A Proof of Theorem 3.1

Proof. Starting from our original objective

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

we proceed as follows:

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

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

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

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

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

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

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

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

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

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

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:

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

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

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

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

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:

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

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

C Lookahead Posteriors

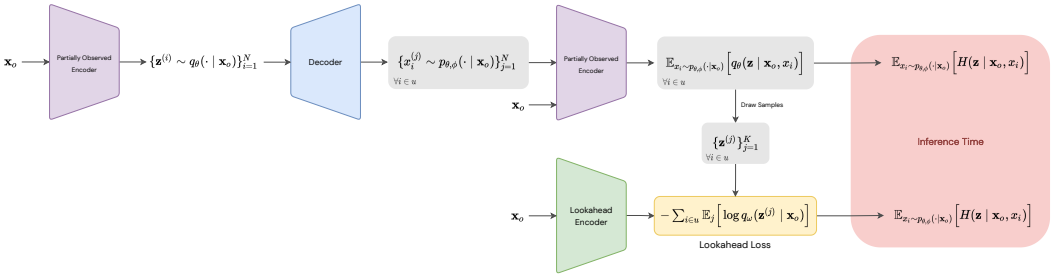


Figure 8: Overview of how Posterior Matching is used to learn "lookahead" posteriors for greedy active feature acquisition. The top path illustrates how samples are produced for use in the lookahead Posterior Matching loss. At inference time, we obtain the entropy of the distributions outputted by the 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 θ 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 β 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 β was 0 (except for MINIBOONE, where the period is 5000 and the delay is 2000). For models where a monotonic schedule was used, β 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
β 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.

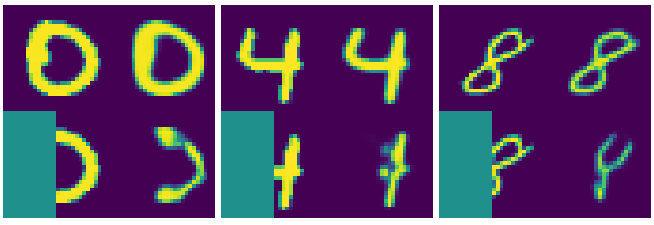


Figure 9: Demonstration of the zero-imputing approach to imputation with VAEs. In each image, the left column shows the inputs to a base VAE's encoder, and the right column shows the corresponding outputs. We show the masked regions of the inputs on the bottom row in green for clarity, even though those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

F Additional Image Samples

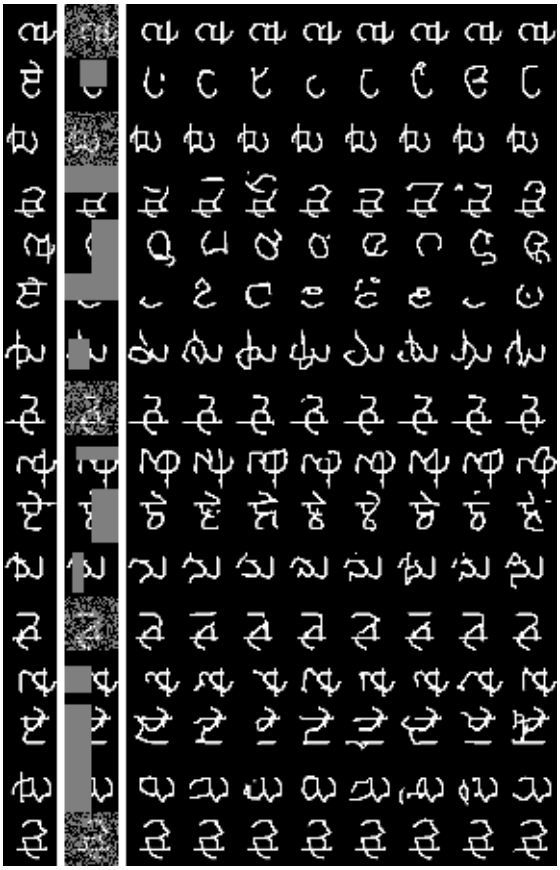


Figure 10: VDVAE OMNIGLOT inpaintings.

Figure 11: VDVAE MNIST inpaintings.

Figure 12: VDVAE CELEBA inpaintings.

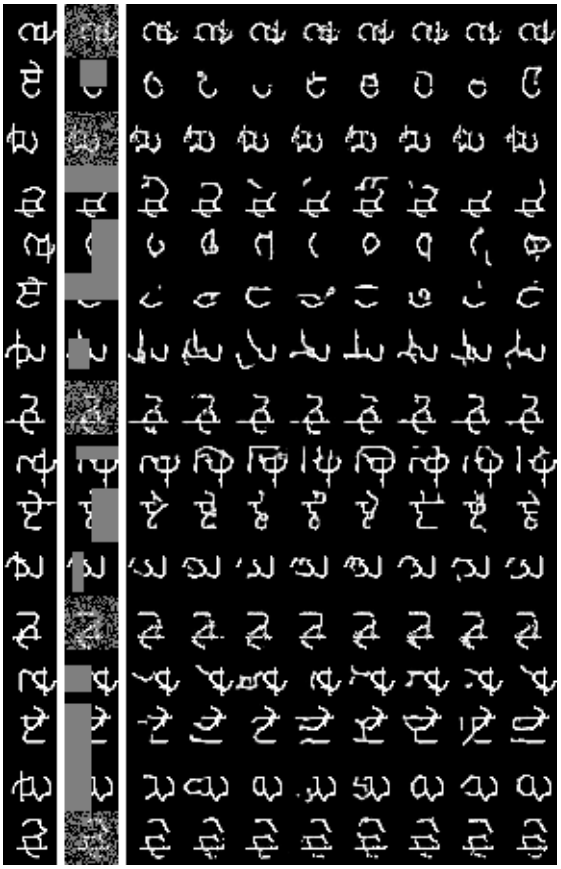


Figure 13: VQ-VAE OMNIGLOT inpaintings.

Figure 14: VQ-VAE MNIST inpaintings.

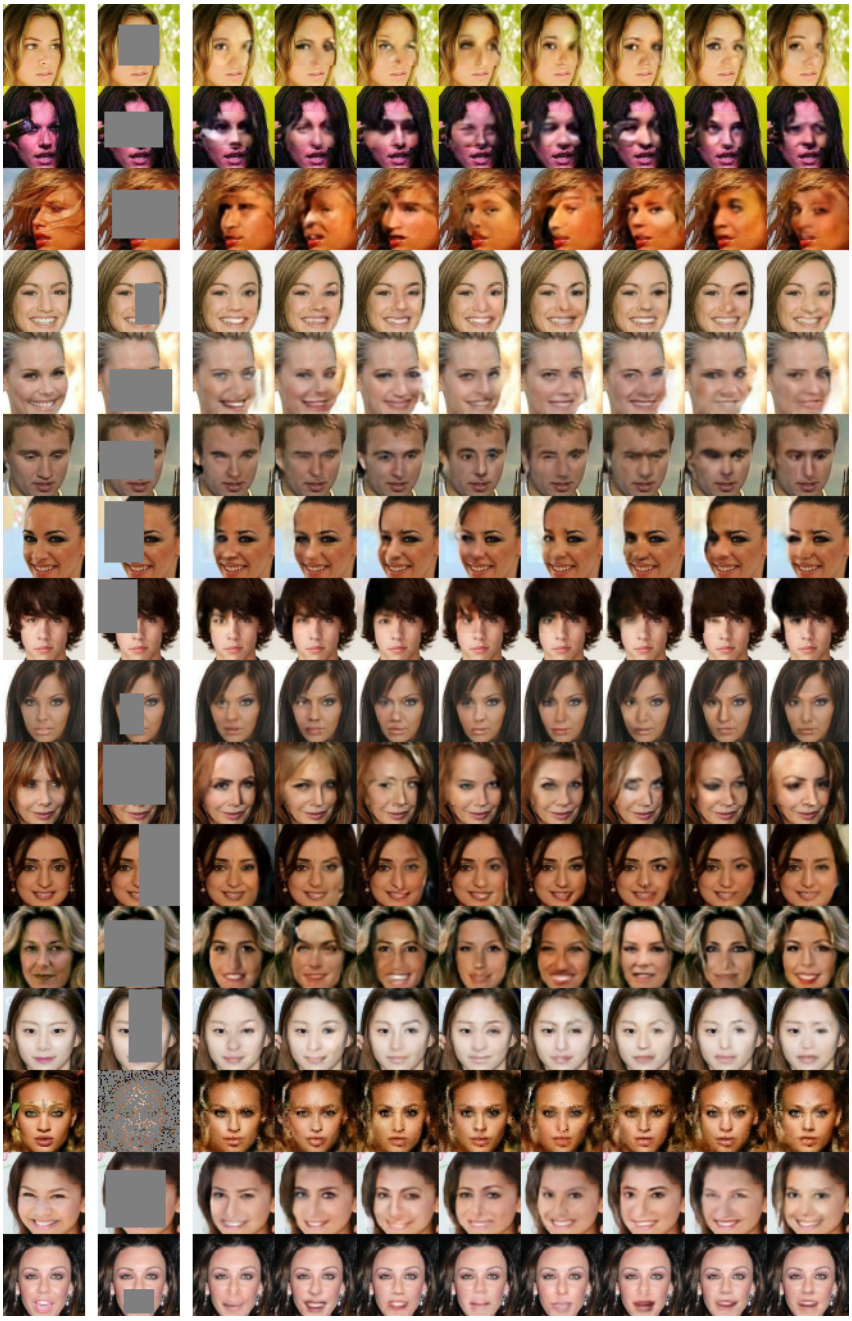
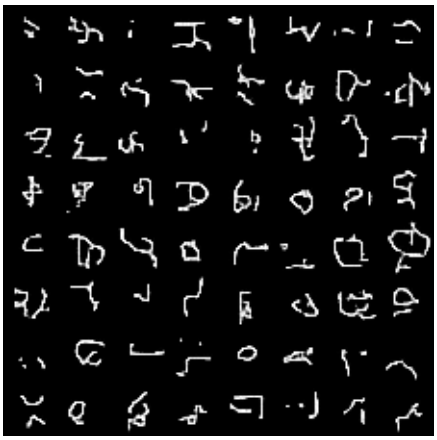


Figure 15: VQ-VAE CELEBA inpaintings.



(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.