Disentangling impact of capacity, objective, batchsize, estimators, and step-size on flow VI

Abhinav Agrawal
University of Massachusetts Amherst

Justin Domke
University of Massachusetts Amherst

Abstract

Normalizing flow-based variational inference (flow VI) is a promising approximate inference approach, but its performance remains inconsistent across studies. Numerous algorithmic choices influence flow VI's performance. We conduct a step-by-step analysis to disentangle the impact of some of the key factors: capacity, objectives, gradient estimators, number of gradient estimates (batchsize), and step-sizes. Each step examines one factor while neutralizing others using insights from the previous steps and/or using extensive parallel computation. To facilitate high-fidelity evaluation, we curate a benchmark of synthetic targets that represent common posterior pathologies and allow for exact sampling. We provide specific recommendations for different factors and propose a flow VI recipe that matches or surpasses leading turnkey Hamiltonian Monte Carlo (HMC) methods.

1 INTRODUCTION

Normalizing flow-based variational inference (flow VI) [Rezende and Mohamed, 2015] approximates posterior distributions [Webb et al., 2019, Agrawal et al., 2020, Ambrogioni et al., 2021b, Glöckler et al., 2022, Durr et al., 2024] by constructing flexible variational families through a series of invertible transformations [Papamakarios et al., 2021, Kobyzev et al., 2020]. While promising, performance of flow VI remains inconsistent—some studies report success [Agrawal et al., 2020, Ambrogioni et al., 2021b, Glöckler et al., 2022, Vaitl et al., 2022, Durr et al., 2024], while others highlight optimization challenges and poor outcomes [Hoffman and Ma, 2020, Behrmann et al., 2021, Baudart and Mandel, 2021, Dhaka et al., 2021, Jaini et al., 2020, Liang et al., 2022, Andrade, 2024].

Multiple factors influence flow VI, making it challenging to determine the cause of inconsistencies: *Is it insufficient capacity, inappropriate objective, high gradient variance, or incorrect step-size?* The "entangling" of such factors limits understanding of flow VI's capabilities and best practices, hindering its wider adoption.

This paper takes a step-by-step approach to disentangle the impact of key factors—capacity (section 3), objective (section 4), gradient estimator and batchsize (section 5), and step-size (section 6). Each step answers a fundamental question about a factor while neutralizing the impact of others by either applying insights from the previous steps or using compute of modern GPU clusters. For example, section 4 asks: *Do we need complicated mode-spanning objectives?* To test this, we neutralize the impact of capacity by borrowing tested flow architectures from section 3 and neutralize the impact of optimization choices by using a massive batchsize alongside extensive hyperparameter sweeps.

This approach requires precise measures of accuracy. However, intractable posteriors complicate high-fidelity evaluations as without true samples, common metrics (applicable to both VI and HMC) are rare—the evidence lower bound and its variants [Burda et al., 2016, Dieng et al., 2017, Domke and Sheldon, 2018, 2019] work only for VI, while convergence measures like effective sample size are used primarily for HMC. Although one might use samples from an established method as proxies for ground truth [Magnusson et al., 2025], reliable, common metrics are still limited—the Wasserstein distance [Villani et al., 2009] is often used [Zhang et al., 2022, Blessing et al., 2024, Sendera et al., 2024, Vargas et al., 2024, Yi and Liu, 2023b] but scales poorly with sample size [Cuturi, 2013], making it unsuitable for high-fidelity evaluations (section 2.2).

To overcome evaluation issues, we curate a benchmark of synthetic targets (section 2.1) that reflect common posterior pathologies: poor conditioning, nonlinear cur-

Proceedings of the 28th International Conference on Artificial Intelligence and Statistics (AISTATS) 2025, Mai Khao, Thailand. PMLR: Volume 258. Copyright 2025 by the author(s).

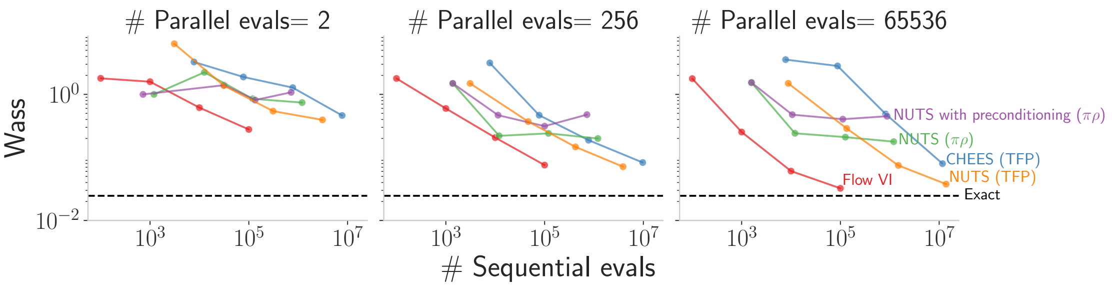


Figure 1: Marginal-Wasserstein metric (eq. 2) against sequential evaluations for Neal's funnel [Neal, 2001] in ten dimensions, with parallel budget increasing from left to right. For flow VI, sequential evaluations count optimization iterations and parallel evaluations represent batchsize. For HMC variants, sequential evaluations count leapfrog steps and parallel evaluations represent number of chains. (We use implementations from NumPyro and TensorFlow Probability, denoted with $\pi\rho$ (read "pyro") and TFP, respectively.) The black dotted line indicates the marginal-Wasserstein metric under exact samples (section 2.3). Flow VI is as accurate as exact inference and faster than HMC when using sufficient parallel budget.

vature, heavy tails, and parameter interdependence. These targets provide ground truth samples, ensuring high-fidelity evaluations (see section 2.3 for evaluation strategy). Additionally, we use a scalable proxy for the Wasserstein distance—marginal-Wasserstein (eq. 2)—which averages the Wasserstein distances between one-dimensional marginals. This scales log-linearly in sample size, offering a precise and efficient comparison of VI and HMC methods (section 2.2).

We present the major findings from analyzing the impact of important factors below.

Capacity. Real-NVP flows [Dinh et al., 2017] can accurately approximate challenging targets (fig. 2). While commonly used [Webb et al., 2019, Agrawal et al., 2020, Glöckler et al., 2022, Xu et al., 2023], some works report poor outcomes with real-NVP [Behrmann et al., 2021, Dhaka et al., 2021, Jaini et al., 2020, Andrade, 2024]. We neutralize the impact of other factors and show that, with appropriate architectural choices, real-NVP has sufficient capacity to represent challenging targets accurately (section 3).

Objectives. With high-capacity flows, complicated objectives are unnecessary (fig. 3). Some works advocate for mode-spanning objectives due to their better coverage [Li and Turner, 2016, Wang et al., 2018, Naesseth et al., 2020], but others show that these are hard to optimize [Geffner and Domke, 2020a, 2021, Dhaka et al., 2020]. We directly optimize mode-spanning $\mathrm{KL}(p \parallel q)$ using exact samples and show this indeed achieves great results (fig. 3). However, the easier to optimize mode-seeking $\mathrm{KL}(q \parallel p)$ is sufficient when capacity is high (fig. 3).

Gradient batchsize and estimators. Large batchsize greatly benefit high-capacity flows (fig. 4). Some works explore lower-variance gradient estimators for flows [Agrawal et al., 2020, Vaitl et al., 2022, 2024]. While these help, we show they are insufficient alone, and a larger batchsize significantly improves the performance of high-capacity flows, suggesting combined usage when possible (section 5).

Step-size and optimization. Maintaining the step-size within a narrow range is crucial for convergence (fig. 5). Automating the choice of step-size is an open problem [Kucukelbir et al., 2017, Welandawe et al., 2024]. We show that even with high-capacity flows, low gradient variance, and adaptive optimizers [Kingma and Ba, 2015], optimization can diverge abruptly after appearing to work for thousands of iterations (section 6). Step-sizes within 10^{-4} to 10^{-3} provide consistent results across targets over long runs (fig. 5).

Suggested Recipe. For easier adoption of various findings of our work, we suggest a *recipe* and encourage practitioners to adapt this to best fit their constraints.

Absolute essentials. Usually these will be easier to accommodate with powerful GPUs.

- Capacity: Use high capacity flows to reduce representational constraints. We use real-NVP flows with at least ten coupling layers and 32 hidden units when not varying the capacity explicitly (see figs. 2 and 3, and section D for details).
- Batchsize: Use a large number of gradient estimates (batchsize) to lower the gradient variance and simplify optimization. We use 512×2^{10} samples for experiments as an exploratory exercise and the performance is good with smaller *but large enough* choices (see results for 4×2^{10} samples in fig. 4).

Works really well. With high capacity and large batchsize, the following work extremely well.

- Objective: Optimize traditional VI objective (eq. 1). While, mode-spanning objectives help, the standard objective is easier to optimize and achieves comparable performance (see fig. 3).
- Estimator: Use sticking-the-landing (STL) gradient

estimator to further reduce gradient variance. This can be tricky for some flows, see section E for details.

- Step-size: Select a fixed step-size in 10^{-4} to 10^{-3} (fig. 5). If resources allow, sweep within this range.
- Optimization: Optimize for many iterations with an adaptive optimizer like Adam [Kingma and Ba, 2015]. At least 10K updates worked extremely well across the targets and dimensions (fig. 5).

Additional ingredients. Some additional recommendations based on initial experimentation.

- Base distribution: Evaluate pre-optimization ELBO under different distributions and choose the one with the highest initial ELBO. For experiments in the paper, we use the standard normal. However, for several targets, setting the Laplace's approximation [Domke and Sheldon, 2018, 2019, Agrawal et al., 2020] or the Student-t distribution [Jaini et al., 2020, Liang et al., 2022, Andrade, 2024] as the base distribution can help with the optimization. We suggest checking pre-optimization and picking the one with the higher initial ELBO value.
- Parameter initialization: We initialize the neural network parameters to some small value as it is equivalent to initializing the real-NVP transformations to identity [Agrawal et al., 2020], helping us control the starting approximation.
- Non-linearity for scale function: Chose hyperbolic tangent for the scale function in the affine transform (see eq. 3 and related discussion in section D). The choice of this non-linearity impacts the stability of the affine coupling flows [Behrmann et al., 2021, Andrade, 2024]. We experimented with several scale non-linearities and found that the hyperbolic tangent wrapped in an exponential (as in eq. 3) provides an easy trade-off of stability and capacity.

Using this recipe, we show flow VI matches or surpasses leading HMC methods on challenging targets in fewer model evaluations (see section 7 and figs. 1 and 6).

2 SETUP

Given a model $p(z, y)$ where z are the latent variables and y are the observed variables, the goal of inference is to approximate the posterior $p(z|y)$. Variational inference learns an approximation q by maximizing evidence lower-bound (ELBO) [Saul et al., 1996, Jordan et al., 1999, Wainwright et al., 2008, Blei et al., 2017], where

$$\text{ELBO} \coloneqq \mathbb{E}_{q}[\log p(z, y) - \log q(z)]. \tag{1}$$

We focus on classical probabilistic models where $p(z, y)$ does not have any unknown parameters apart from z (different from deep-latent variable models [Kingma and Welling, 2014, Rezende et al., 2014]). Maximizing the ELBO is equivalent to minimizing KL $(q \parallel p)$.

We use normalizing flows [Rezende and Mohamed, 2015, Kobyzev et al., 2020, Papamakarios et al., 2021] as the variational family for q. The main idea behind flows is to transform a base density $q_\epsilon(\epsilon)$ using a diffeomorphism T. Let $\epsilon \sim q_\epsilon(\epsilon)$. Then, the transformed variable $z = T(\epsilon)$ has the density $q(z) = q_\epsilon(\epsilon)|\det \nabla T(\epsilon)|^{-1}$. Usually, T is composed of a sequence of neural-network-based transformations designed such that T is invertible and the determinant of the Jacobian $|\det \nabla T(\epsilon)|$ can be calculated efficiently [Dinh et al., 2015, 2017].

We use real-NVP flows [Dinh et al., 2017]. These flows are composed of a sequence of layers, where each layer applies an affine transformation to one-half the variables, with the scale and translation parameters of the affine transform given by a neural network that takes the other half of the variables as input (see section D for details). real-NVP flows have an efficient forward (T) and inverse (T^{-1}) pass, allowing additional gradient estimators [Vaitl et al., 2022, 2024] (used in section 5).

2.1 Targets

We design a benchmark of synthetic targets, capturing common pathologies in real-world posteriors. See section K for full details. (We skip multi-modal targets from our benchmark as posterior distributions in classical probabilistic models do not commonly demonstrate that behavior.) All of these targets allow exact sampling, enabling analysis that is otherwise impossible (sections 3 and 4). The exact samples also allow comparisons for fair evaluations of VI and HMC methods (see section 2.3 for evaluation strategy). See fig. 13 in section K for samples in three dimensions.

2.2 Metrics

Several metrics of performance apply to either VI or HMC, but not both: standard metrics like ELBO, evidence upper bound [Ji and Shen, 2019], and importance-weighted ELBO [Burda et al., 2016] do not apply to HMC; convergence diagnostics like effective sample size do not apply to VI. Other metrics make precise comparisons hard: predictive test likelihoods are unreliable due to the vagaries of test datasets [Agrawal and Domke, 2024, Deshpande et al., 2024] and moments might not exist for heavy-tailed targets.

The Wasserstein distance [Villani et al., 2009] is rare in that it allows fair VI and HMC comparisons, but requires access to reference samples and scales poorly in sample-size [Cuturi, 2013]. In preliminary experiments, we found the noise in the Wasserstein distance estimates made fine comparisons impossible, even when using

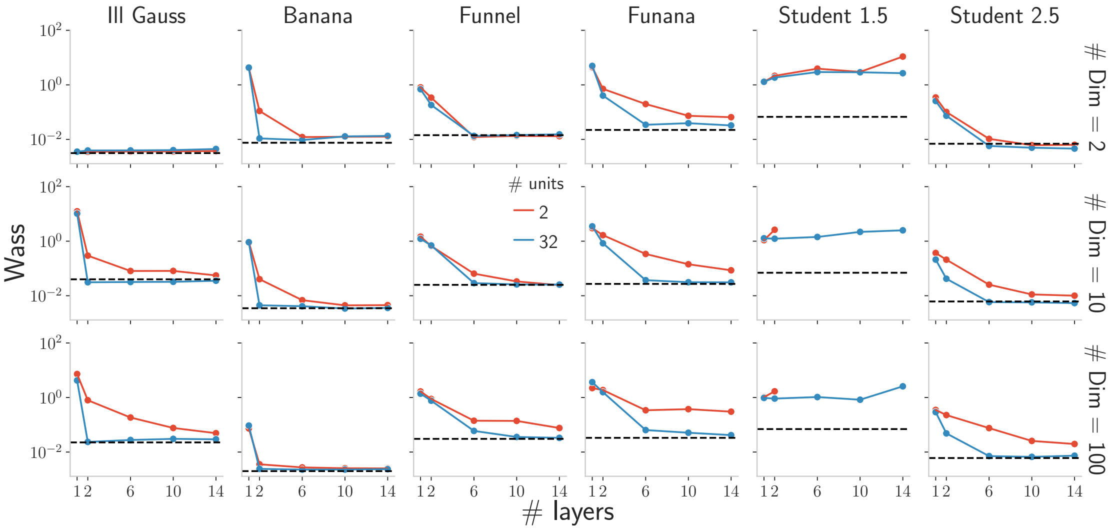


Figure 2: **Rows:** Model dimensions. Marginal-Wasserstein metric (eq. 2) against number of coupling layers for different number of hidden units. Performance improves with increase in either of these levers of capacity for all targets but Student-t with $\nu = 1.5$. Heavy tails of this target create problems when optimizing KL $(p \parallel q)$ [Jaini et al., 2020]. See fig. 3 for comparisons with KL $(q \parallel p)$ optimization. (In the first row, Funana uses 3 dimensions, see section K for target details)

thousands of samples (see fig. 8 in section B).

Fortunately, calculating the Wasserstein distance in the univariate case reduces to sorting. Based on this, we use marginal-Wasserstein metric—the average Wasserstein distance between the corresponding one-dimensional marginal distributions. Let both A and B be $S \times d$ sample matrices, where S is the number of samples, and d is the number of dimensions. Let $A^{\#}$ be the column-wise sorted version of A such that $A^{\#}_{0j}$ is the smallest value in column j. Then,

$$\text{marginal-Wasserstein}(A, B) := \frac{1}{d}\sum_{j=1}^{d} \frac{1}{S}\sum_{i=1}^{S} |A^{\#}_{ij} - B^{\#}_{ij}|. \quad (2)$$

Like the Wasserstein distance, this also suffers from Monte Carlo noise when the sample-size is relatively small, but since it is more scalable, we can use a much larger sample-size to reduce the noise (see section B). Of course, the marginal-Wasserstein metric only looks at marginals, and does not directly measure the correlations between the dimensions. However, its scalability makes it the only choice for high-fidelity evaluations.

2.3 Evaluation Strategy

For reliable performance evaluations, we use marginal-Wasserstein metric (eq. 2). The minimum achievable value of this metric, even with exact inference, depends on the sample size and target geometry [Cuturi, 2013]. To include a measure of ideal performance, we plot the marginal-Wasserstein metric between two independent sets of samples from the target with a black-dotted line (as in fig. 1). When an inference method's value approaches this line, its samples are as good as samples from the target. We use the same set of one million reference samples for all evaluations of a synthetic targets (figs. 1 and 6). In section 7, we also evaluate on some real targets and generate reference samples through long HMC runs (see section J for details).

2.4 Experimental Details

To simulate different scales, we use three dimensions: two, ten, and one hundred (apart from Funana, where at least three dimensions are needed). We implement flow-VI in JAX [Bradbury et al., 2018] and use implementations in TensforFlow probability [Lao et al., 2020] and NumPyro [Phan et al., 2019] for HMC methods. All methods are run on Nvidia A100 GPUs. For full details, see sections F to J.

3 DO REAL-NVP FLOWS HAVE ENOUGH CAPACITY FOR CHALLENGING TARGETS?

Real-NVP flows are a popular choice among inference researchers [Webb et al., 2019, Agrawal et al., 2020, Thin et al., 2020, Dhaka et al., 2021, Samsonov et al., 2022, Glöckler et al., 2022, Siahkoohi et al., 2023, Xu et al., 2023, Andrade, 2024]. However, some works also report unstable performance and poor results [Dhaka et al., 2021, Jaini et al., 2020, Andrade, 2024]. It is unclear whether such results arise due to a lack of representational ability of real-NVP flows or a failure of optimization. To resolve this, we study the impact of capacity while neutralizing other factors, asking:

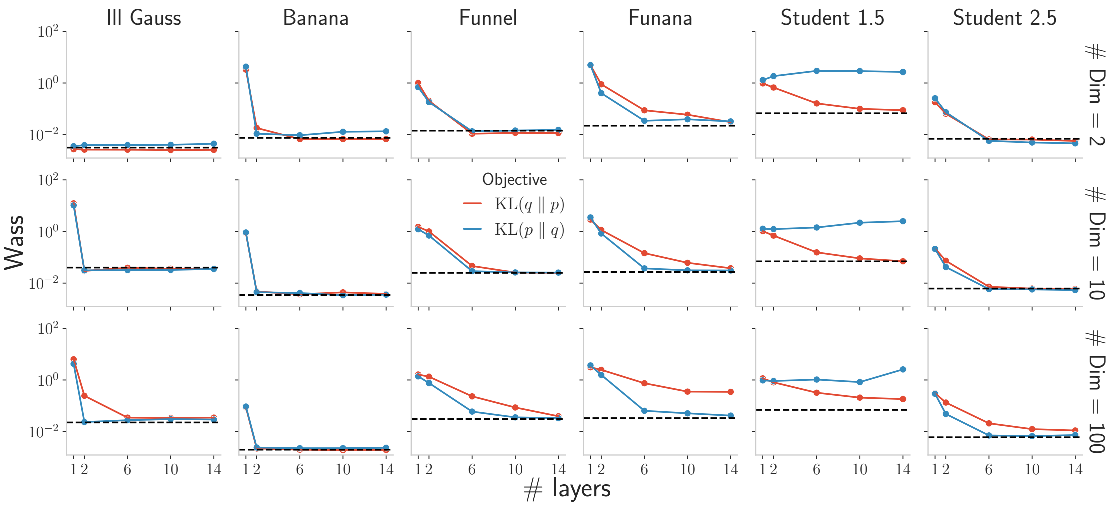


Figure 3: **Rows:** Model dimensions. Marginal-Wasserstein metric (eq. 2) against the number of layers (with 32 hidden units) for different objectives. Performance of $\mathrm{KL}\,(q \parallel p)$ (red) optimization improves as the number of layers increase, often reaching that of exact inference. A gap remains for Funana in one hundred dimensions, indicating this newly proposed density presents a significant challenge. (In the first row, Funana uses 3 dimensions.)

Do real-NVP flows have enough capacity to represent challenging targets accurately?

To answer the above question, we optimize $\mathrm{KL}\,(p \parallel q)$ [Minka et al., 2005, Naesseth et al., 2020, Ou and Song, 2020, Kim et al., 2022]. The practical algorithms for $\mathrm{KL}\,(p \parallel q)$ optimization suffer from high-variance self-normalized importance sampling-based gradients [Geffner and Domke, 2020a, 2021], but these issues get side-stepped when one works with exact samples. We directly optimize $\mathrm{KL}\,(p \parallel q)$ by using a massive batch of samples from the synthetic target, significantly simplifying the optimization.

We neutralize the impact of other optimization factors by sweeping over step-sizes and schedules, and optimizing for a large number of iterations (see section F).

Figure 2 shows the marginal-Wasserstein metric as we vary the capacity by increasing the number of coupling layers or hidden units (see section D for details). The key insight is that with the appropriate choice of these architectural parameters, real-NVP flows have sufficient capacity to represent challenging targets. The one exception is the Student-t with $\nu = 1.5$, where $\mathrm{KL}\,(p \parallel q)$ optimization struggles due to heavy tails [Jaini et al., 2020] ($\mathrm{KL}\,(q \parallel p)$ performs well, see fig. 3). Key findings include the following.

1. **Layers:** Increasing the number of layers is very effective. With ten or more layers, performance often matches exact inference (curves get closer to the black dotted line as layers increase).
2. **Hidden units:** Increasing the number of hidden units benefits higher dimensional problems (difference between the red and the blue curves increases).

While one expects that increasing capacity improves representational power, these results suggest that existing flow models like real-NVP are capable of representing difficult posterior geometries. So, with appropriate optimization choices, one can hope for impressive results from “relatively simple” flows.

4 DOES FLOW-VI NEED MODE-SPANNING OBJECTIVES?

The previous section demonstrates that real-NVP flows have enough capacity to represent challenging targets. However, section 3 relies on exact samples for optimization, making it crucial to reconsider our objective.

Selecting the right optimization objective is an active research area. Some advocate “mode-spanning” objectives, like $\mathrm{KL}\,(p \parallel q)$ from Section 3, as they lead better mass covering approximations [Li and Turner, 2016, Wang et al., 2018, Dieng et al., 2017, Hernandez-Lobato et al., 2016, Margossian et al., 2024, Cai et al., 2024, Zenn and Bamler, 2024]. While these are promising, their practical algorithms can suffer from large gradient variance [Rainforth et al., 2018, Finke and Thiery, 2019, Geffner and Domke, 2021, 2020a]. The primary alternative is to use the “easier” standard VI objective. However, other works show that this also can struggle when used with flows [Behrmann et al., 2021, Dhaka et al., 2021, Andrade, 2024]. Overall, the choice remains unclear, raising questions like: *Do we need complicated mode-spanning objectives? Does the standard VI objective suffice when the capacity is high?*

To answer the above questions, we focus on optimizing the “easy” $\mathrm{KL}\,(q \parallel p)$ while neutralizing the impact of

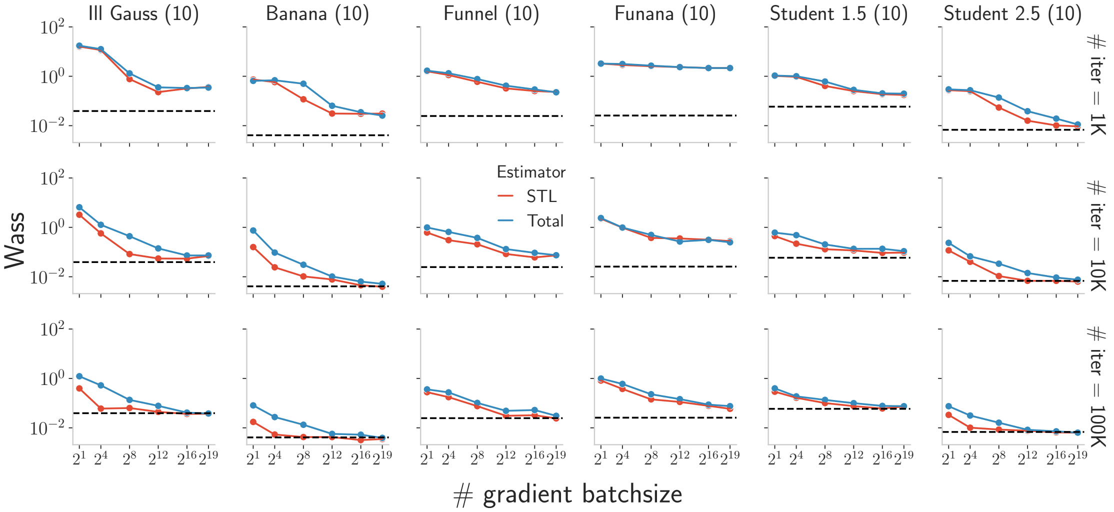


Figure 4: **Rows:** Number of iterations. Marginal-Wasserstein metric against the number of samples used for gradient evaluation for different targets in ten dimensions. STL (red) consistently outperforms total gradient (blue) at smaller batchsizes. However, as the batchsize increases, the difference vanishes. The performance for a given number of iterations (fixed for a row) improves significantly as the batchsize increases, indicating the impact of reduced gradient variance.

other factors. To do so, we use a massive batchsize (to reduce variance), do an exhaustive search over gradient estimators, step-sizes, and step-schedules, and run many iterations (see section G for details).

Figure 3 plots marginal-Wasserstein metric as a function of the number of layers after optimizing $\mathrm{KL}(q \parallel p)$ and $\mathrm{KL}(p \parallel q)$. The key insights are:

1. **Objective:** Optimizing the "mode-spanning" $\mathrm{KL}(p \parallel q)$ (blue) indeed gives better results than the easier "mode-seeking" $\mathrm{KL}(q \parallel p)$ (red), except in the case of the Student-t with $\nu = 1.5$, which is likely due to heavy tails [Jaini et al., 2020].
2. **Capacity:** As the capacity of the flow increases, the performance gap between the two objectives diminishes, with both achieving results close to exact inference. (A noticeable gap remains only for Funana in one hundred dimensions, suggesting this problem has a very challenging geometry).

These suggest an important difference for flow VI compared to VI with simple families like Gaussians. Rather than using a complex divergence to compensate for a mismatch between the target and the variational family, one can simply increase the flow capacity and optimize the standard objective. With enough capacity, the choice of divergence is less important.

5 ARE REDUCED VARIANCE ESTIMATORS HELPFUL?

The previous sections showed that optimizing standard $\mathrm{KL}(q \parallel p)$ objective can approximate challenging targets if the flow has enough capacity. However, the strategy in section 4 relied on impractical exhaustive hyperparameter sweeps. For effective practical use, we need a more efficient procedure. For this, we first ask: *how to estimate the gradient for reliable optimization?*

One major challenge for optimizing ELBO (eq. 1) is the reliance on stochastic gradient estimates. Several works have explored ways to construct lower-variance gradient estimators [Ranganath et al., 2014, Richter et al., 2020, Geffner and Domke, 2020b, Miller et al., 2017, Roeder et al., 2017, Tucker et al., 2019, Bauer and Mnih, 2021, Fujisawa and Sato, 2021, Yi and Liu, 2023a, Burroni et al., 2023]. A popular choice is the sticking-the-landing (STL) [Roeder et al., 2017, Tucker et al., 2019] estimator, which often performs well [Agrawal et al., 2020, Dhaka et al., 2020, Vaitl et al., 2022, 2024, Andrade, 2024]. However, STL requires inverting the flow transformation T [Agrawal et al., 2020, Vaitl et al., 2022, 2024], adding computational cost and potential numerical issues [Behrmann et al., 2021]. STL is also impractical for flows where inversion is expensive, like autoregressive flows [Papamakarios et al., 2017].

In principle, modern GPUs allow a brute-force solution to the variance problem—simply draw a huge number of estimates in parallel and average. This raises the natural question: *How does the choice of estimator interact with the batchsize for high-capacity flows?*

To answer this question, we optimize a high-capacity flow with both STL and the standard total gradient estimator using different batchsizes. We run independent optimization for different numbers of iterations, and sweep over step-sizes and step-schedules to reduce the impact of optimization factors (see section H).

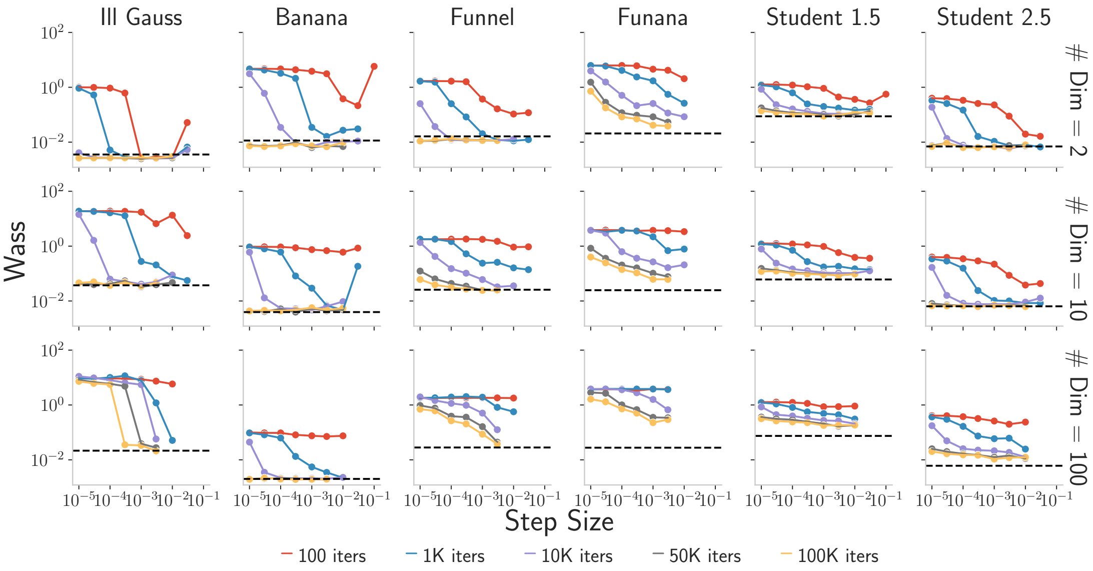


Figure 5: **Rows:** Model dimensions. Marginal-Wasserstein metric versus step-sizes for different iteration counts. Each point represents an independent optimization run; missing points indicate divergence. Notably, certain step-sizes, like 10^{-2}, achieve strong performance initially but diverge after several thousand iterations (see Funnel and Funana). In general, step-sizes within the range 10^{-4} to 10^{-3} perform reliably across the targets when optimized for 10K iterations or more.

Figure 4 plots the marginal-Wasserstein metric against the gradient batchsize for targets in ten dimensions. The key insights are:

1. **Estimator:** STL (red) consistently outperforms the total gradient (blue) for small batchsizes, but the difference reduces as the batchsize increases, and both reach the accuracy of exact samples.
2. **Batchsize:** For any fixed number of iterations (within a row), performance dramatically improves with larger batchsizes, highlighting the importance of reduced gradient variance.

These findings highlight that reducing gradient variance is crucial for strong empirical performance, even for flows that *can* represent challenging targets. Large batchsizes significantly improve performance and should, almost certainly, be used when appropriate parallel compute is available. For powerful GPUs, this increase in the batchsize can come at little or almost no penalty in wall-clock time. For instance, for Nvidia A100, increase in the batchsize by 32,000 times only increases the wall-clock times by 3 to 4 folds (see table 1 in section H). STL also reduces variance and improves convergence and should usually also be used (when efficient, see section E for discussion).

6 CAN OPTIMIZATION BE RELIABLY AUTOMATED?

While the previous section focused on reliably estimating ELBO gradients, the remaining hyperparameter sweeps are still impractical. This leads to the next important question: *how can we reliably optimize?*

Automating stochastic VI optimization is an open problem [Kucukelbir et al., 2017, Ambrogioni et al., 2021a,b, Rouillard et al., 2023, Welandawe et al., 2024, Carmon and Hinder, 2024, Attia and Koren, 2024, Defazio et al., 2024]. Decisions such as the choice of optimizer, step-size, step-schedule, and the number of iterations are not straightforward [Kucukelbir et al., 2017, Agrawal et al., 2020], and the literature offers little concrete guidelines tailored to flow VI.

Several works using flows report success with the Adam optimizer [Kingma and Ba, 2015]. Even when using Adam, selecting the optimal step-size schedule remains a challenge. This raises several questions: *Is there a step-size range that works across the targets for high-capacity flows? Should we scale the step-size with the dimensionality? Can the best step-size be predicted based on performance within a few hundred iterations?*

To transparently answer these questions, we optimize high-capacity flows independently with ten step-sizes for different numbers of iterations. We use a large batchsize and the STL estimator (see section I).

Figure 5 shows the marginal-Wasserstein metric against step-sizes. Some critical insights emerge:

1. **Diverging step-sizes:** Some step-sizes perform well for thousands of iterations and diverge when run for longer. For example, a step-size of 10^{-2} works well for 1K iterations but diverges at 10K

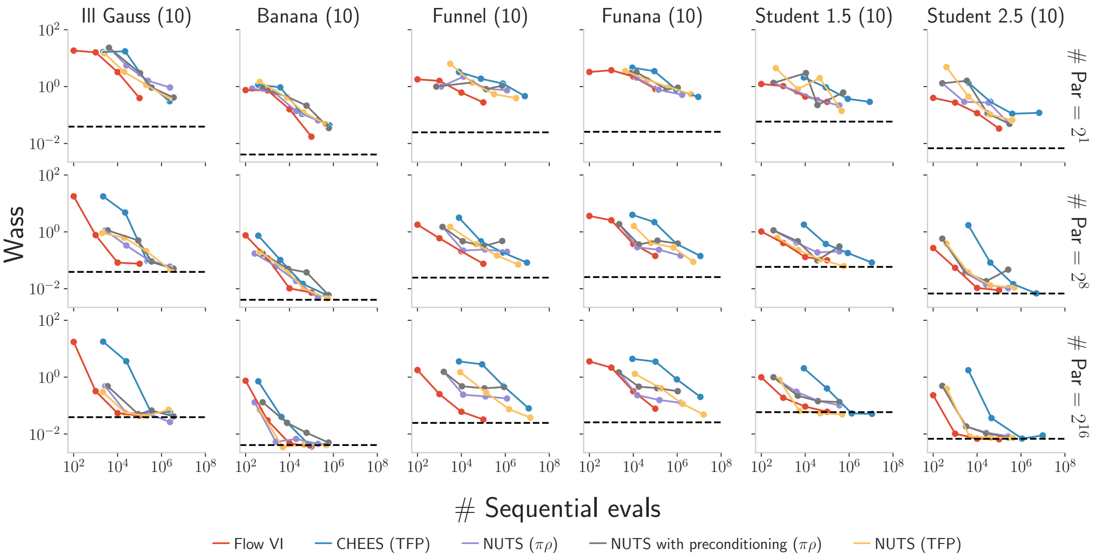


Figure 6: **Rows:** Parallel evaluations. Marginal-Wasserstein metric against number of sequential evaluations for synthetic targets in ten dimensions with parallel evaluations increasing across the rows. With sufficient parallel compute, flow VI is almost as accurate as NUTS (or exact inference); however, flow VI takes much fewer number of sequential evaluations. See fig. 1 and section 7 for details on how the evaluations are computed.

iterations in Funnel with one hundred dimensions.

2. **Stable step-range:** Range of 10^{-4} to 10^{-3} consistently performs well across targets (Funnel in a hundred dimensions is a notable exception where a slightly larger step-size yields better results.)

These results suggest that predicting optimal step-sizes from early performance (a few hundred iterations) is challenging. When using high-capacity flows with low gradient variance, step-sizes within a narrow range from 10^{-4} to 10^{-3} perform optimally across different targets and dimensions when optimized for at least 10K iterations (see purple, gray, and yellow curves).

7 HOW DOES FLOW VI COMPARE TO HMC METHODS?

At this point, previous sections suggest a simple recipe: Use a high capacity flow, traditional VI objective (for easier optimization), the STL estimator (when possible, for lower gradient variance), a large gradient batchsize (to further lower the variance), and a fixed step-size in a small range (see section 1 for the full recipe). Naturally, we ask: *how well does this actually work?* In this section, we compare our recipe to the state-of-the-art turnkey HMC methods.

Using modern GPUs with flow VI is straightforward—simply average more stochastic gradient estimates in each iteration. However, leveraging them for HMC methods is challenging [Margossian et al., 2023, Hoffman et al., 2021, Hoffman and Sountsov, 2022]. For NUTS, a leading turnkey HMC method, complications arise due to nuanced chain-dependent control flows [Lao and Dillon, 2019, Phan and Pradhan, 2019, Radul et al., 2020, Hoffman et al., 2021]. For some recent methods, like CHEES [Hoffman et al., 2021] or MEADS [Hoffman and Sountsov, 2022], the need for efficient cross-chain communication complicates implementations. Currently, running multiple parallel chains is the simplest and the most common way to use GPUs with HMC methods [Margossian et al., 2021, Hoffman et al., 2021, Sountsov et al., 2024].

With real problems, the computational bottleneck is often the model evaluation. Therefore, we count the model evaluations along two axes: parallel (a proxy for accelerator size) and sequential (a proxy for runtime). For HMC, parallel evaluations correspond to the number of chains, and sequential evaluations are the number of leapfrog steps. For VI, parallel evaluations correspond to gradient batchsizes, and sequential evaluations are the number of optimization iterations. Due to the limitations of the current frameworks, we extrapolate warmup-phase leapfrog-steps from the post-warmup phase (see section J for more discussion on how these steps are calculated.)

Figure 6 plots the marginal-Wasserstein metric against the number of sequential evaluations. Figure 7 plots the comparisons on six real-world problems (see section K for model details). Key insights are:

1. **Parallel evals:** Performance for all methods improves significantly as parallel compute increases

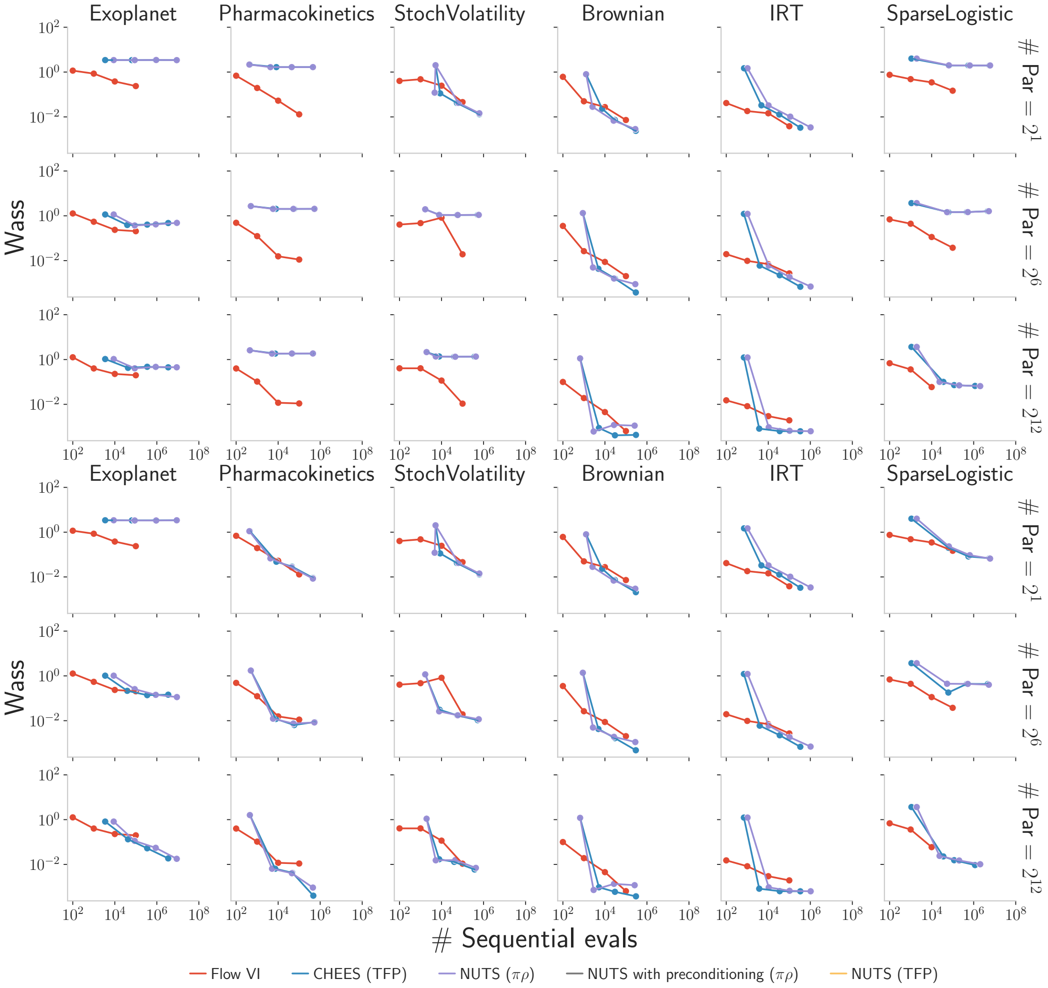


Figure 7: **Rows:** Parallel evaluations. Marginal-Wasserstein metric against number of sequential evaluations for non-synthetic models from Section K.2 with parallel evaluations increasing across the rows. For some models, we found that several HMC chains got stuck in a region of low probability. The first subplot corresponds to Wasserstein values when we do not filter the collapsed chains, and the second subplot corresponds to Wasserstein values when we filter the collapsed chains. The reference samples are generated after filtering the collapsed chains from a separate HMC run (see section J for details). (Such reference samples can potentially bias the comparisons in favor of HMC methods, motivating our synthetic benchmark approach in this work.)

(moving top to bottom across the rows).

2. **Sequential evals:** Flow VI matches or surpasses HMC methods for smaller batchsizes, and requires far fewer number of sequential evaluations when the batchsizes are larger.

These results suggest that flow VI can more effectively utilize the parallel compute through the large batchsizes, avoiding the coordination overhead inherent in HMCs chain-based approaches. More advances in HMC for modern accelerators could temper this conclusion; however, for now, flow VI using our proposed recipe serves as a strong alternative.

8 CONCLUSION

This paper presents a step-by-step analysis to disentangle the impact of key factors of flow VI. Our analysis finds that high-capacity flows and large gradient batchsizes are essential for achieving strong performance. We provide recommendations for successful choices of objectives, gradient estimators, and optimization strategies. Additionally, we show flow VI can match or surpass leading turnkey HMC methods on challenging targets with much fewer sequential steps.

9 ACKNOWLEDGEMENTS

The authors would like to thank Bob Carpenter for constructive comments on the early version of this work; Gilad Turock, Edmond Cunnigham, and Miguel Fuentes for useful discussions and comments; and Michelle Hedlund and Ankit Patel for helping improve the clarity of the manuscript.

References

Eric Agol, Rodrigo Luger, and Daniel Foreman-Mackey. Analytic planetary transit light curves and derivatives for stars with polynomial limb darkening. In *The Astronomical Journal*, 2020.

Abhinav Agrawal and Justin Domke. Understanding and mitigating difficulties in posterior predictive evaluation. *arXiv preprint arXiv:2405.19747*, 2024.

Abhinav Agrawal, Daniel R. Sheldon, and Justin Domke. Advances in black-box VI: normalizing flows, importance weighting, and optimization. In *NeurIPS*, 2020.

Luca Ambrogioni, Kate Lin, Emily Fertig, Sharad Vikram, Max Hinne, Dave Moore, and Marcel Gerven. Automatic structured variational inference. In *AISTATS*, 2021a.

Luca Ambrogioni, Gianluigi Silvestri, and Marcel van Gerven. Automatic variational inference with cascading flows. In *ICML*, 2021b.

Daniel Andrade. Stable training of normalizing flows for high-dimensional variational inference. *arXiv preprint arXiv:2402.16408*, 2024.

Michael Arbel, Alex Matthews, and Arnaud Doucet. Annealed flow transport monte carlo. In *ICML*, 2021.

Amit Attia and Tomer Koren. How free is parameter-free stochastic optimization? In *ICML*, 2024.

Guillaume Baudart and Louis Mandel. Automatic guide generation for stan via numpyro. *arXiv preprint arXiv:2110.11790*, 2021.

Matthias Bauer and Andriy Mnih. Generalized doubly reparameterized gradient estimators. In *ICML*, 2021.

Jens Behrmann, Paul Vicol, Kuan-Chieh Wang, Roger Grosse, and Jörn-Henrik Jacobsen. Understanding and mitigating exploding inverses in invertible neural networks. In *AISTATS*, 2021.

Michael Betancourt. A conceptual introduction to hamiltonian monte carlo. *arXiv preprint arXiv:1701.02434*, 2017.

David M Blei, Alp Kucukelbir, and Jon D McAuliffe. Variational inference: A review for statisticians. In *Journal of the American statistical Association*, 2017.

Denis Blessing, Xiaogang Jia, Johannes Esslinger, Francisco Vargas, and Gerhard Neumann. Beyond elbos: A large-scale evaluation of variational methods for sampling. In *ICML*, 2024.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

James Brofos, Marylou Gabrié, Marcus A Brubaker, and Roy R Lederman. Adaptation of the independent metropolis-hastings sampler with normalizing flow proposals. In *AISTATS*, 2022.

Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. In *ICLR*, 2016.

Javier Burroni, Kenta Takatsu, Justin Domke, and Daniel Sheldon. U-statistics for importance-weighted variational inference. In *TMLR*, 2023.

Alberto Cabezas and Christopher Nemeth. Transport elliptical slice sampling. In *AISTATS*, 2023.

Diana Cai, Chirag Modi, Loucas Pillaud-Vivien, Charles C. Margossian, Robert M. Gower, David M. Blei, and Lawrence K. Saul. Batch and match: black-box variational inference with a score-based divergence. In *ICML*, 2024.

Yair Carmon and Oliver Hinder. The price of adaptivity in stochastic convex optimization. In *CoLT*, 2024.

Saurab Chhachhi and Fei Teng. On the 1-wasserstein distance between location-scale distributions and the effect of differential privacy. *arXiv preprint arXiv:2304.14869*, 2023.

Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *NeurIPS*, 2013.

Marco Cuturi, Laetitia Meng-Papaxanthos, Yingtao Tian, Charlotte Bunne, Geoff Davis, and Olivier Teboul. Optimal transport tools (ott): A jax toolbox for all things wasserstein. *arXiv preprint arXiv:2201.12324*, 2022.

Aaron Defazio, Xingyu Alice Yang, Ahmed Khaled, Konstantin Mishchenko, Harsh Mehta, and Ashok Cutkosky. The road less scheduled. In *NeurIPS*, 2024.

Sameer Deshpande, Soumya Ghosh, Tin D. Nguyen, and Tamara Broderick. Are you using test log-likelihood correctly? In *TMLR*, 2024.

Akash Kumar Dhaka, Alejandro Catalina, Michael R Andersen, Måns Magnusson, Jonathan Huggins, and Aki Vehtari. Robust, accurate stochastic optimization for variational inference. In *NeurIPS*, 2020.

Akash Kumar Dhaka, Alejandro Catalina, Manushi Welandawe, Michael R Andersen, Jonathan Huggins, and Aki Vehtari. Challenges and opportunities in high dimensional variational inference. In *NeurIPS*, 2021.

Adji Bousso Dieng, Dustin Tran, Rajesh Ranganath, John Paisley, and David Blei. Variational inference via χ upper bound minimization. In *NeurIPS*, 2017.

Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. In *ICLR*, 2015.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. In *ICLR*, 2017.

C Djalil. Wasserstein distance between two gaussians. https://djalil.chafai.net/blog/2010/04/30/wasserstein-distance-between-two-gaussians/, April 2010.

Justin Domke and Daniel R Sheldon. Importance weighting and variational inference. In *NeurIPS*, 2018.

Justin Domke and Daniel R Sheldon. Divide and couple: Using monte carlo variational objectives for posterior approximation. In *NeurIPS*, 2019.

Oliver Durr, Stefan Hortling, Danil Dold, Ivonne Kovylov, and Beate Sick. Bernstein flows for flexible posteriors in variational bayes. In *Advances in Statistical Analysis (AStA)*, 2024.

Axel Finke and Alexandre H Thiery. On importance-weighted autoencoders. *arXiv preprint arXiv:1907.10477*, 2019.

Masahiro Fujisawa and Issei Sato. Multilevel monte carlo variational inference. In *JMLR*, 2021.

Marylou Gabrié, Grant M Rotskoff, and Eric Vanden-Eijnden. Adaptive monte carlo augmented with normalizing flows. In *Proceedings of the National Academy of Sciences*. National Acad Sciences, 2022.

Tomas Geffner and Justin Domke. On the difficulty of unbiased alpha divergence minimization. In *ICML*, 2020a.

Tomas Geffner and Justin Domke. A rule for gradient estimator selection, with an application to variational inference. In *AISTATS*, 2020b.

Tomas Geffner and Justin Domke. Empirical evaluation of biased methods for alpha divergence minimization. In *Third Symposium on Advances in Approximate Bayesian Inference*, 2021.

Manuel Glöckler, Michael Deistler, and Jakob H Macke. Variational methods for simulation-based inference. In *ICLR*, 2022.

Heikki Haario, Eero Saksman, and Johanna Tamminen. Adaptive proposal distribution for random walk metropolis algorithm. In *Computational statistics*, 1999.

Paul Hagemann, Johannes Hertrich, and Gabriele Steidl. Stochastic normalizing flows for inverse problems: a markov chains viewpoint. In *SIAM/ASA Journal on Uncertainty Quantification*. SIAM, 2022.

Soichiro Hattori, Lionel Garcia, Catriona Murray, Jiayin Dong, Shashank Dholakia, David Degen, and Daniel Foreman-Mackey. exoplanet-dev/jaxoplanet: Astronomical time series analysis with JAX, 2024. URL https://doi.org/10.5281/zenodo.10736936.

Jose Hernandez-Lobato, Yingzhen Li, Mark Rowland, Thang Bui, Daniel Hernández-Lobato, and Richard Turner. Black-box alpha divergence minimization. In *ICML*, 2016.

Matthew Hoffman and Yian Ma. Black-box variational inference as a parametric approximation to langevin dynamics. In *ICML*, 2020.

Matthew Hoffman, Pavel Sountsov, Joshua V Dillon, Ian Langmore, Dustin Tran, and Srinivas Vasudevan. Neutra-lizing bad geometry in hamiltonian monte carlo using neural transport. *arXiv preprint arXiv:1903.03704*, 2019.

Matthew Hoffman, Alexey Radul, and Pavel Sountsov. An adaptive-mcmc scheme for setting trajectory lengths in hamiltonian monte carlo. In *AISTATS*, 2021.

Matthew D Hoffman and Pavel Sountsov. Tuning-free generalized hamiltonian monte carlo. In *AISTATS*, 2022.

Priyank Jaini, Ivan Kobyzev, Yaoliang Yu, and Marcus Brubaker. Tails of lipschitz triangular flows. In *ICML*, 2020.

Chunlin Ji and Haige Shen. Stochastic variational inference via upper bound. *arXiv preprint arXiv:1912.00650*, 2019.

Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. In *Machine learning*. Springer, 1999.

Kyurae Kim, Jisu Oh, Jacob Gardner, Adji Bousso Dieng, and Hongseok Kim. Markov chain score ascent: A unifying framework of variational inference with markovian gradients. In *NeurIPS*, 2022.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *NeurIPS*, 2016.

David M Kipping. Efficient, uninformative sampling of limb darkening coefficients for two-parameter laws. In *Monthly Notices of the Royal Astronomical Society*, 2013.

Ivan Kobyzev, Simon JD Prince, and Marcus A Brubaker. Normalizing flows: An introduction and review of current methods. In *IEEE transactions on pattern analysis and machine intelligence*, 2020.

Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M Blei. Automatic differentiation variational inference. In *JMLR*, 2017.

J. Lao and J. V. Dillon. Unrolled implementation of no-u-turn sampler. https://github.com/tensorflow/probability/blob/master/discussion/technical_note_on_unrolled_nuts.md, August 2019.

Junpeng Lao, Christopher Suter, Ian Langmore, Cyril Chimisov, Ashish Saxena, Pavel Sountsov, Dave Moore, Rif A Saurous, Matthew D Hoffman, and Joshua V Dillon. tfp. mcmc: Modern markov chain monte carlo tools built for modern hardware. *arXiv preprint arXiv:2002.01184*, 2020.

Yingzhen Li and Richard E Turner. Rényi divergence variational inference. In *NeurIPS*, 2016.

Feynman Liang, Michael Mahoney, and Liam Hodgkinson. Fat–tailed variational inference with anisotropic tail adaptive flows. In *ICML*, 2022.

Måns Magnusson, Jakob Torgander, Paul-Christian Bürkner, Lu Zhang, Bob Carpenter, and Aki Vehtari. posteriordb: Testing, benchmarking and developing bayesian inference algorithms. In *AISTATS*, 2025.

Charles C Margossian, Matthew D Hoffman, Pavel Sountsov, Lionel Riou-Durand, Aki Vehtari, and Andrew Gelman. Nested r: Assessing the convergence of markov chain monte carlo when running many short chains. *arXiv preprint arXiv:2110.13017*, 2021.

Charles C Margossian, Andrew Gelman, and Alexandre Dumas. For how many iterations should we run markov chain monte carlo? *arXiv preprint arXiv:2311.02726*, 2023.

Charles C Margossian, Loucas Pillaud-Vivien, and Lawrence K Saul. An ordering of divergences for variational inference with factorized gaussian approximations. *arXiv preprint arXiv:2403.13748*, 2024.

Alex Matthews, Michael Arbel, Danilo Jimenez Rezende, and Arnaud Doucet. Continual repeated annealed flow transport monte carlo. In *ICML*, 2022.

Andrew Miller, Nick Foti, Alexander D'Amour, and Ryan P Adams. Reducing reparameterization gradient variance. In *NeurIPS*, 2017.

Tom Minka et al. Divergence measures and message passing. Technical report, Technical report, Microsoft Research, 2005.

Christian Naesseth, Fredrik Lindsten, and David Blei. Markovian score climbing: Variational inference with kl (p|| q). In *NeurIPS*, 2020.

Radford M Neal. Annealed importance sampling. In *Statistics and computing*, 2001.

Radford M Neal. Slice sampling. In *The annals of statistics*, 2003.

Zhijian Ou and Yunfu Song. Joint stochastic approximation and its application to learning discrete latent variable models. In *UAI*, 2020.

George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. In *NeurIPS*, 2017.

George Papamakarios, Eric Nalisnick, Danilo Jimenez Rezende, Shakir Mohamed, and Balaji Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. In *JMLR*, 2021.

Matthew D Parno and Youssef M Marzouk. Transport map accelerated markov chain monte carlo. In *SIAM/ASA Journal on Uncertainty Quantification*. SIAM, 2018.

D. Phan and N. Pradhan. Iterative nuts. https://github.com/pyro-ppl/numpyro/wiki/Iterative-NUTS, May 2019.

Du Phan, Neeraj Pradhan, and Martin Jankowiak. Composable effects for flexible and accelerated probabilistic programming in numpyro. *arXiv preprint arXiv:1912.11554*, 2019.

Alexey Radul, Brian Patton, Dougal Maclaurin, Matthew Hoffman, and Rif A Saurous. Automatically batching control-intensive programs for modern accelerators. In *Proceedings of Machine Learning and Systems*, 2020.

Tom Rainforth, Adam Kosiorek, Tuan Anh Le, Chris Maddison, Maximilian Igl, Frank Wood, and Yee Whye Teh. Tighter Variational Bounds are Not Necessarily Better. In *ICML*, 2018.

Rajesh Ranganath, Sean Gerrish, and David Blei. Black Box Variational Inference. In *AISTATS*, 2014.

Danilo Rezende and Shakir Mohamed. Variational inference with normalizing flows. In *ICML*, 2015.

Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, 2014.

Lorenz Richter, Ayman Boustati, Nikolas Nüsken, Francisco Ruiz, and Omer Deniz Akyildiz. Vargrad: a low-variance gradient estimator for variational inference. In *NeurIPS*, 2020.

Geoffrey Roeder, Yuhuai Wu, and David Duvenaud. Sticking the landing: Simple, lower-variance gradient estimators for variational inference. In *NIPS*, 2017.

Louis Rouillard, Alexandre Le Bris, Thomas Moreau, and Demian Wassermann. Pavi: Plate-amortized variational inference. In *TMLR*, 2023.

Sergey Samsonov, Evgeny Lagutin, Marylou Gabrié, Alain Durmus, Alexey Naumov, and Eric Moulines. Local-global mcmc kernels: the best of both worlds. In *NeurIPS*, 2022.

Lawrence K Saul, Tommi Jaakkola, and Michael I Jordan. Mean field theory for sigmoid belief networks. In *Journal of artificial intelligence research*, 1996.

Marcin Sendera, Minsu Kim, Sarthak Mittal, Pablo Lemos, Luca Scimeca, Jarrid Rector-Brooks, Alexandre Adam, Yoshua Bengio, and Nikolay Malkin. On diffusion models for amortized inference: Benchmarking and improving stochastic control and sampling. *arXiv preprint arXiv:2402.05098*, 2024.

Ali Siahkoohi, Gabrio Rizzuti, Rafael Orozco, and Felix J Herrmann. Reliable amortized variational inference with physics-based latent distribution correction. In *Geophysics*. Society of Exploration Geophysicists, 2023.

Pavel Sountsov, Alexey Radul, and contributors. Inference gym, 2020. URL https://pypi.org/project/inference_gym.

Pavel Sountsov, Colin Carroll, and Matthew D Hoffman. Running markov chain monte carlo on modern hardware and software. *arXiv preprint arXiv:2411.04260*, 2024.

Achille Thin, Nikita Kotelevskii, Jean-Stanislas Denain, Leo Grinsztajn, Alain Durmus, Maxim Panov, and Eric Moulines. Metflow: a new efficient method for bridging the gap between markov chain monte carlo and variational inference. *arXiv preprint arXiv:2002.12253*, 2020.

George Tucker, Dieterich Lawson, Shixiang Gu, and Chris J Maddison. Doubly reparameterized gradient estimators for monte carlo objectives. In *ICLR*, 2019.

Lorenz Vaitl, Kim A Nicoli, Shinichi Nakajima, and Pan Kessel. Gradients should stay on path: better estimators of the reverse-and forward kl divergence for normalizing flows. In *Machine Learning: Science and Technology*, 2022.

Lorenz Vaitl, Ludwig Winkler, Lorenz Richter, and Pan Kessel. Fast and unified path gradient estimators for normalizing flows. In *ICLR*, 2024.

Francisco Vargas, Shreyas Padhy, Denis Blessing, and Nikolas Nüsken. Transport meets variational inference: Controlled monte carlo diffusions. In *ICLR*, 2024.

Cédric Villani et al. *Optimal transport: old and new*. Springer, 2009.

Martin J Wainwright, Michael I Jordan, et al. Graphical models, exponential families, and variational inference. In *Foundations and Trends® in Machine Learning*. Now Publishers, Inc., 2008.

Dilin Wang, Hao Liu, and Qiang Liu. Variational inference with tail-adaptive f-divergence. In *NeurIPS*, 2018.

Stefan Webb, Jonathan P Chen, Martin Jankowiak, and Noah Goodman. Improving automated variational inference with normalizing flows. In *ICML Workshop on Automated Machine Learning*, 2019.

Manushi Welandawe, Michael Riis Andersen, Aki Vehtari, and Jonathan H. Huggins. A framework for improving the reliability of black-box variational inference. In *JMLR*, 2024.

Hao Wu, Jonas Köhler, and Frank Noé. Stochastic normalizing flows. In *NeurIPS*, 2020.

Zuheng Xu, Naitong Chen, and Trevor Campbell. Mixflows: principled variational inference via mixed flows. In *ICML*, 2023.

Mingxuan Yi and Song Liu. Bridging the gap between variational inference and wasserstein gradient flows. *arXiv preprint arXiv:2310.20090*, 2023a.

Mingxuan Yi and Song Liu. Sliced wasserstein variational inference. In *Asian Conference on Machine Learning*. PMLR, 2023b.

Johannes Zenn and Robert Bamler. Differentiable annealed importance sampling minimizes the jensen-shannon divergence between initial and target distribution. In *ICML*, 2024.

Lu Zhang, Bob Carpenter, Andrew Gelman, and Aki Vehtari. Pathfinder: Parallel quasi-newton variational inference. In *JMLR*, 2022.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. [Not Applicable]
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. [Not Applicable]
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. [No]
2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. [Not Applicable]
   (b) Complete proofs of all theoretical results. [Not Applicable]
   (c) Clear explanations of any assumptions. [Not Applicable]

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). [Yes‘’]
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). [Yes]
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). [Yes]
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. [Yes]
   (b) The license information of the assets, if applicable. [Not Applicable]
   (c) New assets either in the supplemental material or as a URL, if applicable. [Not Applicable]
   (d) Information about consent from data providers/curators. [Not Applicable]
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. [Not Applicable]

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. [Not Applicable]
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. [Not Applicable]
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. [Not Applicable]

Appendices: Table of Contents

A RELATED WORKS

Several works explore normalizing flows for black-box variational inference (BBVI) [Webb et al., 2019, Baudart and Mandel, 2021, Agrawal et al., 2020, Dhaka et al., 2021, Ambrogioni et al., 2021b, Andrade, 2024]. Often, these studies make axiomatic assumptions about flow abilities of flows and are more focused on the applications. We focus on disentangling the key factors involved in flow VI using a step-by-step approach and our learnings should benefit any such future applications.

Numerous studies aim to automate aspects of BBVI [Kucukelbir et al., 2017, Agrawal et al., 2020, Dhaka et al., 2020, Ambrogioni et al., 2021a,b, Welandawe et al., 2024], aiming to provide turnkey solutions for probabilistic models without requiring manual intervention. Unlike these approaches, our work focuses on understanding the impact of different factors, offering detailed guidelines rather than an automatic solution. We believe our analysis lays the necessary groundwork for future flow-based automatic inference tools.

Agrawal et al. [2020] proposed an approach that combines normalizing flows, STL [Roeder et al., 2017], a step-size search scheme, and a post-hoc importance sampling step to enhance out-of-the-box BBVI performance. They do not dissect the factors affecting flow VI, use a relatively small batchsizes, use CPUs, and skip any insights into their failure cases. In contrast, our work specifically delves into the impact of different factors, leverages modern GPUs to employ massive batchsizes, and also demonstrates how appropriately optimized flow VI can match or surpass HMC methods.

Andrade [2024] focus on stably optimizing real-NVP flows on high-dimensional problems and consider the impact of architectural choices on optimization stability. Our study independently uses an architecture that aligns with their optimal choices, avoiding instabilities. While it is possible that some of their tricks improve our performance, our aim here is to develop a holistic understanding of impact of different factors and not just restrict to architectural choices.

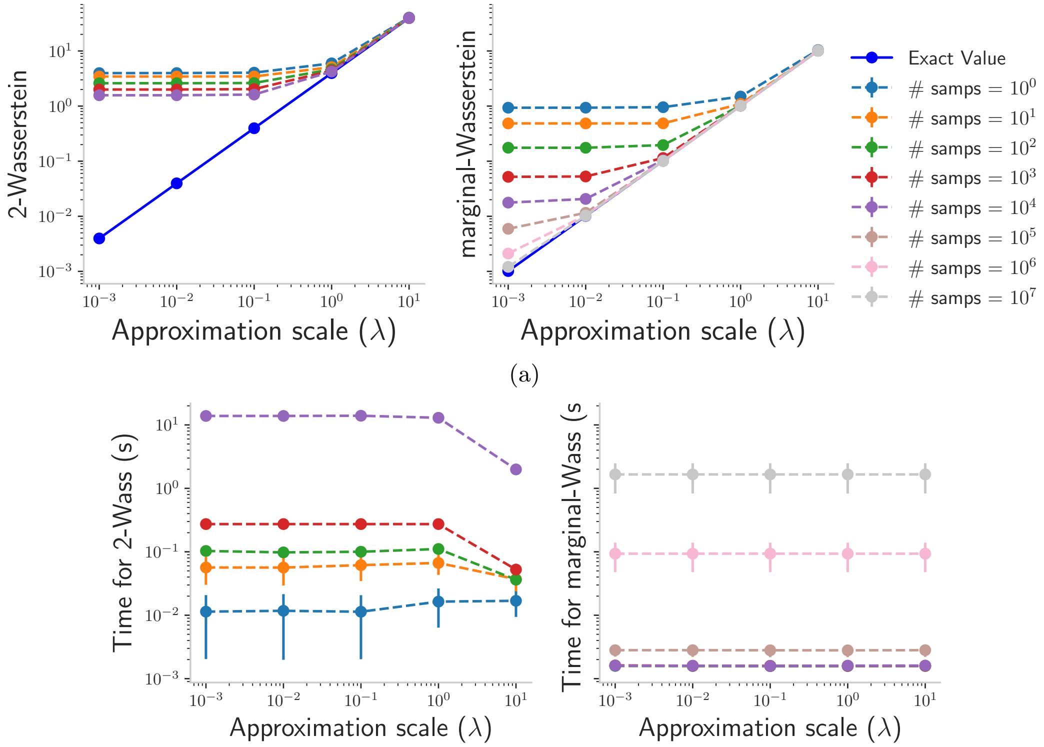
Dhaka et al. [2021] use an importance sampling based diagnostics to assess performance and recommend using normalizing flows but report poor performance due to optimization challenges (see Figures C.2 and C.3 in their appendix). Based on the insights from our study, we conjecture that the relatively lower-capacity flows (causing insufficient representational ability) and smaller batchsizes (leading to higher-gradient variance) might explain some issues they experience.

Jaini et al. [2020] uncover that targets with heavier tails require approximations that have base distributions with identical tail properties. Our findings in fig. 2 corroborate this as optimizing $\mathrm{KL}(p \parallel q)$ results in poor performance on heavy tailed targets. Importantly, in section 4, a high-capacity flow with standard VI objective significantly improves the performance.

Blessing et al. [2024] highlighted the need for standardized evaluation in inference research and introduced a benchmark that includes both synthetic and real-world problems with special focus on multi-modal targets. We also use synthetic densities to create controlled environments and employ integral metrics like the Wasserstein distance. However, we additionally provide a useful measure of ideal performance to better contextualize the numbers (black dotted line in figures, see section 2.3). We also include a collection of six real-world problems to showcase the practical performance of flow VI.

Recent literature also explores the use of flows as proposals for MCMC methods [Parno and Marzouk, 2018, Hoffman et al., 2019, Wu et al., 2020, Naesseth et al., 2020, Arbel et al., 2021, Matthews et al., 2022, Gabrié et al., 2022, Hagemann et al., 2022, Brofos et al., 2022, Kim et al., 2022, Samsonov et al., 2022, Cabezas and Nemeth, 2023] with some using $\mathrm{KL}(q \parallel p)$ optimization to initialize the proposal distribution. These approaches are orthogonal to simply using flow VI, and can potentially benefit from insights discovered in this work (by learning better initial proposals).

B DETAILS OF n-WASSERSTEIN EVALUATION



(a)

(b)

Figure 8: **(a)** Empirical 2−Wasserstein and marginal-Wasserstein metric against approximation scale λ for the example described in section B.1. As λ increases, approximation becomes poor, increasing the true value of either of the Wasserstein metrics (solid blue line). For poor approximations (large λ), the numerical evaluation is accurate with smaller sample-size. However, when the approximation is closer to the target (small λ), a larger sample-size is required for accurate evaluation and fine comparisons. **(b)** Time taken for calculating 2−Wasserstein and marginal-Wasserstein metric against the approximation scale λ. Marginal-Wasserstein scales log-linearly (section 2.2), allowing much larger sample-sizes while maintaining reasonable wall-clock times. (Calculating 2−Wasserstein with 10^{4} samples take more time than 10^{7} samples for marginal-Wasserstein metric).

Wasserstein distance between two distributions is generally not available in closed-form [Villani et al., 2009]. Numerical methods to evaluate it do not scale well with sample size [Cuturi, 2013]. When using a limited number of samples, these lead to noisy evaluations that make the $n-$Wasserstein distance unreliable for fine-grained comparisons. To demonstrate this, we present a simple representative example.

B.1 Example for Wasserstein Calculations

Consider a standard normal target distribution in ten dimensions. We will evaluate $2-$Wasserstein and the marginal-Wasserstein metric between this target and some approximations. Both of the Wasserstein metrics are available in closed form for Gaussian distributions [Djalil, 2010, Chhachhi and Teng, 2023]. We will compare this exact value with numerically calculated values to understand the efficacy of these metrics for making fine-grained comparisons.

To create approximation that are increasingly different from the target (the standard normal), we sample the location from $\mathcal{N}(0, \lambda^2\mathbf{I})$ where we vary the approximation scale λ geometrically from 10^{-4} to 10^1. When λ is small (like $\lambda = 10^{-4}$), the approximation is close to the target, and when λ is large (like $\lambda = 10$), the approximation is likely to be very different from the target. (The covariance matrix is set to be identity for all the approximations.)

Figure 8 shows the performance of empirical calculation of marginal-Wasserstein metric and $2-$Wasserstein. When the approximation is poor (λ is large), the numerical methods require a much smaller sample-size for accurate estimation (Figure 8a). However, when the approximation is close to the target (λ is small), both of these metrics require a much larger sample-size for accurate estimation. Since, marginal-Wasserstein metric is more scalable, we are able to use a much larger sample-size and get extremely accurate measurements in relatively faster wall-clock time (see fig. 8b). We use `ott-jax` [Cuturi et al., 2022] for calculating the $2-$Wasserstein distance and run the experiments on Nvidia A100.

Overall, fig. 8 demonstrates that unless we use a large number of samples, approximations with varying degree of accuracy can evaluate to the same incorrect Wasserstein value. $2-$Wasserstein metric is not suitable for evaluating fine-grained comparisons as it can not scale to large sample sizes required for such accurate evaluations. On the other hand, marginal-Wasserstein metric is a lot more scalable and provides accurate measurements in reasonable time.

C LIMITATIONS

We use synthetic targets in our setup. Our use is intentional to mitigate any possible variability and to facilitate high-fidelity performance comparisons with HMC methods. Future work can exhaustively explore the impact on real-world problems. However, we expect our learnings to apply. In fact, we show that flow VI can match or surpass HMC methods on several real-world problems Figures 6 and 12. We use modern GPUs like the Nvidia A100s which are not yet universally available. Despite computational constraints, we believe researchers and practitioners can still take guidelines we establish to better utilize the resources available to them. We do not go beyond a hundred dimensions for the synthetic targets. While this is not small, our step-by-step analysis required a substantial computational effort, consuming more than 4000 GPU hours on NVIDIA A100s. We expect several of our learnings to apply when scaling to even higher dimensions, and leave the in-depth exploration for future work. Some expert practitioners may find some of our findings trivial. However, we believe the utility of these techniques is hypothetical unless somebody conducts a direct, thorough study like we do.

D DETAILS OF REAL-NVP ARCHITECTURE

We use a real-NVP [Dinh et al., 2017] flow with affine coupling layers. We define each coupling layer to be comprised of two transitions, where a single transition corresponds to affine transformation of one part of the latent variables. For example, if the input variable for the k^{th} layer is $z^{(k)}$, then first transition is defined as

$$\begin{aligned} z_{1:d} &= z^{(k)}_{1:d} \quad \text{and} \\ z_{d+1:D} &= z^{(k)}_{d+1:D} \odot \exp\left(s^a_k(z^{(k)}_{1:d})\right) + t^a_k(z^{(k)}_{1:d}), \end{aligned} \tag{3}$$

where, for the function s and t, super-script a denotes first transition and sub-script k denotes the layer k. For the next transition, the $z_{d+1:D}$ part is kept unchanged and $z_{1:d}$ is affine transformed similarly to obtain the layer output $z^{(k+1)}$ (this time using $s_k^b(z_{d+1:D}^{(k)})$ and $t_k^b(z_{d+1:D}^{(k)})$). This is also referred to as the alternating first half binary mask.

Both, scale(s) and translation(t) functions of single transition are parameterized by the same fully connected neural network(FNN). More specifically, in the above example, a single FNN takes $z_{1:d}^{(k)}$ as input and outputs both $s_k^a(z_{1:d}^{(k)})$ and $t_k^a(z_{1:d}^{(k)})$. The skeleton of the FNN, in terms of the size of the layers, is as $[d, H, H, 2(D-d)]$ where, H denotes the size of the two hidden layers.

The hidden layers of FNN use a leaky rectified linear unit with slope $= 0.01$, while the output layer uses a hyperbolic tangent for s and remains linear for t. We initialize neural network parameters with normal distribution $\mathcal{N}(0, 0.001^2)$. This choice approximates standard normal initialization.

E DETAILS OF STL FOR FLOWS

To understand the issue with implementing sticking-the-landing (STL) [Roeder et al., 2017, Tucker et al., 2019] estimator for flows, it is helpful to keep two fundamental steps in mind. First, generate a sample z from q. Second, evaluate the density of a sample z under q. Both of these steps are integral to estimate ELBO and estimating its gradients. In most flows, these two steps can be carried out in a single forward pass by keeping track of the Jacobian calculation $|\det \nabla_\epsilon T(\epsilon)|$ (to evaluate the density) while transforming input samples ϵ (to generate the sample).

However, a single-forward pass is insufficient for correct STL calculations and extra care is required. To see why, note the main idea of path-gradients (like STL) is to restrict the dependence of learnable parameters to only the sampling step and to ignore any dependence during density evaluation. To implement this, we need to manipulate the flow of gradients such that the updates only depend on the sampling step. This requirement of treating the parameters differently during these two fundamental steps of sampling and evaluation creates the implementation challenge that forbids a single forward-pass evaluation.

Agrawal et al. [2020] implement STL for flows by implementing the two steps separately. This naive implementation increases memory and almost doubles the run-time. Recent work has improved the STL implementation for flows both in terms of memory and speed, but requires more complicated implementation procedures [Vaitl et al., 2022, 2024]. All of these approaches still require both forward and inverse passes of flow to be efficient [Vaitl et al., 2024]. Thus, despite these advances, autoregressive flows [Kingma et al., 2016, Papamakarios et al., 2017] still can not be practically used with STL as they are only efficient in one direction.

F DETAILS FOR Section 3 EXPERIMENTS

We directly optimize $\mathrm{KL}(p \parallel q)$ using samples from the target density. We use a gradient batchsize of 512×2^{10} samples, that is, at each iteration, we draw a fresh batch of 512×2^{10} samples from the target density to approximate the $\mathrm{KL}(p \parallel q)$ gradients. We optimize using Adam [Kingma and Ba, 2015] and sweep over three step-sizes: $0.0001, 0.0003$, and 0.001, and two step-schedules: decayed and constant. We run the optimization for 100K iterations.

Decayed schedule scales down the step-size by a factor of ten, twice during the optimization: once, after half the number of iterations and again after three-fourth number of iterations. For example, if we start with a step-size of 0.01, the step-size is kept constant for the first half and then updated at the half-way point to 0.001. Then, we keep it constant for the next one-fourth of iterations at 0.001 and update it at the three-fourth point to 0.0001.

We vary the capacity by changing the number of hidden units in the two-hidden layer FNN and the number of coupling layers as described in section D. Each coupling layer corresponds to two affine transformations, resulting in a complete transformation of the input variables (section D for details). For hidden units, we use 2 and 32, and for coupling layers, we use $1, 2, 6, 10$, and14.

G DETAILS FOR Section 4 EXPERIMENTS

For $\mathrm{KL}(q \parallel p)$ optimization, we use a gradient batchsize of 512×2^{10} samples, that is, at each iteration, we draw a fresh batch of 512×2^{10} samples from q to approximate the $\mathrm{KL}(q \parallel p)$ gradients. We use Adam for optimization and sweep over gradient estimators: STL and total gradient, three step-sizes: $0.0001, 0.0003$, and 0.001 and two step-schedules: decayed and constant. We optimize for 100K iterations for all these choices.

For capacity and $\mathrm{KL}\,(p \parallel q)$ results, please see the details in section F.

H DETAILS FOR Section 5 EXPERIMENTS

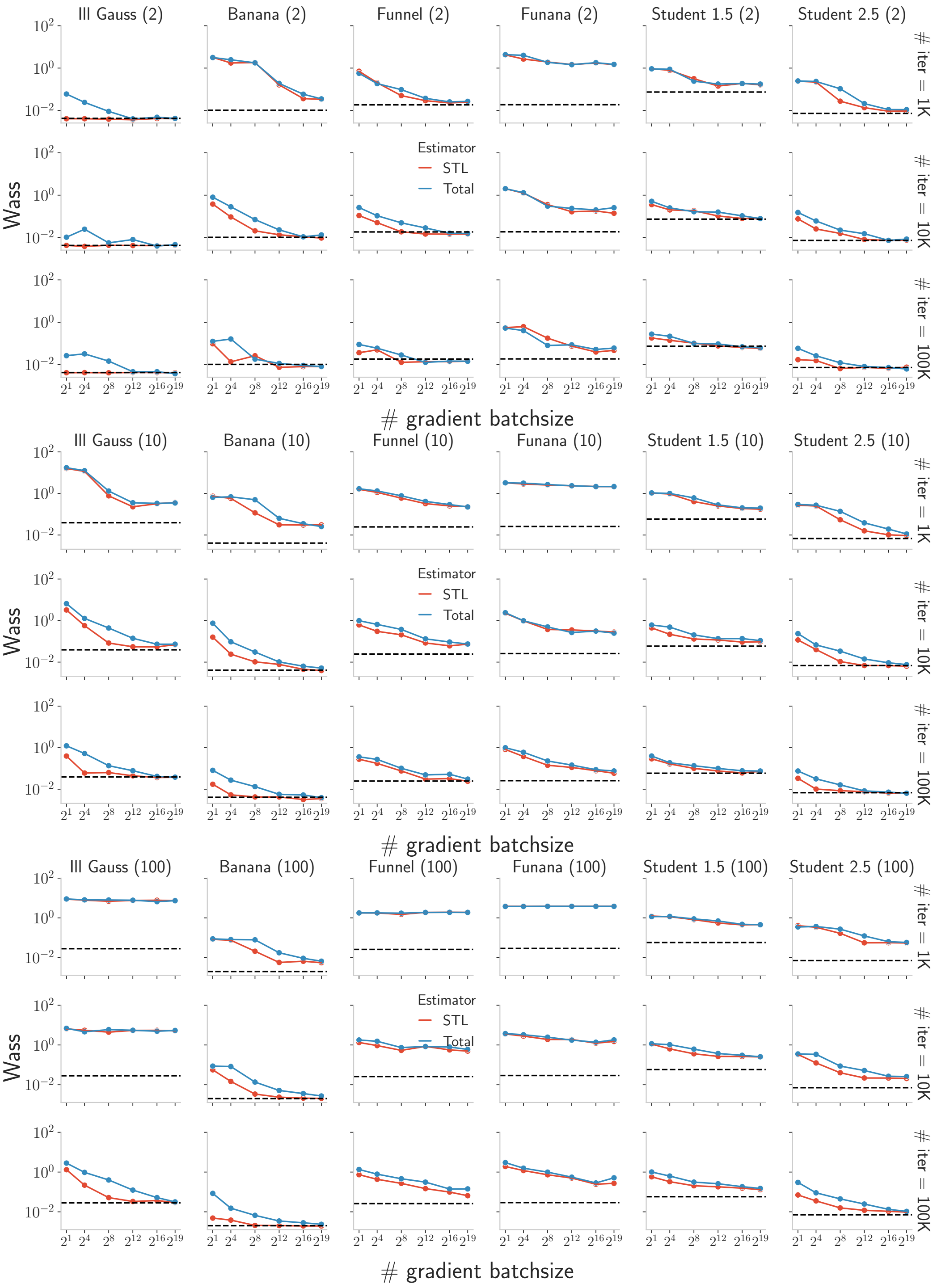


Figure 9: **Rows:** Number of iterations. Dimensions indicated in brackets alongside model names. Figure uses the same setting as Figure 4 and includes the results for different dimensions (fig. 4 uses ten dimensions).

Table 1: We provide the table with scaled wall-clock times for different batchsizes, scaled by the wall-clock time for the batchsize of 2. More precisely, let the total time taken for the batchsize of 16 be T_{16} and for the batchsize of 2 be T_2. Then, the first column below represents T_{16}/T_2. These numbers correspond to results from fig. 4 for $10,000$ update iterations (the number of 100K iterations are very similar. In short, this table shows that for a powerful GPU, we pay a relatively small penalty in wall-clock time for an extremely large increase in the batchsize—*for Nvidia A100, using a batchsize of* 2^{16} *only increase the wall-clock times by 3 to 4 folds compared to batchsize of 2, whereas the batchsize increases by almost 32,000 times.* See table 2 for unscaled wall-clock times.

Model	**# Dim**	T_{2^4}/T_2	T_{2^8}/T_2	$T_{2^{12}}/T_2$	$T_{2^{16}}/T_2$	$T_{2^{19}}/T_2$
Banana	2	1.020	1.072	1.812	1.270	3.857
	10	1.016	0.925	1.475	1.469	5.775
	100	0.998	0.979	1.815	3.119	16.530
Funana	3	1.001	0.947	1.635	1.320	4.335
	10	1.002	0.991	1.482	1.468	5.831
	100	1.013	0.943	1.815	3.154	16.812
Ill Gauss	2	1.005	1.025	2.089	1.247	3.787
	10	1.010	0.918	1.699	1.475	5.859
	100	1.008	0.969	2.376	3.393	18.053
Funnel	2	1.012	1.063	2.102	1.297	3.850
	10	0.996	0.931	1.703	1.495	5.816
	100	0.992	1.015	1.801	3.085	16.452
Student 1.5	2	1.011	1.034	2.084	1.259	3.776
	10	0.993	0.962	1.662	1.454	5.736
	100	1.003	0.951	2.351	3.358	17.712
Student 2.5	2	0.999	1.018	2.072	1.260	3.753
	10	0.996	0.922	1.648	1.444	5.713
	100	1.006	1.001	2.371	3.381	17.877

We use a ten-layered real-NVP flow with 32 hidden units and optimize $\mathrm{KL}\,(q \parallel p)$. We run independent optimization for 1K, 10K and 100K iterations using Adam, and sweep over three step-sizes: $0.0001, 0.0003$, and0.001. Figure 4 uses the decayed step-schedule. Figure 9 plots the results for different dimensions.

We present the scaled and the unscaled wall-clock times for the results corresponding to $10,000$ iterations in table 1 and table 2, respectively. The wall-clock times provide an important observation. When using powerful GPUs like the Nvidia A100, we pay a relatively small penalty in wall-clock time for very large increases in batchsize. For instance, see the scaled times for batchsize of 2^4 and 2^{16} in table 1—a 32,000 times increase in the batchsize results in a 3 to 4 folds increase in the wall-clock times. These benefits disappear for *extremely* large batchsizes like 512×2^{10} but the use of such sizes in exploratory, and recommended in practice (see recipe in section 1). Overall, we believe one should use large batchsizes when the available compute supports it for faster and more accurate results in exchange for a small penalty in wall-clock time.

I DETAILS FOR Section 6 EXPERIMENTS

We use a ten-layered real-NVP flow with 32 hidden units and optimize $\mathrm{KL}\,(q \parallel p)$ using STL. We use a gradient batchsize of 512×2^{10} samples, that is, at each iteration, we draw a fresh batch of 512×2^{10} samples from q to approximate the $\mathrm{KL}\,(q \parallel p)$ gradients. We run independent optimization for 100, 1K, $10K$, 50K, and 100K iterations using Adam. Figure 5 in the paper uses the decayed step-size schedule. Figure 10 plots the results for different step-schedules.

J DETAILS FOR Section 7 EXPERIMENTS

For flow VI, we use a ten-layered real-NVP flow with 32 hidden units, we optimize $\mathrm{KL}\,(q \parallel p)$ using STL estimator. We run independent optimization for 100, 1K, 10K, and 100K iterations using Adam and use a single step-size of

Table 2: Unscaled wall-clock times for the numbers corresponding to table 1.

Model	# **Dim**	T_2	T_{2^4}	T_{2^8}	$T_{2^{12}}$	$T_{2^{16}}$	$T_{2^{19}}$
Banana	2	33.881	34.563	36.332	61.389	43.032	130.688
	10	37.465	38.082	34.642	55.275	55.040	216.347
	100	37.959	37.884	37.161	68.913	118.399	627.483
Funana	3	36.381	36.426	34.463	59.488	48.005	157.704
	10	37.283	37.367	36.958	55.243	54.727	217.384
	100	38.004	38.482	35.830	68.970	119.865	638.935
Ill Gauss	2	34.907	35.076	35.778	72.921	43.512	132.191
	10	37.371	37.745	34.323	63.488	55.136	218.940
	100	37.849	38.153	36.687	89.929	128.410	683.284
Funnel	2	33.999	34.419	36.136	71.463	44.109	130.889
	10	37.557	37.421	34.951	63.951	56.137	218.443
	100	38.358	38.033	38.921	69.101	118.352	631.071
Student 1.5	2	35.204	35.608	36.416	73.350	44.337	132.918
	10	38.242	37.957	36.790	63.554	55.593	219.356
	100	38.578	38.686	36.695	90.687	129.549	683.281
Studentt 2.5	2	35.374	35.332	36.006	73.312	44.571	132.749
	10	38.446	38.294	35.441	63.351	55.500	219.627
	100	38.267	38.489	38.307	90.726	129.397	684.126

3×10^{-4} along with a decayed step-schedule for 100K iterations and a constant step-size of 3×10^{-4} for the rest.

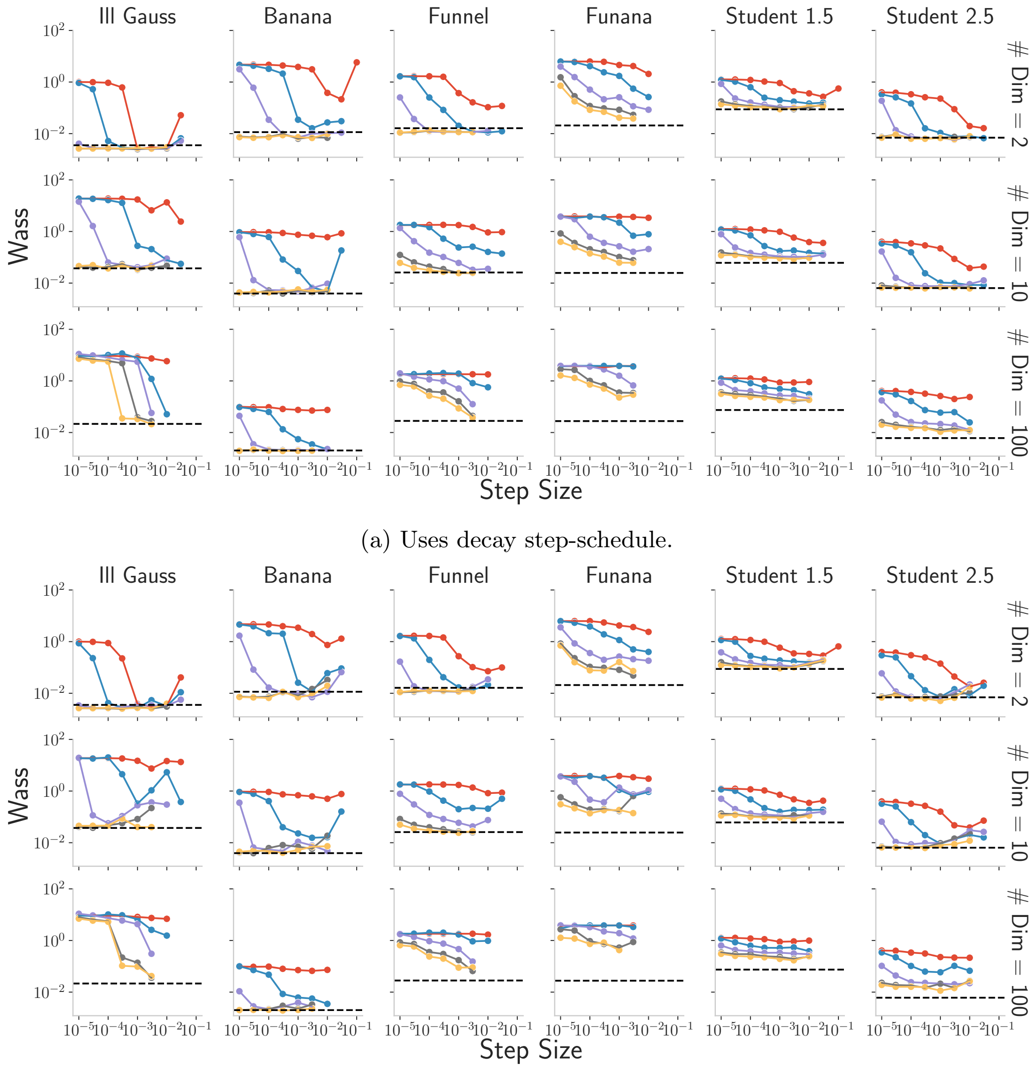
For NUTS as implemented in TensforFlow Probability, we use a target acceptance probability of 0.75, a starting step-size of 0.1, and a maximum-tree-depth of 2^{10} steps. For CHEES as implemented in TensforFlow Probability, we use a target acceptance probability of 0.75 and an initial step-size of 1.0.

For NUTS as implemented in NumPyro, we use a target acceptance probability of 0.9, a starting step-size of 0.1, and a maximum-tree-depth of 2^{10}. We experiment with preconditioning by turning the `adapt_mass_matrix` flag on and off, giving us two variants from NumPyro.

For all HMC methods, we run chains for 10, 100, 1K, and 10K iterations and use as many warm-up iterations, where each iteration can include several leapfrog steps. Currently, most frameworks do not allow tracking the number of leapfrog steps during the warm-up phase. So, we extrapolate the number of leapfrog steps from post-warm-up leapfrog steps of the run with 256 chains and 1K iterations for the synthetic targets and 1024 chains and 1K iterations for the non-synthetic models. While the number of leapfrog steps can be larger during the initial phase, by using "enough" chains for "enough" iterations, we make an honest attempt to estimate the leapfrog steps. Of course, a more sophisticated approach would be to exactly measure every step, without any extrapolation.

For all methods, we experiment with parallel compute budgets. For synthetic targets, we use a budget of $2, 2^8$, and 2^{16}. For VI, this means that we use as many samples for evaluating the gradient, and for HMC, this means that we run as many chains in parallel. HMC chains either collect fewer than or more than a million samples, but not exactly a million samples when run for as many iterations as indicated above (we use a million samples for evaluations). For the chains that collect fewer samples, we simply use as many samples as possible from the chain. For the chains that collect more samples, we use thinning if necessary for memory constraints and use the last million samples collected. For instance, when using a budget of 2^{16} for the non-synthetic models, we run chains for 10K iterations but collect only every 16th sample, resulting in 1048576 samples. Figure 11 plots the results across different dimensions.

For the non-synthetic models, we use a budget of 2, 2^6, and 2^{12}. Again, we use similar procedure as above to get the last million samples from the chains that collect more than a million samples. Figure 12 plots the results for different models. On non-synthetic models, some of the chains get stuck in a region of low probability and do not collect useful samples. We make two plots in fig. 12. The first plot shows the Wasserstein distance when we do



(a) Uses decay step-schedule.

(b) Uses constant step-schedule.

Figure 10: **Rows:** Model dimensions. Figure uses the same setting as Figure 5 and includes the results for different step-schedules (fig. 5 uses decay step-schedule). See section F for details about the decayed step-schedule.

not filter the collapsed chains and the second plot shows the Wasserstein distance when we filter the collapsed chains. For details of the reference samples, see section J.1.

J.1 Reference Samples for Non-Synthetic Models

For non-synthetic models, we need reference samples to evaluate the marginal Wasserstein distance. For this, we run Numpyro's NUTS with preconditioning (as this performed the best based on preliminary experiments) for 2^{14} chains in parallel and 10K iterations (we thin to collect a million samples in total). However, since we select model with non-trivial geometries (see section K.2), we found that several chains were stuck in a region of low probability (indicated by the sample being repeated for the entire duration of the run) and did not collect useful samples. Especially, for the Exoplanet model, the Pharmacokinetics model, and the Sparse Linear Regression model. (We run things in 32-bits. The performance may improve when using 64-bits.) For reference samples, we drop all such instances of stuck chains. This leaves us with at least 497395 for all the models. So, we use 497395 samples for reference samples for all non-synthetic models. When using reference samples, we use as many samples from a method as possible. So, if a method collects more than 497395 samples, we use 497395 samples. If a method collects less than 497395 samples, we use all samples collected.

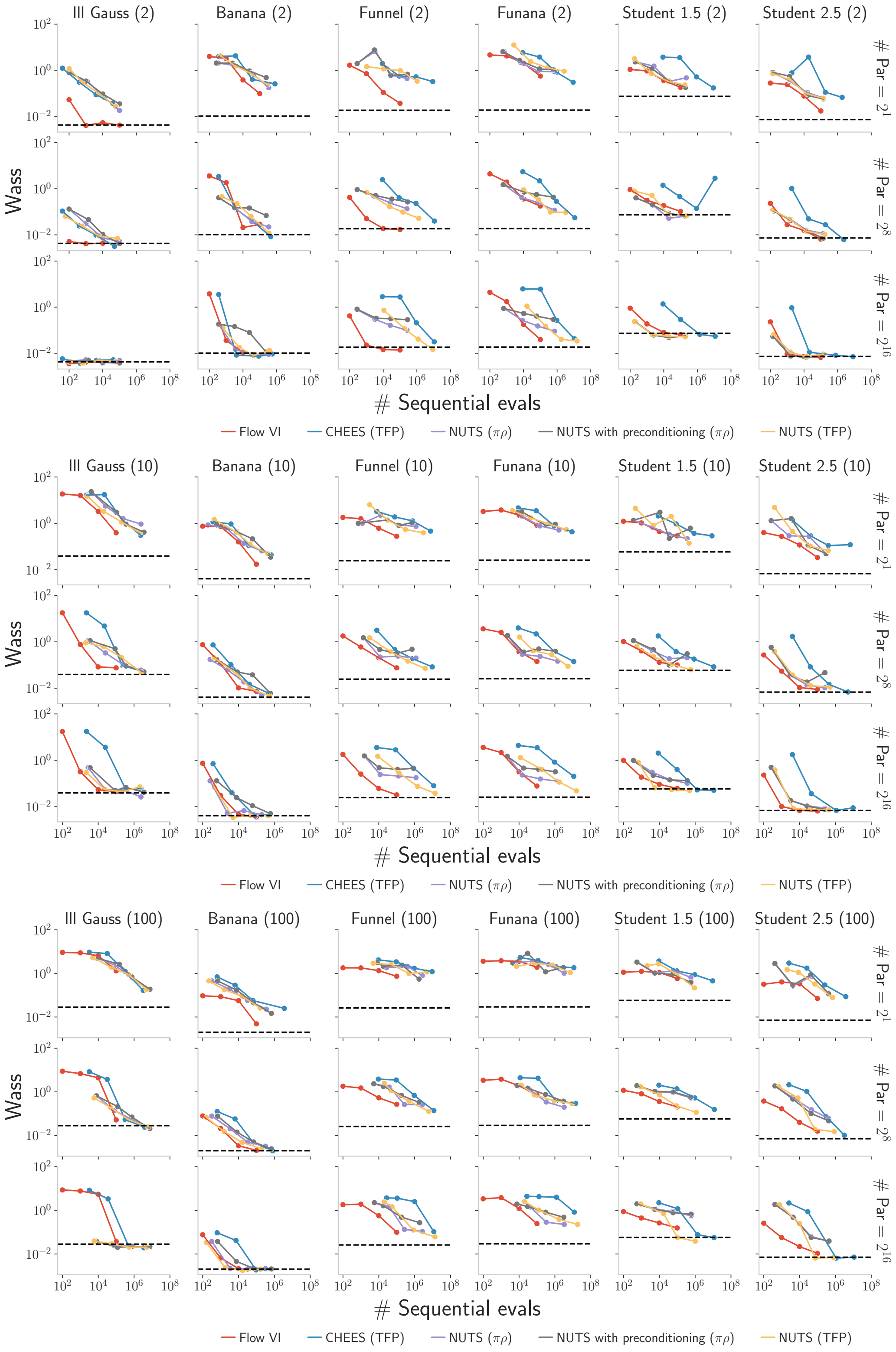


Figure 11: **Rows:** Parallel evaluations. Dimensions indicated in brackets alongside model names. Figure uses the same setting as Figure 6 and includes the results for different dimensions (fig. 6 uses ten dimensions).

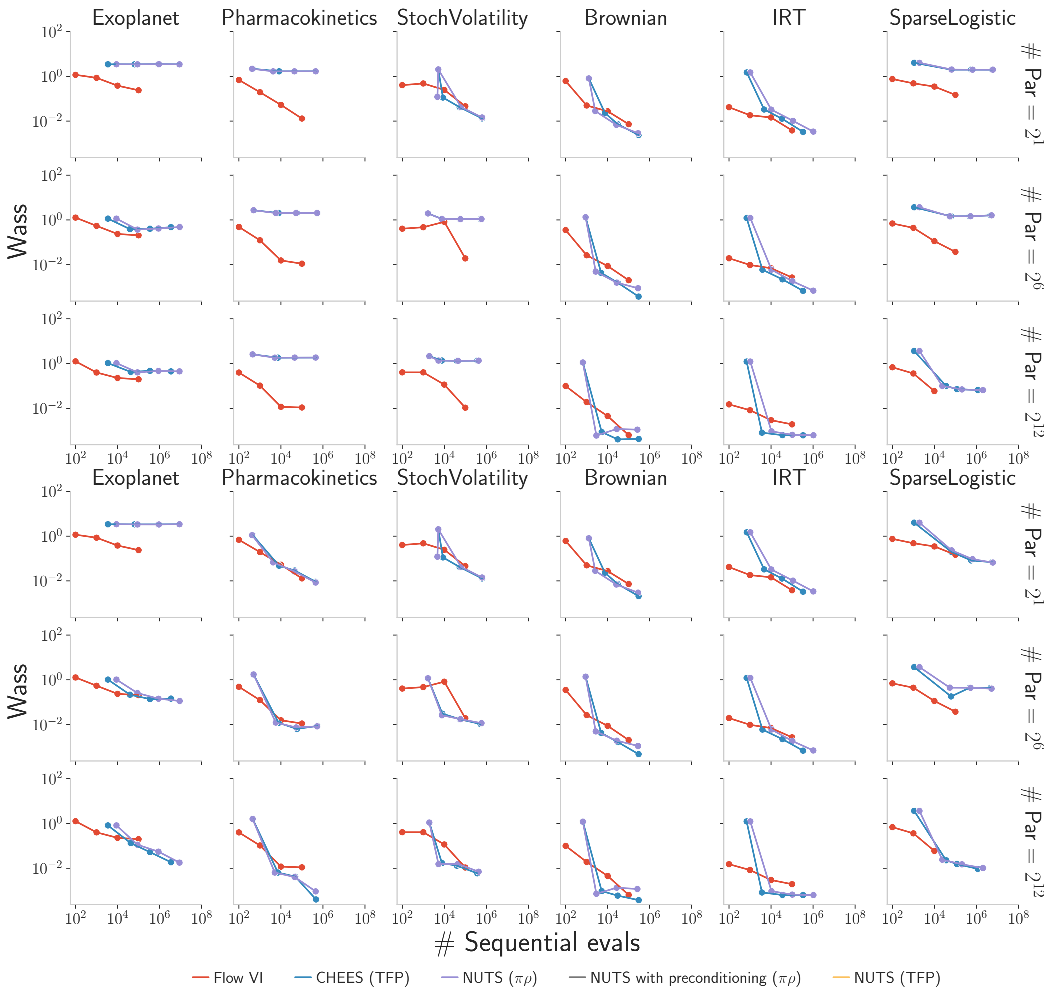


Figure 12: **Rows:** Parallel evaluations. Marginal-Wasserstein metric against number of sequential evaluations for non-synthetic models from Section K.2 with parallel evaluations increasing across the rows. The first subplot corresponds to Wasserstein values when we do not filter the collapsed chains, and the second subplot corresponds to Wasserstein values when we filter the collapsed chains.

K TARGET DETAILS

K.1 Synthetic Targets

Ill-conditioned Gaussian. A multivariate Gaussian with zero mean and a covariance matrix with eigenvalues sampled from a gamma distribution (shape = 0.5, scale = 1) and then rotated by a random orthogonal matrix, ensuring high correlations. We use Inference Gym implementation to ensure reproducibile targets [Sountsov et al., 2020].

Banana [Haario et al., 1999]. This distribution modifies a 2-dimensional normal distribution by applying a non-linear transformation to the second dimension, introducing dependence on the first dimension. The transformation involves a curvature parameter that modifies the scale of the second dimension based on the value of the first dimension. If more than two dimensions are specified, the additional dimensions are modeled as standard normal variables.

For d dimensions, the distribution is

$$\begin{aligned} z_1 &\sim \mathcal{N}(0, 10^2), \\ z_2 &\sim \mathcal{N}(0.03(z_1^2 - 100), 1), \text{ and} \\ z_i &\sim \mathcal{N}(0, 1) \quad \text{for } i = 3, \ldots, d. \end{aligned}$$

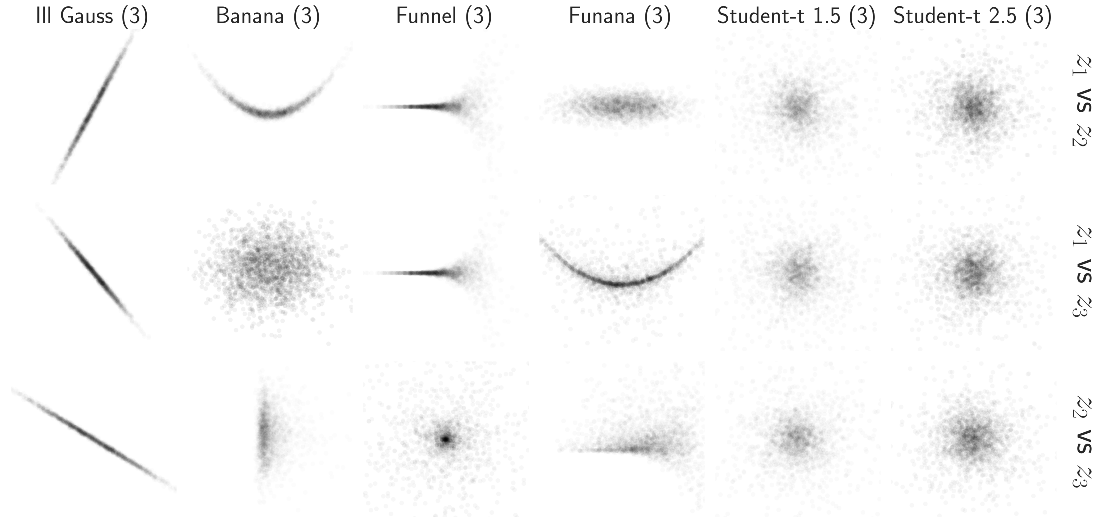


Figure 13: **Rows:** Pair marginals. These targets cover various pathologies: Ill-conditioned Gaussian has high correlations, Banana has non-linear relationships, Neal's funnel has parameters whose spread depends on other parameters, Funana combines funnel-like behavior with non-linearity, and Student-t with $\nu = 1.5$ has heavier tails than Student-t with $\nu = 2.5$.

where z_1 is the primary dimension, sampled from a normal distribution with a mean of 0 and a scale of 10. z_2 is the transformed second dimension, where the mean is adjusted by $0.03(z_1^2 - 100)$, incorporating a quadratic dependence on z_1. z_i for $i \geq 3$ are independent and follow a standard normal distribution, i.e., $\mathcal{N}(0, 1)$.

Neal's funnel [Neal, 2001, 2003]. This distribution is constructed by transforming a multi-dimensional Gaussian distribution, with an initial scale for the first dimension and an exponentially scaled variance for the remaining dimensions. The transformation creates a funnel shape, with a narrow "neck" that poses challenges for some sampling algorithms [Betancourt, 2017], such as HMC. The funnel shape resembles the posterior distributions commonly found in centrally parameterized hierarchical models. For d dimensions, the distribution is:

$$\begin{aligned} z_1 &\sim \mathcal{N}(0, 3^2) \\ z_i &\sim \mathcal{N}(0, \exp(z_1)) \quad \text{for } i = 2, \dots, d \end{aligned}$$

where z_1 is the first dimension, with a normal distribution of mean 0 and scale 3, setting the initial scale for the funnel. z_i for $i \geq 2$ are additional dimensions, with scales that depend on z_1 through the transformation $\exp(z_1/2)$. This transformation introduces increased variance as z_1 grows, widening the distribution as you move away from the origin.

Funana. This new distribution is a hybrid of Neal's Funnel and the Banana distribution, combining features that challenge sampling methods with both a funnel-like narrowing and a non-linear transformation of higher dimensions. The result is a distribution with complex dependencies and variable scaling that can pose significant difficulties for inference algorithms.

For $d \geq 3$ dimensions, the distribution is:

$$\begin{aligned} z_1 &\sim \mathcal{N}(0, 3^2) \\ z_2 &\sim \mathcal{N}(0, 10^2) \\ z_3 &\sim \mathcal{N}(0.03(z_1^2 - 100), \exp(z_1)) \end{aligned}$$

For dimensions $n \geq 4$, we extend z_3 to all additional dimensions, such that,

$$z_n \sim \mathcal{N}(0.03(z_1^2 - 100), \exp(z_1)) \quad \text{for } n \geq 4$$

where z_1 sets the scale for the distribution, much like the leading variable in Neal's Funnel, introducing an exponentially increasing variance for the subsequent dimensions. z_2 follows a normal distribution with a larger scale, contributing to the initial shape of the distribution. z_3 is influenced by both the curvature (as in the Banana distribution) and the scaling behavior (as in Neal's Funnel), making it dependent on z_1 for both mean and variance. Additional dimensions z_n for $n \geq 4$ inherit the complex transformation from z_3, extending the funnel and banana features across higher dimensions.

Funana is challenging to sample from due to the combined narrowing effect of the funnel shape and the curvature introduced by the banana transformation. This mixture of features creates non-linear dependencies and variable scaling, which can confound sampling algorithms.

Student-t. A multivariate Student-t distribution centered at the origin with an identity scale matrix, controlled by the degrees of freedom parameter, ν. The distribution exhibits heavier tails as ν decreases, making it useful for modeling data with outliers or extreme values. We consider two specific cases: $\nu = 1.5$ and $\nu = 2.5$. For $\nu = 1.5$, the distribution has particularly heavy tails, with undefined covariance. For $\nu = 2.5$, the distribution still has heavier tails than a normal distribution, though its covariance exists.

K.2 Non-synthetic Models

We also compare flow VI with HMC on models where the target density is not available in closed-form, and we do not have access to ground truth samples. The details of the models are as follows.

Exoplanet (Dim = 7). This model describes the dimming of a star's light during a transit event, where an exoplanet passes in front of its host star. By analyzing the star's light curve, this model infers properties of the exoplanet, such as its size and orbital period. The model includes both the transit shape and parameters characterizing the exoplanet.

The latent variables governing the transit and exoplanet properties are modeled with the following priors:

$$\begin{aligned}
t_0 &\sim \mathcal{N}(2.26, 1^2), \\
P &\sim \text{LogNormal}(3.66, 0.1^2), \\
D &\sim \text{LogNormal}(0.5, 0.1^2), \\
r &\sim \text{LogNormal}(0.08, 0.1^2), \\
\tilde{b} &\sim \text{Uniform}(0, 1), \quad b = \tilde{b} \times (1 + r), \\
u &\sim p(u),
\end{aligned}$$

where t_0 is the reference time of the transit, assumed to be around 2.26 with a standard deviation of 1. P is the orbital period, with $\log P$ centered at 3.66 and a standard deviation of 0.1. D is the transit duration, with $\log D$ centered at 0.5 and a standard deviation of 0.1. r is the planet-to-star radius ratio, with $\log r$ centered at 0.08 and a standard deviation of 0.1. $\tilde{b}$ is an unscaled impact parameter, uniformly distributed between 0 and 1, which is scaled by $1 + r$ to obtain b. u represents the quadratic limb-darkening parameters, distributed as per Kipping [2013], as implemented in jaxoplanet [Hattori et al., 2024]. The set of latent variables is $z = \{t_0, P, D, r, b, u\}$.

The model predicts the flux y_i^{pred} at each observation time t_i as follows:

$$y_i^{\text{pred}} = L(t_i; z)$$

where $L(t_i; z)$ is the transit light curve model incorporating orbital mechanics and limb-darkening effects [Agol et al., 2020], as implemented in jaxoplanet [Hattori et al., 2024].

The observed flux y_i at time t_i is modeled as a normal distribution around the predicted flux:

$$y_i \sim \mathcal{N}(y_i^{\text{pred}}, 0.03^2)$$

where 0.03 is the observational noise standard deviation.

The joint distribution of the observed flux y and latent variables z is given by:

$$p(y, z) = \left(\prod_i \mathcal{N}(y_i \mid L(t_i; z), 0.03^2) \right) \times p(z)$$

where $p(z)$ represents the prior distributions of the latent variables as specified above.

Observations for y are generated using ancestral sampling over a time range t from 0 to 17 with a step size of 0.05. We condition on data generated from this process. We work in the unconstrained space for ease, so use log-transformations for the parameters P, D, and r.

Pharmacokinetics (Dim = 45)**.** This hierarchical model describes the absorption and distribution of an orally administered drug in the body, using a one-compartment pharmacokinetic system with first-order absorption. The model captures both population-level parameters and individual variability across $n = 20$ patients, enabling it to reflect differences in drug absorption and elimination rates between individuals.

The drug is absorbed from the gut into the bloodstream and distributed to various organs. This process is governed by the following system of differential equations, representing the rate of change in drug concentration over time:

$$\begin{aligned} \frac{dm_{\text{gut}}}{dt} &= -k_1 m_{\text{gut}} \\ \frac{dm_{\text{cent}}}{dt} &= k_1 m_{\text{gut}} - k_2 m_{\text{cent}} \end{aligned}$$

where $m_{\text{gut}}(t)$ is the mass of the drug in the gut compartment. $m_{\text{cent}}(t)$ is the mass of the drug in the central compartment (e.g., bloodstream). k_1 is the rate constant for absorption from the gut to the central compartment. k_2 is the rate constant for elimination from the central compartment.

For $k_1 \neq k_2$, the system has an analytical solution that provides the drug concentration over time:

$$\begin{aligned} m_{\text{gut}}(t) &= m^0_{\text{gut}} \exp(-k_1 t) \\ m_{\text{cent}}(t) &= \frac{\exp(-k_2 t)}{k_1 - k_2} \left(m^0_{\text{gut}} k_1 \left(1 - \exp([k_2 - k_1]t)\right) + (k_1 - k_2) m^0_{\text{cent}} \right) \end{aligned}$$

where m^0_{gut} and m^0_{cent} are the initial amounts of drug in the gut and central compartments, respectively.

Patients receive repeated doses at times 0, 12, and 24 hours. The model incorporates these dosing events by updating the amount of drug in the gut compartment at each dosing time, effectively resetting the boundary conditions for the differential equations. Each patient's observations are taken at various time points after each dose, covering a total time range from 0 to 32 hours.

The model uses a hierarchical Bayesian framework to estimate patient-specific parameters k_1^n and k_2^n, based on population-level parameters with non-centered parameterization

$$\begin{aligned} k_{1,\text{pop}} &\sim \text{LogNormal}(\log 1, 0.1), \\ k_{2,\text{pop}} &\sim \text{LogNormal}(\log 0.3, 0.1) \\ \sigma_1 &\sim \text{LogNormal}(\log 0.15, 0.1), \\ \sigma_2 &\sim \text{LogNormal}(\log 0.35, 0.1), \\ \sigma &\sim \text{LogNormal}(-1, 1) \end{aligned}$$

For each patient n, the individual parameters are given by:

$$\eta_1^n \sim \mathcal{N}(0, 1), \quad \eta_2^n \sim \mathcal{N}(0, 1)$$

$$\begin{aligned} k_1^n &= k_{1,\text{pop}} \cdot \exp(\eta_1^n \sigma_1), \\ k_2^n &= k_{2,\text{pop}} \cdot \exp(\eta_2^n \sigma_2) \end{aligned}$$

The observed drug concentration y_n for patient n at a given time is modeled as:

$$y_n \sim \text{LogNormal}\left(\log m_{\text{cent}}(t; k_1^n, k_2^n), \sigma\right)$$

where $m_{\text{cent}}(t; k_1^n, k_2^n)$ is the concentration in the central compartment, computed using the analytical solution with patient-specific rate constants.

We condition on data generated from ancestral sampling. We work in the unconstrained space for ease, so use log-transformations for the parameters k_1, k_2, σ_1, σ_2, and σ.

Stochastic Volatility (Dim = 103**).** This model captures the volatility of asset prices over time, assuming that volatility follows an AR(1) process with an unknown persistence coefficient and shock scale. The model uses vectorized computations for efficiency and assumes that daily returns are centered around zero with a time-varying volatility. For T timesteps, the model is:

$$\begin{aligned}\phi &\sim 2 \times \text{Beta}(20, 1.5) - 1\\ \mu &\sim \text{Cauchy}(0, 5)\\ \sigma_w &\sim \text{HalfCauchy}(0, 2)\end{aligned}$$

The log volatility h_t follows an AR(1) process:

$$\begin{aligned}h_0 &\sim \mathcal{N}\left(0, \frac{\sigma_w}{\sqrt{1-\phi^2}}\right)\\ h_t &\sim \mathcal{N}(\phi\, h_{t-1}, \sigma_w) \quad \text{for } t = 1, \ldots, T-1\end{aligned}$$

The returns y_t are modeled with a time-varying volatility:

$$y_t \sim \mathcal{N}\left(0, \sqrt{\exp(\mu + h_t)}\right) \quad \text{for } t = 0, \ldots, T-1$$

where ϕ is the persistence of volatility, rescaled to lie between $[-1, 1]$ using a Beta distribution; μ represents the mean log volatility; σ_w is the white noise shock scale, determining the variability of the AR(1) process driving the log volatility; h_t is the latent log volatility at time t, evolving as an AR(1) process; y_t is the centered return at time t, assumed to have a time-varying standard deviation based on the current log volatility.

There are three top-level parameters and 100 per-time-step latent variables, for a total of 103 dimensions. We condition on data y drawn from the prior.

Item Response Theory (Dim = 501**).** This model describes the probability of a set of students answering a set of questions correctly, based on each student's ability and each question's difficulty. The model assumes a one-parameter logistic form, where each student has an individual ability parameter, and each question has a difficulty parameter. Additionally, there is a shared mean ability across all students. For a group of students and questions, the model is:

$$\begin{aligned}\mu &\sim \mathcal{N}(0.75, 1)\\ \alpha_i &\sim \mathcal{N}(0, 1) \quad \text{for } i = 1, \ldots, N_{\text{students}}\\ \beta_j &\sim \mathcal{N}(0, 1) \quad \text{for } j = 1, \ldots, N_{\text{questions}}\end{aligned}$$

For each student-question pair (i, j), the probability of a correct answer is modeled as:

$$y_{i,j} \sim \text{Bernoulli}(\sigma(\alpha_i - \beta_j + \mu))$$

where μ is the mean student ability, shared across all students; α_i is the centered ability of student i, reflecting their specific skill level; β_j is the difficulty of question j, representing how challenging the question is; $y_{i,j}$ is a binary indicator of whether student i answered question j correctly; $\sigma(\cdot)$ is the logistic sigmoid function, translating the linear combination into a probability.

There are $N_{\text{students}} = 100$ students, $N_{\text{questions}} = 400$ questions, and $N_{\text{responses}} = 30105$ responses, for a total of 501 parameters. We condition on data drawn from the prior.

Sparse Logistic Regression (Dim $= 51$)**.** This is a hierarchical logistic regression model with a sparse prior applied to the German credit dataset. We use the variant of the dataset, with the covariates standardized to range between -1 and 1. With the addition of a constant factor, this yields 25 covariates. The model is defined as follows:

$$\begin{aligned}
\tau &\sim \text{Gam}(\alpha = 0.5, \beta = 0.5) \\
\lambda_d &\sim \text{Gam}(\alpha = 0.5, \beta = 0.5) \\
\beta_d &\sim \mathcal{N}(0, 1) \quad (10) \\
y_n &\sim \text{Bern}(\text{sigmoid}(x_n^\top(\tau\beta \circ \lambda)))
\end{aligned}$$

where Gam is the Gamma distribution, τ is the overall scale, λ are per-dimension scales, β are the non-centered covariate weights, $\beta \circ \lambda$ denotes the elementwise product of β and λ, and sigmoid is the sigmoid function. The sparse gamma prior on λ imposes a soft sparsity prior on the weights, which could be used for variable selection. This parameterization uses $D = 51$ dimensions. We log-transform τ and β to make them unconstrained.

Brownian Motion (Dim $= 32$)**.** This model represents a Brownian Motion process with Gaussian observations, where the scale parameters for both the process noise and observation noise are unknown and modeled with log-normal distributions. For T timesteps, the model is:

$$\begin{aligned}
\sigma_{\text{proc}} &\sim \text{LogNormal}(0, 2) \\
\sigma_{\text{obs}} &\sim \text{LogNormal}(0, 2) \\
x_0 &\sim \mathcal{N}(0, \sigma_{\text{proc}}) \\
x_t &\sim \mathcal{N}(x_{t-1}, \sigma_{\text{proc}}) \quad \text{for } t = 1, \ldots, T-1 \\
y_t &\sim \mathcal{N}(x_t, \sigma_{\text{obs}}) \quad \text{for } t = 0, \ldots, T-1
\end{aligned}$$

where σ_{proc} is the process noise scale, which controls the variability in the Brownian Motion from one step to the next; σ_{obs} is the observation noise scale, determining the level of noise in the observed positions; and x_t represents the latent position at time t, while y_t is the observed position subject to observation noise. In this model, we condition on data generated from ancestral sampling. We set the number of timesteps to $T = 30$ and fix the middle ten timesteps to NaNs to simulate missing values.