Supplementary Information
Automatic Variational Inference with Cascading Flows

A Experiments

The experiments where implemented in Python 3.8 using the packages PyTorch, Numpy and Matplotlib. The implementation code is contained in the "modules" folder. The probabilistic program objects are defined in the file "models.py" and the deep architectures are in "networks.py". The experiments can be run from the files "timeseries experiment.py", "collider linear experiments.py", "collider tanh experiments.py" and "amortized experiments.py" in the "experiments" folder.

B Models

In the experiments, we use the following dynamical models.

B.1 Brownian motion (BR)

Dimensionality:	$J = 1$
Drift:	$\mu(x) = x$
Diffusion:	$\sigma^2(x) = 1$
Initial density:	$p(x_0) = \mathcal{N}(0, 1)$
Time horizon:	$T = 40$
Time step:	$\mathrm{d}t = 1$
Regression noise:	$\sigma_{\mathrm{lk}} = 1$
Classification gain:	$k = 2$

B.2 Lorentz system (LZ)

Dimensionality:	$J = 3$
Drift:	$\mu(x_1, x_2, x_3) = (10(x_2 - x_1), x_1(28 - x_3) - x_2, x_1x_2 - 8/3x_3)$
Diffusion:	$\sigma^2(x) = 2^2$
Initial density:	$p(x_0) = \mathcal{N}(3, 20^2)$
Time horizon:	$T = 40$
Time step:	$\mathrm{d}t = 1$
Regression noise:	$\sigma_{\mathrm{lk}} = 3$
Classification gain:	$k = 2$

B.3 Population dynamics (PD)

Dimensionality:	$J = 2$
Drift:	$\mu(x_1, x_2, x_3) = (\mathrm{ReLu}(0.2x_1 - 0.02x_1x_2), \mathrm{ReLu}(0.1x_1x_2 - 0.1x_2))$
Diffusion:	$\sigma^2(x) = 2$
Initial density:	$p(x_0) = \mathcal{N}(0, 1)$
Time horizon:	$T = 100$
Time step:	$\mathrm{d}t = 0.02$
Regression noise:	$\sigma_{\mathrm{lk}} = 3$
Classification gain:	$k = 2$

Note: The ReLu activation was added to avoid instabilities arising from negative values.

B.4 Recurrent neural network (RNN)

Dimensionality:	$J = 3$
Drift:	$\mu(\boldsymbol{x}) = \tanh(W_3 \tanh(W_2 \tanh(W_1\boldsymbol{x} + \boldsymbol{b_1}) + \boldsymbol{b_2}))$
Diffusion:	$\sigma^2(x) = 0.1^2$
Initial density:	$p(x_0) = \mathcal{N}(0, 1)$
Time horizon:	$T = 40$
Time step:	$\mathrm{d}t = 0.02$
Regression noise:	$\sigma_{\mathrm{lk}} = 1$
Classification gain:	$k = 2$

Note: W_1 is a 2×5 matrix, W_2 is a 5×5 matrix, W_3 is a 5×2 matrix, $\boldsymbol{b_1}$ is a 2D vector and $\boldsymbol{b_2}$ is a 5D vector. All the entries of these quantities are sampled from normal distributions with mean 0 and standard deviation (SD) 1. These parameters we re-sampled for every repetition of the experiment.

C Collider dependencies

We tested performance under collider dependencies using a Gaussian binary tree model where the mean of a scalar variable x_j^d in the d-th layer is a function of two variables in the $d-1$-th layer:

$$x_j^d \sim \mathcal{N}\big(\mathrm{link}(\pi_{j1}^{d-1}, \pi_{j2}^{d-1}), \sigma^2\big)$$

where π_{j1}^{d-1} and π_{j2}^{d-1} are the two parents of x_j^d. In the linear experiment the link was $\text{link}(\pi_{j1}^{d-1}, \pi_{j2}^{d-1}) = \pi_{j1}^{d-1} - \pi_{j2}^{d-1}$ The SD σ was 0.15 and the initial SD was 0.2. In non-linear experiment the link was $\text{link}(\pi_{j1}^{d-1}, \pi_{j2}^{d-1}) = \tanh \pi_{j1}^{d-1} - \tanh \pi_{j2}^{d-1}$. The SD σ was 0.05 and the initial SD was 0.1.

D Amortized experiments

The amortized experiments had the same details and parameters as the corresponding timeseries experiment. We use the following inference network for the MF and GF baselines (PyTorch code):

```
class InferenceNet(nn.Module):

    def __init__(self, in_size, out_size, n_hidden, out_shape=None, eps=10**-5):
        super(InferenceNet, self).__init__()
        self.in_size = in_size
        self.out_size = out_size
        self.eps = eps
        self.l1 = nn.Linear(in_size, n_hidden)
        self.l2 = nn.Linear(n_hidden, 2*out_size)
        if out_shape is not None:
            self.out_shape = out_shape
        else:
            self.out_shape = (out_size,1)

    def __call__(self, x):
        h = self.l2(F.relu(self.l1(x)))
        M = h.shape[0]
        out_shape = (M, self.out_shape[0], self.out_shape[1])
        return h[:,:self.out_size].view(out_shape), (F.softplus(h[:,self.out_size:]) + self.eps).view(out_shape)
```

In the case of GF, the network provided means and scales of the pre-transformation normal variables.