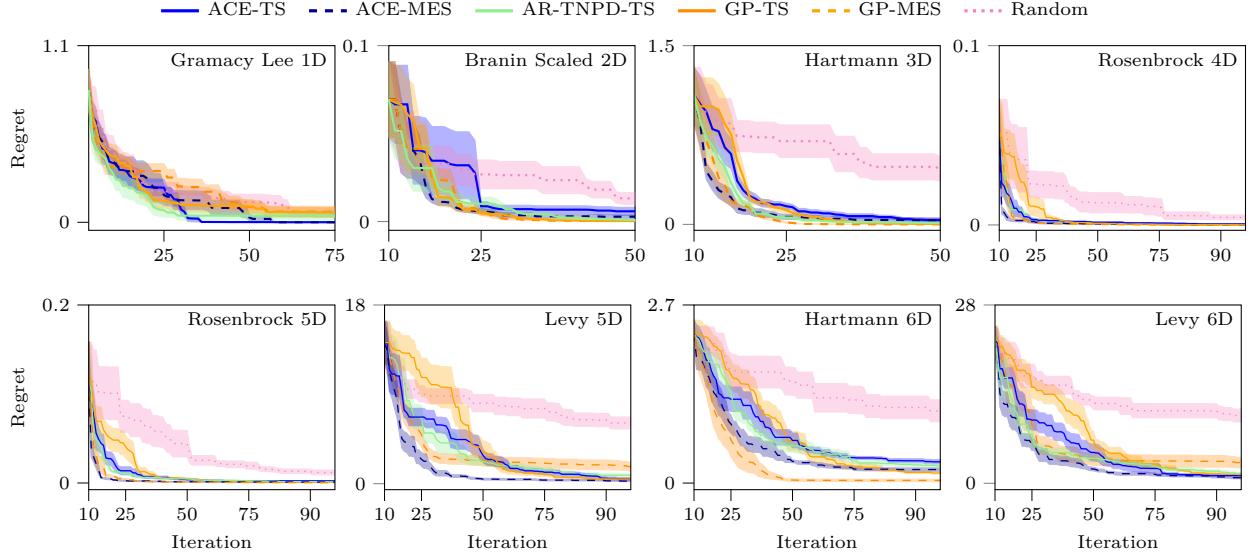


Figure 5: **Bayesian optimization results.** Regret comparison (mean  $\pm$  standard error) for different methods across benchmark tasks.

Müller et al., 2023); Fig. 6. Unlike other methods that employ heuristics or complex approximations, ACE’s architecture affords seamless incorporation of a prior  $p(\mathbf{x}_{\text{opt}})$ . Here, we consider two types of priors: strong and weak, represented by Gaussians with a standard deviation equal to, respectively, 10% and 25% of the optimization box width in each dimension, and mean drawn from a Gaussian centered on the true optimum and same standard deviation (see Appendix C.3.3).

In Fig. 5, we show the performance of ACE Thompson sampling (ACE-TS) and MES (ACE-MES) with GP-based MES (GP-MES; Wang and Jegelka, 2017), GP-based Thompson Sampling (GP-TS; Balandat et al., 2020), and Autoregressive TNP-D based Thompson Sampling (AR-TNPD-TS; Bruinsma et al., 2023; Nguyen and Grover, 2022) on several benchmark functions (see Appendix C.3.4 for details). ACE-MES frequently outperforms ACE-TS and often matches the gold-standard GP-MES. In the prior setting, we compare ACE without (ACE-TS) and with prior (ACEP-TS) against  $\pi$ BO-TS, a state-of-the-art heuristic for prior injection in BO (Hvarfner et al., 2022), as well as the GP-TS baseline. ACEP-TS shows significant improvement over its no-prior variant ACE-TS while showing competitive performance compared to  $\pi$ BO-TS in both weak and strong prior case (Fig. 6).

### 4.3 Simulation-based inference (SBI)

We now apply ACE to simulation-based inference (SBI; Cranmer et al., 2020). With ACE, we can predict the posterior distribution of (latent) model parameters, simulate data based on parameters, predict missing data given partial observations, and set priors at run-

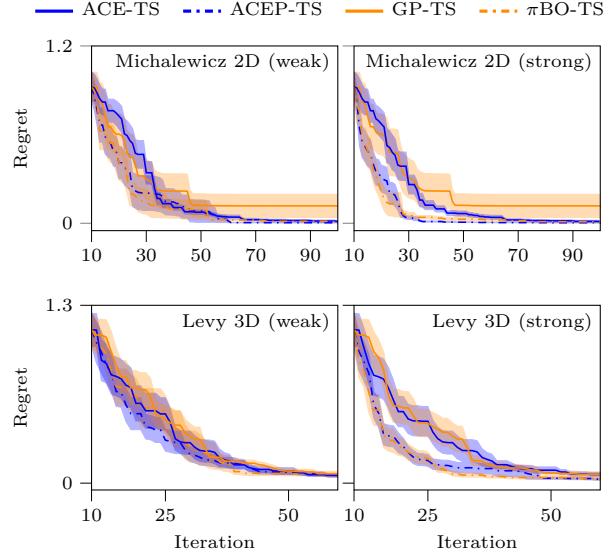


Figure 6: **Bayesian optimization with prior over  $\mathbf{x}_{\text{opt}}$ .** Regret comparison (mean  $\pm$  standard error) on 2D and 3D optimization benchmarks. Left: Weak Gaussian prior (25%), Right: Strong prior (10%). ACEP-TS performs competitively compared to  $\pi$ BO-TS.

time. We consider two benchmark time-series models, each with two latents: the Ornstein-Uhlenbeck Process (OUP; Uhlenbeck and Ornstein 1930) and the Susceptible-Infectious-Recovered model (SIR; Kermack and McKendrick 1927); and a third more complex engineering model from the field of radio propagation (Turin; Turin et al., 1972), which has four parameters and produces 101-dimensional data representing a radio signal. See Appendix C.4 for all model descriptions.

		NPE	NRE	Simformer	ACE	ACEP <sub>weak prior</sub>	ACEP <sub>strong prior</sub>
OUP	log-probs <sub><math>\theta</math></sub> ( $\uparrow$ )	<b>1.09</b> (0.10)	<b>1.07</b> (0.13)	<b>1.03</b> (0.04)	<b>1.03</b> (0.02)	1.05(0.02)	1.44(0.03)
	RMSE <sub><math>\theta</math></sub> ( $\downarrow$ )	<b>0.48</b> (0.01)	0.49(0.00)	0.50(0.02)	<b>0.48</b> (0.00)	0.43(0.01)	0.27(0.00)
	MMD <sub><math>y</math></sub> ( $\downarrow$ )	-	-	<b>0.43</b> (0.02)	0.51(0.00)	0.37(0.00)	0.35(0.00)
SIR	log-probs <sub><math>\theta</math></sub> ( $\uparrow$ )	6.53(0.11)	6.24(0.16)	<b>6.89</b> (0.09)	6.78(0.02)	6.62(0.10)	6.69(0.10)
	RMSE <sub><math>\theta</math></sub> ( $\downarrow$ )	<b>0.02</b> (0.00)	0.03(0.00)	<b>0.02</b> (0.00)	<b>0.02</b> (0.00)	0.02(0.00)	0.02(0.00)
	MMD <sub><math>y</math></sub> ( $\downarrow$ )	-	-	<b>0.02</b> (0.00)	<b>0.02</b> (0.00)	0.02(0.00)	0.00(0.00)
Turin	log-probs <sub><math>\theta</math></sub> ( $\uparrow$ )	1.99(0.05)	2.33(0.07)	<b>3.16</b> (0.03)	<b>3.14</b> (0.02)	3.58(0.04)	4.87(0.08)
	RMSE <sub><math>\theta</math></sub> ( $\downarrow$ )	0.26(0.00)	0.28(0.00)	0.25(0.00)	<b>0.24</b> (0.00)	0.21(0.00)	0.13(0.00)
	MMD <sub><math>y</math></sub> ( $\downarrow$ )	-	-	<b>0.35</b> (0.00)	<b>0.35</b> (0.00)	0.35(0.00)	0.34(0.00)

Table 1: **Comparison metrics for SBI models** on parameters ( $\theta$ ) and data ( $y$ ) prediction; mean and (standard deviation) from 5 runs. *Left:* Statistically significantly (see Appendix C.4.2) best results are **bolded**. ACE shows performance comparable to the other dedicated methods on latents prediction. In the data prediction task, ACE performs similarly to Simformer with much lower sampling cost at runtime (see text). *Right:* ACE can leverage probabilistic information provided at runtime by informative priors (ACEP), yielding improved performance.

We compare ACE with Neural Posterior Estimation (NPE; Greenberg et al. 2019), Neural Ratio Estimation (NRE; Miller et al. 2022), and Simformer (Gloeckler et al., 2024), from established to state-of-the-art methods in amortized SBI. We evaluate ACE in three different scenarios. For the first one, we use only the observed data as context. For the other two scenarios, we inform ACE with priors over the parameters (ACEP), to assess their impact on posterior prediction. These priors are Gaussians with standard deviation equal to 25% (weak) or 10% (strong) of the parameter range, and mean drawn from a Gaussian centered on the true parameter value and the same standard deviation.

We evaluate the performance of posterior estimation using the log probabilities of the ground-truth parameters and the root mean squared error (RMSE) between the true parameters and posterior samples. Since both ACE and Simformer can predict missing data from partial observations – an ability that previous SBI methods lack – we also test them on a data prediction task. For each observed dataset, we randomly designate half of the data points as missing and use the remaining half as context for predictions. We then measure performance via the maximum mean discrepancy (MMD) between the true data and the predicted distributions.

**Results.** Results are reported in Table 1; see Appendix C.4 for details. For ACE, we see that joint training to predict data and latents does not compromise its posterior estimation performance compared to NPE and NRE, even achieving better performance on the Turin model. ACE and Simformer obtain similar results. However, as Simformer uses diffusion, data sampling is substantially slower. For example, we measured the time required to generate 1,000 posterior samples for 100 sets of observations on the OUP model using a CPU (GPU) across 5 runs: the average time for

Simformer is  $\sim 130$  minutes (14 minutes), whereas ACE takes 0.05 seconds (0.02 s). When we provide ACE with informative priors (ACEP; Table 1 right), performance improves in proportion to the provided information. Importantly, simulation-based calibration checks (Talts et al., 2018) show that both ACE and ACEP output good approximate posteriors (Appendix C.4.4).

Finally, we applied ACE to a real-world outbreak dataset (Avilov et al., 2023) using an extended, four-parameter version of the SIR model. We show that ACE can handle real data, providing reasonable results under a likely model mismatch (see Appendix C.4.5).

## 5 RELATED WORK

Our work combines insights from different fields such as neural processes, meta-learning, and simulation-based inference, providing a new unified and versatile framework for amortized inference.

**Neural processes.** ACE relates to the broader work on neural processes (Garnelo et al., 2018a,b; Kim et al., 2019; Gordon et al., 2020; Markou et al., 2022; Nguyen and Grover, 2022; Huang et al., 2023a) and shares similarities with autoregressive diffusion models (Hoogeboom et al., 2022), whose permutation-invariant conditioning set mimics the context set in neural processes. Unlike previous methods that focused on predictive data distributions conditioned on observed data, ACE also explicitly models and conditions on latent variables of interest. While Latent Neural Processes (LNPs) capture correlations via non-interpretable latent vectors that yield an intractable learning objective (Garnelo et al., 2018a), ACE uses autoregressive sampling to model correlations (Bruinsma et al., 2023), and ‘latents’ in ACE denote explicitly modelled task variables.

**Meta-learning.** Our approach falls within the fields of amortized inference, meta-learning, and pre-trained models, which overlap significantly with neural process literature (Finn et al., 2017, 2018). Prior-Fitted Networks (PFNs) demonstrated the use of transformers for Bayesian inference (Müller et al., 2022) and optimization (Müller et al., 2023), focusing on predictive posterior distributions over data using fixed bins ('Riemannian' distributions). In particular, Müller et al. (2023) allow users to indicate a 'high-probability' interval for the optimum location at runtime within a set of specified ranges. ACE differs from these methods by allowing the specification of more flexible priors at runtime, using learnable mixtures of Gaussians for continuous values, and, most importantly, for yielding explicit predictive distributions over both data and task-relevant, interpretable latents. While other works target specific optimization tasks (Liu et al., 2020; Simpson et al., 2021; Amos, 2022), we provide a generative model for random functions with known optima, directly amortizing predictive distributions useful for optimization (see Appendix C.3). More broadly, ACE relates to efforts in amortized inference over explicit task-relevant latents (Mittal et al., 2023), though our focus extends beyond posterior estimation to flexible conditioning and prediction of both data and latents. Additionally, Mittal et al. (2024) show that enforcing a bottleneck in transformers captures task-relevant latents but does not necessarily improve generalization. Unlike their learned latents, ACE conditions on pre-defined, interpretable task variables, enabling direct control and adaptability. By providing methods to condition on interpretable latents as well as injecting probabilistic knowledge (priors), our work aligns with the principles of *informed meta-learning* (Kobalczyk and van der Schaar, 2024). Subsequent to our work, *distribution transformers* were recently proposed as amortized meta-learning models for inference with flexible priors and posteriors (Whittle et al., 2025).

**Simulation-based inference.** ACE is related to the extensive literature on amortized or neural simulation-based inference (SBI) (Cranmer et al., 2020), where deep networks are used to (a) recover the posterior distribution over model parameters (Lueckmann et al., 2017), and (b) emulate the simulator or likelihood (Papamakarios et al., 2019). While these two steps are often implemented separately, recent work has started to combine them (Radev et al., 2023). In particular, the recently proposed Simformer architecture allows users to freely condition on observed variables and model parameters, in a manner similar to ACE (Gloeckler et al., 2024). Simformer uses a combination of a transformer architecture and a denoising diffusion model (Song and Ermon, 2019), which makes it suitable for continuous predictions but does not immediately apply

to discrete outputs. Notably, Simformer allows the user to specify *intervals* for variables at runtime, but not (yet) general priors. To our knowledge, before our paper the only work in amortized SBI that afforded some meaningful flexibility in prior specification at runtime was Elsemüller et al. (2024). Even so, the choice there is between a limited number of fixed priors or a global prior scale parameter, with the main focus being sensitivity analysis (Elsemüller et al., 2024). Importantly, all these works focus entirely on SBI settings, while our paper showcases the applicability of ACE to a wider variety of machine learning tasks.

## 6 DISCUSSION

In this paper, we introduced the Amortized Conditioning Engine (ACE), a unified transformer-based architecture that affords arbitrary probabilistic conditioning and prediction over data and task-relevant variables for a wide variety of tasks. In each tested domain ACE performed on par with state-of-the-art, bespoke solutions, and with greater flexibility. As a key feature, ACE allows users to specify probabilistic information at runtime (priors) without the need for retraining as required instead by the majority of previous amortized inference methods (Cranmer et al., 2020).

**Limitations and future work.** As all amortized and machine learning methods, ACE's predictions become unreliable if applied to data unseen during training due to mismatch between simulated and real data. This is an active area of research in terms of developing both more robust training objectives (Huang et al., 2023b) and diagnostics (Schmitt et al., 2023).

The method's quadratic complexity in context size could benefit from sub-quadratic attention variants (Feng et al., 2023), while incorporating equivariances (Huang et al., 2023a) and variable covariate dimensionality (Liu et al., 2020; Dutordoir et al., 2023; Müller et al., 2023) could further improve performance.

While ACE can learn full joint distributions through 1D marginals (Bruinsma et al., 2023), scaling to many data points and latents remains challenging. Similar scaling challenges affect our prior-setting approach, currently limited to few latents under factorized, smooth priors. Future work should also explore discovering interpretable latents (Mittal et al., 2024) and handling multiple tasks simultaneously (Kim et al., 2022; Ashman et al., 2024), extending beyond our current single-task supervised learning approach.

**Conclusions.** ACE shows strong promise as a new unified and versatile method for amortized probabilistic conditioning and prediction, able to perform various probabilistic machine learning tasks.

## Acknowledgments

PC, DH, UR, SK, and LA were supported by the Research Council of Finland (Flagship programme: Finnish Center for Artificial Intelligence FCAI). NL was funded by Business Finland (project 3576/31/2023). LA was also supported by Research Council of Finland grants 358980 and 356498. SK was also supported by the UKRI Turing AI World-Leading Researcher Fellowship, [EP/W002973/1]. The authors wish to thank the Finnish Computing Competence Infrastructure (FCCI), Aalto Science-IT project, and CSC-IT Center for Science, Finland, for the computational and data storage resources provided, including access to the LUMI supercomputer, owned by the EuroHPC Joint Undertaking, hosted by CSC (Finland) and the LUMI consortium (LUMI project 462000551).

## References

- Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Ali Eslami, and Yee Whye Teh. Neural processes. In *ICML Workshop on Theoretical Foundations and Applications of Deep Generative Models*, 2018a.
- Brandon Amos. Tutorial on amortized optimization for learning to optimize over continuous domains. *arXiv e-prints*, pages arXiv–2202, 2022.
- Kyle Cranmer, Johann Brehmer, and Gilles Louppe. The frontier of simulation-based inference. *Proceedings of the National Academy of Sciences*, 117(48):30055–30062, 2020.
- Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*, 30, 2017.
- Tung Nguyen and Aditya Grover. Transformer Neural Processes: Uncertainty-aware meta learning via sequence modeling. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 123–134. PMLR, 2022.
- Samuel Müller, Noah Hollmann, Sebastian Pineda Arango, Josif Grabocka, and Frank Hutter. Transformers can do Bayesian inference. In *International Conference on Learning Representations*, 2022.
- Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459, 2015.
- Roman Garnett. *Bayesian optimization*. Cambridge University Press, 2023.
- Philipp Hennig and Christian J Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13(6), 2012.
- José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. *Advances in Neural Information Processing Systems*, 27, 2014.
- Zi Wang and Stefanie Jegelka. Max-value entropy search for efficient Bayesian optimization. In *International Conference on Machine Learning*, pages 3627–3635. PMLR, 2017.
- Vu Nguyen and Michael A Osborne. Knowing the what but not the where in Bayesian optimization. In *International Conference on Machine Learning*, pages 7317–7326. PMLR, 2020.
- Artur Souza, Luigi Nardi, Leonardo B Oliveira, Kunle Olukotun, Marius Lindauer, and Frank Hutter. Bayesian optimization with a prior for the optimum. In *Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part III 21*, pages 265–296. Springer, 2021.
- Carl Hvarfner, Danny Stoll, Artur Souza, Luigi Nardi, Marius Lindauer, and Frank Hutter. πBO: Augmenting acquisition functions with user beliefs for bayesian optimization. In *International Conference on Learning Representations*, 2022.
- Andrew YK Foong, Wessel P Bruinsma, Jonathan Gordon, Yann Dubois, James Requeima, and Richard E Turner. Meta-learning stationary stochastic process prediction with convolutional neural processes. In *Advances in Neural Information Processing Systems*, volume 33, pages 8284–8295, 2020.
- Stratis Markou, James Requeima, Wessel P Bruinsma, Anna Vaughan, and Richard E Turner. Practical conditional neural processes via tractable dependent predictions. In *International Conference on Learning Representations*, 2022.
- Marta Garnelo, Dan Rosenbaum, Chris J Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo J Rezende, and SM Ali Eslami. Conditional neural processes. In *International Conference on Machine Learning*, pages 1704–1713, 2018b.
- Carl Edward Rasmussen and Christopher KI Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

- Wessel P Bruinsma, Stratis Markou, James Requeima, Andrew YK Foong, Tom R Andersson, Anna Vaughan, Anthony Buonomo, J Scott Hosking, and Richard E Turner. Autoregressive conditional neural processes. In *International Conference on Learning Representations*, 2023.
- Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabás Póczos, Ruslan Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in Neural Information Processing Systems*, volume 30, pages 3391–3401, 2017.
- Leo Feng, Hossein Hajimirsadeghi, Yoshua Bengio, and Mohamed Osama Ahmed. Latent bottlenecked attentive neural processes. In *The Eleventh International Conference on Learning Representations, ICLR 2023*. PMLR (Proceedings of Machine Learning Research), 2023.
- Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation. *Journal of Machine Learning Research*, 17(205):1–37, 2016.
- Lasse Elsemüller, Hans Olischläger, Marvin Schmitt, Paul-Christian Bürkner, Ullrich Koethe, and Stefan T. Radev. Sensitivity-aware amortized bayesian inference. *Transactions on Machine Learning Research*, 2024.
- Li Deng. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.
- Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.
- Vincent Dutordoir, Alan Saul, Zoubin Ghahramani, and Fergus Simpson. Neural diffusion processes. In *International Conference on Machine Learning*, pages 8990–9012. PMLR, 2023.
- Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. Large language models to enhance Bayesian optimization. *International Conference on Learning Representations*, 2024.
- Samuel Müller, Matthias Feurer, Noah Hollmann, and Frank Hutter. Pfns4bo: In-context learning for bayesian optimization. In *International Conference on Machine Learning*, pages 25444–25470. PMLR, 2023.
- Maximilian Balandat, Brian Karrer, Daniel Jiang, Samuel Daulton, Ben Letham, Andrew G Wilson, and Eytan Bakshy. BoTorch: A framework for efficient Monte-Carlo Bayesian optimization. *Advances in Neural Information Processing Systems*, 33:21524–21538, 2020.
- George E Uhlenbeck and Leonard S Ornstein. On the theory of the Brownian motion. *Physical Review*, 36 (5):823, 1930.
- William O Kermack and Anderson G McKendrick. A contribution to the mathematical theory of epidemics. *Proceedings of the Royal Society of London. Series A, Containing papers of a mathematical and physical character*, 115(772):700–721, 1927.
- 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.
- David Greenberg, Marcel Nonnenmacher, and Jakob Macke. Automatic posterior transformation for likelihood-free inference. In *International Conference on Machine Learning*, pages 2404–2414. PMLR, 2019.
- Benjamin K Miller, Christoph Weniger, and Patrick Forré. Contrastive neural ratio estimation. *Advances in Neural Information Processing Systems*, 35:3262–3278, 2022.
- Manuel Gloeckler, Michael Deistler, Christian Weilbach, Frank Wood, and Jakob H Macke. All-in-one simulation-based inference. In *International Conference on Machine Learning*. PMLR, 2024.
- Sean Talts, Michael Betancourt, Daniel Simpson, Aki Vehtari, and Andrew Gelman. Validating bayesian inference algorithms with simulation-based calibration. *arXiv preprint arXiv:1804.06788*, 2018.
- Konstantin Avilov, Qiong Li, Lewi Stone, and Daihai He. The 1978 english boarding school influenza outbreak: Where the classic seir model fails. *SSRN 4586177*, 2023.
- Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. In *International Conference on Learning Representations*, 2019.
- Jonathan Gordon, Wessel P Bruinsma, Andrew YK Foong, James Requeima, Yann Dubois, and Richard E Turner. Convolutional conditional neural processes. In *International Conference on Learning Representations*, 2020.
- 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, 2023a.
- Emiel Hoogeboom, Alexey A. Gritsenko, Jasmin Bastings, Ben Poole, Rianne van den Berg, and Tim

- Salimans. Autoregressive diffusion models. In *International Conference on Learning Representations*, 2022.
- 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.
- Chelsea Finn, Kelvin Xu, and Sergey Levine. Probabilistic model-agnostic meta-learning. *Advances in Neural Information Processing Systems*, 31, 2018.
- Sulin Liu, Xingyuan Sun, Peter J Ramadge, and Ryan P Adams. Task-agnostic amortized inference of Gaussian process hyperparameters. In *Advances in Neural Information Processing Systems*, volume 33, pages 21440–21452, 2020.
- Fergus Simpson, Ian Davies, Vidhi Lalchand, Alessandro Vullo, Nicolas Durrande, and Carl Edward Rasmussen. Kernel identification through transformers. In *Advances in Neural Information Processing Systems*, volume 34, pages 10483–10495, 2021.
- Sarthak Mittal, Niels Leif Bracher, Guillaume Lajoie, Priyank Jaini, and Marcus A Brubaker. Exploring exchangeable dataset amortization for bayesian posterior inference. In *ICML 2023 Workshop on Structured Probabilistic Inference and Generative Modeling*, 2023.
- Sarthak Mittal, Eric Elmoznino, Leo Gagnon, Sangnie Bhardwaj, Dhanya Sridhar, and Guillaume Lajoie. Does learning the right latent variables necessarily improve in-context learning? *arXiv preprint arXiv:2405.19162*, 2024.
- Katarzyna Kobalczyk and Mihaela van der Schaar. Informed meta-learning. *arXiv preprint arXiv:2402.16105*, 2024.
- George Whittle, Juliusz Ziomek, Jacob Rawling, and Michael A Osborne. Distribution transformers: Fast approximate Bayesian inference with on-the-fly prior adaptation. *arXiv preprint arXiv:2502.02463*, 2025.
- Jan-Matthis Lueckmann, Pedro J Goncalves, Giacomo Bassetto, Kaan Öcal, Marcel Nonnenmacher, and Jakob H Macke. Flexible statistical inference for mechanistic models of neural dynamics. *Advances in Neural Information Processing Systems*, 30, 2017.
- George Papamakarios, David Sterratt, and Iain Murray. Sequential neural likelihood: Fast likelihood-free inference with autoregressive flows. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 837–848. PMLR, 2019.
- Stefan T Radev, Marvin Schmitt, Valentin Pratz, Umberto Picchini, Ullrich Köthe, and Paul-Christian Bürkner. JANA: Jointly amortized neural approximation of complex Bayesian models. In *Uncertainty in Artificial Intelligence*, pages 1695–1706. PMLR, 2023.
- Yang Song and Stefano Ermon. Generative modeling by estimating gradients of the data distribution. *Advances in Neural Information Processing Systems*, 32, 2019.
- Daolang Huang, Ayush Bharti, Amauri Souza, Luigi Acerbi, and Samuel Kaski. Learning robust statistics for simulation-based inference under model misspecification. *Advances in Neural Information Processing Systems*, 36, 2023b.
- Marvin Schmitt, Paul-Christian Bürkner, Ullrich Köthe, and Stefan T Radev. Detecting model misspecification in amortized Bayesian inference with neural networks. In *DAGM German Conference on Pattern Recognition*, pages 541–557. Springer, 2023.
- Donggyun Kim, Seongwoong Cho, Wonkwang Lee, and Seunghoon Hong. Multi-task neural processes. In *International Conference on Learning Representations*, 2022.
- Matthew Ashman, Cristiana Diaconu, Adrian Weller, and Richard E Turner. In-context in-context learning with transformer neural processes. In *Proceedings of the 6th Symposium on Advances in Approximate Bayesian Inference*, 2024.
- Ryan L Murphy, Balasubramanian Srinivasan, Vinayak Rao, and Bruno Ribeiro. Janossy pooling: Learning deep permutation-invariant functions for variable-size inputs. In *International Conference on Learning Representations*, 2019.
- James Hensman, Alexander Matthews, and Zoubin Ghahramani. Scalable variational Gaussian process classification. In *Artificial Intelligence and Statistics*, pages 351–360. PMLR, 2015.
- Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015*. PMLR (Proceedings of Machine Learning Research), 2015.
- Christian P. Robert and George Casella. *Monte Carlo Statistical Methods*. Springer Texts in Statistics. Springer, 2nd edition, 2004.
- Jan-Matthis Lueckmann, Jan Boelts, David Greenberg, Pedro Goncalves, and Jakob Macke. Benchmarking simulation-based inference. In *Proc. AISTATS*, pages 343–351. PMLR, 2021.
- Troels Pedersen. Stochastic multipath model for the in-room radio channel based on room electromagnetics. *IEEE Transactions on Antennas and Propagation*, 67(4):2591–2603, 2019.
- Ayush Bharti, Ramoni Adeogun, and Troels Pedersen. Estimator for stochastic channel model without

multipath extraction using temporal moments. In *2019 IEEE 20th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*, pages 1–5. IEEE, 2019.

Alvaro Tejero-Cantero, Jan Boelts, Michael Deistler, Jan-Matthis Lueckmann, Conor Durkan, Pedro J. Gonçalves, David S. Greenberg, and Jakob H. Macke. sbi: A toolkit for simulation-based inference. *Journal of Open Source Software*, 5(52):2505, 2020.

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. *Advances in Neural Information Processing Systems*, 30, 2017.

Teemu Säilynoja, Paul-Christian Bürkner, and Aki Vehtari. Graphical test for discrete uniformity and its applications in goodness-of-fit evaluation and multiple sample comparison. *Statistics and Computing*, 32(2):32, 2022.

Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research*, 2018.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32, 2019.

## 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] See description in [Section 3](#) and further details in [Appendix B](#) and [Appendix C](#).
- (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Not Applicable] Our architecture is based on the standard attention mechanism ([Vaswani et al., 2017](#)) which has well-known properties.
- (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [Yes] Our source code is available at <https://github.com/acerbilab/amortized-conditioning-engine/>.

2. For any theoretical claim, check if you include:

- (a) Statements of the full set of assumptions of all theoretical results. [Not Applicable]

- (b) Complete proofs of all theoretical results. [Not Applicable]
  - (c) Clear explanations of any assumptions. [Not Applicable]
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 linked GitHub repo contains working examples demonstrating our main results, in addition to the full codebase.
  - (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes] In [Appendix C](#) we provide all experimental details.
  - (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] See [Section 4.1](#), [Section 4.2](#), [Section 4.3](#) and [Appendix C](#).
  - (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes] See [Appendix C.5](#).
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] See [Appendix C.5](#).
  - (b) The license information of the assets, if applicable. [Yes] See [Appendix C.5](#).
  - (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]

## Supplementary Material

---

### Contents

<b>A TABLE OF ACRONYMS</b>	<b>15</b>
<b>B METHODS</b>	<b>15</b>
B.1 Details and experiments with prior injection . . . . .	16
B.2 Architecture . . . . .	20
B.3 Training batch construction . . . . .	21
B.4 Autoregressive predictions . . . . .	21
<b>C EXPERIMENTAL DETAILS</b>	<b>21</b>
C.1 Gaussian process (GP) experiments . . . . .	21
C.2 Image completion and classification . . . . .	22
C.3 Bayesian optimization . . . . .	24
C.4 Simulation-based inference . . . . .	33
C.5 Computational resources and software . . . . .	39

## A TABLE OF ACRONYMS

For ease of reference, [Table S1](#) reports a list of key acronyms and abbreviations used in the paper.

Acronym	Full Name	Description
<b>Architectures</b>		
TPM-D	Transformer Prediction Map – Diagonal	Family of transformer architectures for diagonal prediction maps, including all architectures below
ACE	Amortized Conditioning Engine	Our transformer-based meta-learning model for probabilistic tasks with explicit latent variables
ACEP	Amortized Conditioning Engine (with Priors)	ACE variant allowing runtime injection of priors over latent variables
CNP	Conditional Neural Process	Context-to-target mapping with permutation invariance
PFN	Prior-Fitted Network	Meta-learning approach using transformers for inference and introducing Riemannian output distributions
TNP-D	Transformer Neural Process – Diagonal	Transformer neural process variant with independent target predictions
<b>Bayesian Optimization (BO) Terms</b>		
BO	Bayesian Optimization	Black-box function optimization using surrogate models
MES	Max-Value Entropy Search	Acquisition function based on uncertainty over optimum value
TS	Thompson Sampling	Optimization via sampling from the posterior over optimum location
$\pi$ BO	Prior-information BO	BO incorporating prior knowledge on optimum location
AR-TNP-D-TS	Autoregressive TNP-D Thompson Sampling	TNP extension with autoregressive sampling for BO
<b>Simulation-Based Inference (SBI) Terms</b>		
SBI	Simulation-Based Inference	Parameter posterior inference using synthetic data
NPE	Neural Posterior Estimation	Direct posterior modeling with neural networks
NRE	Neural Ratio Estimation	Likelihood-ratio-based posterior inference
OUP	Ornstein–Uhlenbeck Process	Mean-reverting stochastic process
SIR	Susceptible–Infectious–Recovered	Epidemiological disease spread model

Table S1: Key acronyms used in the paper, grouped by category.

## B METHODS

This section details several technical aspects of our paper, such as the prior amortization techniques, neural network architecture and general training and inference details.

## B.1 Details and experiments with prior injection

**Prior generative process.** To expose ACE to a wide array of distinct priors during training, we generate priors following a hierarchical approach that generates smooth priors over a bounded range. The process is as follows, separately for each latent variable  $\theta_l$ , for  $1 \leq l \leq L$ :

- We first sample the type of priors for the latent variable. Specifically, with 80% probability, we sample from a mixture of Gaussians to generate a smooth prior, otherwise, we create a flat prior with uniform distribution.
- If we sample from a mixture of Gaussians:
  - We first sample the number of Gaussian components  $K$  from a geometric distribution with  $q = 0.5$ :

$$K \sim \text{Geometric}(0.5). \quad (\text{S1})$$

- If  $K > 1$ , we randomly choose among three configurations with equal probability:
  1. Same means and different standard deviations.
  2. Different means and same standard deviations.
  3. All different means and standard deviations.
- Given the predefined global priors for mean and standard deviation (uniform distributions whose ranges are determined by the range of the corresponding latent variable), we sample the means and standard deviations for each component from the predefined uniform distributions.
- The weights of the mixture components are sampled from a Dirichlet distribution:

$$\mathbf{w} \sim \text{Dirichlet}(\alpha_0 = 1). \quad (\text{S2})$$

- Finally, we convert the mixture of Gaussians into a normalized histogram over a grid  $\mathcal{G}$  with  $N_{\text{bins}}$  uniformly-spaced bins. For each bin  $b$ , we compute the probability mass  $\mathbf{p}_l^{(b)}$  by calculating the difference between the cumulative distribution function values at the bin edges. This is done for each Gaussian component and then summed up, weighted by the mixture weights.
- We normalize the bin probabilities to ensure a valid probability distribution:

$$\mathbf{p}_l = \frac{\mathbf{p}_l}{\sum_{b=1}^{N_{\text{bins}}} \mathbf{p}_l^{(b)}}. \quad (\text{S3})$$

- If we sample from a uniform distribution:
  - We assign equal probability to each bin over the grid:

$$\mathbf{p}_l = \frac{1}{N_{\text{bins}}} \mathbf{1}_{N_{\text{bins}}}, \quad (\text{S4})$$

where  $\mathbf{1}_{N_{\text{bins}}}$  is a vector of ones of length  $N_{\text{bins}}$ .

For all experiments, we select  $N_{\text{bins}} = 100$  as the number of bins for the prior grid. See Fig. S1 for some examples of sampled priors.

**Investigation of prior injection with a Gaussian toy model.** To investigate the effect of the injected prior, we test our method with a simple 1D Gaussian model with two latent variables: mean  $\mu$  and standard deviation  $\sigma$ . The data is the samples drawn from this distribution,  $\mathcal{D}_N = \{y_n\}_{n=1}^N \sim \mathcal{N}(\mu, \sigma^2)$ . We can numerically compute the exact Bayesian posterior on the predefined grid given the data and any prior, and subsequently compare the ground-truth posterior with the ACE’s predicted posterior after injecting the same prior.

We first sample random priors using the generative process described above. Then we sample  $\mu$  and  $\sigma$  from the priors and generate the corresponding data  $\mathcal{D}_N$ . We pass the data along with the prior to ACE to get the predictive distributions  $p(\mu|\mathcal{D}_N)$  and  $p(\sigma|\mathcal{D}_N)$  as well as the autoregressive predictions  $p(\mu|\sigma, \mathcal{D}_N)$  and  $p(\sigma|\mu, \mathcal{D}_N)$ . With these equations, we can autoregressively compute our model’s prediction for  $p(\mu, \sigma|\mathcal{D}_N)$  on the grid. The true posterior is calculated numerically via Bayes rule on the grid. See Fig. S2 for several examples.

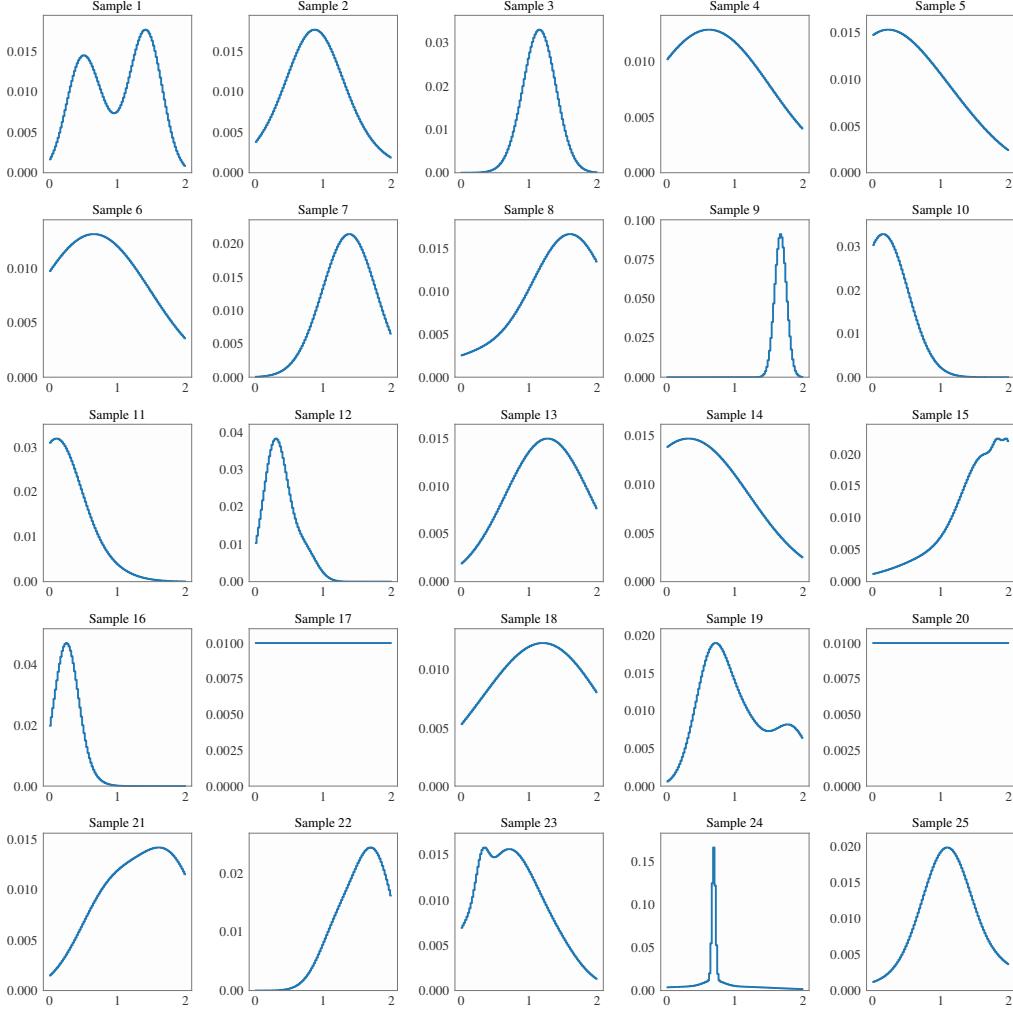


Figure S1: Examples of randomly sampled priors over the range  $[0, 2]$ . Samples include mixtures of Gaussians and Uniform distributions.

To quantitatively assess the quality of our model's predicted posteriors, we compare the posterior mean and standard deviation (i.e., the first two moments<sup>3</sup>) for  $\mu$  and  $\sigma$  of predicted vs. true posteriors, visualized in Fig. S3. The scatter points are aligned along the diagonal line, indicating that the moments of the predicted posterior closely match the moments of true posterior. These results show that ACE is effectively incorporating the information provided by the prior and adjusts the final posterior accordingly. In Appendix C.4.4 we perform a more extensive analysis of posterior calibration in ACE with a complex simulator model.

<sup>3</sup>We prefer standard deviation to variance as it has the same units as the quantity of interest, as opposed to squared units which are less interpretable.

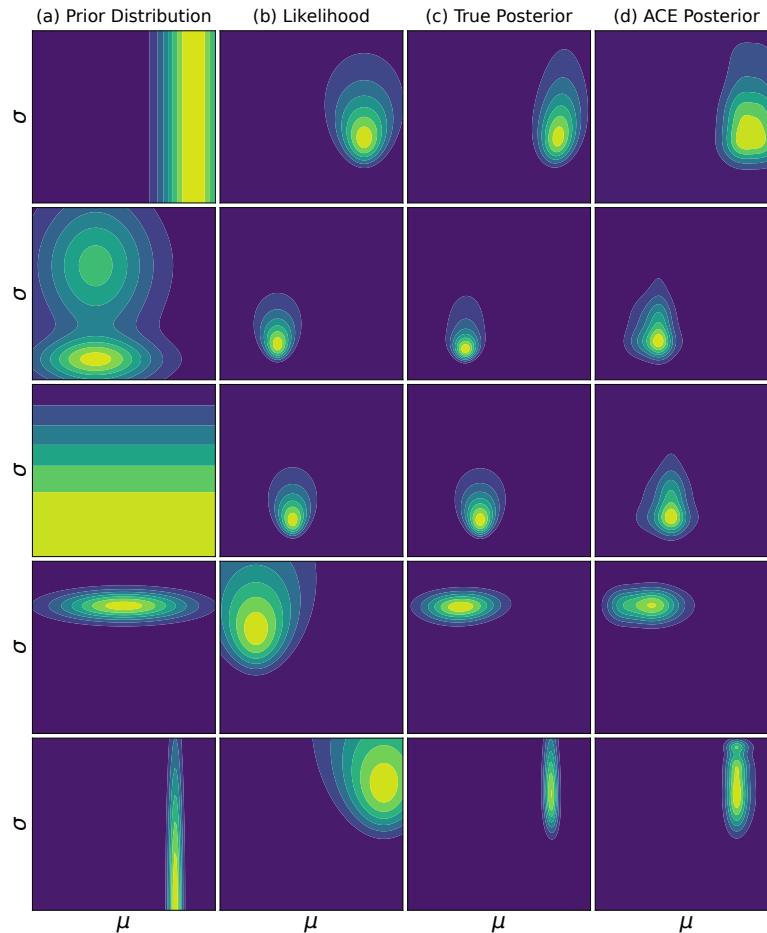


Figure S2: Examples of the true and predicted posterior distributions in the toy 1D Gaussian case. (a) Prior distribution over  $\theta = (\mu, \sigma)$  set at runtime. (b) Likelihood for the observed data (the data themselves are not shown). (c) Ground-truth Bayesian posterior. (d) ACE's predicted posterior, based on the user-set prior and observed data, approximates well the true posterior.

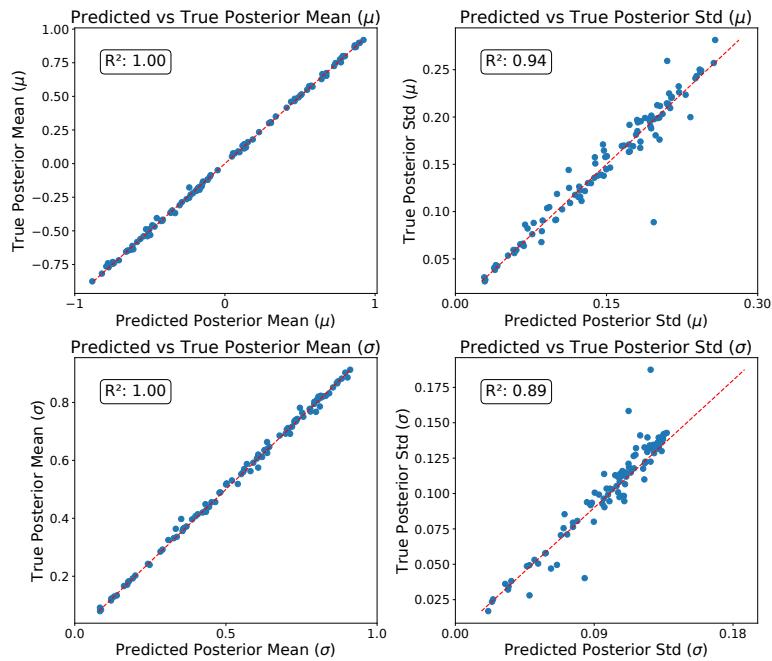


Figure S3: The scatter plots compare the predicted and true posterior mean and standard deviation values for both  $\mu$  and  $\sigma$  across 100 examples. We can see that the points lie closely along the diagonal red dashed line, indicating that the moments (mean and standard deviation) of the predicted posteriors closely match the true posteriors.

## B.2 Architecture

Here we give an overview of two key architectures used in our paper. First, we show the TNP-D (Nguyen and Grover, 2022) architecture in Fig. S4, which our method extends. Fig. S5 shows the ACE architecture introduced in this paper.

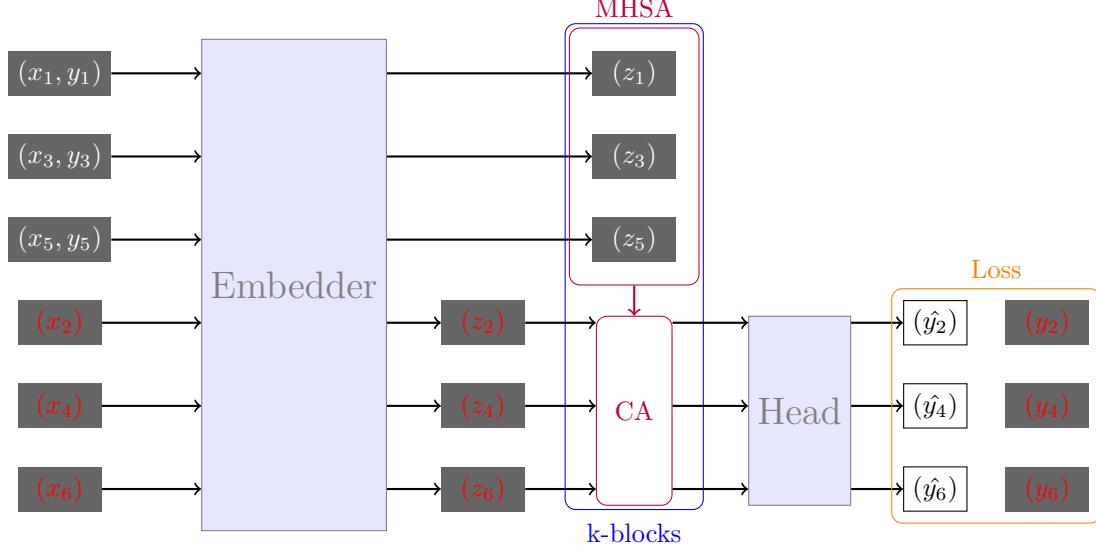


Figure S4: **A conceptual figure of TNP-D architecture.** The TNP-D architecture can be summarized in the embedding layer, attention layers and output head. The  $x$  denotes locations where the output is unknown (target inputs). The  $z$  is the embedded data, while MHSA stands for multi head cross attention and CA for cross attention. The head for TNP-D is Gaussian, so it outputs a mean and variance for each target point.

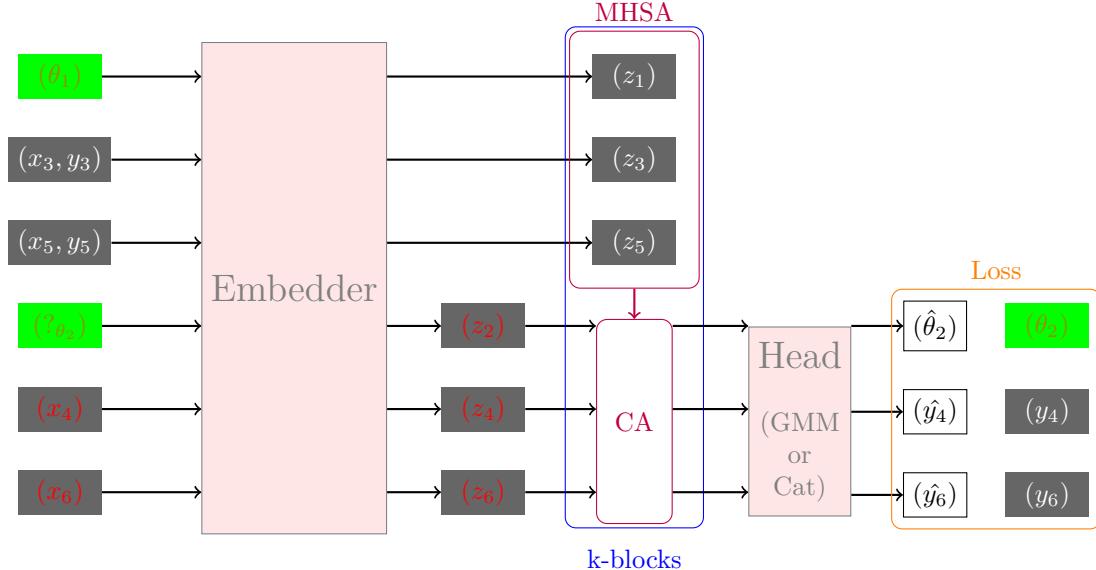


Figure S5: **A conceptual figure of ACE architecture.** The diagram shows key differences between ACE and TNP-D. The differences boil down to the embedder layer that now incorporates latents  $\theta_l$  (and possibly priors over these) and the output head that is now a Gaussian mixture model (GMM, for continuous variables) or categorical (Cat, for discrete variables). Both latent and data can be of either type.

### B.3 Training batch construction

ACE can condition on and predict data, latent variables, and combinations of both. Here, we outline the sampling process used to construct the training batch.

- First, we generate our dataset by following the steps outlined for the respective cases (GP, Image Completion, BO, SBI); see [Appendix C](#). For example, in the GP emulation case, we draw  $n_{\text{data}}$  points from a function sampled from a GP along with its respective latent variables.
- Next, we sample the number of context points,  $n_{\text{context}}$ , uniformly between the minimum and maximum context points,  $\text{min\_ctx}$  and  $\text{max\_ctx}$ . We then split our data based on this  $n_{\text{context}}$  value; the remaining data points that are not in the context set are allocated to the target set.
- We then determine whether the context includes any latent variables at all with a 50% probability. If latent variables are to be included in the context set, we sample the number of latents residing in the context set, uniformly from 1 to  $n_{\text{latents}}$ . All latent variables not in the context set are assigned to the target set.

The above steps are applied for each element (dataset) in the training batch. In the implementation, we also ensure that, within each batch, the number of context points remains consistent across all elements, as does the number of target points, to facilitate batch training for our model. However, the number of latents in the context set may vary for each element, introducing variability that improves the model’s training process.

### B.4 Autoregressive predictions

While ACE predicts conditional marginals independently, we can still obtain *joint* predictions over both data and latents autoregressively ([Nguyen and Grover, 2022](#); [Bruinsma et al., 2023](#)). Suppose we want to predict the joint target distribution  $p(\mathbf{z}_{1:M}^* | \boldsymbol{\xi}_{1:M}^*, \mathfrak{D}_N)$ , where we use compact indexing notation. We can write:

$$p(\mathbf{z}_{1:M}^* | \boldsymbol{\xi}_{1:M}^*, \mathfrak{D}_N) = \prod_{m=1}^M p(z_m^* | \mathbf{z}_{1:m-1}^*, \boldsymbol{\xi}_{1:m}^*, \mathfrak{D}_N) = \mathbb{E}_{\boldsymbol{\pi}} \left[ \prod_{m=1}^M p(z_{\pi_m}^* | \mathbf{z}_{\pi_1:\pi_{m-1}}^*, \boldsymbol{\xi}_{\pi_1:\pi_m}^*, \mathfrak{D}_N) \right], \quad (\text{S5})$$

where  $\boldsymbol{\pi}$  is a permutation of  $(1, \dots, M)$ , i.e. an element of the symmetric group  $\mathcal{S}_M$ . The first passage follows from the standard rules of probability and the second passage follows from permutation invariance of the joint distribution with respect to the ordering of the variables  $\boldsymbol{\xi}_{1:M}$ . The last expression can be used to enforce permutation invariance and validity of our joint predictions even if sequential predictions of the model are not natively invariant ([Murphy et al., 2019](#)). In practice, for moderate to large  $M$  ( $M \gtrsim 4$ ) we approximate the expectation over permutations via Monte Carlo sampling.

## C EXPERIMENTAL DETAILS

In this section, we show additional experiments to validate our method and provide additional details about sampling, training, and model architecture.

### C.1 Gaussian process (GP) experiments

We now demonstrate the use of ACE for performing amortized inference tasks in the Gaussian processes (GP) model class. GPs are a Bayesian non-parametric method used as priors over functions (see [Rasmussen and Williams, 2006](#)). To perform inference with a GP, one must first define a kernel function  $\kappa_{\theta}$  parameterized by hyperparameters  $\theta$  such as lengthscales and output scale. As a flexible model of distributions over functions used for regression and classification, GPs are a go-to generative model for meta-learning and feature heavily in the (conditional) neural process literature (CNP; [Garnelo et al., 2018b](#)). ACE can handle many of the challenges faced when applying GPs. Firstly, it can accurately amortize the GP predictive distribution as is usually shown in the CNP literature. In addition, ACE can perform other crucial tasks in the GP workflow, such as amortized learning of  $\theta$ , usually found through optimization of the marginal likelihood (Gaussian likelihood) or via approximate inference for non-Gaussian likelihoods (e.g., [Hensman et al. 2015](#)). Furthermore, we can also do kernel selection by treating the kernel as a latent discrete variable, and incorporate prior knowledge about  $\theta$ .

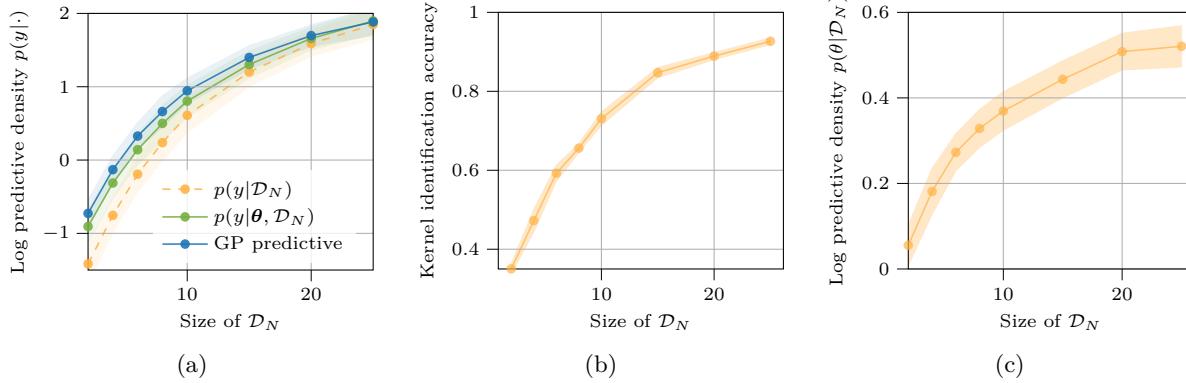


Figure S6: (a) Conditioning on the latent variable  $\theta$  (kernel hyperparameters and type) improves predictive performance, approaching the GP upper bound for the log predictive density. (b) ACE can identify the kernel  $\kappa$ . (c) ACE can learn kernel hyperparameters.

**Results.** The main results from our GP regression experiments are displayed in Fig. S6. We trained ACE on samples from four kernels, the RBF and Matérn-( $1/2, 3/2, 5/2$ ), using the architecture described in Section 3; see below for details. In Fig. S6a, we show ACE’s ability to condition on provided information: data only, or data and  $\theta$  (kernel hyperparameters and type). As expected, there is an improvement when conditioning on more information, specifically when the context set  $\mathcal{D}_N$  is small. As an upper bound, we show the ground-truth GP predictive performance. The method can accurately predict the kernel, *i.e.* model selection (Fig. S6b), while at the same time learn the hyperparameters (Fig. S6c), both improving as a function of the context set size.

**Sampling from a GP.** Both the GP experiments and the Bayesian optimization experiments reported in Section 4.2 and further detailed in Appendix C.3 use a similar sampling process to generate data.

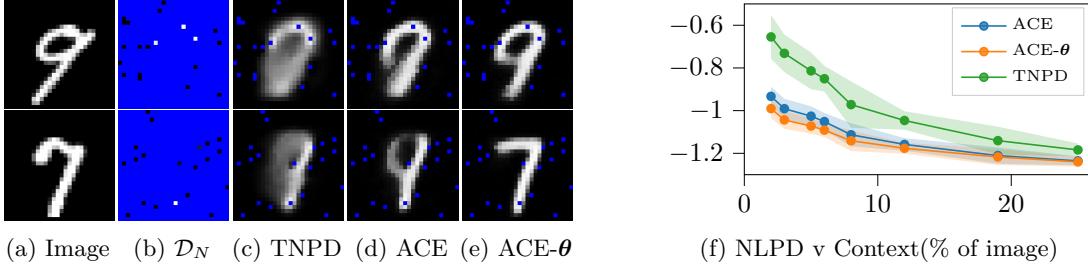
- We first sample the latents. These are kernel hyperparameters, the output scale  $\sigma_f$  and lengthscale  $\ell$ . Each input dimension of  $\mathbf{x}$  is assigned its own lengthscale  $\ell = (\ell^{(1)}, \ell^{(2)}, \dots)$  and a corresponding kernel. For all GP examples the RBF and three Matérn-( $1/2, 3/2, 5/2$ ) kernels were used with equal weights. The kernel output scale  $\sigma_f \sim U(0.1, 1)$  and each  $k$ -th lengthscale is  $\ell^{(k)} \sim \mathcal{N}(1/3, 0.75)$  truncated between [0.05, 2].
- Once all latent information is defined, we draw from a GP prior from a range  $[-1, 1]$  for each input dimension. The realizations from the prior form our context data  $\mathcal{D}_N$  where the size of the context set  $N$  is drawn from a discrete uniform distribution 3, 4, ..., 50. The target data  $(\mathbf{X}^*, \mathbf{y}^*)$  of size  $200 - N$  is then drawn from the predictive posterior of the GP conditioned on  $\mathcal{D}_N$ .

**Architecture.** The ACE model used in the GP experiments had embedding dimension 256 and 6 transformer layers. The attention blocks had 6 heads and the MLP block had hidden dimension 128. The output head had  $K = 20$  MLP components with hidden dimension 256. The model was trained for  $5 \times 10^4$  steps with batch size 32, using learning rate  $1 \times 10^{-4}$  with cosine annealing. Following Nguyen and Grover (2022), and unlike the original transformer implementation (Vaswani et al., 2017), we do not use dropout in any of our experiments.

## C.2 Image completion and classification

In this section, we detail the image experiments in Section 4.1 as well as report additional experiments. Image completion experiments have long been a benchmark in the neural process literature treating them as regression problems (Garnelo et al., 2018a; Kim et al., 2019). We use two standard datasets, MNIST (Deng, 2012) and CELEBA (Liu et al., 2015). The MNIST results presented are with the full image size  $28 \times 28$ , while CELEBA results were downsized to  $32 \times 32$ . However, as shown in Fig. S11, ACE can also handle the full image size. All image datasets were normalised based on the complete dataset average and standard deviation. The data input  $\mathbf{x}$  for image experiments is the 2-D image pixel-coordinates and the data value  $y$  for MNIST is one output dimension, while CELEBA uses all three RGB channels and thus is a multi-output  $\mathbf{y}$ .

The experiments on images demonstrate the versatility of the ACE method and its advantages over conventional



**Figure S7: Image regression (MNIST).** Image (a) serves as the reference for the problem, while (b) is the context where 10% of the pixels are observed. Figures (c) - (e) are the respective model predictions, while (f) shows performance over varying context (mean and 95% confidence interval). In (e) the model is also conditioned on the class label, showing a clear improvement in performance.

CNPs. We outperform the current state-of-the-art TNP-D on the standard image completion task (Fig. 3). Given a random sample from the image space as context  $\mathcal{D}_N$ , the model predicts the remaining  $M$  image pixel values at  $\mathbf{x}^*$ . The total number of points  $N + M$  for MNIST is thus 784 points and 1024 for CELEBA where the split is randomly sampled (see below for details). The model is then trained as detailed in Section 3.3. Thus, the final trained model can perform image completion, also sometimes known as in-painting.

In addition to image completion, our method can condition on and predict latent variables  $\theta$ . For MNIST, we use the class labels as latents, so 0, 1, 2, ..., which were encoded into a single discrete variable. Meanwhile, for CELEBA we use as latents the 40 binary features that accompany the dataset, e.g. BROWNHAIR, MAN, SMILING, trained with the sampling procedure discussed below. We recall that in ACE the latents  $\theta$  can be both conditioned on and predicted. Thus, we can do conditional generation based on the class or features or, given a sample of an image, predict its class or features, as initially promised in Fig. 1a.

**Results.** The main image completion results for the CELEBA dataset are shown in Fig. 3, with the same experiment performed on MNIST and displayed in Fig. S7. In both figures, we display some example images and predictions and negative log-probability density for different context sizes (shaded area is 95% confidence interval). Our method demonstrates a clear improvement over the TNP-D method across all context sizes on both datasets (Fig. 3 and Fig. S7). Moreover, incorporating latent information for conditional generation further enhances the performance of our base method. A variation of the image completion experiment is shown in Fig. S9, where the context is no longer randomly sampled from within the image but instead selected according

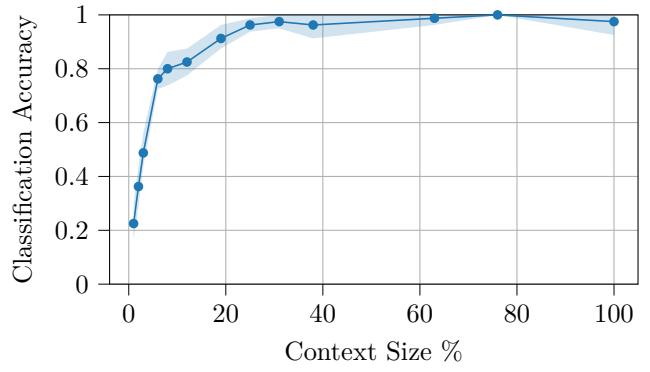


Figure S8: Classification accuracy for MNIST for varying context size.

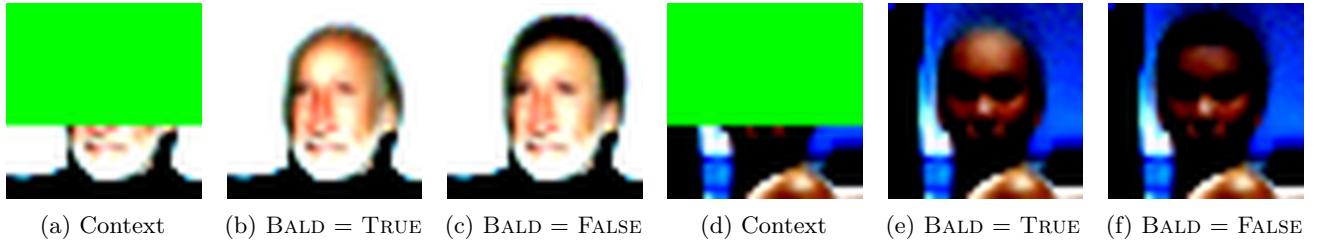


Figure S9: Example of ACE conditioning on the value of the BALD feature when the image is masked for the first 22 rows. (a) and (d) show the context points used for prediction, where (b) and (e) show predictions where the BALD feature is conditioned on TRUE. Meanwhile, (c) and (f) are conditioned on FALSE.

to a top 22-row image mask. For this example, the latent information BALD is either conditioned on TRUE or FALSE. The results show that the model adjusts its generated output based on the provided latent information, highlighting the potential of conditional generation. Furthermore, in Fig. S10, we show examples of ACE’s ability to perform image classification showing a subset of the 40 features in CELEBA dataset. Despite only having 10% of the image available, ACE can predict most features successfully. Finally, in Fig. S8 the accuracy for predicting the correct class label for MNIST is reported.

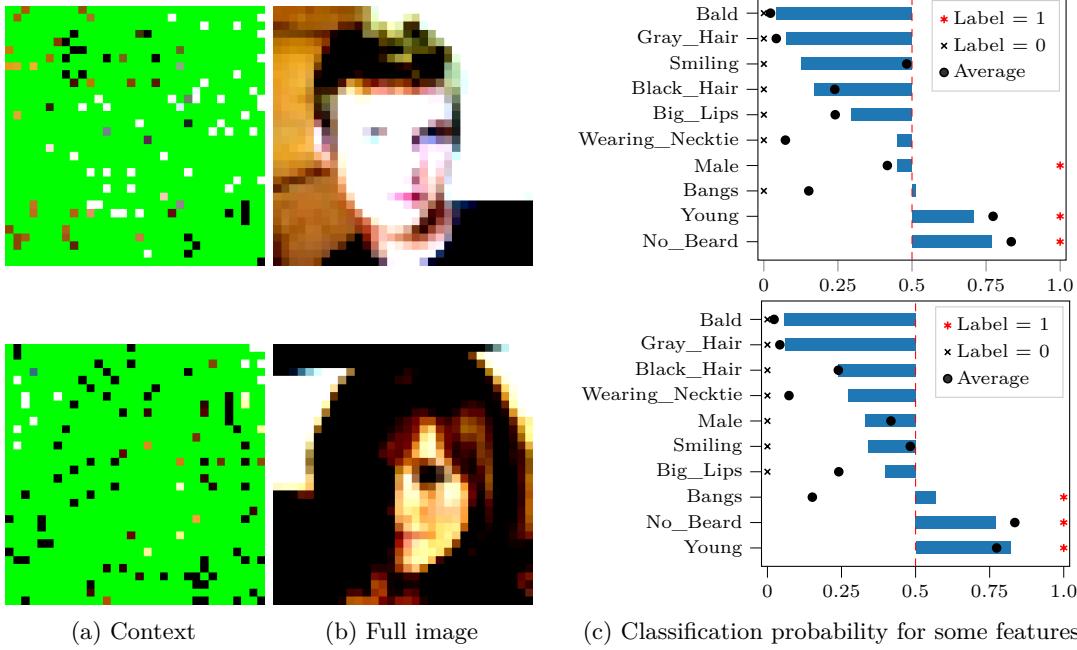


Figure S10: An example showing the classification ability of ACE. (a) is the context available of the full image displayed in the panel (b). The probabilities for a subset of features are in (c).

**Sampling for Image experiments.** For sampling, we use the full available dataset for both MNIST and CELEBA, detailed in Appendix B.3. In the MNIST dataset there is one latent class label and for CELEBA all 40 features were used. In Fig. S9, the sampling procedure was adjusted to represent features that would influence the top 22 rows of the images. Therefore, we selected a subset of seven features, which were BALD, BANGS, BLACK\_HAIR, BLOND\_HAIR, BROWN\_HAIR, GRAY\_HAIR and EYEGLASSES. The same sampling procedure was performed again but, now on a smaller set of features.

**Architecture and training.** For the image experiments, we used the same embedder layer as in the other experiments. Through grid search, we found that a transformer architecture with 8 heads, 6 layers, and a hidden dimension of 128 performed best. For the MLP layer, we used a dimension of 256. Finally, we reduced the number of components for the output head to  $K = 3$ . We trained the model for 80,000 iterations using a cosine scheduler with ADAM (Kingma and Ba, 2015), with a learning rate 0.00005 and a batch size of 64.

### C.3 Bayesian optimization

This section presents ACE’s Bayesian Optimization (BO) experiments (Section 4.2 in the main paper) in more detail, including the training data generation, algorithms, benchmark functions, and baselines used in this paper.

#### C.3.1 Bayesian Optimization dataset, architecture and training details

**Dataset.** The BO datasets are generated by sampling from a GP, following the approach described in Appendix C.1. The sampling procedure is adjusted to include the known optimum location and value of the function within the generative process. The detailed dataset generation procedure is outlined as follows:

1. Sampling GP hyper-parameters, to determine the base function shape:

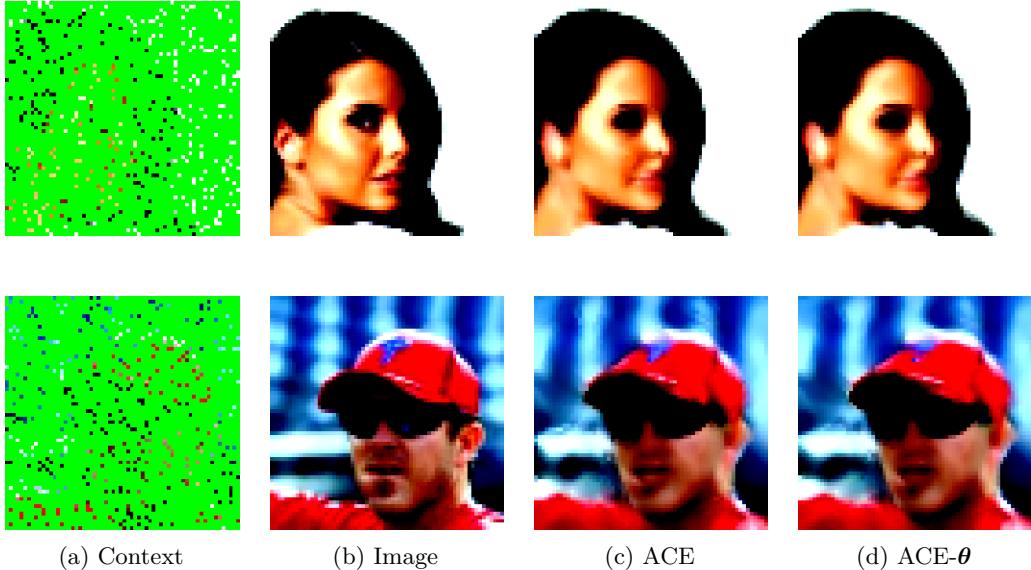


Figure S11: Examples of ACE on 64x64 image size.

- First, we randomly select a kernel from a set comprising the RBF kernel and three Matérn kernels (Matérn-1/2, Matérn-3/2, and Matérn-5/2) based on predefined weights [0.35, 0.1, 0.2, 0.35], corresponding respectively to the RBF kernel and the Matérn kernels in the specified order.
  - Then, we sample whether the kernel is isotropic or not with  $p = 0.5$ .
  - The output scale  $\sigma_f$  and lengthscales  $l^{(k)}$  are sampled following the procedure outlined in [Appendix C.1](#).
  - We assume the GP (constant) mean to be 0 for now.

2. Sampling the latent values,  $\mathbf{x}_{\text{opt}}$  and  $y_{\text{opt}}$ :

  - We sample the optimum location  $\mathbf{x}_{\text{opt}}$  uniformly inside  $[-1, 1]^D$ .
  - We sample the value of the global minimum  $y_{\text{opt}}$  from a minimum-value distribution of a zero-mean Gaussian with variance equal to the output variance of the GP. The number of samples for the minimum-value distribution,  $N$ , is approximated as the number of uncorrelated samples from the GP in the hypercube, determined based on the GP's length scale. This approach ensures that  $y_{\text{opt}}$  roughly respects the statistics of optima for the GP hyperparameters.
  - With  $p = 0.1$  probability, we add  $\Delta y \sim \exp(1)$  to the mean function to represent an “unexpectedly low” optimum.

3. Sampling from GP posterior to get the context and target sets:

  - We build a posterior GP with the above specification and a single observation at  $(\mathbf{x}_{\text{opt}}, y_{\text{opt}})$ .
  - We sample a total of  $100 \cdot D$  (context + target) locations where the number of context points is sampled similarly to the GP dataset generation. The maximum number of context points is 50 for the 1D case and 100 for both 2D and 3D cases.
  - Then, the values of this context set are jointly sampled from a GP posterior conditioned on one observation at  $(\mathbf{x}_{\text{opt}}, y_{\text{opt}})$ .
  - Instead, the target points are sampled *independently* from a GP posterior conditioned on  $\mathcal{D}_N$  (the previously sampled context points) and  $(\mathbf{x}_{\text{opt}}, y_{\text{opt}})$ . Independent sampling of the targets speeds up GP data generation and is valid since *during training* we only predict 1D marginal distributions at the target points.

4. Further adjustment of  $y$ , and consequently  $y_{\text{opt}}$ :

  - To ensure that the global optimum is at  $(\mathbf{x}_{\text{opt}}, y_{\text{opt}})$  we add a convex envelope (a quadratic component)

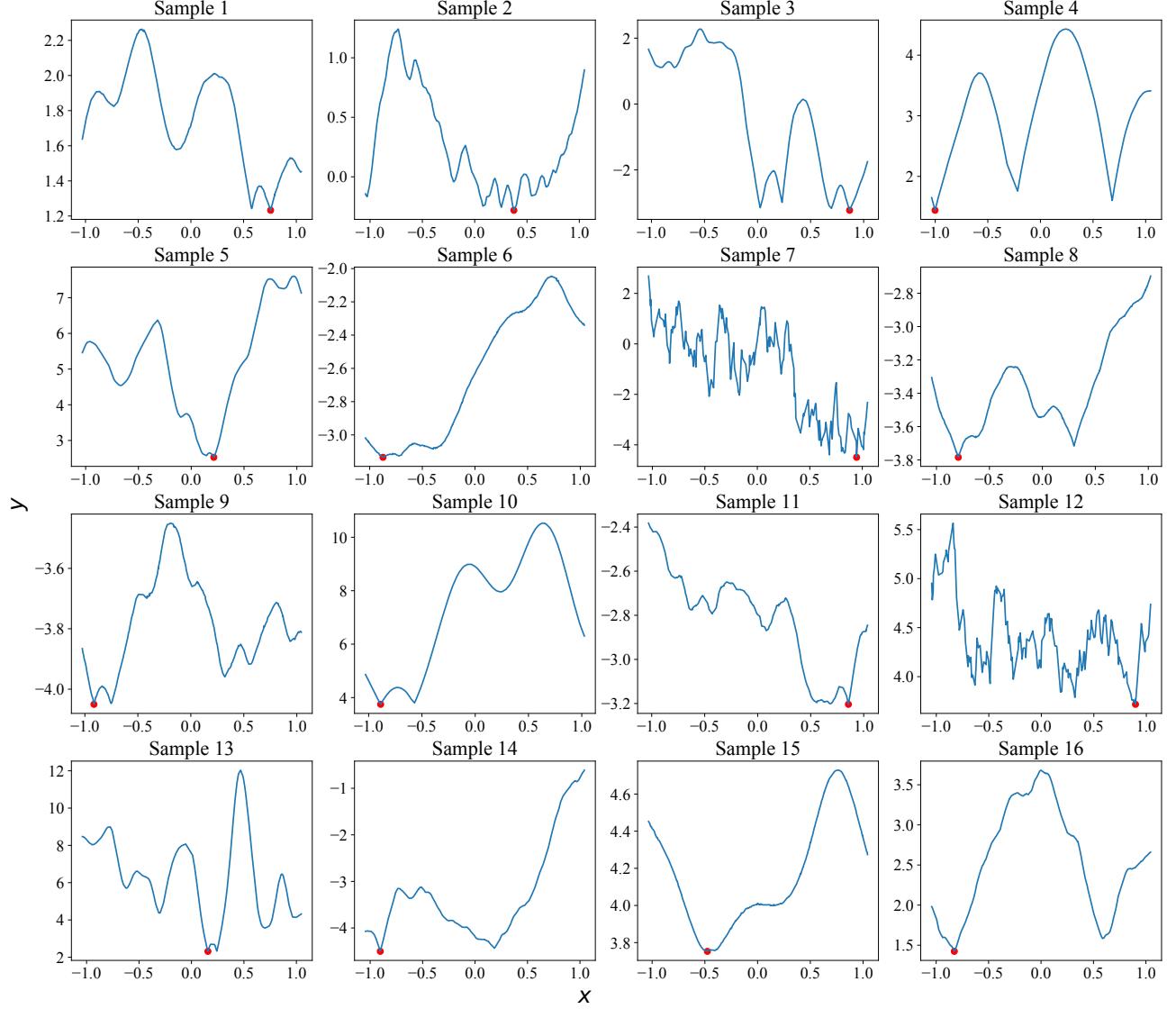


Figure S12: One-dimensional Bayesian optimization dataset samples, with their global optimum (red dot).

- Lastly, we add an offset to the  $y'$  values of sampled points uniformly drawn from  $[-5, 5]$ , meaning that  $y_{\text{opt}} \in [-5, 5]$ .

One and two-dimensional examples of the sampled functions are illustrated in Fig. S12 and Fig. S13, respectively.

**Architecture and training details.** In the Bayesian Optimization (BO) experiments, the ACE model was configured differently depending on the dimensionality of the problem. For the 1-3D cases, the model used an embedding dimension of  $D_{\text{emb}} = 256$  with six transformer layers. Each attention block had 16 heads, while the MLP block had a hidden dimension of 128. The output head consisted of  $K = 20$  MLP components, each with a hidden dimension of 128. For the 4-6D cases, the model was configured with embedding dimension of  $D_{\text{emb}} = 128$  while still using six transformer layers. Each attention block had 8 heads, and the MLP block had a hidden dimension of 512. The output head structure remained unchanged, consisting of  $K = 20$  MLP components, each with a hidden dimension of 128. The model configuration varied with problem dimensionality to balance capacity and efficiency.

The model was trained for  $5 \times 10^5$  steps with a batch size of 64 for 1-3D cases and  $3.5 \times 10^5$  steps and 128 batch size for 4-6D cases, using learning rate  $5 \times 10^{-4}$  with cosine annealing. We apply loss weighing to give more

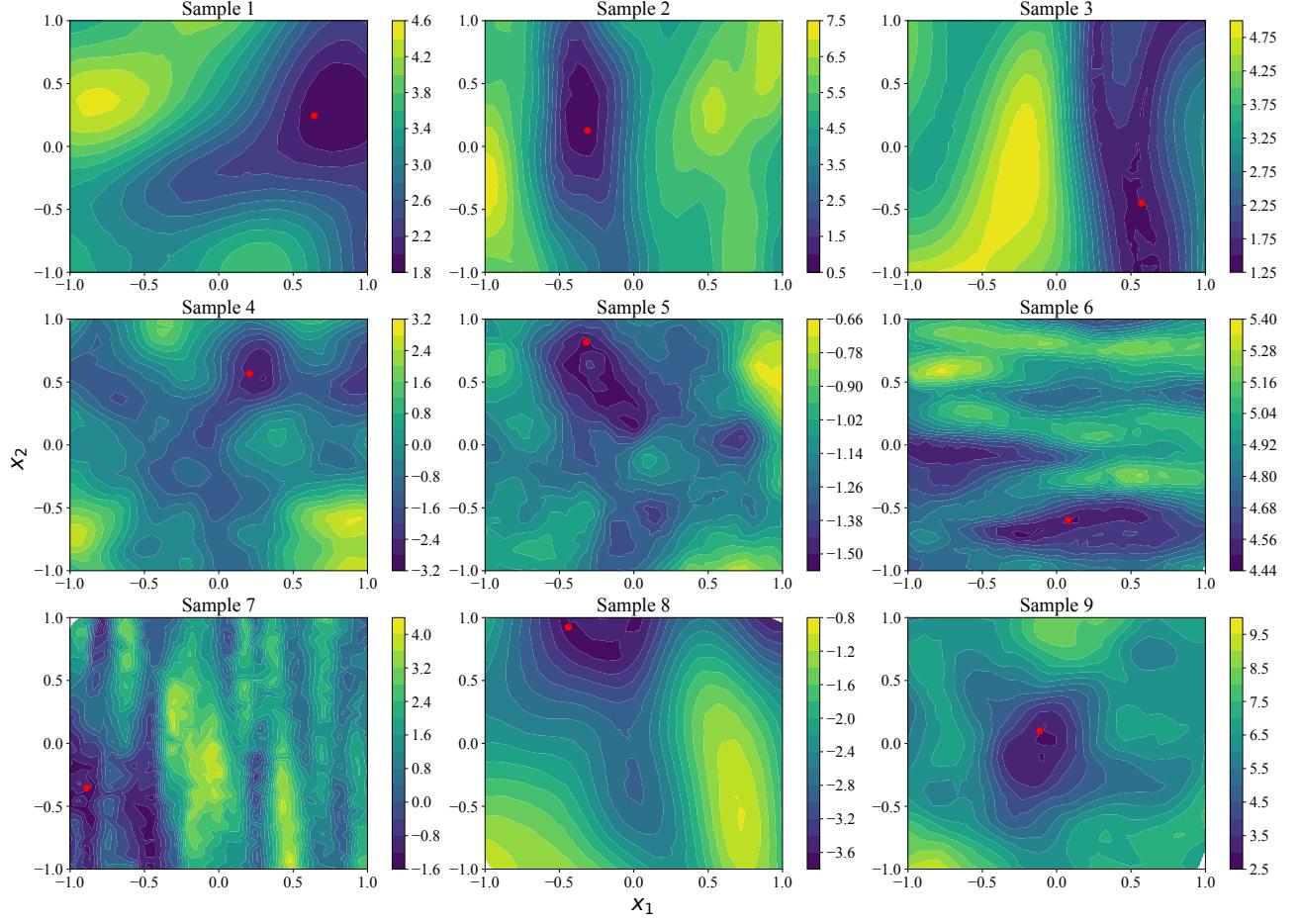


Figure S13: Two-dimensional Bayesian optimization dataset samples, with their optimum (red dot).

importance to the latent variables during training. This adjustment accounts for the fact that the number of latent variables,  $n_{\text{latent}}$ , is generally much smaller than the number of data points,  $(n_{\text{total}} - n_{\text{latent}})$ . The weight assigned to the latent loss is calculated as  $w_{\text{latent}} = (n_{\text{total}} - 1/2(\max_{\text{ctx}} + \min_{\text{ctx}})/n_{\text{latent}})^T$  where  $T$  is a tunable parameter,  $\max_{\text{ctx}}$  and  $\min_{\text{ctx}}$  are the maximum and minimum number of context points during the dataset generation. We conducted a grid search over  $T = 1, 2/3, 1/3, 0$  to identify the best-performing model. In our experiments, the optimal  $T$  values are  $T = 1$  for 1D,  $T = 2/3$  for 2D and 3D, and  $T = 0$  for 4D–6D. Note that ACE has different models trained with different datasets for each input dimensionality.

### C.3.2 ACE-BO Algorithm

**Bayesian optimization with Thompson sampling (ACE-TS).** For Thompson sampling, we sample the query point at each step from  $p(\mathbf{x}_{\text{opt}}|\mathcal{D}_N, y_{\text{opt}} < \tau)$  where  $\tau$  is a threshold lower than the minimum point seen so far. This encourages exploration to sample a new point that is lower than the current optimum. We set  $\tau = y_{\min} - \alpha \max(1, y_{\max} - y_{\min})$ , where  $y_{\max}$  and  $y_{\min}$  are the maximum and minimum values currently observed so far, and  $\alpha$  a parameter controlling the minimum improvement. We set  $\alpha = 0.01$  throughout all experiments. First, we sample  $y_{\text{opt}}$  from a truncated mixture of Gaussian obtained from ACE's predictive distribution  $p(y_{\text{opt}}|\mathcal{D}_N)$ , truncated for  $y_{\text{opt}} < \tau$ . After that, we sample  $\mathbf{x}_{\text{opt}}$  conditioned on that sampled  $y_{\text{opt}}$  (i.e., sample from  $p(\mathbf{x}_{\text{opt}}|\mathcal{D}_N, y_{\text{opt}} < \tau)$ ). For higher dimension ( $D > 1$ ) we sample  $\mathbf{x}_{\text{opt}}$  in an autoregressive manner, one dimension at a time. The order of the dimensions is randomly permuted to mitigate order bias among the dimensions. The detailed pseudocode for ACE-TS ( $D > 1$ ) is presented in Algorithm [Algorithm 1](#). An example evolution of ACE-TS is reported in [Fig. S14](#).

---

**Algorithm 1** ACE-Thompson Sample (D>1)

**Input:** observed data points  $\mathcal{D}_N = \{\mathbf{x}_{1:N}, y_{1:N}\}$ , improvement parameter  $\alpha$ , input dimensionality  $D \in \mathbb{N}^+$ , whether to condition on  $y_{\text{opt}}$  or not flag  $c \in \{\text{True}, \text{False}\}$ .

- 1: **Initialization**  $y_{\min} \leftarrow \min y_{1:N}$ ,  $y_{\max} \leftarrow \max y_{1:N}$ .
- 2: **if**  $c$  is True **then**
- 3:     set threshold value:  $\tau \leftarrow y_{\min} - \alpha \max(1, y_{\max} - y_{\min})$ .
- 4:     sample  $y_{\text{opt}}$  from mixture truncated at  $\tau$ :  $y_{\text{opt}} \sim p(y_{\text{opt}} | \mathcal{D}_N, y_{\text{opt}} < \tau)$ .
- 5: **end if**
- 6: randomly permute dimension indices:  $(1, \dots, D) \rightarrow (\pi_1, \dots, \pi_D)$ .  $\triangleright \boldsymbol{\pi}$  is permutation of  $(1, \dots, D)$
- 7: **for**  $i \leftarrow \pi_1, \dots, \pi_D$  **do**
- 8:     **if**  $c$  is True **then**
- 9:         sample  $x_{\text{opt}}^i$  conditioned on  $y_{\text{opt}}$ ,  $\mathcal{D}_N$ , and already sampled  $\mathbf{x}_{\text{opt}}$  dimensions if any:
- 10:          $x_{\text{opt}}^i \sim p(x_{\text{opt}}^i | \mathcal{D}_N, y_{\text{opt}}, x_{\text{opt}}^{0:i-1})$ .
- 11:     **else**
- 12:         sample  $x_{\text{opt}}^i$  conditioned on  $\mathcal{D}_N$  and already sampled  $\mathbf{x}_{\text{opt}}$  dimensions if any:
- 13:          $x_{\text{opt}}^i \sim p(x_{\text{opt}}^i | \mathcal{D}_N, x_{\text{opt}}^{0:i-1})$ .
- 14:     **end if**
- 15: **end for**
- 16: get full value of  $\mathbf{x}_{\text{opt}}$  using the true indices:  $\mathbf{x}_{\text{opt}} \leftarrow (x_{\text{opt}}^1, \dots, x_{\text{opt}}^D)$ .
- 17: **return**  $\mathbf{x}_{\text{opt}}$

---

**Algorithm 2** ACE-MES

**Input:** observed data points  $\mathcal{D}_N = \{x_{1:N}, y_{1:N}\}$ , number of candidate points  $N_{\text{cand}}$ , Thompson sampling ratio for candidate point  $TS_{\text{ratio}}$ .

- 1: **Initialization**  $N_{TS1} \leftarrow N_{\text{cand}} \times TS_{\text{ratio}}$ ,  $N_{TS2} \leftarrow N_{\text{cand}} \times (1 - TS_{\text{ratio}})$ .
- 2: propose  $N_{\text{cand}}$  candidate points  $X_{1:N_{\text{cand}}}^*$  according to  $TS_{\text{ratio}}$ :
- 3: sample  $X_{1:N_{TS1}}^*$  using ACE-TS with conditioning on  $y_{\text{opt}}$  ( $c = \text{True}$ ).
- 4: sample  $X_{N_{TS1}+1:N_{TS1}+N_{TS2}}^*$  using ACE-TS without conditioning on  $y_{\text{opt}}$  ( $c = \text{True}$ ).
- 5: **for**  $i \leftarrow 1$  **to**  $N_{\text{cand}}$  **do**:
- 6:     sample  $y_{\text{opt}}$  for conditioning:  $y_{\text{opt}} \sim p(y_{\text{opt}} | \mathcal{D}_N)$ .
- 7:      $\alpha_{(i)}(\mathbf{x}_{(i)}^*) = H[p(y_{(i)}^* | \mathbf{x}_{(i)}^*, \mathcal{D}_N)] - \mathbb{E}(H[p(y_{(i)}^* | \mathbf{x}_{(i)}^*, \mathcal{D}_N, y_{\text{opt}})])$   $\triangleright$  see [Appendix C.3.2](#) for more detail
- 8: **end for**
- 9:  $\mathbf{x}_{\text{opt}} = \arg \max \boldsymbol{\alpha}$ .
- 10: **return**  $\mathbf{x}_{\text{opt}}$

---

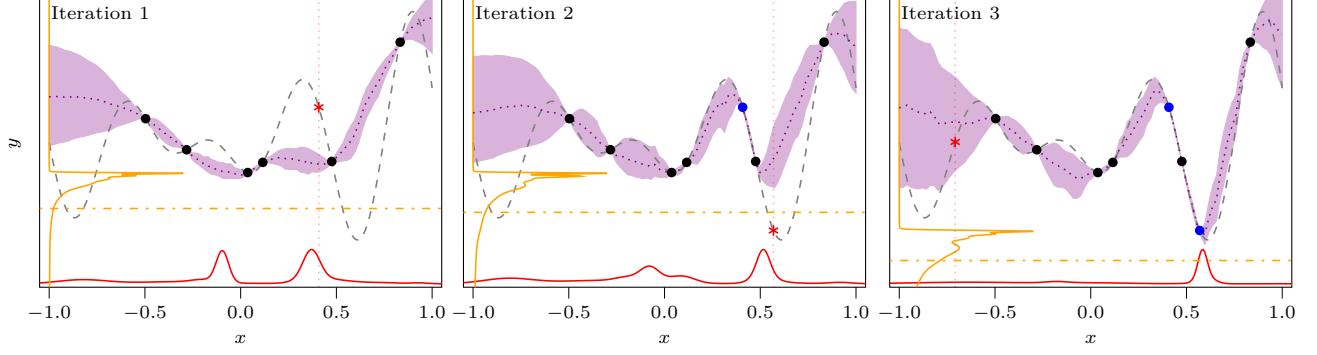


Figure S14: **Bayesian optimization example.** We show here an example evolution of ACE-TS on a 1D function. The orange pdf on the left of each panel is  $p(y_{\text{opt}}|\mathcal{D}_N)$ , the red pdf at the bottom of each panel is  $p(x_{\text{opt}}|y_{\text{opt}}, \mathcal{D}_N)$ , for a sampled  $y_{\text{opt}}$  (orange dashed-dot line). The queried point at each iteration is marked with a red asterisk, while black and blue dots represent the observed points. Note how ACE is able to learn complex conditional predictive distributions for  $x_{\text{opt}}$  and  $y_{\text{opt}}$ .

**Bayesian optimization with Minimum-value Entropy Search (ACE-MES).** For Minimum-value Entropy Search (MES; Wang and Jegelka, 2017), the procedure is as follows:

1. First, we propose  $N_{\text{candidate}}$  points. We generate these candidate points by sampling 80% of them using the conditional Thompson sampling approach described earlier, i.e.,  $p(\mathbf{x}_{\text{opt}}|\mathcal{D}_N, y_{\text{opt}} < \tau)$ , and the remaining 20% directly from  $p(\mathbf{x}_{\text{opt}}|\mathcal{D}_N)$ . In our experiments we use  $N_{\text{candidate}} = 20$ .
2. For each candidate point  $\mathbf{x}^*$ , we evaluate the acquisition function, which in our case is the gain in mutual information between the maximum  $y_{\text{opt}}$  and the candidate point  $\mathbf{x}^*$  (Eq. (5)).
3. To compute the first term of the right-hand side of Eq. (5), for a candidate point  $\mathbf{x}^*$ , we calculate the predictive distribution  $p(y^*|\mathbf{x}^*, \mathcal{D}_N)$  represented in our model by a mixture of Gaussians. We compute its entropy via numerical integration over a grid.
4. For the second term of the right-hand side of Eq. (5), we perform Monte Carlo sampling to evaluate the expected entropy. For each candidate point  $\mathbf{x}^*$ , we draw  $N_{\text{mc}}$  samples of  $y_{\text{opt}}$  from the predictive distribution  $p(y_{\text{opt}}|\mathcal{D}_N)$ . We set  $N_{\text{mc}} = 20$  to ensure the procedure remains efficient while maintaining accuracy.
5. For each sampled  $y_{\text{opt}}$ , we determine the predictive distribution  $p(y^*|\mathbf{x}^*, \mathcal{D}_N, y_{\text{opt}})$ . Then, for each mixture, we compute the entropy as in step 2. We then average over samples to compute the expectation.
6. To compute the estimated MES value of candidate point  $\mathbf{x}^*$  we subtract the computed first term to the second term of the equation Eq. (5).
7. We repeat this procedure for all candidate points  $\mathbf{x}^*$  and select the point with the highest information gain. This point is expected to yield the lowest uncertainty about the value of the minimum, thus guiding our next query in the Bayesian optimization process.

To illustrate the implementation details of ACE-MES, we present its pseudocode in [Algorithm 2](#).

### C.3.3 Bayesian optimization with prior over $\mathbf{x}_{\text{opt}}$

ACE is capable of injecting a prior over latents when predicting values and latents. In the context of BO, this prior could incorporate information about the location of the optimum,  $\mathbf{x}_{\text{opt}}$ . Several works, such as (Souza et al., 2021; Hvarfner et al., 2022; Müller et al., 2023), have explored the use of priors in BO to improve predictive performance. In our experiments, we evaluate two types of priors: strong and weak, to assess the robustness of the model under varying levels of prior knowledge. As a baseline, we utilize a  $\pi$ BO-like procedure (Hvarfner et al., 2022), as described below, to perform Thompson sampling across all experiments.

**Training.** For training, we generate a prior distribution similar to Appendix B.1, but with slight adjustments: when sampling the mixture distribution, we include a 50% chance of adding a uniform component. If present, the uniform distribution weight  $w_{\text{unif}}$  is sampled uniformly from 0.0 to 0.2 (otherwise  $w_{\text{unif}} = 0$ ). The uniform component is then added as follows:

$$\mathbf{p} = (w_{\text{unif}} \cdot \mathbf{p}_{\text{unif}}) + (1 - w_{\text{unif}}) \cdot \mathbf{p}_{\text{mixture}}, \quad (\text{S6})$$

where  $\mathbf{p}_{\text{unif}}$  represents the uniform component, and  $\mathbf{p}_{\text{mixture}}$  is the sampled mixture. The inclusion of a uniform component during training means that the prior can be a mixture of an informative and a non-informative (flat) component, which will be useful later. Using this binned distribution, we then sample  $\mathbf{x}_{\text{opt}}$  and  $y_{\text{opt}}$ , and use these two latent samples to construct our function, as outlined in Appendix C.3.1.

**Testing.** During the BO testing phase, we consider two scenarios:

1. Strong prior: We first sample a mean for the  $\mathbf{x}_{\text{opt}}$  prior by drawing from a Gaussian distribution centered on the true  $\mathbf{x}_{\text{opt}}$  with a standard deviation set to 10% of the domain (in our case  $[-1, 1]$ ), resulting in a standard deviation of 0.2. We use this sampled prior mean and standard deviation to construct the binned prior.
2. Weak prior: The same steps are applied to generate the prior, but with a standard deviation of 25%, which translates to 0.5 for our domain.

In both scenarios, we add a uniform prior component with  $w_{\text{uniform}} = 0.1$ . The uniform component helps with model and prior misspecification, by allowing the model to explore outside the region indicated by the prior.

We compare ACE with Prior Thompson Sampling (ACEP-TS) to the no-prior ACE-TS and a baseline GP-TS. We also consider a state-of-the-art heuristic for prior injection in BO,  $\pi\text{BO}$  (Hvarfner et al., 2022), with the TS acquisition procedure described below ( $\pi\text{BO-TS}$ ). The procedure is repeated 10 times for each case, with different initial points sampled at random.

**$\pi\text{BO-TS}$ .** The main technique in  $\pi\text{BO}$  for injecting a prior in the BO procedure consists of rescaling the chosen acquisition function  $\alpha(\mathbf{x})$  by the user-provided prior over the optimum location  $\pi(\mathbf{x})$  (Eq. 6 in Hvarfner et al., 2022),

$$\alpha_{\pi\text{BO}}(\mathbf{x}; \alpha) \propto \alpha(\mathbf{x})\pi(\mathbf{x})^{\gamma_n}, \quad (\text{S7})$$

where  $n$  is the BO iteration and  $\gamma_n$  governs the relative influence of the prior with respect to the acquisition function, which is heuristically made to decay over iterations to reflect the increased role of the observed data. As in Hvarfner et al. (2022), we set  $\gamma_n = \frac{\beta}{n}$  where  $\beta$  is a hyperparameter reflecting the user confidence on the prior.

To implement Thompson sampling (TS) with  $\pi\text{BO}$ , we first note that the TS acquisition function  $\alpha_{\text{TS}}(\mathbf{x})$  corresponds to the current posterior probability over the optimum location, and the TS procedure consists of drawing one sample from this acquisition function (as opposed to optimizing it). Thus, the  $\pi\text{BO}$  variant of TS ( $\pi\text{BO-TS}$ ) corresponds to sampling from Eq. (S7), where the current posterior over the optimum takes the role of  $\alpha(\mathbf{x})$ . We sample from Eq. (S7) using a self-normalized importance sampling-resampling approach (Robert and Casella, 2004). Namely, we sample  $N_{\text{TS}} = 100$  points from  $\alpha_{\text{TS}}$  using batch GP-TS, then resample one point from this batch using importance sampling weight  $w \propto \frac{\alpha_{\text{TS}}(\mathbf{x})\pi(\mathbf{x})^{\beta/n}}{\alpha_{\text{TS}}(\mathbf{x})} = \pi(\mathbf{x})^{\beta/n}$ , where all weights are then normalized to sum to 1. Following (Hvarfner et al., 2022), we set  $\beta = 10$ , i.e., equal to their setting when running BO experiments with 100 iterations, as in our case.

### C.3.4 Benchmark functions and baselines

**BO benchmarks.** We use a diverse set of benchmark functions with input dimensions ranging from 1D to 6D to thoroughly evaluate ACE’s performance on the BO task. These include (1) the non-convex Ackley function in both 1D and 2D, (2) the 1D Gramacy-Lee function, known for its multiple local minima, (3) the 1D Negative Easom function, characterized by a sharp, narrow global minimum and deceptive flat regions, (4) the non-convex 2D Branin Scaled function with multiple global minima, (5) the 2D Michalewicz function, which features a complex landscape with multiple local minima, (6) the 3D, 5D, and 6D Levy function, with numerous local minima due to its sinusoidal component, (7) the 5D and 6D Griewank function, which is highly multimodal and regularly spaced local minima, but a single smooth global minimum, (8) the 4D and 5D Rosenbrock function,

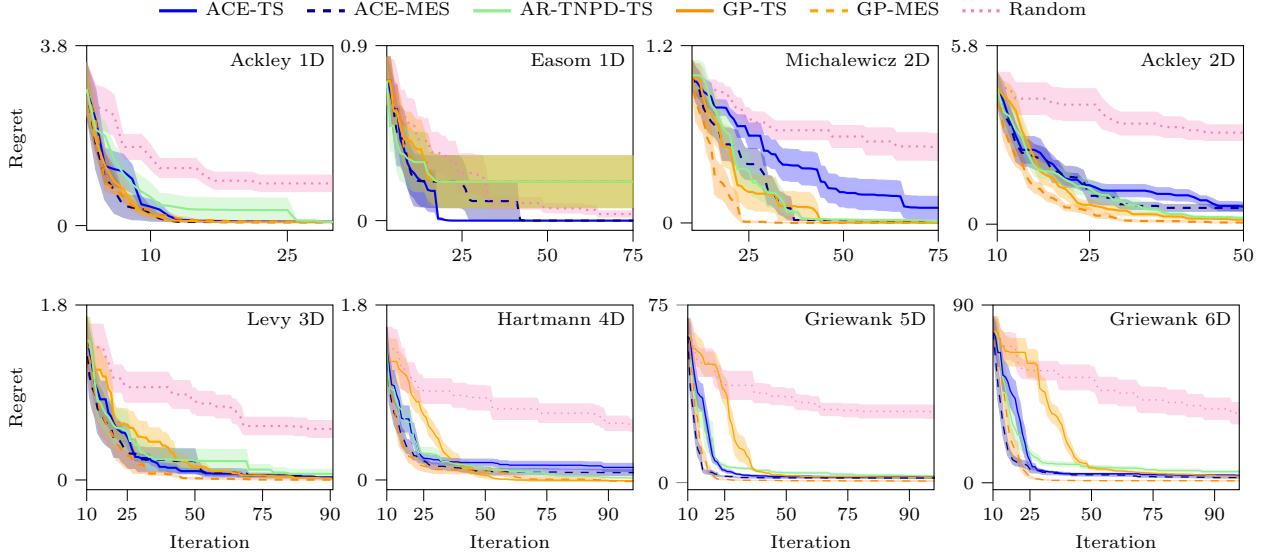


Figure S15: **Bayesian optimization additional results.** Regret comparison (mean  $\pm$  standard error) on extended BO benchmark results on distinct test functions.

which has a narrow, curved valley containing the global minimum, and (9) the 3D, 4D, and 6D Hartmann function, a widely used standard benchmark. These functions present a range of challenges, allowing us to effectively test the robustness and accuracy of ACE across different scenarios.

**BO baselines.** For our baselines, we employ three methods: autoregressive Thompson Sampling with TNP-D (AR-TNPD-TS) (Nguyen and Grover, 2022), Gaussian Process-based Bayesian Optimization with the MES acquisition function (GP-MES) (Wang and Jegelka, 2017), and Gaussian Process-based Thompson Sampling (GP-TS) with 5000 candidate points. In addition, we use  $\pi$ BO-TS for the prior injection case as the baseline Hvarfner et al. (2022) (using the same number of candidate points used in GP-TS). We optimize the acquisition function in GP-MES using the ‘shotgun’ procedure detailed later in this section (with 1000 candidate points for minimum value approximation via Gumbel sampling). Both GP-MES and GP-TS implementations are written using the BOTorch library (Balandat et al., 2020). For AR-TNPD-TS, we use the same network architecture configurations as ACE, but with a non-linear embedder and a single Gaussian head (Nguyen and Grover, 2022). Additionally, AR-TNPD-TS uses autoregressive sampling, as described in (Bruinsma et al., 2023).

We conducted our experiments with 100 BO iterations across all benchmark functions. The number of initial points was set to 3 for 1D experiments and 10 for 2D-6D experiments. These initial points were drawn uniformly randomly within the input domain. We evaluated the runtime performance of our methods and baseline algorithms on a local machine equipped with a 13th Gen Intel(R) Core(TM) i5-1335U processor and 15GB of RAM. On average, the runtime for 100 BO iterations was as follows: ACE-TS and ACEP-TS completed in approximately 5 seconds; ACE-MES required about 1.3 minutes; GP-TS and  $\pi$ BO-TS took roughly 2 minutes; GP-MES took about 1.4 minutes; and AR-TNPD-TS was the slowest, requiring approximately 10 minutes, largely due to the computational cost of its autoregressive steps.

**Shotgun optimizer.** To perform fast optimization in parallel, we first sample 10000 points from a quasirandom grid using the Sobol sequence. Then we pick the point with the highest acquisition function value, referred to as  $\mathbf{x}_0$ . Subsequently, we sample 1000 points around  $\mathbf{x}_0$  using a multivariate normal distribution with diagonal covariance  $\sigma^2 \mathbf{I}$ , where the initial  $\sigma$  is set to the median distance among the points. We re-evaluate the acquisition function over this neighborhood, including  $\mathbf{x}_0$ , and select the best point. After that, we reduce  $\sigma$  by a factor of five and repeat the process, iterating from the current best point. This ‘shotgun’ approach allows us to zoom into a high-valued region of the acquisition function while exploiting large parallel evaluations.

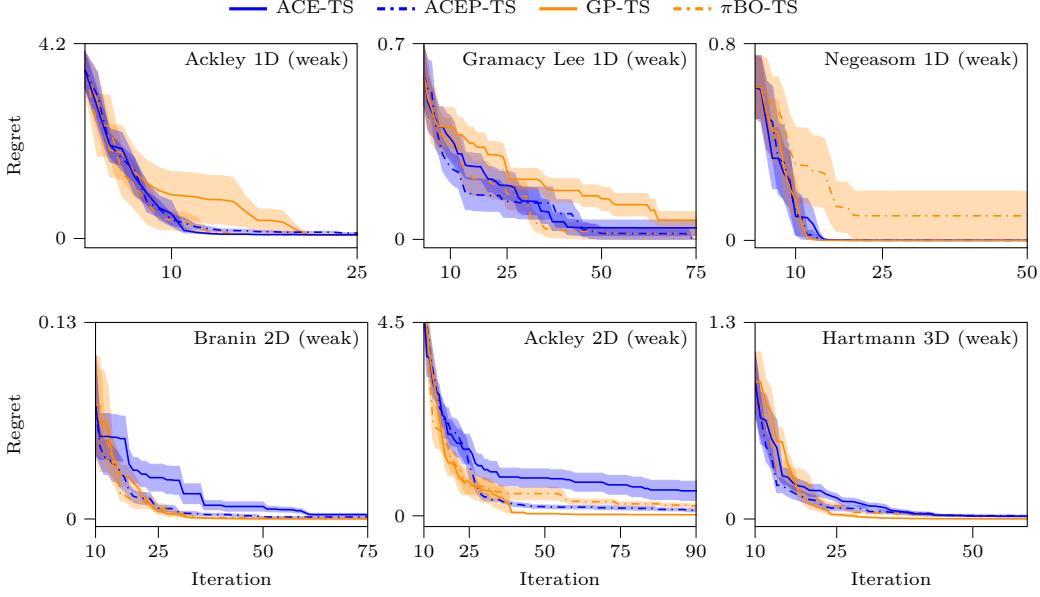


Figure S16: **Bayesian optimization with weak prior.** Simple regret (mean  $\pm$  standard error). Prior injection can improve the performance of ACE, making it perform competitively compared to  $\pi$ BO-TS.

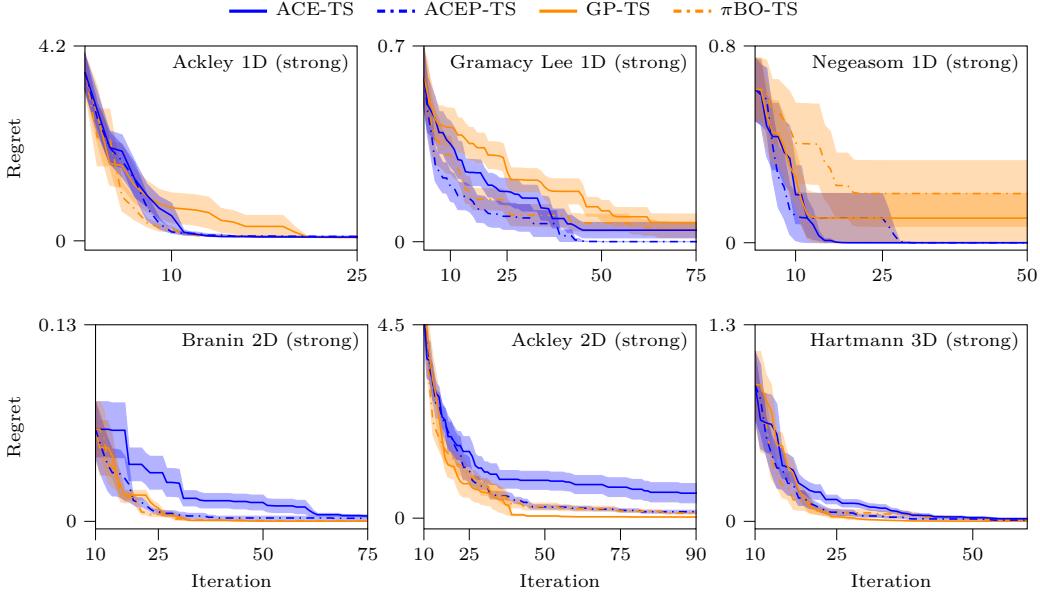


Figure S17: **Bayesian optimization with strong prior.** Simple regret (mean  $\pm$  standard error). When strong priors are used, the gap between ACE-TS and ACEP-TS is more evident compared to weak priors.

### C.3.5 Additional Bayesian optimization results.

**Standard BO setting additional results.** Additional results in Fig. S15 complement those in Fig. 5. While our method performs generally well across different benchmark functions, we find that it struggles on the Michalewicz function, likely because its sharp, narrow optima and highly irregular landscape differ significantly from the function classes used during training. Conversely, ACE performs competitively on Griewank, where the structured landscape aligns well with our approach. On the 2D Ackley function, the challenge may stem from its highly non-stationary nature, while our method was trained only on draws from stationary kernels. Addressing functions like Michalewicz and Ackley may require extending our relatively simple function generation process and incorporating specialized techniques like input and output warping (Müller et al., 2023) to better handle non-stationarity.

**BO with prior over  $x_{\text{opt}}$  additional results.** Additional results on the weak prior scenario are presented in Fig. S16 and with strong prior in Fig. S17. The results indicate that ACEP-TS consistently outperforms ACE-TS, particularly when using a strong prior. In this case, the model benefits from the prior information, leading to a notable improvement in performance. Specifically, the strong prior allows the model to converge more rapidly toward the optimum.

## C.4 Simulation-based inference

### C.4.1 Simulators

The experiments reported in Section 4.3 used three time-series models to simulate the training and test data. This section describes the simulators in more details.

**Ornstein-Uhlenbeck Process (OUP)** is widely used in financial mathematics and evolutionary biology due to its ability to model mean-reverting stochastic processes (Uhlenbeck and Ornstein, 1930). The model is defined as:

$$y_{t+1} = y_t + \Delta y_t, \quad \Delta y_t = \theta_1 [\exp(\theta_2) - y_t] \Delta t + 0.5w, \quad \text{for } t = 1, \dots, T,$$

where  $T = 25$ ,  $\Delta t = 0.2$ ,  $x_0 = 10$ , and  $w \sim \mathcal{N}(0, \Delta t)$ . We use a uniform prior  $U([0, 2] \times [-2, 2])$  for the latent variables  $\boldsymbol{\theta} = (\theta_1, \theta_2)$  to generate the simulated data.

**Susceptible-Infectious-Recovered (SIR)** is a simple compartmental model used to describe infectious disease outbreaks (Kermack and McKendrick, 1927). The model divides a population into susceptible (S), infectious (I), and recovered (R) individuals. Assuming population size  $N$  and using  $S_t$ ,  $I_t$ , and  $R_t$  to denote the number of individuals in each compartment at time  $t$ ,  $t = 1, \dots, T$ , the disease outbreak dynamics can be expressed as

$$\Delta S_t = -\beta \frac{I_t S_t}{N}, \quad \Delta I_t = \beta \frac{I_t S_t}{N} - \gamma I_t, \quad \Delta R_t = \gamma I_t,$$

where the parameters  $\beta$  and  $\gamma$  denote the contact rate and the mean recovery rate. An observation model with parameters  $\phi$  is used to convert the SIR model predictions to observations  $(t, y_t)$ . The experiments carried out in this work consider two observation models and simulator setups.

The setups considered in this work are as follows. First, we consider a SIR model with fixed initial condition and 10 observations  $y_t \sim \text{Bin}(1000, I_t/N)$  collected from  $T = 160$  time points at even interval, as proposed in (Lueckmann et al., 2021). Here the population size  $N = 10^6$  and the initial condition is fixed as  $S_0 = N - 1$ ,  $I_0 = 1$ ,  $R_0 = 0$ . We use uniform priors  $\beta \sim U(0.01, 1.5)$  and  $\gamma \sim U(0.02, 0.25)$ . We used this model version in the main experiments presented in Section 4.3 and Appendix C.4.2.

In addition we consider a setup where  $N$  and  $I_0$  are unknown and we collect 25 observations  $y_t \sim \text{Poi}(\phi I_t/N)$  from  $T = 250$  time points at even interval. We use  $\beta \sim U(0.5, 3.5)$ ,  $\gamma \sim U(0.0001, 1.5)$ ,  $\phi \sim U(50, 5000)$ , and  $I_0/N \sim U(0.0001, 0.01)$  with  $S_0/N = 1 - I_0/N$  and  $R_0/N = 0$  to generate simulated samples. We used this model version in an additional experiment to test ACE on real world data, presented in Appendix C.4.5.

**Turin** model is a time-series model used to simulate radio propagation phenomena, making it useful for testing and designing wireless communication systems (Turin et al., 1972; Pedersen, 2019; Bharti et al., 2019). The model generates high-dimensional complex-valued time-series data and is characterized by four key parameters that control different aspects of the radio signal:  $G_0$  controls the reverberation gain,  $T$  determines the reverberation time,  $\nu$  specifies the arrival rate of the point process, and  $\sigma_W^2$  represents the noise variance.

The model starts with a frequency bandwidth  $B = 0.5$  GHz and simulates the transfer function  $H_k$  over  $N_s = 101$  equidistant frequency points. The measured transfer function at the  $k$ -th point,  $Y_k$ , is given by:

$$Y_k = H_k + W_k, \quad k = 0, 1, \dots, N_s - 1,$$

where  $W_k$  denotes additive zero-mean complex circular symmetric Gaussian noise with variance  $\sigma_W^2$ . The transfer function  $H_k$  is defined as:

$$H_k = \sum_{l=1}^{N_{\text{points}}} \alpha_l \exp(-j2\pi\Delta f k \tau_l),$$

where  $\tau_l$  are the time delays sampled from a one-dimensional homogeneous Poisson point process with rate  $\nu$ , and  $\alpha_l$  are complex gains. The gains  $\alpha_l$  are modeled as i.i.d. zero-mean complex Gaussian random variables conditioned on the delays, with a conditional variance:

$$\mathbb{E}[|\alpha_l|^2 | \tau_l] = \frac{G_0 \exp(-\tau_l/T)}{\nu}.$$

The time-domain signal  $\tilde{y}(t)$  can be obtained by taking the inverse Fourier transform:

$$\tilde{y}(t) = \frac{1}{N_s} \sum_{k=0}^{N_s-1} Y_k \exp(j2\pi k \Delta f t),$$

with  $\Delta f = B/(N_s - 1)$  being the frequency separation. Our final real-valued output is calculated by taking the absolute square of the complex-valued data and applying a logarithmic transformation  $y(t) = 10 \log_{10}(|\tilde{y}(t)|^2)$ .

The four parameters of the model are sampled from the following uniform priors:  $G_0 \sim \mathcal{U}(10^{-9}, 10^{-8})$ ,  $T \sim \mathcal{U}(10^{-9}, 10^{-8})$ ,  $\nu \sim \mathcal{U}(10^7, 5 \times 10^9)$ ,  $\sigma_W^2 \sim \mathcal{U}(10^{-10}, 10^{-9})$ .

#### C.4.2 Main experiments

ACE was trained on examples that included simulated time series data and model parameters divided between target and context. In these experiments, the time series data were divided into context and target data by sampling  $N_d$  data points into the context set and including the rest in the target set. The context size  $N_d \sim U(10, 25)$  in the OUP experiments,  $N_d \sim U(5, 10)$  in the SIR experiments, and  $N_d \sim U(50, 101)$  in the Turin experiments. In addition, the model parameters were randomly assigned to either the context or target set. NPE and NRE cannot handle partial observations and was trained with the full time series data in both cases.

The ACE model used in these experiments had embedding dimension 64 and 6 transformer layers. The attention blocks had 4 heads and the MLP block had hidden dimension 128. The output head had  $K = 20$  MLP components with hidden dimension 128. The model was trained for  $5 \times 10^4$  steps with batch size 32, using learning rate  $5 \times 10^{-4}$  with cosine annealing.

We used the `sbi` package (Tejero-Cantero et al., 2020) (<https://sbi-dev.github.io/sbi/>, Version: 0.22.0, License: Apache 2.0) to implement NPE and NRE. Specifically, we chose the NPE-C (Greenberg et al., 2019) and NRE-C (Miller et al., 2022) with Masked Autoregressive Flow (MAF) (Papamakarios et al., 2017) as the inference network. We used the default configuration with 50 hidden units and 5 transforms for MAF, and training with a fixed learning rate  $5 \times 10^{-4}$ . For Simformer (Gloeckler et al., 2024), we used their official package (<https://github.com/mackelab/simformer>, Version: 2, License: MIT). We used the same configuration as in our setup for the transformer, while we used their default configuration for the diffusion part. For a fair comparison, we pre-generated  $10^4$  parameter-simulation pairs for all methods. We also normalized the parameters of the Turin model when feeding into the networks. For evaluation, we randomly generated 100 observations and assessed each method across 5 runs. For the RMSE evaluation, given  $N_{\text{obs}}$  observations, with  $N_{\text{post}}$  posterior samples generated for each observation, and  $L$  latent parameters, our RMSE metric is calculated as:

$$\text{RMSE} = \frac{1}{N_{\text{obs}}} \sum_{i=1}^{N_{\text{obs}}} \sqrt{\frac{1}{L \cdot N_{\text{post}}} \sum_{l=1}^L \sum_{j=1}^{N_{\text{post}}} (\theta_{i,l} - \hat{\theta}_{i,l,j})^2},$$

where  $\theta_{i,l}$  represents the true value of the  $l$ -th latent parameter for the  $i$ -th observation, and  $\hat{\theta}_{i,l,j}$  represents the  $j$ -th posterior sample of the  $l$ -th latent parameter for the  $i$ -th observation. This approach first calculates the RMSE for each observation (averaging across all latent dimensions and posterior samples for that observation), and then averages these observation-specific RMSE values to obtain the final metric. For MMD, we use an exponentiated quadratic kernel with a lengthscale of 1.

**Statistical comparisons.** We evaluate models based on their average results across multiple runs and perform pairwise comparisons to identify models with comparable performance. The results from pairwise comparisons are used in Table 1 to highlight in bold the models that are considered best in each experiment. The following procedure is used to determine the best models:

- First, we identify the model (A) with the highest empirical mean and highlight it in bold.
- For each alternative model (B), we perform  $10^5$  bootstrap iterations to resample the mean performance for both model A and model B.
- We then calculate the proportion of bootstrap iterations where model B outperforms model A.
- If this proportion is larger than the significance level ( $\alpha = 0.05$ ), model B is considered statistically indistinguishable from model A.
- All models that are not statistically significantly different from the best model are highlighted in bold.

#### C.4.3 Ablation study: Gaussian vs. mixture-of-Gaussians output heads

To assess the impact of using a Gaussian versus a mixture-of-Gaussians output head in ACE, we conduct an ablation study on the SBI tasks. In theory, a mixture-of-Gaussians head should improve performance when the predictive or posterior data distributions are non-Gaussian. Table S2 shows the results. As expected, we observe improvements in OUP and Turin when using a mixture-of-Gaussians head. This suggests that more flexible distributional families better capture complex distributions. However, for the SIR task, the performance difference is negligible as the posteriors are largely Gaussian. These findings align with our expectations.

		Gaussian (ablation)	Mixture-of-Gaussians (ACE)
OUP	log-probs $_{\theta}$ ( $\uparrow$ )	0.90(0.01)	<b>1.03</b> (0.02)
	RMSE $_{\theta}$ ( $\downarrow$ )	0.48(0.01)	0.48(0.00)
	MMD $_y$ ( $\downarrow$ )	0.52(0.00)	<b>0.51</b> (0.00)
SIR	log-probs $_{\theta}$ ( $\uparrow$ )	6.80(0.02)	6.78(0.02)
	RMSE $_{\theta}$ ( $\downarrow$ )	0.02(0.00)	0.02(0.00)
	MMD $_y$ ( $\downarrow$ )	0.02(0.00)	0.02(0.00)
Turin	log-probs $_{\theta}$ ( $\uparrow$ )	2.73(0.02)	<b>3.14</b> (0.02)
	RMSE $_{\theta}$ ( $\downarrow$ )	0.24(0.00)	0.24(0.00)
	MMD $_y$ ( $\downarrow$ )	0.36(0.00)	<b>0.35</b> (0.00)

Table S2: Ablation study comparing single Gaussian versus mixture-of-Gaussians output heads across SBI tasks. Mean and standard deviation from 5 runs are reported. mixture-of-Gaussians heads benefit complex distributions (OUP and Turin), while maintaining similar performance on simpler tasks (SIR).

#### C.4.4 Simulation-based calibration

To evaluate the calibration of the approximate posteriors obtained by ACE, we apply simulation-based calibration (SBC; [Talts et al. 2018](#)) on the Turin model to evaluate whether the approximate posteriors produced by ACE are calibrated. We recall that SBC checks if a Bayesian inference process is well-calibrated by repeatedly simulating data from parameters drawn from the prior and inferring posteriors under those priors and simulated datasets. If the inference is calibrated, the average posterior should match the prior. Equivalently, when ranking the true parameters within each posterior, the ranks should follow a uniform distribution ([Talts et al., 2018](#)).

We use the following procedure for SBC: for a given prior, we first sample 1000 samples from the prior distribution and generate corresponding simulated data. Then we use ACE to approximate the posteriors and subsequently compare the true parameter values with samples drawn from the inferred posterior distribution. To visualize the calibration, we plot the density of the posterior samples against the prior samples. If the model is well-calibrated, the posterior distribution should recover the true posterior, which results in a close match between the density of the posterior samples and the prior. We also present the fractional rank statistic against the ECDF difference ([Säilynoja et al., 2022](#)). Ideally, the ECDF difference between the rank statistics and the theoretical uniform distribution should remain close to zero, indicating well-calibrated posteriors.

[Fig. S18](#) shows that our ACE is well-calibrated with pre-defined uniform priors across all four latents. Since ACEP allows conditioning on different priors at runtime, we also test the calibration of ACEP using randomly generated priors (following [Appendix B.1](#)). For comparison, we show what happens if we forego prior-injection, using vanilla ACE instead of ACEP. The visualization on one set of priors is shown in [Fig. S19](#). As expected,

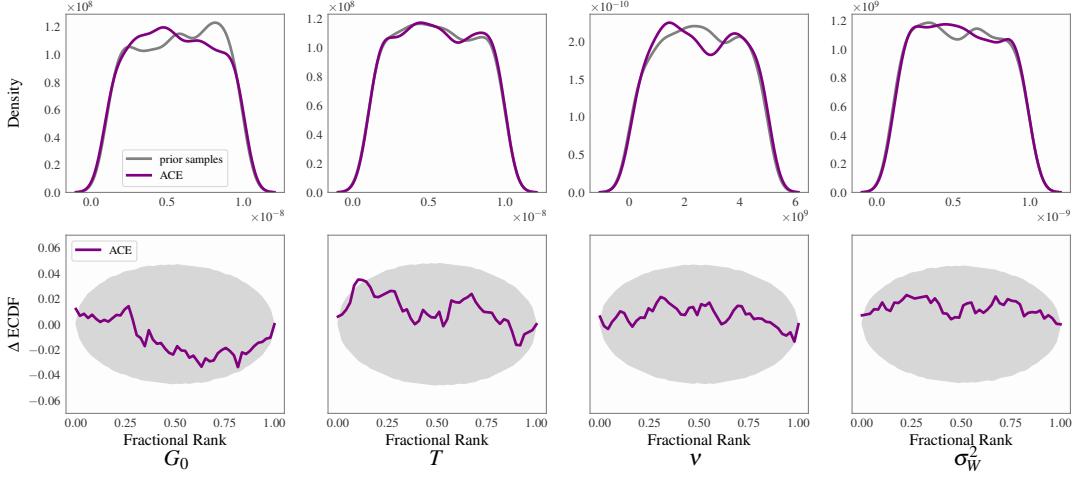


Figure S18: Simulation-based calibration of ACE on the Turin model. The top row shows the density of the posterior samples from ACE compared with the prior samples. The bottom row shows the fractional rank statistic against the ECDF difference with 95% confidence bands. ACE is well-calibrated.

vanilla ACE (without prior-injection) does not include the correct prior information and shows suboptimal calibration performance, whereas ACEP correctly leverages the provided prior information and shows closer alignment with the prior and lower ECDF deviations. We also calculate the average absolute deviation over 100 randomly sampled priors. In the prior-injection setting, ACEP demonstrates better calibration, with an average deviation of  $0.03 \pm 0.01$  compared to  $0.10 \pm 0.05$  for ACE without the correct prior.

#### C.4.5 Extended SIR model on real-world data

We present here results obtained by considering an extended four-parameter version of the SIR model then applied to real-world data. We further include details on the training data and model configurations used in the real-data experiment as well as additional evaluation results from experiments carried out with simulated data. As our real-world data, we used a dataset that describes an influenza outbreak in a boarding school. The dataset is available in the R package `outbreaks` (<https://cran.r-project.org/package=outbreaks>, Version: 1.9.0, License: MIT).

**Methods.** The four-parameter SIR model we used is detailed in Appendix C.4.1 (last paragraph). The ACE models were trained with samples constructed based on simulated data as follows. The observations were divided into context and target points by sampling  $N_d \sim U(2, 20)$  data points into the context set and 2 data points into the target set. The examples included 50% interpolation tasks where the context and target points were sampled at random (without overlap) and 50% forecast tasks where the points were sampled in order. The model parameters were divided between the context and target set by sampling the number to include  $N_l \sim U(0, 4)$  and sampling the  $N_l$  parameters from the parameter set at random. The parameters were normalized to range  $[-1, 1]$  and the observations were square-root compressed and scaled to the approximate range  $[-1, 1]$ .

The ACE models had the same architecture as the models used in the main experiment, but the models were trained for  $10^5$  steps with batch size 32. In this experiment, we generated the data online during the training, which means that the models were trained with  $3.2 \times 10^6$  samples. The NPE models used in this experiment had the same configuration as the model used in the main experiment, for fair comparison, the models were now trained with  $3.2 \times 10^6$  samples. Each sample corresponded to a unique simulation and the full time series was used as the observation data.

To validate model predictions, we note that ground-truth parameter values are not available for real data. Instead, we examined whether running the simulator with parameters sampled from the posterior can replicate the observed data. For reference, we also included MCMC results. The MCMC posterior was sampled with Pyro (Bingham et al., 2018) (<https://pyro.ai/>, Version: 1.9.0, License: Apache 2.0) using the random walk kernel

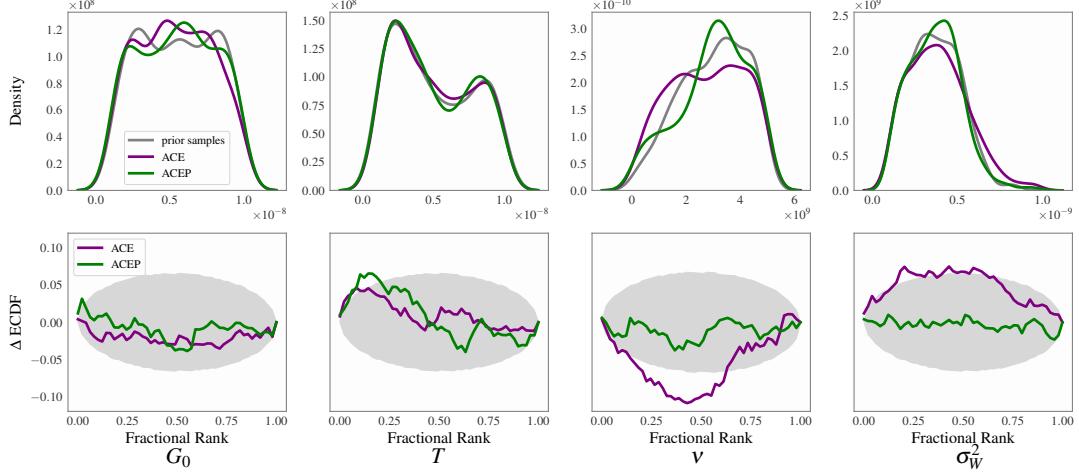


Figure S19: Simulation-based calibration of ACE and ACEP on the Turin model with an example custom prior. ACEP demonstrates improved calibration by closely following the prior distribution and showing lower deviations in the ECDF difference, highlighting its ability to condition on user-specified priors effectively.

and sampling 4 chains with  $5 \times 10^4$  warm-up steps and  $5 \times 10^4$  samples.

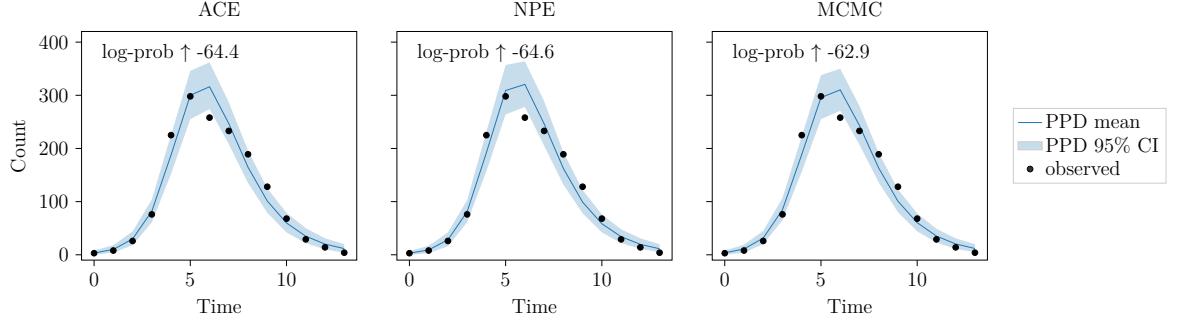


Figure S20: **SIR model on a real dataset.** Posterior predictive distributions based on the ACE, NPE, and MCMC posteriors. The dataset is mildly misspecified, in that even MCMC does not fully match the data.

**Results.** The posterior predictive distributions and log-probabilities for observed data calculated based on ACE, NPE, and MCMC results are shown in Fig. S20. For this visualization, ACE and NPE models were trained once, and simulations were carried out with 5000 parameters sampled from each posterior distribution. The log-probabilities observed in this experiment are  $-64.4$  with ACE,  $-64.6$  with NPE. Repeating ACE and NPE training and posterior estimation 10 times, the average log-probabilities across the 10 runs were  $-65.1$  (standard deviation 0.4) with ACE and  $-65.5$  (standard deviation 0.7) with NPE, showing a similar performance. The ACE predictions used in this experiment are sampled autoregressively (see Appendix B.4). These results show that ACE can handle inference with real data.

**Validation on simulated data.** For completeness, we performed a more extensive validation of ACE and other methods with the extended SIR model using simulated data. Specifically, we assessed the ACE and NPE models on simulated data and evaluated the same ACE models in a data completion task with the TNP-D baseline. All the training details remain the same as in the real-world experiment for ACE and NPE. The TNP-D models had the same overall architecture as ACE but used a different embedder and output head. The MLP block in the TNP-D embedder had hidden dimension 64 and the MLP block in the single-component output head hidden dimension 128. The TNP-D models were trained for  $10^5$  steps with batch size 32. The evaluation set used in these experiments included 1000 simulations sampled from the training distribution and the evaluation metrics included log-probabilities and coverage probabilities calculated based on 95% quantile intervals that were

Table S3: Comparison between ACE and NPE in posterior estimation task in the extended SIR model. The ACE predictions were generated autoregressively so both methods target the joint posterior. The estimated posteriors are compared based on log-probabilities and 95% marginal coverage probabilities. The evaluation set includes 1000 examples and we report the mean and (standard deviation) from 10 runs. ACE log-probabilities are on average better than NPE log-probabilities and the coverage probabilities are close to the nominal level 0.95.

	log-probs ( $\uparrow$ )	cover $\beta$	cover $\gamma$	cover $\phi$	cover $I_0$	cover ave
NPE	6.63 (0.16)	0.92 (0.01)	0.94 (0.01)	0.94 (0.01)	0.92 (0.01)	0.93 (0.01)
ACE (AR)	7.38 (0.04)	0.96 (0.00)	0.97 (0.00)	0.97 (0.00)	0.96 (0.00)	0.97 (0.00)

Table S4: ACE posterior estimation based on incomplete data with  $M$  observation points using either independent or autoregressive predictions. The estimated posteriors are evaluated using (a) log-probabilities and (b) average 95% marginal coverage probabilities. We report the mean and (standard deviation) from 10 runs. The log-probabilities improve when the context size  $M$  increases and when autoregressive predictions are used.

		$M = 25$	$M = 20$	$M = 15$	$M = 10$	$M = 5$
(a)	ACE	4.94 (0.04)	4.55 (0.03)	3.87 (0.02)	2.82 (0.03)	0.88 (0.03)
	ACE (AR)	7.38 (0.04)	6.93 (0.04)	6.21 (0.04)	5.11 (0.04)	2.91 (0.05)
(b)	ACE	0.97 (0.00)	0.96 (0.00)	0.95 (0.00)	0.95 (0.00)	0.96 (0.00)
	ACE (AR)	0.97 (0.00)	0.97 (0.00)	0.96 (0.00)	0.96 (0.00)	0.97 (0.00)

estimated based on 5000 samples.

We start with the posterior estimation task where we used ACE and NPE to predict simulator parameters based on the simulated observations with 25 observation points. The results are reported in Table S3. We observe that the ACE log-probabilities are on average better than NPE log-probabilities and that both methods have marginal coverage probabilities close to the nominal level 0.95.

The simulated observations used in the previous experiment were complete with 25 observation points. Next, we evaluate ACE posteriors estimated based on incomplete data with 5–20 observation points. NPE is not included in this experiment since it cannot handle incomplete observations. Instead, we use this experiment to compare independent and autoregressive ACE predictions. The results are reported in Table S4. The log-probabilities indicate that both independent and autoregressive predictions improve when more observation points are available while the coverage probabilities are close to the nominal level in all conditions. That autoregressive predictions result in better log-probabilities than independent predictions indicates that ACE is able to use dependencies between simulator parameters.

Table S5: Comparison between ACE and TNP-D in data completion task in the extended SIR model. The estimated predictive distributions are compared based on (a) log-probabilities and (a) 95% coverage probabilities. We report the mean and (standard deviation) from 10 runs. ACE log-probabilities are on average better than TNP-D log-probabilities and improve both when the context size  $M$  increases or when predictions are conditioned on the simulator parameters  $\theta$ .

		$M = 20$	$M = 15$	$M = 10$	$M = 5$
(a)	TNP-D	10.1 (0.11)	9.99 (0.09)	9.44 (0.10)	8.02 (0.07)
	ACE	14.2 (0.31)	13.8 (0.31)	13.2 (0.31)	11.4 (0.28)
	ACE + $\theta$	14.7 (0.31)	14.6 (0.31)	14.6 (0.30)	14.3 (0.30)
(b)	TNP-D	0.96 (0.00)	0.96 (0.00)	0.95 (0.00)	0.95 (0.00)
	ACE	0.97 (0.00)	0.96 (0.00)	0.96 (0.00)	0.95 (0.00)
	ACE + $\theta$	0.96 (0.00)	0.96 (0.00)	0.96 (0.00)	0.96 (0.00)

The same ACE models that have been evaluated in the posterior estimation (latent prediction) task can also make predictions about the unobserved values in incomplete data. To evaluate ACE in the data completion task,

we selected 5 target observations from each evaluation sample and used 5–20 remaining observations as context. We used ACE to make target predictions either based on the context data alone or based on both context data and the simulator parameters  $\theta$ . For comparison, we also evaluated data completion with TNP-D. The results are reported in [Table S5](#). We observe that ACE log-probabilities are on average better than TNP-D log-probabilities and improve when simulator parameters are available as context. In these experiments, both ACE and TNP-D were used to make independent predictions.

### C.5 Computational resources and software

For the experiments and baselines, we used a GPU cluster containing AMD MI250X GPUs. All experiments can be run using a single GPU with a VRAM of 50GB. Most of the experiments took under 6 hours, with the exception of a few BO experiments that took around 10 hours. The core code base was built using Pytorch ([Paszke et al., 2019](#)) (<https://pytorch.org/> Version: 2.2.0, License: modified BSD license) and based on the Pytorch implementation for TNP ([Nguyen and Grover, 2022](#)) (<https://github.com/tung-nd/TNP-pytorch>, License: MIT). Botorch ([Balandat et al., 2020](#)) (<https://github.com/pytorch/botorch> Version: 0.10.0, License: MIT) was used for the implementation of GP-MES, GP-TS, and  $\pi$ BO-TS.