# Posterior Matching for Arbitrary Conditioning

**Ryan R. Strauss**
Department of Computer Science
UNC at Chapel Hill
rrs@cs.unc.edu

**Junier B. Oliva**
Department of Computer Science
UNC at Chapel Hill
joliva@cs.unc.edu

## Abstract

Arbitrary conditioning is an important problem in unsupervised learning, where we seek to model the conditional densities $p(\mathbf{x}_u \mid \mathbf{x}_o)$ that underly some data, for all possible non-intersecting subsets $o, u \subset \{1, \ldots, d\}$. However, the vast majority of density estimation only focuses on modeling the joint distribution $p(\mathbf{x})$, in which important conditional dependencies between features are opaque. We propose a simple and general framework, coined Posterior Matching, that enables Variational Autoencoders (VAEs) to perform arbitrary conditioning, without modification to the VAE itself. Posterior Matching applies to the numerous existing VAE-based approaches to joint density estimation, thereby circumventing the specialized models required by previous approaches to arbitrary conditioning. We find that Posterior Matching is comparable or superior to current state-of-the-art methods for a variety of tasks with an assortment of VAEs (e.g. discrete, hierarchical, VaDE).

## 1 Introduction

Variational Autoencoders (VAEs) [21] are a widely adopted class of generative model that have been successfully employed in numerous areas [4, 15, 26, 33, 16]. Much of their appeal stems from their ability to probabilistically represent complex data in terms of lower-dimensional latent codes.

Like most other generative models, VAEs are typically designed to model the joint data distribution, which communicates likelihoods for particular configurations of all features at once. This can be useful for some tasks, such as generating images, but the joint distribution is limited by its inability to explicitly convey the conditional dependencies between features. In many cases, conditional distributions, which provide the likelihood of an event given some known information, are more relevant and useful. Conditionals can be obtained in theory by marginalizing the joint distribution, but in practice, this is generally not analytically available and is expensive to approximate.

Easily assessing the conditional distribution over *any subset* of features is important for tasks where decisions and predictions must be made over a varied set of possible information. For example, some medical applications may require reasoning over: the distribution of *blood pressure* given *age* and *weight*; or the distribution of *heart-rate* and *blood-oxygen level* given *age*, *blood pressure*, and *BMI*; etc. For flexibility and scalability, it is desirable for a *single* model to provide all such conditionals at inference time. More formally, this task is known as *arbitrary conditioning*, where the goal is to model the conditional density $p(\mathbf{x}_u \mid \mathbf{x}_o)$ for any arbitrary subsets of unobserved features $\mathbf{x}_u$ and observed features $\mathbf{x}_o$. In this work, we show, by way of a simple and general framework, that traditional VAEs can perform arbitrary conditioning, without modification to the VAE model itself.

Our approach, which we call Posterior Matching, is to model the distribution $p(\mathbf{z} \mid \mathbf{x}_o)$ that is induced by some VAE, where $\mathbf{z}$ is the latent code. In other words, we consider the distribution of latent codes given partially observed features. We do this by having a neural network output an approximate partially observed posterior $q(\mathbf{z} \mid \mathbf{x}_o)$. In order to train this network, we develop a straightforward maximum likelihood estimation objective and show that it is equivalent to maximizing $p(\mathbf{x}_u \mid \mathbf{x}_o)$,

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

the quantity of interest. Unlike prior works that use VAEs for arbitrary conditioning, we do not make special assumptions or optimize custom variational lower bounds. *Rather, training via Posterior Matching is simple, highly flexible, and without limiting assumptions on approximate posteriors (e.g., $q(\mathbf{z} \mid \mathbf{x}_o)$ need not be reparameterized and can thus be highly expressive).*

We conduct several experiments in which we apply Posterior Matching to various types of VAEs for a myriad of different tasks, including image inpainting, tabular arbitrary conditional density estimation, partially observed clustering, and active feature acquisition. We find that Posterior Matching leads to improvements over prior VAE-based methods across the range of tasks we consider.

## 2 Background

**Arbitrary Conditioning** A core problem in unsupervised learning is *density estimation*, where we are given a dataset $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^{N}$ of *i.i.d.* samples drawn from an unknown distribution $p(\mathbf{x})$ and wish to learn a model that best approximates the probability density function $p$. A limitation of only learning the joint distribution $p(\mathbf{x})$ is that it does not provide direct access to the conditional dependencies between features. *Arbitrary conditional density estimation* [18, 24, 38] is a more general task where we want to estimate the conditional density $p(\mathbf{x}_u \mid \mathbf{x}_o)$ for all possible subsets of observed features $o \subset \{1, \ldots, d\}$ and unobserved features $u \subset \{1, \ldots, d\}$ such that $o$ and $u$ do not intersect. Here, $\mathbf{x}_o \in \mathbb{R}^{|o|}$ and $\mathbf{x}_u \in \mathbb{R}^{|u|}$. Estimation of joint or marginal likelihoods is a special case where $o = \emptyset$. Note that, while not strictly necessary for arbitrary conditioning methods [24, 38], we assume $\mathcal{D}$ is fully observed, a requirement for training traditional VAEs.

**Variational Autoencoders** Variational Autoencoders (VAEs) [21] are a class of generative models that assume a generative process in which data likelihoods are represented as $p(\mathbf{x}) = \int p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z}) \, \mathrm{d}\mathbf{z}$, where $\mathbf{z}$ is a latent variable that typically has lower dimensionality than the data $\mathbf{x}$. A tractable distribution that affords easy sampling and likelihood evaluation, such as a standard Gaussian, is usually imposed on the prior $p(\mathbf{z})$. These models are learned by maximizing the *evidence lower bound* (ELBO) of the data likelihood:

$$\log p(\mathbf{x}) \geq \mathbb{E}_{\mathbf{z} \sim q_\psi(\cdot \mid \mathbf{x})}[\log p_\phi(\mathbf{x} \mid \mathbf{z})] - \mathrm{KL}(q_\psi(\mathbf{z} \mid \mathbf{x}) \mid\mid p(\mathbf{z})),$$

where $q_\psi(\mathbf{z} \mid \mathbf{x})$ and $p_\phi(\mathbf{x} \mid \mathbf{z})$ are the encoder (or approximate posterior) and decoder of the VAE, respectively. The encoder and decoder are generally neural networks that output tractable distributions (e.g., a multivariate Gaussian). In order to properly optimize the ELBO, samples drawn from $q_\psi(\mathbf{z} \mid \mathbf{x})$ must be differentiable with respect to the parameters of the encoder (often called the *reparameterization trick*). After training, a new data point $\hat{\mathbf{x}}$ can be easily generated by first sampling $\mathbf{z}$ from the prior, then sampling $\hat{\mathbf{x}} \sim p_\phi(\cdot \mid \mathbf{z})$.

## 3 Posterior Matching

In this section we describe our framework, coined Posterior Matching, to model the underlying arbitrary conditionals in a VAE. In many respects, Posterior Matching *cuts the Gordian knot* to uncover the conditional dependencies. Following our insights, we show that our approach is *direct* and *intuitive*. Notwithstanding, we are the first to apply this direct methodology for arbitrary conditionals in VAEs and are the first to connect our proposed loss with arbitrary conditional likelihoods $p(\mathbf{x}_u \mid \mathbf{x}_o)$. Note that we are *not* proposing a new type of VAE. Rather, we are formalizing a simple and intuitive methodology that can be applied to numerous existing (or future) VAEs.

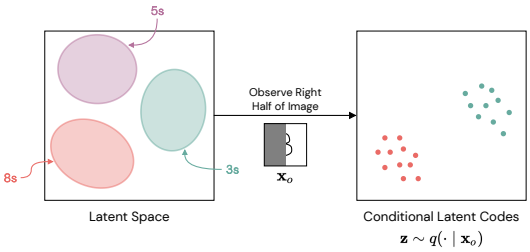



Figure 1: A two-dimensional VAE latent space (left) that represents the distribution of codes of handwritten 3s, 5s, and 8s. Conditioning on the subset of pixels shown in the center should then result in a distribution over latent codes that only correspond to 3s and 8s, as shown on the right.

### 3.1 Motivation

Let us begin with a motivating example, depicted in Figure 1. Suppose we have trained a VAE on images of handwritten 3s, 5s, and 8s. This VAE has thus learned to represent these images in a low-dimensional latent space. Any given code (vector) in this latent space represents a distribution over images in the original data space, which can be retrieved by passing that code through the VAE's decoder. Some regions in the latent space will contain codes that represent 3s, some will represent 5s, and some will represent 8s. There is typically only an interest in mapping from a given image $\mathbf{x}$ to a distribution over the latent codes that could represent that image, i.e., the posterior $q(\mathbf{z} \mid \mathbf{x})$. *However, we can just as easily ask which latent codes are feasible having only observed part of an image.*

For example, if we only see the right half the image shown in Figure 1, we know the digit could be a 3 or an 8, but certainly not a 5. Thus, the distribution over latent codes that could correspond to the full image, that is $p_\psi(\mathbf{z} \mid \mathbf{x}_o)$ (where $\psi$ is the encoder's parameters), should only include regions that represent 3s or 8s. Decoding any sample from $p_\psi(\mathbf{z} \mid \mathbf{x}_o)$ will produce an image of a 3 or an 8 that aligns with what has been observed.

The important *insight* is that we can think about how conditioning on $\mathbf{x}_o$ changes the distribution over latent codes without explicitly worrying about what the (potentially higher-dimensional and more complicated) conditional distribution over $\mathbf{x}_u$ looks like. Once we know $p_\psi(\mathbf{z} \mid \mathbf{x}_o)$, we can easily move back to the original data space using the decoder.

### 3.2 Approximating the Partially Observed Posterior

The partially observed approximate posterior of interest is not readily available, as it is implicitly defined by the VAE:

$$p_\psi(\mathbf{z} \mid \mathbf{x}_o) = \mathbb{E}_{\mathbf{x}_u \sim p(\cdot \mid \mathbf{x}_o)}\Big[q_\psi(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u)\Big], \tag{1}$$

where $q_\psi(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u) = q_\psi(\mathbf{z} \mid \mathbf{x})$ is the VAE's encoder. Thus, we introduce a neural network in order to approximate it.

Given a network that outputs the distribution $q_\theta(\mathbf{z} \mid \mathbf{x}_o)$ (i.e. the partially observed encoder in Figure 2), we now discuss our approach to training it. Our approach is guided by the priorities of simplicity and generality. We minimize (with respect to $\theta$) the following likelihoods, where the samples are coming from our target distribution as defined in Equation 1:

$$\mathbb{E}_{\mathbf{x}_u \sim p(\cdot \mid \mathbf{x}_o)}\Big[\mathbb{E}_{\mathbf{z} \sim q_\psi(\cdot \mid \mathbf{x}_o, \mathbf{x}_u)}[-\log q_\theta(\mathbf{z} \mid \mathbf{x}_o)]\Big]. \tag{2}$$

We discuss how this is optimized in practice in Section 3.4.

Due to the relationship between negative log-likelihood minimization and KL-divergence minimization [3], we can interpret Equation 2 as minimizing:

$$\mathbb{E}_{\mathbf{x}_u \sim p(\cdot \mid \mathbf{x}_o)}\Big[\mathrm{KL}\left(q_\psi(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u) \mid\mid q_\theta(\mathbf{z} \mid \mathbf{x}_o)\right)\Big]. \tag{3}$$

We can directly minimize the KL-divergence in Equation 3 if it is analytically available between the two posteriors, for instance if both posteriors are Gaussians. However, Equation 2 is more general in that it allows us to use more expressive (e.g., autoregressive) distributions for $q_\theta(\mathbf{z} \mid \mathbf{x}_o)$ with which the KL-divergence cannot be directly computed. This is important given that $p_\psi(\mathbf{z} \mid \mathbf{x}_o)$ is likely to be complex (e.g., multimodal) and not easily captured by a Gaussian (as in Figure 1). Importantly, there is no requirement for $q_\theta(\mathbf{z} \mid \mathbf{x}_o)$ to be reparameterized, which would further limit the class of distributions that can be used. There is a high degree of flexibility in the choice of distribution for the partially observed posterior. Note that this objective does not utilize the decoder.

### 3.3 Connection with Arbitrary Conditioning

While the Posterior Matching objective from Equation 2 and Equation 3 is intuitive, it is not immediately clear how this approach relates back to the arbitrary conditioning objective of maximizing $p(\mathbf{x}_u \mid \mathbf{x}_o)$. We formalize this connection in Theorem 3.1 (see Appendix for proof).

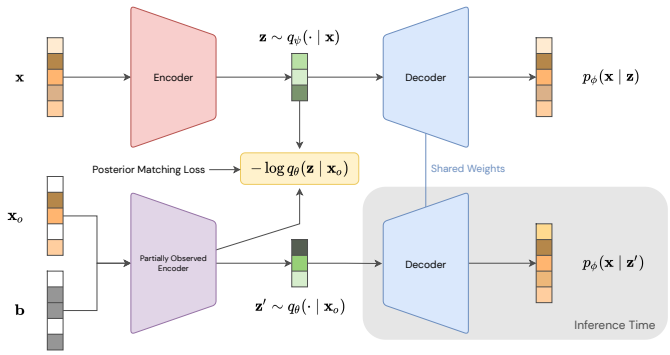



Figure 2: Overview of the Posterior Matching framework. The partially observed encoder can be appended to any existing VAE to enable arbitrary conditioning. At inference time, we can decode samples from $q_\theta(\mathbf{z} \mid \mathbf{x}_o)$ to perform imputation or compute $p(\mathbf{x}_u \mid \mathbf{x}_o)$.

**Theorem 3.1.** *Let $q_\psi(\mathbf{z} \mid \mathbf{x})$ and $p_\phi(\mathbf{x} \mid \mathbf{z})$ be the encoder and decoder, respectively, for some VAE. Additionally, let $q_\theta(\mathbf{z} \mid \mathbf{x}_o)$ be an approximate partially observed posterior. Then minimizing $\mathbb{E}_{\mathbf{x}_u \sim p(\cdot \mid \mathbf{x}_o)} \left[ KL\left(q_\psi(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u) \mid\mid q_\theta(\mathbf{z} \mid \mathbf{x}_o)\right)\right]$ is equivalent to minimizing*

$$\mathbb{E}_{\mathbf{x}_u \sim p(\cdot \mid \mathbf{x}_o)} \Big[ -\log p_{\theta,\phi}(\mathbf{x}_u \mid \mathbf{x}_o) + KL\left(q_\psi(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u) \mid\mid q_\theta(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u)\right) \Big], \tag{4}$$

*with respect to the parameters $\theta$.*

The first term inside the expectation in Equation 4 gives us the explicit connection back to the arbitrary conditioning likelihood $p(\mathbf{x}_u \mid \mathbf{x}_o)$, which is being maximized when minimizing Equation 4. The second term acts as a sort of regularizer by trying to make the partially observed posterior match the VAE posterior when conditioned on all of $\mathbf{x}$ — intuitively, this makes sense as a desirable outcome.

### 3.4 Implementation

A practical training loss follows quickly from Equation 2. For the outer expectation, we do not have access to the true distribution $p(\mathbf{x}_u \mid \mathbf{x}_o)$, but for a given instance $\mathbf{x}$ that has been partitioned into $\mathbf{x}_o$ and $\mathbf{x}_u$, we do have one sample from this distribution, namely $\mathbf{x}_u$. So we approximate this expectation using $\mathbf{x}_u$ as a single sample. This type of single-sample approximation is common with VAEs, e.g., when estimating the ELBO. For the inner expectation, we have access to $q_\psi(\mathbf{z} \mid \mathbf{x})$, which can easily be sampled in order to estimate the expectation. In practice, we generally use a single sample for this as well. This gives us the following Posterior Matching loss:

$$\mathcal{L}_{\text{PM}}(\mathbf{x}, o, \theta, \psi) = -\mathbb{E}_{\mathbf{z} \sim q_\psi(\cdot \mid \mathbf{x})} \left[\log q_\theta(\mathbf{z} \mid \mathbf{x}_o)\right], \tag{5}$$

where $o$ is the set of observed feature indices. During training, $o$ can be randomly sampled from a problem-specific distribution for each minibatch.

Figure 2 provides a visual overview of our approach. In practice, we represent $\mathbf{x}_o$ as a concatenation of $\mathbf{x}$ that has had unobserved features set to zero and a bitmask $\mathbf{b}$ that indicates which features are observed. This representation has been successful in other arbitrary conditioning models [24, 38]. However, this choice is not particularly important to Posterior Matching itself, and alternative representations, such as set embeddings, are valid as well.

As required by VAEs, samples from $q_\psi(\mathbf{z} \mid \mathbf{x})$ will be reparameterized, which means that minimizing $\mathcal{L}_{\text{PM}}$ will influence the parameters of the VAE's encoder in addition to the partially observed posterior network. In some cases, this may be advantageous, as the encoder can be guided towards learning a latent representation that is more conducive to arbitrary conditioning. However, it might also be desirable to train the VAE independently of the partially observed posterior, in which case we can choose to stop gradients on the samples $\mathbf{z} \sim q_\psi(\cdot \mid \mathbf{x})$ when computing $\mathcal{L}_{\text{PM}}$.

Similarly, the partially observed posterior can be trained against an existing pretrained VAE. In this case, the parameters of the VAE's encoder and decoder are frozen, and we only optimize $\mathcal{L}_{\text{PM}}$ with respect to $\theta$. Otherwise, we jointly optimize the VAE's ELBO and $\mathcal{L}_{\text{PM}}$.

We emphasize that there is a high degree of flexibility with the choice of VAE, i.e. we have not imposed any unusual constraints. However, there are some potentially limiting practical considerations that have not been explicitly mentioned yet. First, the training data must be fully observed, as with traditional VAEs, since $\mathcal{L}_{\text{PM}}$ requires sampling $q_\psi(\mathbf{z} \mid \mathbf{x})$. However, given that the base VAE requires fully observed training data anyway, this is generally not a relevant limitation for our purposes. Second, it is *convenient* in practice for the VAE's decoder to be factorized, i.e. $p(\mathbf{x} \mid \mathbf{z}) = \prod_i p(x_i \mid \mathbf{z})$, as this allows us to easily sample from $p(\mathbf{x}_u \mid \mathbf{z})$ (sampling $\mathbf{x}_u$ is less straightforward with other types of decoders). However, it is standard practice to use factorized decoders with VAEs, so this is ordinarily not a concern. We also note that, while useful for easy sampling, a factorized decoder is *not necessary* for optimizing the Posterior Matching loss, which does not incorporate the decoder.

### 3.5 Posterior Matching Beyond Arbitrary Conditioning

The concept of matching VAE posteriors is quite general and has other uses beyond the application of arbitrary conditioning. We consider one such example, which still has ties to arbitrary conditioning, in order to give a flavor for other potential uses.

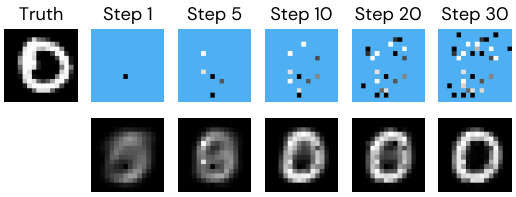



Figure 3: Example acquisitions from our CNN model using the lookahead posteriors (see Section 5.5). Top row shows $\mathbf{x}_o$, and blue pixels are unobserved. Bottom row shows imputation of $\mathbf{x}_u$.

A common application of arbitrary conditioning is *active feature acquisition* [13, 23, 25], where informative features are sequentially acquired on an instance-by-instance basis. In the unsupervised case, the aim is to acquire as few features as possible while maximizing the ability to reconstruct the remaining unobserved features (see Figure 3 for example).

One approach to active feature acquisition is to greedily select the feature that will maximize the expected amount of information to be gained about the currently unobserved features [23, 25]. For VAEs, Ma et al. [25] show that this is equivalent to selecting each feature according to

$$\operatorname*{argmax}_{i \in u} H(\mathbf{z} \mid \mathbf{x}_o) - \mathbb{E}_{x_i \sim p(\cdot \mid \mathbf{x}_o)} \Big[ H(\mathbf{z} \mid \mathbf{x}_o, x_i) \Big] = \operatorname*{argmin}_{i \in u} \mathbb{E}_{x_i \sim p(\cdot \mid \mathbf{x}_o)} \Big[ H(\mathbf{z} \mid \mathbf{x}_o, x_i) \Big]. \tag{6}$$

For certain families of posteriors, such as multivariate Gaussians, the entropies in Equation 6 can be analytically computed. In practice, approximating the expectation in Equation 6 is done via entropies of the posteriors $p^{(i)}(\mathbf{z} \mid \mathbf{x}_o) \equiv \mathbb{E}_{x_i \sim p_{\theta,\phi}(\cdot \mid \mathbf{x}_o)} \Big[ q_\theta(\mathbf{z} \mid \mathbf{x}_o, x_i) \Big]$, where samples from $p_{\theta,\phi}(x_i \mid \mathbf{x}_o)$ are produced by first sampling $\mathbf{z} \sim q_\theta(\cdot \mid \mathbf{x}_o)$ and then passing $\mathbf{z}$ through the VAE's decoder $p_\phi(x_i \mid \mathbf{z})$ (we call $p^{(i)}(\mathbf{z} \mid \mathbf{x}_o)$ the "lookahead" posterior for feature $i$, since it is obtained by imagining what the posterior will look like after one acquisition into the future). Hence, computing the resulting entropies requires one network evaluation per sample of $x_i$ to encode $z$, for $i \in u$. Thus, if using $k$ samples for each $x_i$, each greedy step will be $\Omega(k \cdot |u|)$, which may be prohibitive in high dimensions.

In analogous fashion to the Posterior Matching approach that has already been discussed, we can train a neural network to directly output the lookahead posteriors for all features at once. The Posterior Matching loss in this case is

$$\mathcal{L}_{\text{PM-Lookahead}}(\mathbf{x}, o, u, \omega, \theta, \phi) = \sum_{i \in u} \mathbb{E}_{x_i \sim p_{\theta,\phi}(\cdot \mid \mathbf{x}_o)} \Bigg[ \mathbb{E}_{\mathbf{z} \sim q_\theta(\cdot \mid \mathbf{x}_o, x_i)} \Big[ -\log q_\omega^{(i)}(\mathbf{z} \mid \mathbf{x}_o) \Big] \Bigg], \tag{7}$$

where $\omega$ is the parameters of the lookahead posterior network. In practice, we train a single shared network with a final output layer that outputs the parameters of all $q_\omega^{(i)}(\mathbf{z} \mid \mathbf{x}_o)$. Note that given the distributions $q_\omega^{(i)}(\mathbf{z} \mid \mathbf{x}_o)$ for all $i$, computing the greedy acquisition choice consists of doing a forward evaluation of our network, then choosing the feature $i \in u$ such that the entropy of $q_\omega^{(i)}(\mathbf{z} \mid \mathbf{x}_o)$ is minimized. In other words, we may bypass the individual samples of $x_i$, and use a single shared network for a faster acquisition step. In this setting, we let $q_\omega^{(i)}(\mathbf{z} \mid \mathbf{x}_o)$ be a multivariate Gaussian so that the entropy computation is trivial. See Appendix for a diagram of the entire process. This use of Posterior Matching leads to large improvements in the computational efficiency of greedy active feature acquisition (demonstrated empirically in Section 5.5).

# 4 Prior Work

A variety of approaches to arbitrary conditioning have been previously proposed. ACE is an autoregressive, energy-based method that is the current state-of-the-art for arbitrary conditional likelihood estimation and imputation, although it can be computationally intensive for very high dimensional data [38]. ACFlow is a variant of normalizing flows that can give analytical arbitrary conditional likelihoods [24]. Several other methods, including Sum-Product Networks [32, 6], Neural Conditioner [2], and Universal Marginalizer [10], also have the ability to estimate conditional likelihoods.

Rezende et al. [34] were among the first to suggest that VAEs can be used for imputation. More recently, VAEAC was proposed as a VAE variant designed for arbitrary conditioning [18]. Unlike Posterior Matching, VAEAC is not a general framework and cannot be used with typical pretrained VAEs. EDDI is a VAE-based approach to active feature acquisition and relies on arbitrary conditioning [25]. The authors introduce a "Partial VAE" in order to perform the arbitrary conditioning, which, similarly to Posterior Matching, tries to model $p(\mathbf{z} \mid \mathbf{x}_o)$. Unlike Posterior Matching, they do this by maximizing a variational lower bound on $p(\mathbf{x}_o)$ using a partial inference network $q(\mathbf{z} \mid \mathbf{x}_o)$ (there is no standard VAE posterior $q(\mathbf{z} \mid \mathbf{x})$ in EDDI). Gong et al. [13] use a similar approach that is based on the Partial VAE of EDDI. The major drawback of these methods is that, unlike with Posterior Matching, $q(\mathbf{z} \mid \mathbf{x}_o)$ must be reparameterizable in order to optimize the lower bound (the authors use a diagonal Gaussian). Thus, certain more expressive distributions (e.g., autoregressive) cannot be used. Additionally, these methods cannot be applied to existing VAEs. The methods of Ipsen et al. [17] and Collier et al. [9] are also similar to EDDI, where the former optimizes an approximation of $p(\mathbf{x}_o, \mathbf{b})$ and the latter optimizes a lower bound on $p(\mathbf{x}_o \mid \mathbf{b})$. Ipsen et al. [17] also focuses on imputation for data that is missing "not at random", a setting that is outside the focus of our work.

There are also several works that have considered learning to identify desirable regions in latent spaces. Engel et al. [11] start from a pretrained VAE, but then train a separate GAN [14] with special regularizers to do their conditioning. They only condition on binary vectors, $\mathbf{y}$, that correspond to a small number of predefined attributes, whereas we allow for conditioning on arbitrary subsets of continuous features $\mathbf{x}_o$ (a more complicated conditioning space). Also, their resulting GAN does not make the likelihood $q(\mathbf{z} \mid \mathbf{y})$ available, whereas Posterior Matching directly (and flexibly) models $q(\mathbf{z} \mid \mathbf{x}_o)$, which may be useful for downstream tasks (e.g. Section 5.5) and likelihood evaluation (see Appendix). Furthermore, Posterior Matching trains directly through KL, without requiring an additional critic. Whang et al. [40] learn conditional distributions, but not *arbitrary* conditional distributions (a much harder problem). They also consider normalizing flow models, which are limited to invertible architectures with tractable Jacobian determinants and latent spaces that have the same dimensionality as the data (unlike VAEs). Cannella et al. [7] similarly do conditional sampling from a model of the joint distribution, but are also restricted to normalizing flow architectures and require a more expensive MCMC procedure for sampling.

# 5 Experiments

In order to empirically test Posterior Matching, we apply it to a variety of VAEs aimed at different tasks. We find that our models are able to match or surpass the performance of previous specialized VAE methods. All experiments were conducted using JAX [5] and the DeepMind JAX Ecosystem [1]. Code is available at `https://github.com/lupalab/posterior-matching`.

Our results are dependent on the choice of VAE, and the particular VAEs used in our experiments were *not* the product of extensive comparisons and did not undergo thorough hyperparameter tuning — that is not the focus of this work. With more carefully selected or tuned VAEs, and as new VAEs continue to be developed, we can expect Posterior Matching's downstream performance to improve accordingly on any given task. We emphasize that our experiments span a *diverse set of task, domains, and types of VAE*, wherein Posterior Matching was effective.

## 5.1 MNIST

In this first experiment, our goal is to demonstrate that Posterior Matching replicates the intuition depicted in Figure 1. We do this by training a convolutional VAE with Posterior Matching on the MNIST dataset. The latent space of this VAE is then mapped to two dimensions with UMAP [27] and visualized in Figure 4. In the figure, black points represent samples from $q_\theta(\mathbf{z} \mid \mathbf{x}_o)$, and for select

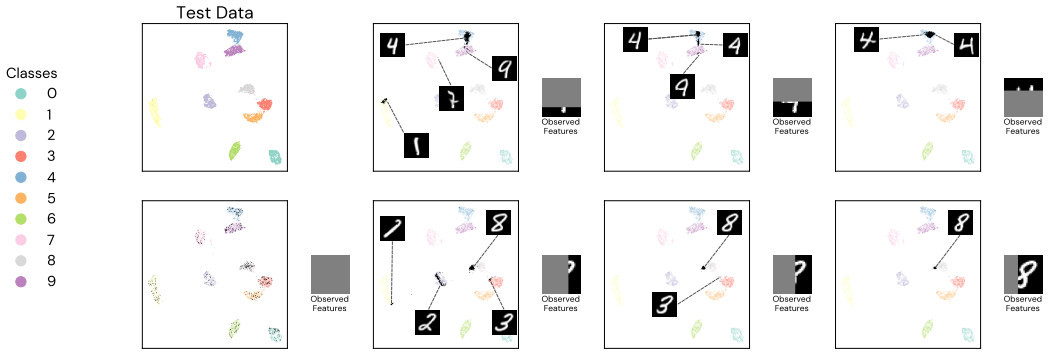



Figure 4: UMAP [27] visualization of the latent space of a VAE trained on MNIST. Black dots represent samples from the distribution $q_\theta(\mathbf{z} \mid \mathbf{x}_o)$ learned via Posterior Matching. Images of $\mathbf{x}_o$ are to the right of their respective plot. Samples from each mode are decoded and shown.

Table 1: Peak signal-to-noise ratio (PSNR) and precision/recall scores [35] for image inpaintings. We report mean and standard deviation across five evaluations with different random masks. VAEAC and ACFlow results are taken from Li et al. [24]. Higher is better for all metrics.

| | MNIST | | | OMNIGLOT | | | CELEBA | | |
|---|---|---|---|---|---|---|---|---|---|
| | PSNR | Precision | Recall | PSNR | Precision | Recall | PSNR | Precision | Recall |
| VDVAE + PM (ours) | **21.603 ± 0.022** | **0.996** | **0.996** | 18.256 ± 0.038 | **0.995** | **0.994** | **27.190 ± 0.049** | **0.995** | **0.995** |
| VQ-VAE + PM (ours) | 19.981 ± 0.021 | 0.989 | 0.993 | 17.954 ± 0.046 | 0.979 | 0.973 | 25.531 ± 0.036 | 0.982 | 0.984 |
| VAEAC | 19.613 ± 0.042 | 0.877 | 0.975 | 17.693 ± 0.023 | 0.525 | 0.926 | 23.656 ± 0.027 | 0.966 | 0.967 |
| ACFlow | 17.349 ± 0.018 | 0.945 | 0.984 | 15.572 ± 0.031 | 0.962 | 0.971 | 22.393 ± 0.040 | 0.970 | 0.988 |
| ACFlow+BG | 20.828 ± 0.031 | 0.947 | 0.983 | **18.838 ± 0.009** | 0.967 | 0.970 | 25.723 ± 0.020 | 0.964 | 0.987 |

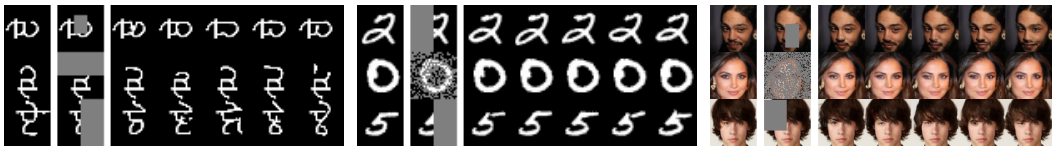
samples, the corresponding reconstruction is shown. The encoded test data is shown, colored by true class label, to highlight which regions correspond to which digits. We see that the experimental results nicely replicate our earlier intuitions — the learned distribution $q_\theta(\mathbf{z} \mid \mathbf{x}_o)$ puts probability mass only in parts of the latent space that correspond to plausible digits based on what is observed and successfully captures multimodal distributions (see the second column in Figure 4).

### 5.2 Image Inpainting

One practical application of arbitrary conditioning is image inpainting, where only part of an image is observed and we want to fill in the missing pixels with visually coherent imputations. As with prior works [18, 24], we assume pixels are missing completely at random. We test Posterior Matching as an approach to this task by pairing it with both discrete and hierarchical VAEs.

**Vector Quantized-VAEs** We first consider VQ-VAE [30], a type of VAE that is known to work well with images. VQ-VAE differs from the typical VAE with its use of a discrete latent space. That is, each latent code is a grid of discrete indices rather than a vector of continuous values. Because the latent space is discrete, Oord et al. [30] model the prior distribution with a PixelCNN [29, 36] after training the VQ-VAE. We similarly use a conditional PixelCNN to model $q_\theta(\mathbf{z} \mid \mathbf{x}_o)$. First, a convolutional network maps $\mathbf{x}_o$ to a vector, and that vector is then used as a conditioning input to the PixelCNN. More architecture and training details can be found in the Appendix. We train VQ-VAEs with Posterior Matching for the MNIST, OMNIGLOT, and CELEBA datasets. Table 1 reports peak signal-to-noise ratio (PSNR) and precision/recall [35] for inpaintings produced by our model. We find that Posterior Matching with VQ-VAE consistently achieves better precision/recall scores than previous models while having comparable PSNR.

**Hierarchical VAEs** Hierarchical VAEs [22, 37, 39] are a powerful extension of traditional VAEs that allow for more expressive priors and posteriors by partitioning the latent variables into subsets $\mathbf{z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_L\}$. A hierarchy is then created by factorizing the prior $p(\mathbf{z}) = \prod_i p(\mathbf{z}_i \mid \mathbf{z}_{<i})$ and posterior $q(\mathbf{z} \mid \mathbf{x}) = \prod_i q(\mathbf{z}_i \mid \mathbf{z}_{<i}, \mathbf{x})$. These models have demonstrated impressive performance on images and can even outperform autoregressive models [8]. Posterior Matching can be



(a) OMNIGLOT (b) MNIST (c) CELEBA

Figure 5: Example image inpaintings from VDVAE with Posterior Matching. First two columns on the left are $\mathbf{x}$ and $\mathbf{x}_o$. Inpainting samples are on the right.

Table 2: Test normalized root-mean-square error (NRMSE) and arbitrary conditional log-likelihoods (LL) for UCI datasets. Lower is better for NRMSE, and higher is better for LL. Results for methods other than Posterior Matching are reproduced from Strauss and Oliva [38]. Mean and standard deviation are reported over 5 random observed masks for each instance.

| | POWER | | GAS | | HEPMASS | | MINIBOONE | | BSDS | |
|---|---|---|---|---|---|---|---|---|---|---|
| | NRMSE | LL | NRMSE | LL | NRMSE | LL | NRMSE | LL | NRMSE | LL |
| Posterior Matching | $0.834 \pm 0.001$ | $0.246 \pm 0.002$ | $0.330 \pm 0.013$ | $5.964 \pm 0.005$ | $0.857 \pm 0.000$ | $-8.963 \pm 0.007$ | $0.450 \pm 0.002$ | $-3.116 \pm 0.175$ | $0.573 \pm 0.000$ | $77.488 \pm 0.012$ |
| VAEAC | $0.880 \pm 0.001$ | $-0.042 \pm 0.002$ | $0.574 \pm 0.033$ | $2.418 \pm 0.006$ | $0.896 \pm 0.001$ | $-10.082 \pm 0.010$ | $0.462 \pm 0.002$ | $-3.452 \pm 0.067$ | $0.615 \pm 0.000$ | $74.850 \pm 0.005$ |
| ACE | $0.828 \pm 0.002$ | $0.631 \pm 0.002$ | $0.335 \pm 0.027$ | $9.643 \pm 0.005$ | $0.830 \pm 0.001$ | $-3.859 \pm 0.005$ | $0.432 \pm 0.003$ | $0.310 \pm 0.054$ | $0.525 \pm 0.000$ | $86.701 \pm 0.008$ |
| ACE Proposal | $0.828 \pm 0.002$ | $0.583 \pm 0.003$ | $0.312 \pm 0.033$ | $9.484 \pm 0.005$ | $0.832 \pm 0.001$ | $-4.417 \pm 0.005$ | $0.436 \pm 0.004$ | $-0.241 \pm 0.056$ | $0.535 \pm 0.000$ | $85.228 \pm 0.009$ |
| ACFlow | $0.877 \pm 0.001$ | $0.561 \pm 0.003$ | $0.567 \pm 0.050$ | $8.086 \pm 0.010$ | $0.909 \pm 0.000$ | $-8.197 \pm 0.008$ | $0.478 \pm 0.004$ | $-0.972 \pm 0.022$ | $0.603 \pm 0.000$ | $81.827 \pm 0.007$ |
| ACFlow+BG | $0.833 \pm 0.002$ | $0.528 \pm 0.003$ | $0.369 \pm 0.016$ | $7.593 \pm 0.011$ | $0.861 \pm 0.001$ | $-6.833 \pm 0.006$ | $0.442 \pm 0.001$ | $-1.098 \pm 0.032$ | $0.572 \pm 0.000$ | $81.399 \pm 0.008$ |

naturally applied to hierarchical VAEs, where the partially observed posterior is represented as $q(\mathbf{z} \mid \mathbf{x}_o) = \prod_i q(\mathbf{z}_i \mid \mathbf{z}_{<i}, \mathbf{x}_o)$. We adopt the Very Deep VAE (VDVAE) architecture used by Child [8] and extend it to include the partially observed posterior (see Appendix for training and architecture details). We note that due to our hardware constraints, we trained smaller models and for fewer iterations than Child [8]. Inpainting results for our VDVAE models are given in Table 1. We see that they achieve better precision/recall scores than the VQ-VAE models and, unlike VQ-VAE, are able to attain better PSNR than ACFlow for MNIST and CELEBA. Figure 5 shows some example inpaintings, and additional samples are provided in the Appendix. The fact that we see better downstream performance when using VDVAE than when using VQ-VAE is illustrative of Posterior Matching's ability to admit easy performance gains by simply switching to a more powerful base VAE.

### 5.3 Real-valued Datasets

We evaluate Posterior Matching on real-valued tabular data, specifically the benchmark UCI repository datasets from Papamakarios et al. [31]. We follow the experimental setup used by Li et al. [24] and Strauss and Oliva [38]. In these experiments, we train basic VAE models while simultaneously learning the partially observed posterior. Given the flexibility that Posterior Matching affords, we use an autoregressive distribution for $q_\theta(\mathbf{z} \mid \mathbf{x}_o)$. Further details can be found in the Appendix.

Table 2 reports the arbitrary conditional log-likelihoods and normalized root-mean-square error (NRMSE) of imputations for our models (with features missing completely at random). Likelihoods are computed using an importance sampling estimate (see Appendix for details). We primarily compare to VAEAC as a baseline in the VAE family, however we also provide results for ACE and ACFlow for reference. We see that Posterior Matching is able to consistently produce more accurate imputations and higher likelihoods than VAEAC. While our models don't match the likelihoods achieved by ACE and ACFlow, Posterior Matching is comparable to them for imputation NRMSE.

### 5.4 Partially Observed Clustering

Probabilistic clustering often views cluster assignments as a latent variable. Thus, when applying Posterior Matching in this setting, we may perform "partially observed" clustering, which clusters instances based on a subset of observed features. We consider VaDE, which uses a mixture of Gaussians as the prior, allowing it to do unsupervised clustering by treating each Gaussian component as one of the clusters [19]. Despite differences in how VaDE is trained compared to a classic VAE, training a partially observed encoder via Posterior Matching remains exactly the same.

We train models on both MNIST and FASHION MNIST (see Appendix for experimental details). Figure 6 shows the clustering accuracy of these models as the percentage of (randomly selected)

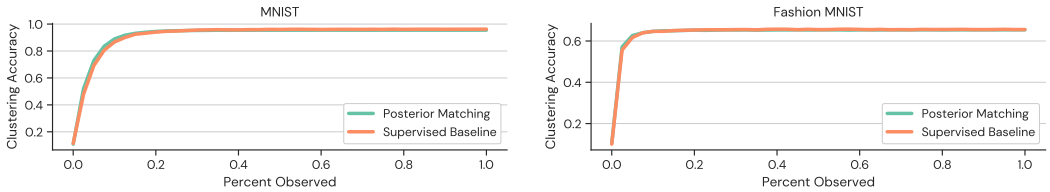



Figure 6: Partially observed clustering accuracy achieved by Posterior Matching with VaDE.

observed features changes. As a baseline, we train a supervised model where the labels are the cluster predictions from the VaDE model when all of the features are observed. We see that Posterior Matching is able to match the performance of the baseline, and even slightly outperform it for low percentages of observed features. Unlike the supervised approach, Posterior Matching has the advantage of being generative.

### 5.5 Very Fast Greedy Feature Acquisition

As discussed in Section 3.5, we can use Posterior Matching outside of the specific task of arbitrary conditioning. Here, we consider the problem of greedy active feature acquisition. We train a VAE with a Posterior Matching network that outputs the lookahead posteriors described in Section 3.5, using the loss in Equation 7. Note that we are also still using Posterior Matching in order to learn $q_\theta(\mathbf{z} \mid \mathbf{x}_o)$ and therefore to produce reconstructions. Training details can be found in the Appendix.

We consider the MNIST dataset and compare to EDDI as a baseline, using the authors' publicly available code. We downscale images to $16 \times 16$ since EDDI has difficulty scaling to high-dimensional data. We also only evaluate on the first 1000 instances of the MNIST test set, as the EDDI code was very slow when computing the greedy acquisition policy. EDDI also uses a particular architecture that is not compatible with convolutions. Thus we train a MLP-based VAE on flattened images in order to make a fair comparison. However, since Posterior Matching does not place any limitations on the type of VAE being used, we also train a convolutional version. For our models, we greedily select the feature to acquire using the more expensive sampling-based approach (similar to EDDI) as well as with the lookahead posteriors (which requires no sampling). In both cases, imputations are computed with an expectation over 50 latent codes, as is done for EDDI. An example acquisition trajectory is shown in Figure 3.

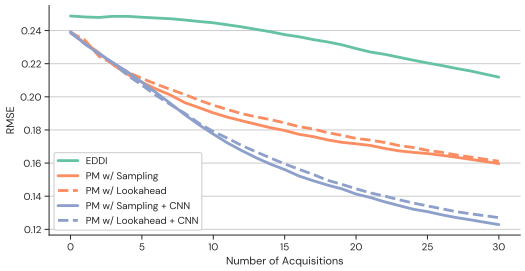



Figure 7: Average RMSE of reconstructions for greedy active feature acquisition.

Figure 7 presents the root-mean-square error, averaged across the test instances, when imputing $\mathbf{x}_u$ with different numbers of acquired features. We see that our models are able to achieve lower error than EDDI. We also see that acquiring based on the lookahead posteriors incurs only a minimal increase in error compared to the sampling-based method, despite being far more efficient. Computing the greedy choice with our model using the sampling-based approach takes 68 ms $\pm$ 917 $\mu$s (for a single acquisition on CPU). Using the lookahead posteriors, the time is only 310 $\mu$s $\pm$ 15.3 $\mu$s, **a roughly 219x speedup**.

## 6 Conclusions

We have presented an elegant and general framework, called Posterior Matching, that allows VAEs to perform arbitrary conditioning. That is, we can take an existing VAE that only models the joint distribution $p(\mathbf{x})$ and train an additional model that, when combined with the VAE, is able to assess any likelihood $p(\mathbf{x}_u \mid \mathbf{x}_o)$ for arbitrary subsets of unobserved features $\mathbf{x}_u$ and observed features $\mathbf{x}_o$.

We applied this approach to a variety of VAEs for a multitude of different tasks. We found that Posterior Matching outperforms previous specialized VAEs for arbitrary conditioning with tabular data and for image inpainting. Importantly, we find that one can switch to a more powerful base VAE and get immediate improvements in downstream arbitrary conditioning performance "for free," without making changes to Posterior Matching itself. We can also use Posterior Matching to perform clustering based on partially observed inputs and to improve the efficiency of greedy active feature acquisition by several orders of magnitude at negligible cost to performance.

With this work, we hope to make arbitrary conditioning more widely accessible. Arbitrary conditioning no longer requires specialized methods, but can instead be achieved by applying one general framework to common VAEs. As advances are made in VAEs for joint density estimation, we can expect to immediately reap the rewards for arbitrary conditioning.

## Acknowledgments and Disclosure of Funding

We would like to thank Google's TPU Research Cloud program for providing free access to TPUs. This research was partly funded by NSF grant IIS2133595 and by NIH 1R01AA02687901A1.

## References

[1] I. Babuschkin, K. Baumli, A. Bell, S. Bhupatiraju, J. Bruce, P. Buchlovsky, D. Budden, T. Cai, A. Clark, I. Danihelka, C. Fantacci, J. Godwin, C. Jones, T. Hennigan, M. Hessel, S. Kapturowski, T. Keck, I. Kemaev, M. King, L. Martens, V. Mikulik, T. Norman, J. Quan, G. Papamakarios, R. Ring, F. Ruiz, A. Sanchez, R. Schneider, E. Sezener, S. Spencer, S. Srinivasan, W. Stokowiec, and F. Viola. The DeepMind JAX Ecosystem, 2020. URL `http://github.com/deepmind`.

[2] M. I. Belghazi, M. Oquab, Y. LeCun, and D. Lopez-Paz. Learning about an exponential amount of conditional distributions. *arXiv preprint arXiv:1902.08401*, 2019.

[3] C. M. Bishop. Pattern recognition. *Machine learning*, 128(9), 2006.

[4] S. R. Bowman, L. Vilnis, O. Vinyals, A. M. Dai, R. Jozefowicz, and S. Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

[5] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL `http://github.com/google/jax`.

[6] C. J. Butz, J. S. Oliveira, A. E. dos Santos, and A. L. Teixeira. Deep convolutional sum-product networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 3248–3255, 2019.

[7] C. Cannella, M. Soltani, and V. Tarokh. Projected latent markov chain monte carlo: Conditional sampling of normalizing flows. *arXiv preprint arXiv:2007.06140*, 2020.

[8] R. Child. Very deep vaes generalize autoregressive models and can outperform them on images. *arXiv preprint arXiv:2011.10650*, 2020.

[9] M. Collier, A. Nazabal, and C. K. Williams. Vaes in the presence of missing data. *arXiv preprint arXiv:2006.05301*, 2020.

[10] L. Douglas, I. Zarov, K. Gourgoulias, C. Lucas, C. Hart, A. Baker, M. Sahani, Y. Perov, and S. Johri. A universal marginalizer for amortized inference in generative models. *arXiv preprint arXiv:1711.00695*, 2017.

[11] J. Engel, M. Hoffman, and A. Roberts. Latent constraints: Learning to generate conditionally from unconditional generative models. *arXiv preprint arXiv:1711.05772*, 2017.

[12] H. Fu, C. Li, X. Liu, J. Gao, A. Celikyilmaz, and L. Carin. Cyclical annealing schedule: A simple approach to mitigating kl vanishing. *arXiv preprint arXiv:1903.10145*, 2019.

[13] W. Gong, S. Tschiatschek, S. Nowozin, R. E. Turner, J. M. Hernández-Lobato, and C. Zhang. Icebreaker: Element-wise efficient information acquisition with a bayesian deep latent gaussian model. 2019.

[14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[15] K. Gregor, I. Danihelka, A. Graves, D. Rezende, and D. Wierstra. Draw: A recurrent neural network for image generation. In *International Conference on Machine Learning*, pages 1462–1471. PMLR, 2015.

[16] K. Han, H. Wen, J. Shi, K.-H. Lu, Y. Zhang, D. Fu, and Z. Liu. Variational autoencoder: An unsupervised model for encoding and decoding fmri activity in visual cortex. *NeuroImage*, 198: 125–136, 2019. ISSN 1053-8119. doi: https://doi.org/10.1016/j.neuroimage.2019.05.039. URL `https://www.sciencedirect.com/science/article/pii/S1053811919304318`.

[17] N. B. Ipsen, P.-A. Mattei, and J. Frellsen. not-miwae: Deep generative modelling with missing not at random data. *arXiv preprint arXiv:2006.12871*, 2020.

[18] O. Ivanov, M. Figurnov, and D. Vetrov. Variational autoencoder with arbitrary conditioning. *arXiv preprint arXiv:1806.02382*, 2018.

[19] Z. Jiang, Y. Zheng, H. Tan, B. Tang, and H. Zhou. Variational deep embedding: An unsupervised and generative approach to clustering. *arXiv preprint arXiv:1611.05148*, 2016.

[20] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

[22] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling. Improved variational inference with inverse autoregressive flow. *Advances in neural information processing systems*, 29, 2016.

[23] Y. Li and J. Oliva. Active feature acquisition with generative surrogate models. In *International Conference on Machine Learning*, pages 6450–6459. PMLR, 2021.

[24] Y. Li, S. Akbar, and J. Oliva. Acflow: Flow models for arbitrary conditional likelihoods. In *International Conference on Machine Learning*, pages 5831–5841. PMLR, 2020.

[25] C. Ma, S. Tschiatschek, K. Palla, J. M. Hernández-Lobato, S. Nowozin, and C. Zhang. Eddi: Efficient dynamic discovery of high-value information with partial vae. *arXiv preprint arXiv:1809.11142*, 2018.

[26] L. Maaløe, C. K. Sønderby, S. K. Sønderby, and O. Winther. Auxiliary deep generative models. In *International conference on machine learning*, pages 1445–1453. PMLR, 2016.

[27] L. McInnes, J. Healy, and J. Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*, 2018.

[28] C. Nash and C. Durkan. Autoregressive energy machines. In *International Conference on Machine Learning*, pages 1735–1744. PMLR, 2019.

[29] A. v. d. Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu. Conditional image generation with pixelcnn decoders. *arXiv preprint arXiv:1606.05328*, 2016.

[30] A. v. d. Oord, O. Vinyals, and K. Kavukcuoglu. Neural discrete representation learning. *arXiv preprint arXiv:1711.00937*, 2017.

[31] G. Papamakarios, T. Pavlakou, and I. Murray. Masked autoregressive flow for density estimation. *arXiv preprint arXiv:1705.07057*, 2017.

[32] H. Poon and P. Domingos. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690. IEEE, 2011.

[33] A. Razavi, A. van den Oord, and O. Vinyals. Generating diverse high-fidelity images with vq-vae-2. In *Advances in neural information processing systems*, pages 14866–14876, 2019.

[34] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International conference on machine learning*, pages 1278–1286. PMLR, 2014.

[35] M. S. Sajjadi, O. Bachem, M. Lucic, O. Bousquet, and S. Gelly. Assessing generative models via precision and recall. *arXiv preprint arXiv:1806.00035*, 2018.

[36] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.

[37] C. K. Sønderby, T. Raiko, L. Maaløe, S. K. Sønderby, and O. Winther. Ladder variational autoencoders. *Advances in neural information processing systems*, 29, 2016.

[38] R. R. Strauss and J. B. Oliva. Arbitrary conditional distributions with energy. *arXiv preprint arXiv:2102.04426*, 2021.

[39] A. Vahdat and J. Kautz. Nvae: A deep hierarchical variational autoencoder. *Advances in Neural Information Processing Systems*, 33:19667–19679, 2020.

[40] J. Whang, E. Lindgren, and A. Dimakis. Composing normalizing flows for inverse problems. In *International Conference on Machine Learning*, pages 11158–11169. PMLR, 2021.

## Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 3.4.
   (c) Did you discuss any potential negative societal impacts of your work? [No] We do not believe this work has any particular negative societal impacts.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] We include std. dev. for relevant experiments (e.g. Table 2).
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

# A Proof of Theorem 3.1

*Proof.* Starting from our original objective

$$\underset{\theta}{\operatorname{argmin}}\, \mathbb{E}_{\mathbf{x}_u \sim p(\cdot \mid \mathbf{x}_o)} \left[ \mathrm{KL}\left( q_\psi(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u) \mid\mid q_\theta(\mathbf{z} \mid \mathbf{x}_o) \right) \right], \tag{8}$$

we proceed as follows:

$$= \underset{\theta}{\operatorname{argmin}}\, \mathbb{E}_{\mathbf{x}_u \sim p(\cdot \mid \mathbf{x}_o)} \left[ \int q_\psi(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u) \log \frac{q_\psi(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u)}{q_\theta(\mathbf{z} \mid \mathbf{x}_o)} \, \mathrm{d}\mathbf{z} \right] \tag{9}$$

$$= \underset{\theta}{\operatorname{argmin}}\, \mathbb{E}_{\mathbf{x}_u \sim p(\cdot \mid \mathbf{x}_o)} \left[ \int q_\psi(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u) \log \frac{q_\psi(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u)}{\frac{p_\phi(\mathbf{x}_u \mid \mathbf{z}, \mathbf{x}_o) q_\theta(\mathbf{z} \mid \mathbf{x}_o)}{p_\phi(\mathbf{x}_u \mid \mathbf{z}, \mathbf{x}_o)}} \, \mathrm{d}\mathbf{z} \right] \tag{10}$$

$$= \underset{\theta}{\operatorname{argmin}}\, \mathbb{E}_{\mathbf{x}_u \sim p(\cdot \mid \mathbf{x}_o)} \left[ \int q_\psi(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u) \log \frac{q_\psi(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u)}{\frac{p_{\theta,\phi}(\mathbf{x}_u, \mathbf{z} \mid \mathbf{x}_o)}{p_\phi(\mathbf{x}_u \mid \mathbf{z}, \mathbf{x}_o)}} \, \mathrm{d}\mathbf{z} \right] \tag{11}$$

$$= \underset{\theta}{\operatorname{argmin}}\, \mathbb{E}_{\mathbf{x}_u \sim p(\cdot \mid \mathbf{x}_o)} \left[ \int q_\psi(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u) \log \frac{q_\psi(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u)}{\frac{p_{\theta,\phi}(\mathbf{x}_u \mid \mathbf{x}_o) q_\theta(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u)}{p_\phi(\mathbf{x}_u \mid \mathbf{z}, \mathbf{x}_o)}} \, \mathrm{d}\mathbf{z} \right] \tag{12}$$

$$= \underset{\theta}{\operatorname{argmin}}\, \mathbb{E}_{\mathbf{x}_u \sim p(\cdot \mid \mathbf{x}_o)} \left[ \int q_\psi(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u) \log \frac{p_\phi(\mathbf{x}_u \mid \mathbf{z}, \mathbf{x}_o) q_\psi(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u)}{p_{\theta,\phi}(\mathbf{x}_u \mid \mathbf{x}_o) q_\theta(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u)} \, \mathrm{d}\mathbf{z} \right] \tag{13}$$

$$= \underset{\theta}{\operatorname{argmin}}\, \mathbb{E}_{\mathbf{x}_u \sim p(\cdot \mid \mathbf{x}_o)} \left[ \int q_\psi(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u) \left( \log \frac{q_\psi(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u)}{q_\theta(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u)} + \log p_\phi(\mathbf{x}_u \mid \mathbf{z}, \mathbf{x}_o) - \log p_{\theta,\phi}(\mathbf{x}_u \mid \mathbf{x}_o) \right) \mathrm{d}\mathbf{z} \right] \tag{14}$$

$$= \underset{\theta}{\operatorname{argmin}}\, \mathbb{E}_{\mathbf{x}_u \sim p(\cdot \mid \mathbf{x}_o)} \left[ -\log p_{\theta,\phi}(\mathbf{x}_u \mid \mathbf{x}_o) + \int q_\psi(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u) \left( \log \frac{q_\psi(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u)}{q_\theta(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u)} + \log p_\phi(\mathbf{x}_u \mid \mathbf{z}, \mathbf{x}_o) \right) \mathrm{d}\mathbf{z} \right] \tag{15}$$

$$= \underset{\theta}{\operatorname{argmin}}\, \mathbb{E}_{\mathbf{x}_u \sim p(\cdot \mid \mathbf{x}_o)} \left[ -\log p_{\theta,\phi}(\mathbf{x}_u \mid \mathbf{x}_o) + \mathrm{KL}\Big( q_\psi(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u) \mid\mid q_\theta(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u) \Big) + \mathbb{E}_{\mathbf{z} \sim q_\psi(\cdot \mid \mathbf{x}_o, \mathbf{x}_u)} \left[ \log p_\phi(\mathbf{x}_u \mid \mathbf{z}, \mathbf{x}_o) \right] \right] \tag{16}$$

$$= \underset{\theta}{\operatorname{argmin}}\, \mathbb{E}_{\mathbf{x}_u \sim p(\cdot \mid \mathbf{x}_o)} \left[ -\log p_{\theta,\phi}(\mathbf{x}_u \mid \mathbf{x}_o) + \mathrm{KL}\Big( q_\psi(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u) \mid\mid q_\theta(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u) \Big) \right]. \tag{17}$$

In the final step, we can ignore the term $\mathbb{E}_{\mathbf{z} \sim q_\psi(\cdot \mid \mathbf{x}_o, \mathbf{x}_u)} \left[ \log p_\phi(\mathbf{x}_u \mid \mathbf{z}, \mathbf{x}_o) \right]$ since it does not depend on $\theta$. □

# B Likelihood Estimation

Often, it is of interest to evaluate likelihoods with generative models. In the VAE setting, the joint likelihood is frequently estimated with importance sampling:

$$p_{\psi,\phi}(\mathbf{x}) = \mathbb{E}_{\mathbf{z} \sim q_\psi(\cdot \mid \mathbf{x})} \left[ \frac{p_\phi(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})}{q_\psi(\mathbf{z} \mid \mathbf{x})} \right]. \tag{18}$$

With Posterior Matching, we can estimate arbitrary conditional likelihoods by additionally estimating

$$p_{\theta,\phi}(\mathbf{x}_o) = \mathbb{E}_{\mathbf{z} \sim q_\theta(\cdot \mid \mathbf{x}_o)} \left[ \frac{p_\phi(\mathbf{x}_o \mid \mathbf{z}) p(\mathbf{z})}{q_\theta(\mathbf{z} \mid \mathbf{x}_o)} \right] \tag{19}$$

$$= \mathbb{E}_{\mathbf{z} \sim q_\theta(\cdot \mid \mathbf{x}_o)} \left[ \frac{\int p_\phi(\mathbf{x} = (\mathbf{x}_o, \mathbf{x}_u) \mid \mathbf{z}) \, \mathrm{d}\mathbf{x}_u \, p(\mathbf{z})}{q_\theta(\mathbf{z} \mid \mathbf{x}_o)} \right] \tag{20}$$

in order to obtain $p_{\theta,\psi,\phi}(\mathbf{x}_u \mid \mathbf{x}_o) = p_{\psi,\phi}(\mathbf{x}) / p_{\theta,\phi}(\mathbf{x}_o)$. This is the estimator we use in our experiments that report likelihoods. Note that if the decoder is factorized (which it always is in our experiments), then we can write Equation 20 as:

$$\mathbb{E}_{\mathbf{z} \sim q_\theta(\cdot \mid \mathbf{x}_o)} \left[ \frac{\prod_{i \in o} p_\phi(x_i \mid \mathbf{z}) p(\mathbf{z})}{q_\theta(\mathbf{z} \mid \mathbf{x}_o)} \right]. \tag{21}$$

If the decoder is not factorized, then the integral in Equation 20 needs to be estimated.

# C Lookahead Posteriors

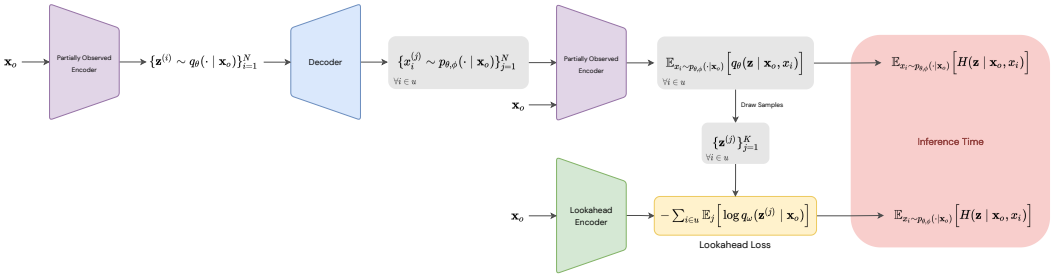



Figure 8: Overview of how Posterior Matching is used to learn "lookahead" posteriors for greedy active feature acquisition. The top path illustrates how samples are produced for use in the lookahead Posterior Matching loss. At inference time, we obtain the entropy of the distributions outputted by the rightmost networks. Taking the top path to obtain these entropies is the more expensive sampling-based approach. Using the learned Lookahead Encoder on the bottom path is the much faster approach that only requires a single network evaluation.

# D Experimental Details

All experiments, except for VDVAE models (see Section D.2.2), were run on a single GeForce GTX 1080 Ti GPU, with the longest running models taking no more than 12 hours to train.

## D.1 Real-valued Datasets

For experiments on the UCI datasets, we use multi-layer perceptrons (MLP) with residual connections for all networks. For these experiments, we found that only optimizing $\mathcal{L}_{\text{PM}}$ with respect to $\theta$ gave the best results (i.e., we stop gradients on samples from $q_\psi(\mathbf{z} \mid \mathbf{x}_o)$ when computing $\mathcal{L}_{\text{PM}}$). We also use a schedule for the $\beta$ coefficient of the KL term in the ELBO, as this helped avoid degeneracy at the start of training. For models where a cyclical schedule [12] was used, the period was 50000 training steps and the schedule began after an initial 1000 steps where $\beta$ was 0 (except for MINIBOONE, where the period is 5000 and the delay is 2000). For models where a monotonic schedule was used, $\beta$ was 0 for the first 30000 training steps and then linearly annealed to 1 at the final training step. We used the autoregressive distribution described in Section D.5, with 256 hidden units and 3 residual blocks, for the partially observed posterior. During training, a small amount of Gaussian noise ($\sigma = 0.001$) is added to each minibatch. We use the Adam [20] optimizer with an initial learning rate of 0.001 and an exponential decay schedule with a rate of 0.9 every 5000 steps (except for MINIBOONE, where the decay is every 1000 steps). During training and test time, observed masks were drawn from a Bernoulli distribution with $p = 0.5$. At test time, likelihoods are computed with the estimator in Appendix B. Additional hyperparameters are given in Table 3.

Table 3: Additional hyperparameters for UCI experiments. Hidden Units, Residual Blocks, and Layer Normalization refer to the VAE encoder/decoder networks.

| | POWER | GAS | HEPMASS | MINIBOONE | BSDS |
|---|---|---|---|---|---|
| Batch Size | 512 | 512 | 512 | 1024 | 1024 |
| Latent Dimension | 16 | 16 | 16 | 32 | 64 |
| Hidden Units | 256 | 256 | 256 | 256 | 256 |
| Residual Blocks | 2 | 2 | 2 | 5 | 5 |
| Layer Normalization | No | No | No | Yes | Yes |
| Training Steps | 200000 | 200000 | 200000 | 22000 | 200000 |
| $\beta$ Schedule | Cyclical | Cyclical | Cyclical | Cyclical | Monotonic |

### D.2 Image Inpainting

#### D.2.1 Vector Quantized-VAEs

We first train VQ-VAE models as described in Oord et al. [30]. The VQ-VAE encoder and decoder are convolutional networks with residual blocks, following the implementation found at `https://github.com/deepmind/dm-haiku/blob/main/examples/vqvae_example.ipynb`. We use two residual blocks with 32 hidden units in the residual layers. We use the exponential moving average version of the VQ-VAE training procedure in order to update the quantized vectors. We use a decay rate of 0.99 and a commitment cost of 0.25. The quantized vectors have a dimensionality of 64. Our decoder outputs a multivariate Gaussian with covarance matrix that is a scalar multiple of the identity matrix, where the scalar is a learnable parameter. All models are trained with the Adam [20] optimizer with learning rate 0.0003. Additional hyperparemeters for each of the datasets can be found in Table 4.

We then train a conditional PixelCNN [29] for each pretrained VQ-VAE model, where the PixelCNN is modeling the partially observed posterior. First, we use the VQ-VAE encoder to obtain the discrete indices $\mathbf{z}$ that correspond to a given $\mathbf{x}$. We then use an encoder network with the same architecture as the VQ-VAE encoder to map $\mathbf{x}_o$ to a 512-dimensional conditioning vector. This vector is then used as conditioning input to the PixelCNN when computing the log-likelihood of $\mathbf{z}$. This gives us $-\log q(\mathbf{z} \mid \mathbf{x}_o)$, which is our usual Posterior Matching loss. We use a learnable embedding lookup as the first layer in the PixelCNN in order to map $\mathbf{z}$ to continuous values. The PixelCNN outputs categorical logits (with as many classes as there are discrete latent vectors). Convolutional layers in the PixelCNN use 128 filters. For CelebA, 12 residual blocks are used, and 8 are used for MNIST and Omniglot. Dropout is used with a rate of 0.5. We use the Adam optimizer with an initial learning rate of 0.0003 and decay of 0.999995 every step. The batch size is 32. Models are trained for 150000 steps. When training and evaluating the PixelCNN models, we randomly generate the masks of observed values according to the same distributions used by Li et al. [24]. When evaluating the models, we compute PSNR by averaging over 10 decoded samples from the partially observed posterior.

Table 4: Dataset-dependent hyperparameters for VQ-VAE.

| | MNIST | OMNIGLOT | CELEBA |
|---|---|---|---|
| Batch Size | 32 | 32 | 64 |
| # of Embeddings | 256 | 256 | 512 |
| Hidden Units | 32 | 32 | 128 |
| Training Steps | 60000 | 60000 | 100000 |

#### D.2.2 Hierarchical VAEs

We use the hierarchical VAE architecture proposed by Child [8] by closely following their original implementation at: `https://github.com/openai/vdvae`. However, we make the following modifications to incorporate the partially observed posterior. First, a second encoder network is added which accepts $\mathbf{x}_o$ as input. We then also add an additional residual block in each top-down block in the decoder to output $q(\mathbf{z}_i \mid \mathbf{z}_{<i}, \mathbf{x}_o)$. These new residual blocks are identical to the ones used to output the posteriors, except they accept the activations from the partially observed encoder instead of the fully observed encoder.

We use Gaussians for the partially observed posteriors, and so we directly compute the KL-divergence between the posterior and partially observed posterior at each level in the hierarchy. In the hierarchical setting, the full Posterior Matching KL-divergence can be computed as

$$\mathrm{KL}\left(q_\psi(\mathbf{z} \mid \mathbf{x}_o, \mathbf{x}_u) \mid\mid q_\theta(\mathbf{z} \mid \mathbf{x}_o)\right) = \sum_{i=1}^{L} \mathbb{E}_{\mathbf{z}_{<i} \sim q_\psi(\cdot \mid \mathbf{x}_o, \mathbf{x}_u)} \left[\mathrm{KL}\left(q_\psi(\mathbf{z}_i \mid \mathbf{z}_{<i}, \mathbf{x}_o, \mathbf{x}_u) \mid\mid q_\theta(\mathbf{z}_i \mid \mathbf{z}_{<i}, \mathbf{x}_o)\right)\right], \tag{22}$$

and so during training we approximate this by simply summing the individual KL terms from all of the levels (this is analogous to how the KL term in the ELBO is computed for VDVAE). We stop gradients in the model such that the Posterior Matching loss is only computed with respect to the parameters of the partially observed encoder and the residual blocks that output the partially observed posteriors (i.e. only the new parameters that we introduced to the model).

We follow the training setup used by Child [8] as well. Our MNIST and OMNIGLOT models have 20 levels in the hierarchy, and convolutions use 192 filters. The models were trained for 500000 steps on 8 TPU-v2 cores, which took about 3 days. Our CELEBA model has 46 levels in the hierarchy, and convolutions use 384 filters. The model was trained for 1000000 steps on 8 TPU-v3 cores, which took about 5.5 days. TPUs were provided by Google's TPU Research Cloud program.

When evaluating likelihoods, we used the importance sampling estimator described in Appendix B with 10,000 samples (estimates had converged with this number of samples).

### D.3 Partially Observed Clustering

We implemented and trained VaDE models as described in Jiang et al. [19]. However, we use convolutional encoders and decoders in our experiments instead of fully-connected networks. Otherwise, all hyperparameters are the same as in Jiang et al. [19]. In a straightforward adaptation of how VaDE typically predicts the cluster for $\mathbf{x}$, we predict the cluster based on $\mathbf{x}_o$ with:

$$q(c \mid \mathbf{x}_o) = \mathbb{E}_{\mathbf{z} \sim q(\cdot \mid \mathbf{x}_o)} \left[ \frac{p(\mathbf{z} \mid c) p(c)}{\sum_{c'} p(\mathbf{z} \mid c') p(c')} \right]. \tag{23}$$

We use 50 samples when estimating the expectation in Equation 23.

Training the partially observed posterior network is then done as usual. We use the same network architecture as the VaDE model's encoder for this network. We use the autoregressive distribution described in Section D.5 for the partially observed posterior, with 256 hidden units and 2 residual blocks. Observed masks are sampled from a uniform distribution during training.

The supervised baseline is trained by first using the pretrained VaDE model to predict the class label for each instance (based on fully observed information). Those labels are then used as the ground truth to train a supervised model (that accepts partially observed inputs) with a standard cross-entropy loss. We use the same network architecture as the VaDE encoder and the partially observed posterior network for the supervised classifier. As before, observed masks are randomly generated during the training of this classifier.

### D.4 Very Fast Greedy Feature Acquisition

In the feature acquisition experiments, we use relatively simple VAE models. For flattened MNIST, our encoder and decoder are MLPs with hidden layers of sizes 50, 100, and 200. We use this same architecture for EDDI as well. For the convolutional model, the encoder has four layers with the following `(hidden units, kernel, stride)`: `(32, 3, 1)`, `(32, 3, 2)`, `(64, 3, 2)`, `(64, 1, 1)`. The decoder has the layers `(64, 8, 1)`, `(64, 5, 2)`, `(32, 5, 1)`, `(32, 5, 1)`, `(1, 3, 1)`. For both versions, the encoder outputs a diagonal Gaussian posterior, and the decoder outputs Bernoulli distributions. The latent dimension is 10 for all models.

For the partially observed posterior, we use a network with the same architecture as the fully observed encoder. Since we want to be able to compute the posterior entropy analytically when computing the information gains, we also use a Gaussian for the partially observed posterior. However, rather than letting it be diagonal, we parameterize the Gaussian with a lower triangular matrix $L$ such that the covariance matrix is $C = LL^\top$. We do not stop gradients on samples from the VAE encoder when computing the Posterior Matching loss.

We first train the VAEs with Posterior Matching, before learning the lookahead posteriors. During training, we uniformly generate masks that set between 0% and 20% of the features as observed. Our models are trained for 200000 steps with a batch size of 128. We use the Adam [20] optimizer with an initial learning rate of 0.001 and an exponential decay schedule with a rate of 0.9 every 5000 steps.

After the VAE with Posterior Matching has been trained, we then freeze this model and train the lookahead posterior network. This network outputs one diagonal Gaussian for each feature. Given that the number of features can be relatively large and we are limited by the memory of the GPU, we randomly select a subset of these distributions to update at each training step. That is, we subsample the terms in the sum in Equation 7. For the MLP model, we subsample 128 indices, and for the convolutional model we subsample 32 indices. We then estimate the expectation in Equation 7 over multiple samples from the already trained VAE model. We found using multiple samples to

be important for getting good performance. For the MLP model, we use 64 samples, and for the convolutional model we use 16 samples. The MLP model is trained for 50000 steps with a batch size of 64 and the convolutional model is trained for 60000 steps with a batch size of 32. We again use the Adam optimizer with an initial learning rate of 0.001 and an exponential decay schedule with a rate of 0.9 every 5000 steps.

### D.5 Autoregressive Posterior Details

As described in the main text, one of the advantages of Posterior Matching is the freedom to use highly expressive distributions for the partially observed posterior, as we do not require it to be reparameterizable. Here, we describe an autoregressive distribution that we use in some of our experiments. It is based on the proposal distributions used in Strauss and Oliva [38] and Nash and Durkan [28], which were shown by Strauss and Oliva [38] to outperform prior state-of-the-art arbitrary conditioning methods for likelihoods and imputation, despite being very simple.

The distribution consists of an MLP with residual connections that outputs a mixture of Gaussians for each covariate. This network accepts partially observed inputs as well as a conditioning vector as input. In our case, the conditioning vector is the output of the partially observed posterior encoder. We then compute $q(\mathbf{z} \mid \mathbf{x}_o)$ in an autoregressive fashion as

$$q(\mathbf{z} \mid \mathbf{x}_o) = \prod_{i=1}^{D} q(z_i \mid \mathbf{x}_o, \mathbf{z}_{<i}),$$

where $D$ is the dimensionality of $\mathbf{z}$. Each $q(z_i \mid \mathbf{x}_o, \mathbf{z}_{<i})$ term is obtained from a separate evaluation of the autoregressive distribution's network, where the partially observed inputs change to reflect the appropriate $\mathbf{z}_{<i}$. The conditioning vector that represents $\mathbf{x}_o$ remains constant for all of these evaluations though. Note that when computing likelihoods, these evaluations can be done efficiently in parallel. Sampling, however, requires an $O(D)$ sequential procedure. However, we do not need to sample this distribution during training, and $D$ is generally small anyway.

We chose this particular distribution for its combination of simplicity and good performance. However, we did not experiment extensively with other types of autoregressive distribution for the partially observed posterior. As previously noted, though, there is a large degree of flexibility in this choice.

## E Zero Imputation with Base VAE

One approach to imputation with VAEs that could be considered is to simply replace missing values with zeros before passing the input to the original VAE encoder and then subsequently using the decoder to obtain a reconstruction/imputation. While this method is straightforward and doesn't require any additional components in the model, it suffers greatly from distribution shift because the original VAE encoder was never trained to encounter those types of inputs. Thus, it is expected that this approach generally has worse performance. We include some simple results in Figure 9 illustrating this, where we use the base VAE from our MNIST experiments. We see that the reconstructions of the zero-imputed inputs faithfully reproduce the zeros (i.e. do not impute anything) and/or degrade the quality of the observed pixels' reconstructions.

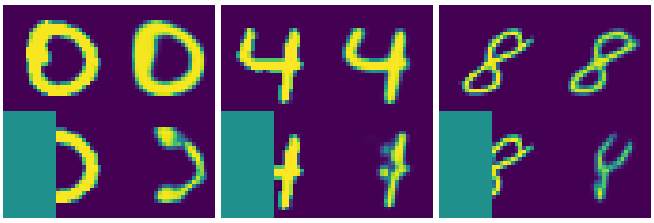


Figure 9: Demonstration of the zero-imputing approach to imputation with VAEs. In each image, the left column shows the inputs to a base VAE's encoder, and the right column shows the corresponding outputs. We show the masked regions of the inputs on the bottom row in green for clarity, even though those values are actually zero when passed to the encoder.

## F Additional Image Samples

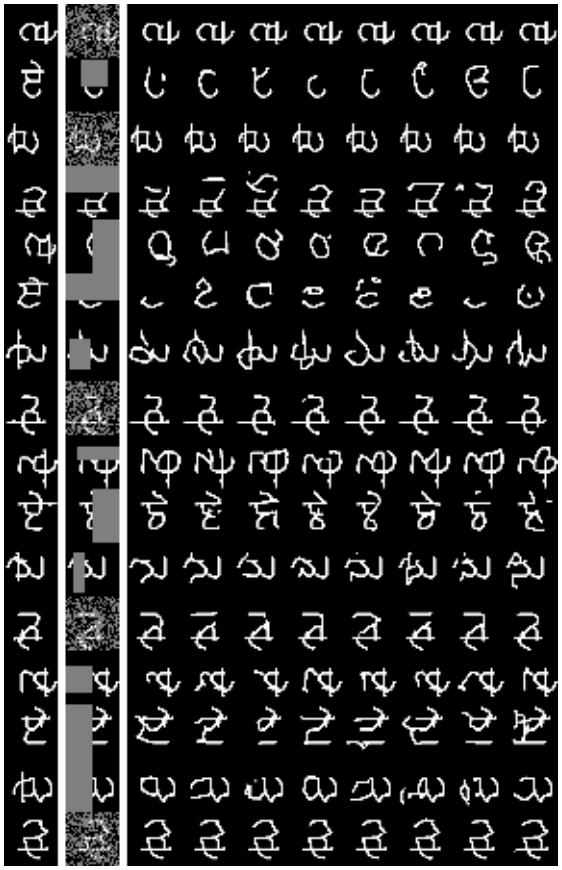


Figure 10: VDVAE OMNIGLOT inpaintings.

Figure 11: VDVAE MNIST inpaintings.

Figure 12: VDVAE CELEBA inpaintings.

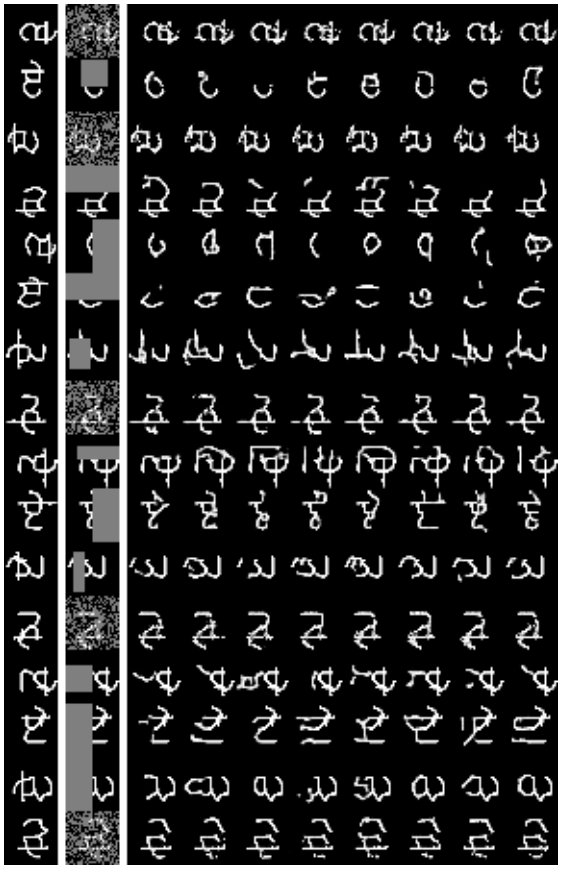


Figure 13: VQ-VAE OMNIGLOT inpaintings.

Figure 14: VQ-VAE MNIST inpaintings.

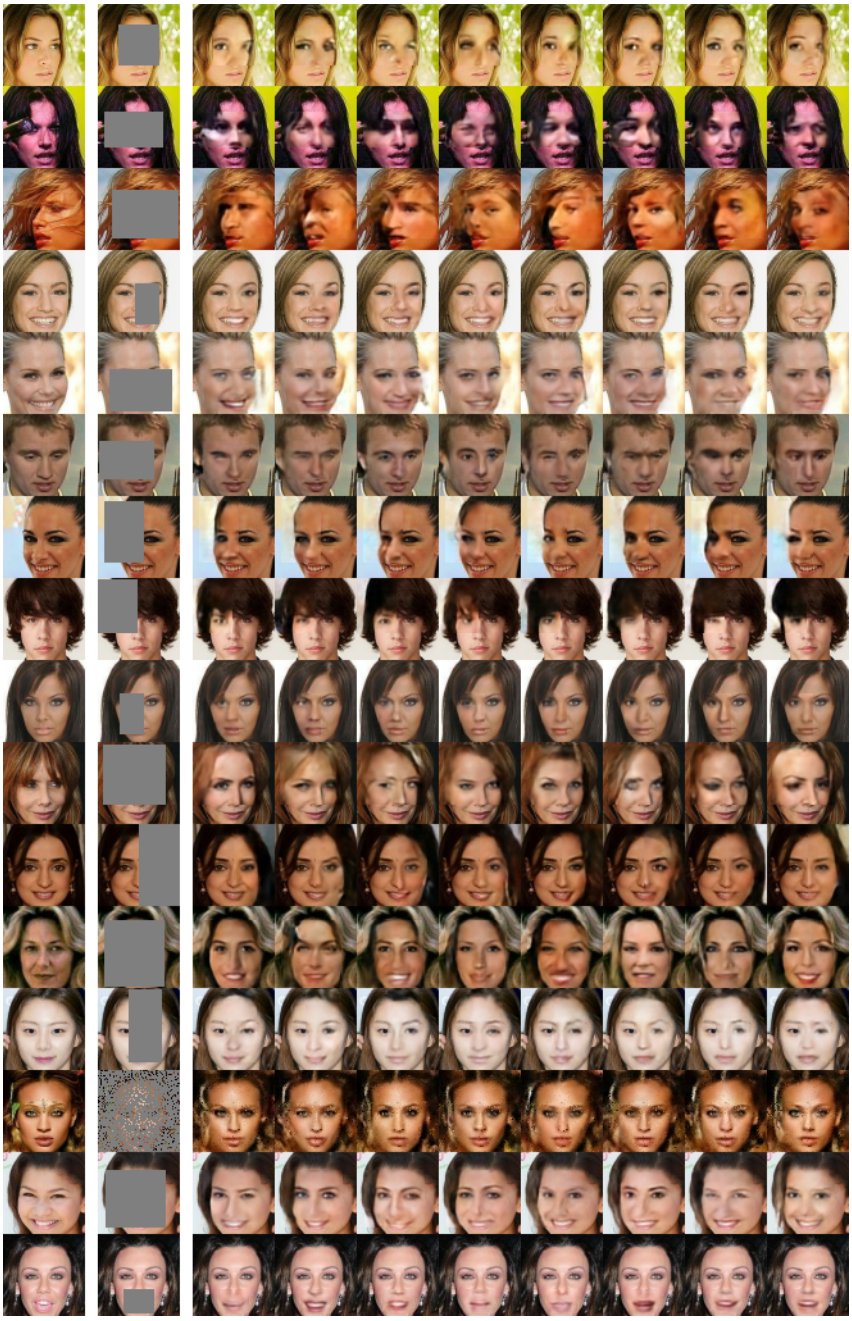
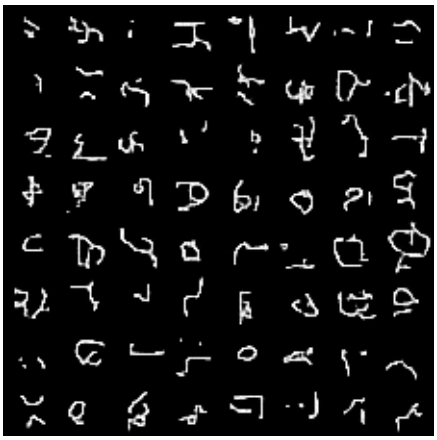


Figure 15: VQ-VAE CELEBA inpaintings.



(a) OMNIGLOT

(b) MNIST

(c) CELEBA

Figure 16: VQ-VAE image samples from the joint distribution, a special case obtained by sampling $q_\theta(\mathbf{z} \mid \mathbf{x}_o = \emptyset)$. Note that the models were not explicitly trained to model the joint and never saw $\mathbf{x}_o = \emptyset$ during training.