ASTRA: Understanding the Practical Impact of Robustness for Probabilistic Programs

Zixin Huang[1] **Saikat Dutta**[1] **Sasa Misailovic**[1]

[1]Department of Computer Science, University of Illinois Urbana-Champaign, Urbana, Illinois, USA

Abstract

We present the first systematic study of effectiveness of robustness transformations on a diverse set of 24 probabilistic programs representing generalized linear models, mixture models, and time-series models. We evaluate five robustness transformations from literature on each model. We quantify and present insights on (1) the improvement of the posterior prediction accuracy and (2) the execution time overhead of the robustified programs, in the presence of three input noise models.

To automate the evaluation of various robustness transformations, we developed ASTRA – a novel framework for quantifying the robustness of probabilistic programs and exploring the trade-offs between robustness and execution time. Our experimental results indicate that the existing transformations are often suitable only for specific noise models, can significantly increase execution time, and have non-trivial interaction with the inference algorithms.

1 INTRODUCTION

Probabilistic programming (PP) has recently emerged as a general and flexible approach for Bayesian inference [Goodman et al., 2012, Carpenter et al., 2016, Bingham et al., 2018]. PP decouples model specification from the inference procedures, and thus allows the users to update their models while automatically applying general inference algorithms for Markov Chain Monte Carlo (MCMC) sampling [Robert and Casella, 2013] or Variational Inference (VI) [Beal, 2003]. In recent years, PP has been applied in various real-world machine learning applications, e.g., forecasting [Taylor and Letham, 2018], recommendations in social networks and predicting user locations [Ai et al., 2019, Gokkaya et al., 2018], rating players in games [Gordon et al., 2014], and COVID-19 modeling [Bherwani et al., 2021].

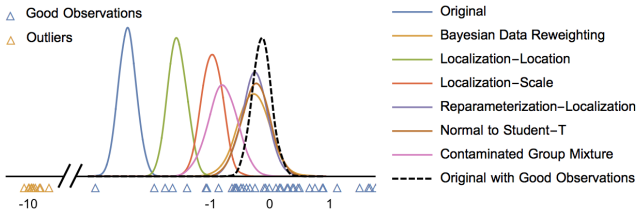

Figure 1: Posteriors of Different Robust Models Fitted with Corrupted Data

Automatically deploying probabilistic programs on such a diverse set of real-world applications raises the question of how much the results of their inferences change in presence of outliers and other deviations of data from model's assumptions (which we summarily call *noise*). *Robustness* is the property of systems, including probabilistic programs, to remain unaffected by data noise [Huber and Ronchetti, 1981]. For instance, many statistical models assume a Gaussian prior or likelihood, but few data points that are far away from the rest can significantly change the inferred mean. In contrast, inference using robust models is more likely to yield posteriors that are not affected by such noise.

Robustness transformations have been traditionally custom-designed for specific models, such as linear regression. Some common transformations can however be applied across different model classes, for instance, by replacing Gaussian likelihood with Student-T. However, it remains unknown (1) which robustness transformation to apply to obtain most robust inference result for a given model and noise model, and (2) how off-the-shelf inference algorithms in popular PP languages interact with these transformations– e.g., which ones have higher execution overhead. Most previous works consider only a few examples, without any thorough comparisons or systematic run time measurements. However, these questions become particularly important when the models are deployed in real-world settings where both accuracy (inference results) and execution cost matter.

Example: How Good are the Robust Models? Figure 1 presents the posteriors fitted on a corrupted dataset when

Accepted for the 39th Conference on Uncertainty in Artificial Intelligence (UAI 2023).

applying different robustness transformations on the original model. The model $y \sim \mathcal{N}(\beta, 1)$ fits the parameter β as the mean of data y. In the dataset, 80% data points are good (non-noisy) observations centered at 0, while 20% are outliers centered at -10, far from the good observations. We present the posterior distributions of β obtained from original and robust models as different lines on the plot.

We observe that all robust models are not equally useful: while some (e.g., Reweighting) yield posteriors that are close to ideal in presence of outliers, some others (e.g., Localization) are not too different than the original (non-robust) model. These observations motivate the need for a systematic study of robustness transformations.

Our Work. The goal of our work is to develop a *systematic understanding* of how various robustness transformations perform in different scenarios through rigorous empirical evaluation on a broad range of subjects. We study the impact on the performance (accuracy) and execution cost of *four factors*: (1) Inference Algorithms, (2) Noise Models and Noise Levels, (3) Model Class, (4) User time budget.

We present the first extensive study of different robustness transformations on 24 probabilistic models from three classes: generalized linear models, mixture models, and time-series models. To help users understand both the practical and fundamental properties of probabilistic robustness transformations, we developed the **ASTRA** framework. ASTRA automatically modifies the program code to apply the robust transformation (and check for its legality) and systematically evaluates different robustness transformations for user-defined input noise models and posterior accuracy metrics. ASTRA then ranks the transformed programs by predictive accuracy. ASTRA is *extensible*: users can easily add new noise models, transformations, and accuracy metrics.

We implemented three common *noise models* for corrupting the datasets (Section 5): (1) *Simple Outliers* randomly changes the value of several data points, (2) *Introducing Hidden Groups* corrupts the data by adding a new distribution mode, and (3) *Skewing Data* adds non-symmetric error to most data points to skew the distribution. We also implemented five robustness transformations from literature for each model (we describe them in Section 3): (1) *Bayesian Data Reweighting* [Wang et al., 2017], (2) *Localization* [Wang et al., 2018], (3) *Robust Reparameterization* combines reparameterization from [Stan User's Guide. Chapter 25.7] with localization, (4) *StudentT* transformation of Gaussian variables, and (5) *Contaminated Group Mixture* [Berger et al., 1994].

We analyze the posterior predictive accuracy of the robustified models and their execution times using two state-of-the-art inference algorithms: No U-Turn Sampler (NUTS) and Automatic Differentiation Variational Inference (ADVI), implemented in Stan [Carpenter et al., 2016].

Results and Insights. Our study yields several interesting insights and observations:

- Different inference algorithms respond differently to each robustness transformation. For instance, for Simple Outliers noise model, Student-T always performs better than Reparameterization for ADVI but for NUTS, Reparameterization outperforms Student-T.
- Robustness transformations can be effective for some noise models – in particular for Simple Outliers – even when 10% of the data has been replaced with outliers the robustness transformations reduce the error by up to 3x, compared to the original program on the same data. However, most transformations do not generalize well across different noise models. For instance, all transformations provide very limited benefits for Hidden Group and Skewed data attacks – this motivates future research to develop novel robustness transformations for these attacks.
- Robustness transformations incur greater overheads for NUTS than ADVI. The run time overheads (over original model) for ADVI range between 1.04x and 7.03x, while for NUTS they are between 1.76x and 14.5x. Hence, some transformations may be impractical in scenarios with tight time budgets. We present more insights in Section 6.

Contributions. This paper makes several contributions:

- **Automated Robustness Evaluation:** We develop ASTRA, a novel automated system that efficiently evaluates the robustness transformations for probabilistic programs.
- **Systematic Evaluation of Robustness:** We present an extensive study of 24 probabilistic programs with multiple robustness transformations, input noise models, and inference algorithms. Our results inform how users select a robustness transformation for their use cases.
- **Insights:** We demonstrate that the robustness transformations can effectively improve predictive accuracy for some models of noisy data, but they may also incur significant execution time overhead. Using ASTRA, we obtained numerous useful insights that are beneficial for both the users and researchers of the probabilistic programming community in particular and AI in general.

ASTRA is open sourced at `https://github.com/uiuc-arc/astra`.

2 ROBUSTNESS METRICS

To evaluate model robustness, we follow the standard methodology in existing research on robust Bayesian modelling [Wang et al., 2017, 2018], by injecting noise in the observed data and computing the relative change in posterior predictive accuracy of the model.

Given a probabilistic program P, and the observed dataset y (we will also call it *uncorrupted*), we fit P to a *corrupted dataset* y^{Noise} that is generated by injecting noise in y. We use the fitted posterior of P to generate predicted data $\hat{y}$,

Table 1: Robustness Transformations: Original and Transformed Models ($\alpha, \beta, s, \eta, \nu, \rho, \sigma, z$ are parameter variable placeholders; y is a data variable placeholder; F, π are distribution placeholders)

(a) Reweighting	(b) Localization	(c) Normal-to-Student-T	(d) Reparameterization	(e) Cont. Mixture
$\beta \sim \pi_\beta(\alpha)$ $y_{i=1}^D \sim F(\beta)$ $\Downarrow$ $w_{i=1}^D \sim \textit{Beta}(\gamma, \zeta)$ $\beta \sim \pi_\beta(\alpha)$ $y_{i=1}^D \sim F(\beta)^{w_i}$	$\beta \sim \pi_\beta(\alpha)$ $y_{i=1}^D \sim F(\beta)$ $\Downarrow$ $\beta \sim \pi_\beta(\alpha)$ $s \sim \textit{Unif}(0,1)$ $\eta_{i=1}^D \sim \mathcal{N}(\beta, s)$ $y_{i=1}^D \sim F(\eta_i)$	$\beta \sim \pi_\beta(\alpha)$ $y_{i=1}^D \sim \mathcal{N}(\beta, \sigma)$ $\Downarrow$ $\nu \sim \pi_\nu(\gamma)$ $\beta \sim \pi_\beta(\alpha)$ $y_{i=1}^D \sim \mathcal{T}(\nu, \beta, \sigma)$	$\beta \sim \pi_\beta(\alpha)$ $y_{i=1}^D \sim \mathcal{N}(\beta, \sigma)$ $\Downarrow$ $\nu \sim \pi_\nu(\gamma)$ $\tau_{i=1}^D \sim \textit{Gamma}(\frac{\nu}{2}, \frac{\nu}{2})$ $\beta \sim \pi_\beta(\alpha)$ $y_{i=1}^D \sim \mathcal{N}(\beta, \frac{\sigma}{\sqrt{\tau_i}}\cdot)$	$\beta \sim \pi_\beta(\alpha)$ $y_{i=1}^D \sim F(\beta, \sigma)$ $\Downarrow$ $\rho_{out}, \eta_{out}, \sigma_{out} \sim \pi(\gamma)$ $\nu \sim \mathcal{N}(\eta_{out}, \sigma_{out})$ $\beta \sim \pi_\beta(\alpha)$ $z_{i=1}^D \sim \textit{Bernoulli}(\rho_{out})$ $y_i \vert z_i = 0 \sim F(\beta, \sigma)$ $y_i \vert z_i = 1 \sim F(\beta, \sqrt{e^\nu})$

and we evaluate the *robustness* of this program through the mean squared error (MSE) metric, as

$$\textit{MSE}(\hat{y}, y) = \frac{1}{D}\sum_{i=1}^{D}(\hat{y}_i - y_i)^2, \tag{1}$$

where D is the size of the dataset. Intuitively, *MSE* quantifies by how much the posterior predictive accuracy changes in presence of data corruptions. Computing *MSE* using predictive data is recommended by [Gelman et al., 2013] as the posterior predictive check to evaluate model fitting and is also used by [Wang et al., 2018] as the predictive R2 metric to evaluate model robustness.

Since the value of *MSE* depends on the scale of data values, we standardize the *MSE*s on the original model, following Saad et al. [2019]. Specifically, let $\textit{MSE}(\hat{y}, y)$ and $\textit{MSE}(\hat{y}_T, y)$ be the estimated robustness of the original model P and a transformed model P_T, respectively. Then we define the *relative improvement of robustness* of the transformed model as:

$$\textit{RIMSE}(\hat{y}, \hat{y}_T, y) = \textit{MSE}(\hat{y}, y) \,/\, \textit{MSE}(\hat{y}_T, y). \tag{2}$$

Intuitively, *RIMSE* denotes the relative improvement of the "robustified" model over the original model. *RIMSE* >1 indicates improved robustness, *RIMSE* of 1 indicates no improvement, and *RIMSE* <1 indicates that the accuracy of robustified model is lower than the original model. In our example (Figure 1), the best transformation (Reweight) yields a RIMSE of 5.22, whereas the least useful transformation (Localization-Location) yields a RIMSE of 1.31.

3 ROBUSTNESS TRANSFORMATIONS

We describe various robustness transformations for probabilistic models from the literature that we use in our study.

Bayesian Data Reweighting. This transformation changes the contribution of each data sample (observation) by raising its likelihood term in the model to its own weight [Wang et al., 2017]. The weights are then exposed as latent variables and inferred along with the rest of the model's parameters. During inference, the outliers are automatically assigned lower weights, which improves prediction. Table 1(a) presents an example of an original model and its transformed version. The transformation introduces a vector of weights w with a Beta prior. $y_i \sim F(\beta)^{w_i}$ denotes that the likelihood F for each data sample is raised to the power of its weight.

Localization. This transformation allows each likelihood term to depend on its own copy of latent variable [Wang et al., 2018]. Table 1(b) presents an example original and robustified probabilistic models. In the transformed model, there are D local versions of the latent variables: η_i, one for each data point y_i. All the auxiliary local variables are sampled from prior π_η. We use a Gaussian prior for η_i in evaluation, following the examples in the original work [Wang et al., 2018]. Unlike [Wang et al., 2018] that designs a specialized E-M algorithm to fit s in the Gaussian prior, we fit s with other parameters via Bayesian inference.

Normal to Student-T. Normal distribution is not robust to outliers or over-dispersed data. An easy alternative is the Student-T distribution [Berger et al., 1994]. Intuitively, the fatter tail of Student-T can better capture the data points far away from the majority. In this transformation, ASTRA replaces a Normal distribution with Student-T by preserving the location and scale parameters in the program, while adding a new parameter ν as the degree of freedom (DOF). Table 1(c) presents the transformation. In the transformed model, ν is from the prior π_ν. Since we may not have prior knowledge, we use a uniform (non-informative) prior for ν.

Reparameterization and Localization of the Scale Parameter. This transformation changes the Gaussian likelihood distribution to an equivalent of Student-T distribution and also localizes the additional parameter τ. Table 1(d) presents an example. The transformation adds D parameters τ_i to adjust the standard deviation of the likelihood for each data point. This has similar effects as the Localization transformation. τ_i is from a *Gamma* prior with hyper-parameter ν. If we integrate out all the τ_is, ν will be equivalent to the DOF parameter in the Normal to Student-T transformation [Stan User's Guide. Chapter 25.7] (but can be more amenable when sampled with MCMC algorithms). This transformation is only applicable for Normal distributions.

Contaminated Group Mixture. To make the model capture a small amount of corruption in data, we can encode in the

model that the data is from a mixture of the original model and a outlier group [Berger et al., 1994]. Table 1(e) presents an example. With probability $1 - \rho_{out}$, the data point is from the original model; with ρ_{out}, the data point comes from another distribution with a different (likely larger) variance. Benefitting from the outlier group, the contaminated data will not directly affect the original model's parameters. ρ_{out} and the scale of the new group are latent parameters which can adapt to the user's data. To ensure a positive scale parameter, we set the outlier group scale parameter to be $\sqrt{e^{\nu}}$ where ν is another hyper-parameter.

4 ASTRA

At a high level, ASTRA takes a probabilistic program P, a dataset y, the desired noise model A, inference algorithm I, and a set of transformations T to apply on P. ASTRA first generates the transformed programs by applying each transformation in T to P. ASTRA then compares each transformed program against the original program and returns the list of transformed programs and their corresponding robustness *scores*, sorted in decreasing order of their robustness.

4.1 PROBABILISTIC PROGRAM TRANSFORMATIONS

Probabilistic Programs. ASTRA takes a probabilistic program (PP) in Stan probabilistic programming language [Carpenter et al., 2016] as input, which is to encode a probabilistic model in the form of a program. Figure 2 shows the Stan program for the original model in the motivating example (Figure 1). The representation is intuitive: the `data` block declares N observations of data y; the `parameters` block declares one parameter `b` in the model; and the `model` block encodes that each data observation is conditional on `b`. Given such a probabilistic program, Stan can automatically apply inference algorithms like MCMC or VI to compute the posterior of parameters.

```
1 data {
2  int<lower=0> N;
3  vector[N] y;
4 }
5 parameters {
6  real b;
7 }
8 model {
9  for (i in 1:N)
10  y[i]~normal(b,1);
11 }
```

Figure 2: Example PP

Transformations. To allow automated transformations on the probabilistic program, we use Storm-IR [Dutta et al., 2019] as our internal representation. Storm-IR can represent program constructs like sampling from distributions (Dist) and conditioning on data (factor) as a graph with program elements as nodes, and control flow as edges (similar to a compiler CFG [Allen, 1970]). Since Storm-IR supports multiple languages (e.g., Stan, Pyro, Edward), it allows ASTRA to be language-agnostic. ASTRA first parses the original probabilistic program into abstract syntax tree and converts to Storm-IR. On this IR, searching for the code pattern from Table 1 amounts to searching for a subgraph that encodes the pattern (e.g., statements corresponding to $\beta \sim \pi_{\beta}(\alpha)$ and $y_{i=1}^{D} \sim F(\beta)$; which do not need be adjacent), while remembering the concrete variable names (e.g., $\beta \mapsto \texttt{b}$, $y \mapsto \texttt{y}$) and distributions (e.g., $F \mapsto \mathcal{N}(\texttt{b}, \texttt{1})$). ASTRA uses the identified distributions/variables to instantiate the transformation template and update the program. For example, to apply the Normal-to-Student-T transformation on Figure 2, ASTRA will replace the normal distribution on Line 10 with a Student-T distribution, as `student_t(nu,b,1)`, where `nu` is a new parameter for the degree of freedom. ASTRA will also place a uniform prior on `nu`.

We show the details of Storm-IR syntax in Appendix A, the code transformation patterns (on Storm-IR) in Appendix B, and the proof of correctness (in the sense of code transformations matching the models from Table 1) in Appendix C.

4.2 ASTRA ALGORITHM

Algorithm 1 presents ASTRA's main algorithm. First, ASTRA initializes a set, *Results*, for storing the robustness scores of all transformed programs (L.2). ASTRA generates the transformed programs $\boldsymbol{P_T}$ (L.3). Next, ASTRA evaluates the robustness of each transformed program (L.4-13). For each transformed program, $P_T \in \boldsymbol{P_T}$, ASTRA performs the following steps `N` times: it first generates a noisy dataset, y^{Noise}, using the specified noise model A (L.7). It runs the inference algorithm I selected by the user to estimate the latent parameters (or posterior data predictions), $\hat{y}$, in program P using the noisy dataset y^{Noise} (L.8). Besides, the user also specifies other inference specifications such as number of samples (for MCMC) or number of iterations (for VI). The *Infer* method encapsulates this step. ASTRA infers the parameters of the transformed program P_T on the same noisy dataset (L.9), and computes the robustness score using the *RIMSE* metric (L.10).

Algorithm 1 ASTRA Algorithm

Input: Program P, Data y, Noise Model A, Inference Algo I, Transformations T
Output: Transformed Programs Ranked by Robustness

1: **procedure** ASTRA(P, y, A, I, T)
2: $\quad$ *Results* $\leftarrow \emptyset$
3: $\quad$ $\boldsymbol{P_T} \leftarrow$ *ApplyTransforms*(P, T)
4: $\quad$ **for** $P_T \in \boldsymbol{P_T}$ **do**
5: $\quad\quad$ *Score* $\leftarrow \emptyset$
6: $\quad\quad$ **for** $i \leftarrow 1$ to `N` **do**
7: $\quad\quad\quad$ $y^{Noise} \leftarrow A(y)$
8: $\quad\quad\quad$ $\hat{y} \leftarrow$ *Infer*(P, y^{Noise}, I)
9: $\quad\quad\quad$ $\hat{y}_T \leftarrow$ *Infer*(P_T, y^{Noise}, I)
10: $\quad\quad\quad$ *Score* $\leftarrow$ *Score* $\cup \{$*RIMSE*$(\hat{y}, \hat{y}_T, y)\}$
11: $\quad\quad$ **end for**
12: $\quad\quad$ *Results* $\leftarrow$ *Results* $\cup \{(P_T,$ *Avg*$($*Score*$))\}$
13: $\quad$ **end for**
14: **return** *Sort*(*Results*)

ASTRA computes the average score (e.g. arithmetic or geometric mean) for the transformed program P_T and appends the result to the *Results* set (L.12). Averaging the scores over multiple runs (and different noisy data-sets) produces a better estimate of the robustness of a transformation. Finally, ASTRA returns the list of transformed programs in descending order of their robustness scores (L.14).

ASTRA also supports other user-specified robustness metrics, which can be specified as a simple function using our python interface. Further, unlike Wang et al. [2018]'s approach that uses only synthetic data (simulated from the original model with known parameters) as y, ASTRA allows users to provide the uncorrupted data as y if the true data model is unknown. Given the uncorrupted data y, ASTRA helps users to compare how different models fit to y.

5 METHODOLOGY

Probabilistic Models. To evaluate the transformations in ASTRA, we obtain a set of 24 probabilistic programs from a popular repository [StanExampleModels] including 13 Regression models, 5 Time-Series/State-Space models, and 6 Mixture models. Table 2 presents the details of all probabilistic programs we evaluate using ASTRA, their description, number of parameters and data items, and run times (in seconds) for ADVI and NUTS inference algorithms with Stan. The TimeSeries models start with "S-", Mixture models with "M-", and Regression models with "R-".

Automated Inference. We use NUTS [Hoffman and Gelman, 2014] and ADVI [Kucukelbir et al., 2015] inference algorithms from Stan [Carpenter et al., 2016] – a popular probabilistic programming language – to run each transformed program and compare their relative behaviors. For NUTS, we run each program with 4 chains, 1000 warmup iterations, and 1000 sampling iterations with a timeout of 8 minutes for each chain. We exclude the programs that timed out. For ADVI, we use 10000 iterations and 1000 posterior samples for comparison. For our evaluation, we use Azure VMs, each with 4 cores, 2.3 GHz CPU, and 16 GB RAM.

Robustness Metric. We use the *RIMSE* metric defined in Section 2. For each model, we repeat generating noise and inference 5 times and compute the geometric mean of *RIMSE* scores. We chose *RIMSE* instead of other information criteria used in [Gelman et al., 2013] for model selection because there are several challenges for applying them on general probabilistic programs: AIC [Akaike, 1974] does not work under strong priors; DIC [Ando, 2010] gives poor results when the distributions are not well summarized by mean; WAIC [Watanabe and Opper, 2010] and Cross-Validation [Stone, 1974] require data partitioning, which is hard to automate for structured models; Bayes factor method [Kass and Raftery, 1995] only works well for discrete models [McElreath, 2020].

Convergence Metric. The sampling-based automated inference may suffer from non-convergence and result in inaccurate estimation of the result. Robustness transformations that introduce new parameters can make the program harder to converge and thus affect its accuracy and robustness. Hence, for evaluation, we also measure the convergence score using Gelman-Rubin Diagnostic [Gelman et al., 2013]. A score significantly larger than 1 indicates non-convergence.

Noise Models. The dataset $\mathcal{D}$ is usually composed of response data (labels) and explanatory data (features) with the same length. In this work, we only add noise to the response data. We select five noise levels for the fraction of perturbed data inputs between 2%, 4%, 6%, 8%, and 10%. Hereon we denote the response data as $\boldsymbol{y}$ and its size as D:

• **Adding Outliers.** We randomly select a subset of data points and add random noise to them. Let $sd(\boldsymbol{y})$ be standard deviation estimated from the original dataset $\boldsymbol{y} = \{y_1, y_2, \ldots, y_D\}$, then we simulate the outliers by:

$$z_{i=1\ldots D} \sim \textit{Bernoulli}(k\%)$$
$$y_i^{\textit{Outliers}} | z_i = 1 \sim \mathcal{N}(c \cdot y_i, |y_i| \cdot sd(\boldsymbol{y}))$$

where $k\%$ corresponds to the amount of noise, can be specified by the user. The constant $c > 1$ allows us generate outliers far from the typical observations. In our experiment, we let $c = k$. This noise model simulates a scenario where some observations get corrupted due to some exception or failure (e.g., of a sensor, storage, or network).

• **Introducing Hidden Groups.** This strategy introduces a hidden group (with its own mode) that does not agree with the modeling assumptions. The location and scale of the hidden group is controlled by the noise level k. Similar to previous case, we allow the user to specify the size of data subset to be changed (e.g., our experiments use $c = 20\%$).

$$z_{i=1\ldots D} \sim \textit{Bernoulli}(c)$$
$$y_i^{\textit{Hidden\_Group}} | z_i = 1 \sim \mathcal{N}(y_i + \frac{k}{2} \cdot sd(\boldsymbol{y}), 0.1k)$$

• **Skewing data.** Using this strategy, we skew the distribution of data points. Skewing causes the mean of the distribution to shift and lose symmetry. It also makes it harder for the inference strategy to sample using a non-robust model. In this work, we apply positive skew to the datasets – for data $\boldsymbol{y}$ we generate skewed data as follows:

$$y_i^{\textit{Skewed}} = \left(\frac{y_i - y_{min}}{y_{max} - y_{min}} \right)^{(1+0.1k)} \cdot (y_{max} - y_{min}) + y_{min}$$

where k is the noise level for this noise model. We first scale all the data to $[0, 1]$, then we raise it to a chosen power to skew the data, and finally scale it back to $y_{min} = \min_{i=1\ldots D} y_i$ and $y_{max} = \max_{i=1\ldots D} y_i$.

A reproducible version of ASTRA is available at `https://figshare.com/s/38668524113696505ef4`.

Table 2: Description of Benchmarks

Prog	Name	Description	#Param	#Data	ADVI	NUTS
RA	anova_radon_nopred_chr	Multi-level linear model with set up for ANOVA with Choo-Hoffman Parametrization	88	919	10.38	20.28
RE	electric_chr	Multi-level linear model with varying intercept with Choo-Hoffman Parametrization	100	192	5.65	12.57
RG	flight_simulator_17.3	Varying intercept model	17	281	4.25	12.28
RK	hiv	Multi-level linear model with varying slope and intercept	173	369	7.00	20.33
RL	lightspeed	Linear model with no predictors	2	66	0.68	0.69
RN	pilots	Multi-level linear model with varying intercept and redundant parameterization	17	40	1.57	3.17
RQ	radon_no_pool	Multi-level linear model without pooling	89	919	12.64	13.60
RR	radon.pooling	Multi-level linear model with complete pooling	3	919	7.18	5.53
RU	radon_vary_si	Multi-level linear model with group level predictors	175	919	14.77	33.95
RV	unemployment	Linear model with one predictor	3	57	0.48	1.46
RW	wells_dae	Logitistic regression model	4	3020	14.06	113.85
RX	y_x	Linear model with one predictor	3	919	7.93	9.17
RY	kidscore_momwork	Linear model with discrete predictor	5	434	3.98	9.26
SA	gp-fit-latent	Gaussian process (GP) with exponentiated quadratic kernel and Gaussian likelihood	104	101	150.63	732.07
SB	stochastic-volatility	Moving average model for time-series	503	500	11.52	170.22
SC	gp-fit-pois	GP with exponentiated quadratic kernel and Poisson likelihood	104	101	108.98	697.92
SD	gp-fit-ARD	GP with ARD-parameterized exponentiated quadratic kernel and Gaussian likelihood	105	101	141.25	1188.97
SE	koyck	Geometric lag time-series	4	200	1.88	6.61
MA	normal_mixture_k_prop	Mixture model with unknown locations, scales and mixing proportion	11	1000	9.98	601.02
MB	normal_mixture_k	Mixture model with unknown locations, scales and mixing proportion	9	1000	1.98	80.11
MC	normal_mixture	Mixture model with known scale	3	1000	25.49	27.91
MD	gauss_mix_asym_prior	Mixture model with non-exchangeable priors	5	100	1.75	3.28
ME	gauss_mix_given_theta	Mixture model with known mixing proportion	4	1000	19.81	54.51
MF	gauss_mix_ordered_prior	Mixture model with ordered priors	5	1000	14.08	30.30

6 EVALUATION

6.1 PERFORMANCE OF TRANSFORMATIONS

We apply the following transformations (discussed in Section 3): Robust reweighting data (*Reweight*), Localization of location parameter (*Local-Loc*), Localization of scale parameter (*Local-Scale*), Reparameterization and Localization of scale parameter (*Reparam*), Normal to StudentT (*StudentT*) and Contaminated Group Mixture (*Mixture*). To evaluate the transformations using ASTRA, we apply 3 different noise models (Outliers, Hidden Groups, and Skewing) on the datasets obtained from 24 programs.

General Trends of Different Transformations. Figures 3 and 4 present the geomean of the relative improvement of MSE (RIMSE) by different robustness transformations at different noise levels for ADVI and NUTS algorithms respectively. Each sub-plot presents the results for one noise model. The X-axis represents the noise level while the Y-axis represents the *geometric mean* of RIMSE over all programs. Each line in the plots represent the performance of one transformation. We also present the MSE of the original (non-robust) program at all noise levels below the X-axis (as "Orig MSE"). The robust transformations reduce MSE by the factor represented on the Y-axis (e.g., up to 3.31x for StudentT transformation for Outliers (ADVI)).

Overall, the transformations are most effective for the *Outliers* noise model. The improvements are significantly smaller for *Hidden Group*. For *Skewed Data* noise model, none of the transformations are effective because the noisy samples are harder to distinguish from typical observations. In general, RIMSE increases with higher noise level, showing that the transformations are more helpful when there is more corruption in the data.

Insight 1. *Our results show that most transformations do not generalize well beyond the Outliers noise model and provide limited benefits. Hence, there is a need to develop novel robustness transformations, especially for Hidden Group and Skewed Data noise models.*

For the Outliers and Hidden Group noise models, *StudentT* transformation is the best in most cases, closely followed by *Reparam*. However, Reparam requires inferring many more parameters than StudentT (e.g., for D data points, StudentT transformation adds one more parameter while Reparam adds $D + 1$ auxiliary parameters), which increases the run time of inference (see also RQ3).

The Local-Loc and Local-Scale transformations provide less protection from noisy data. Local-Scale may help improve the accuracy with NUTS, but it is likely to diverge when using ADVI (Table 4), leading to inaccurate results. One potential cause for this may be that we infer the hyper-parameters for localization transformations in the Bayesian model using automated inference along with other parameters. It may be possible to obtain a better result by applying the E-M algorithm proposed in [Wang et al., 2018], which is customized for each model. However, it is unclear how to automatically apply such an algorithm for general probabilistic programs.

6.2 PREDICTIVE ACCURACY IMPROVEMENT

Table 3 presents the RIMSE scores for all programs with the Outliers noise model at noise level 10 for ADVI and NUTS.

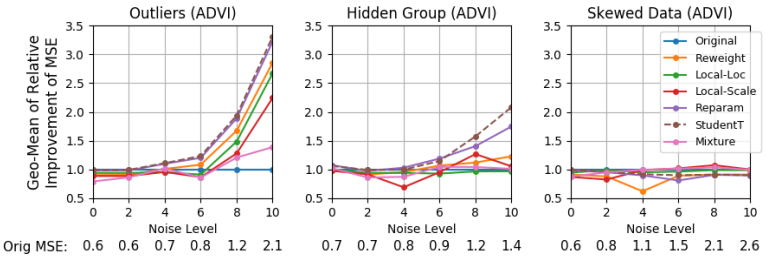

Figure 3: Mean Improvement of Transformed Programs at Different Noise Levels (ADVI)

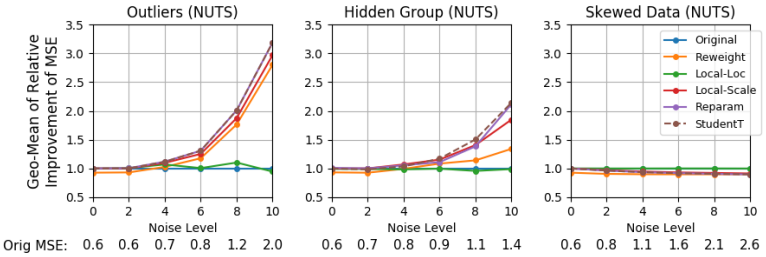

Figure 4: Mean Improvement of Transformed Programs at Different Noise Levels (NUTS)

Table 3: MSE Improvement at Noise Level 10 (Outliers)

Prog	ADVI		NUTS	
RE	256.42	(StudentT)	412.60	(StudentT)
RV	28.04	(StudentT)	31.94	(Reparam)
MC	27.48	(Local1)	1.00	(Original)
SE	14.23	(StudentT)	16.02	(Reweight)
RK	8.41	(StudentT)	9.25	(Reparam)
RN	7.11	(Reparam)	6.25	(Local2)
RU	3.42	(StudentT)	3.75	(StudentT)
RA	3.31	(StudentT)	3.19	(Reparam)
MF	3.27	(StudentT)	2.81	(Reparam)
RQ	3.23	(StudentT)	3.78	(StudentT)
RR	2.95	(Reparam)	3.00	(Reparam)
RX	2.93	(StudentT)	3.18	(Reparam)
SD	2.52	(StudentT)	3.52	(StudentT)
MD	2.21	(Reweight)	6.08	(Reweight)
ME	1.27	(StudentT)	1.41	(Reparam)
RY	1.25	(StudentT)	1.00	(Original)
MB	1.14	(StudentT)	1.22	(StudentT)
RG	1.04	(StudentT)	1.03	(Reweight)
SA	1.02	(Mixture)	1.56	(Reparam)
RW	1.00	(Reweight)	1.00	(Reweight)
SB	1.00	(StudentT)	1.00	(Original)
SC	1.00	(Original)	1.05	(Local1)
RL	1.00	(Original)	1.00	(Original)
MA	1.00	(Original)	1.68	(StudentT)

(R-): Regression, (M-): Mixture, (S-): TimeSeries

Each row represents one program. Each column presents the largest improvement of MSE and the name of the transformation that enabled this improvement in parentheses. For example, "256.42 (StudentT)" means that StudentT is the best among all the transformations (and the original one) and yields 256.42x reduction of the MSE of the original program. For the RIMSE scores with other noise models, see Appendix D. A larger value means the posterior obtained by the transformation is closer to the posterior based on the non-noisy data. “Local1" stands for Local-Loc and “Local2" stands for Local-Scale. We do not apply Hidden Group noise model on Mixture models and Skewed Data noise model on programs with binary data since they are unsuitable. In summary, when using ADVI, StudentT provides the best improvement on 15 benchmarks, followed by Original which is the best on 3 benchmarks, while the other transformations only lead on fewer than 3 benchmarks each. When using NUTS, Reparam is the best on 8 benchmarks; StudentT is the best on 6 benchmarks; Reweight and Original both dominate 4 benchmarks; Local1 and Local2 both dominate one.

Characteristics for Different Model Categories. Generally, Regression (R-) models show the largest improvement in robustness, while Time-Series models (S-) show the smallest improvement. We observe substantial improvements in most linear regression models (e.g. RE and RV), since most transformations are designed for such models. However, for a logistic regression model (RW), we observe very small improvements (up to 1.00x). This is because this model already has a high tolerance for noise compared to other models, since the noise is limited between 0 and 1 for binary data, and thus makes most transformations redundant.

Most Time-Series models model the auto-correlation within data points or fit a correlation matrix for Gaussian processes. As a result, small noise in the data may not affect the fitted correlation. For instance, we observe that the MSE scores of the original models of SA and SB are not affected as the noise level increases. Further, since the robustness transformations are generally designed for exchangeable data [Wang et al., 2017], they are unlikely to work well for many Time-Series models. For instance, for models SC and SD, the transformations are not as effective as other models. Unlike other time-series models, the model SE does not model the correlation but describes a regression equation between the past and current observations and thus can benefit more from the robustness transformations.

Mixture models are less robust to outliers than other classes, because they require fitting a large number of parameters, i.e. the locations, scales, and the probabilities of multiple groups. Several robustness transformations could help fit the locations correctly, however, they tend to classify outliers into one of the groups and infer a less accurate scale or probability, which is the case for models MD, MB, and ME. Also, mixture models are more expensive to fit (due to the label switching problem [Stan User’s Guide. Chapter 23.2]) and thus are likely to diverge when they are not robust to outliers. We observe that at the noise level 10, for the original mixture models, the geomean of the convergence score is 8.94, which is much larger than that of all the original models (2.09). As a result, models like MC and MD can occasionally show a large improvement (up to 27.48x and 6.08x) when the original model diverges due to outliers but the transformed model converges to correct result.

Table 4: (Geometric-)Mean of Convergence Score (Gelman-Rubin Diagnostic) at Noise Level 10

Transformations	Outliers		Hidden Group		Skewed Data	
	ADVI	NUTS	ADVI	NUTS	ADVI	NUTS
Original	2.12	1.60	1.25	1.00	2.15	1.19
Reweighting	1.40	1.15	1.20	1.01	1.36	1.06
Localized-Loc	3.97	1.44	2.13	1.20	5.07	1.31
Localized-Scale	2.77	1.23	1.88	1.05	5.48	1.19
Reparam-Local	2.15	1.34	1.29	1.13	2.27	1.25
StudentT	1.69	1.36	1.15	1.05	1.87	1.35
Cont. Group Mixture	9.41	–	9.53	–	9.94	–

Insight 2. *Overall, we observe the transformations are most useful for most regression models. However, for most time-series models and some classes of regression models, the benefits of transformations are limited since they are already tolerant to input noise. For mixture models, due to the model complexity, transformations generally have less protection against noise, however, they might occasionally protect the original model from divergence.*

Convergence of Transformed Models under Noise. Table 4 shows the geometric mean of convergence scores by different transformations averaged by the models with three noise models at noise level 10 when running with ADVI and NUTS. We present the convergence scores at noise levels 2 and 6 in Appendix E.

The convergence score with NUTS is generally better than that with ADVI: for all the transformed programs, the geometric mean of the convergence score for NUTS is 1.33, while for ADVI it is 2.74. We observe that StudentT generally has better average convergence score than Reparam with ADVI. For example, at noise level 10 with Outliers attack, the average convergence score for Reparam is 2.15 while for StudentT it is 1.69 (the lower the better). When using NUTS, the heavy-tailed nature of StudentT can make sampling less efficient [Stan User's Guide. Chapter 25.7]. Hence, StudentT transformation has *worse* convergence score with NUTS than with ADVI. This also explains why StudentT has slightly higher RIMSE than Reparam when using ADVI, while their RIMSEs are similar using NUTS.

The Local-Loc and Local-Scale transformations introduce a strong dependency between the original parameter and new parameters, as described in [Gorinova et al., 2019]. This creates a complex posterior geometry, which is difficult for both algorithms to explore [Stan User's Guide. Chapter 25.7]. For instance, for ADVI, at noise level 10, the geometric mean of the convergence score over all models for Local-Loc is 3.69 while for Local-Scale it is 3.18. NUTS does not work well with mixture models (including the Mixture transformation) [Stan User's Guide. Chapter 23.2]. For ADVI, *Cont. Mixture* provides best improvements only for two models. Finally, good convergence may not necessarily lead to high accuracy. For example, the Reweighting transformation obtains the best convergence score but only provides the best improvement for four models.

Insight 3. *The performance of a transformation depends on both the inference algorithm and the convergence quality.*

6.3 THE OVERHEAD OF ROBUSTNESS

Overhead of Transformations for Different Model Categories. Table 5 presents the time overhead for different transformations using ADVI and NUTS (over the original program). We divide the benchmarks into three categories: generalized linear models (GLM), Time-Series (TS) and Mixture Models (Mix). The overhead is calculated by dividing the run time of a transformed model by the run time of the original model, and then computing geometric mean over all the benchmarks in the corresponding category. For instance, applying Reweight on GLM is 2.25x times (on average) slower than running the original program with ADVI. NUTS generally has a higher overhead than ADVI. Also, for Mixture Models, the transformations incur the largest overheads among the three model categories, followed by GLM, and Time-Series. The significant increase in execution time for Mixture Models is because the transformations add additional dependency between the parameters in these models, making inference more difficult and slow. On the other hand, since Time-Series Models already have strong dependencies between the parameters, the robust transformations do not affect their execution times much.

Trade-off of Time vs Performance. We evaluate how the choice for best transformation changes (based on posterior predictive accuracy) when the user has limited time budget. For this experiment, we consider different overhead time budgets (from 1x to ∞). For each budget, we filter out transformations that exceed the budget and choose the best transformation among the rest. Figures 5 and 6 present the results for ADVI and NUTS respectively, for the Outliers noise model. The X-axis represents time budgets. The Y-axis represents percentage of models for which a transformation obtained the best improvement in predictive accuracy. Each line shows the mean across all noise levels for either a transformation or the original model.

For lower time budgets (1-3x), the transformations often produce unacceptable execution overheads, which makes the original model more preferable than the transformed models, especially for NUTS. For ADVI, we observe that StudentT consistently dominates other transformations across all overhead budgets, while Reparam and Reweight assume the second place in most cases and yield best results for similar number of cases. For NUTS, StudentT and Reweight provide better gains than Reparam for overhead budgets of up to 10x. However, for higher budgets, Reparam dominates Reweight and shows closer performance to StudentT.

Insight 4. *Overall, since we observe a larger variance of overheads for NUTS, the users should carefully select a robustness transformation based on the maximum tolerable execution overhead in their applications.*

Table 5: GeoMean Time Overhead

Transforms	ADVI			NUTS		
	GLM	TS	Mix	GLM	TS	Mix
Reweight	2.25x	1.60x	3.13x	6.41x	1.76x	4.88x
Local-Loc	1.73x	1.79x	6.01x	14.75x	3.30x	18.12x
Local-Scale	2.34x	1.81x	6.84x	22.75x	8.00x	30.15x
Reparam	2.47x	1.75x	7.03x	13.05x	3.64x	14.50x
StudentT	1.24x	1.04x	2.21x	6.13x	2.44x	3.15x
Mixture	4.05x	2.70x	–	–	–	–

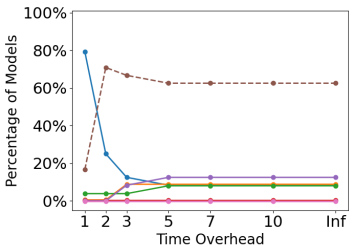

Figure 5: (Outliers) ADVI

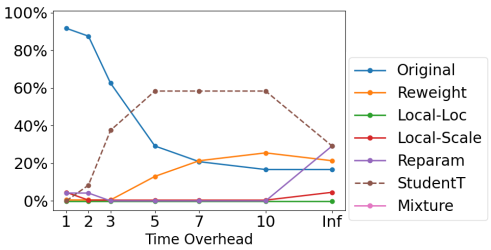

Figure 6: (Outliers) NUTS

7 DISCUSSION

Since *MSE* supports a wide variety of model classes and is easily automated, we only report the evaluation result based on *MSE* in this work. However, our methodology may not tell the entire story about the robustness for all models: (1) it does not take into account the uncertainty in predictions; and (2) it does not use a held-out test set to compute the MSE.

Therefore, after the posterior predictive checks in ASTRA show how well the robustness transformations perform, we suggest ASTRA users to further evaluate the subset of well-performing transformations (semi-automatically) for specific model classes using specialized methods (e.g., predictive on test data, cross-validation, sensitivity analysis, etc.) [Gelman et al., 2013]. These methods can play a complementary role to ASTRA automated analyses.

For example, one may further study the model performance on future observations using the following procedure: if ASTRA shows a poor fit (high *MSE*) using its noise models, then it indicates that prediction of future data is also likely highly inaccurate. If ASTRA shows a good fit (low *MSE*), it means one could also apply other analyses to improve user confidence in the model. In particular, for regression models, one could conduct a cross-validation by splitting the existing data into train/validation/test. For time-series models, one can manually split the data and apply the leave-future out (LFO) cross-validation [Bürkner et al., 2020].

We anticipate our work and further automation of applying model robustness transformations and testing for model robustness can lead to future works on 1) general techniques for improving PP robustness, 2) libraries of techniques applicable for specific, but broad, classes of probabilistic models. In addition, we believe that symbolic techniques for robustness analysis and inference (e.g. [Huang et al., 2018, 2021]) can further help improve the reliability of the implementations of robust probabilistic programs.

8 RELATED WORK

Robust Probabilistic Modeling. We evaluated various robustness transformations previously proposed in literature [Wang et al., 2017, 2018, Berger et al., 1994]. These works did only evaluate small number of programs which makes the generality of their methods unclear. Wang et al. [2017] evaluated on six models, and they compared to [Wang et al., 2018] on a single model. Wang et al. [2018] evaluated their method on four models. Also, these works did not the report run time of the robust models. In addition to these approaches, Futami et al. [2017] proposed a robust version of KL divergence to make variational inference robust of outliers. Their approach focuses on making the inference more robust instead of the model itself. Gbohounme et al. [2017] proposed special measures to improve the robustness of logistic regression models.

Robustness of Neural Networks. Despite their tremendous success in various domains, neural networks are known to be vulnerable to adversarial examples. Researchers have proposed ways to both design attacks for testing the robustness of neural networks [Carlini and Wagner, 2017, Gopinath et al., 2017] and defending against adversarial observations [Gu and Rigazio, 2014, Papernot et al., 2016, Shaham et al., 2018]. In this work, we consider multiple attack (or noise) models used previously for probabilistic models. The source of our noise may not necessarily be adversarial but may stem from practical sources such as erroneous measurements, data corruptions, or random failures.

9 CONCLUSION

We presented ASTRA – the first system for automatically evaluating the robustness of probabilistic programs against various noise patterns. Our study on 24 benchmarks is the first systematic study of robustness of a diverse set of probabilistic programs. We highlight the benefits of robustness transformations when applied on different models. We show the tradeoffs between the level of robustness and the time overhead for different transformations, which can allow users deploy those robust model versions that best fit their needs.

ACKNOWLEDGEMENTS

This research was supported in part by NSF Grants No. CCF-1846354, CCF-1956374, CCF-2008883, a research award from C3.ai Digital Transformation Institute, and Facebook PhD Fellowship.

References

Jessica Ai, Nimar S Arora, Ning Dong, Beliz Gokkaya, Thomas Jiang, Anitha Kubendran, Arun Kumar, Michael Tingley, and Narjes Torabi. Hackppl: a universal probabilistic programming language. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 20–28, 2019.

Hirotugu Akaike. A new look at the statistical model identification. *IEEE transactions on automatic control*, 19(6): 716–723, 1974.

Frances E Allen. Control flow analysis. *ACM Sigplan Notices*, 5(7), 1970.

Tomohiro Ando. *Bayesian model selection and statistical modeling*. CRC Press, 2010.

Matthew James Beal. *Variational algorithms for approximate Bayesian inference*. University of London, 2003.

James O Berger, Elías Moreno, Luis Raul Pericchi, M Jesús Bayarri, José M Bernardo, Juan A Cano, Julián De la Horra, Jacinto Martín, David Ríos-Insúa, Bruno Betrò, et al. An overview of robust bayesian analysis. *Test*, 3(1): 5–124, 1994.

Hemant Bherwani, Saima Anjum, Suman Kumar, Sneha Gautam, Ankit Gupta, Himanshu Kumbhare, Avneesh Anshul, and Rakesh Kumar. Understanding covid-19 transmission through bayesian probabilistic modeling and gis-based voronoi approach: a policy perspective. *Environment, Development and Sustainability*, 23(4):5846–5864, 2021.

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

Paul-Christian Bürkner, Jonah Gabry, and Aki Vehtari. Approximate leave-future-out cross-validation for bayesian time series models. *Journal of Statistical Computation and Simulation*, 90(14):2499–2523, 2020.

Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE, 2017.

Bob Carpenter, Andrew Gelman, Matt Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Michael A Brubaker, Jiqiang Guo, Peter Li, Allen Riddell, et al. Stan: A probabilistic programming language. *JSTATSOFT*, 20(2), 2016.

Saikat Dutta, Wenxian Zhang, Zixin Huang, and Sasa Misailovic. Storm: program reduction for testing and debugging probabilistic programming systems. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 729–739. ACM, 2019.

Futoshi Futami, Issei Sato, and Masashi Sugiyama. Variational inference based on robust divergences. *arXiv preprint arXiv:1710.06595*, 2017.

Idelphonse Léandre Tawanou Gbohounme, Oscar Owino Ngesa, and Jude Eggoh. Self-selecting robust logistic regression model. *Int. J. Statist. Probab.*, 6(3):1–9, 2017.

Andrew Gelman, Hal S Stern, John B Carlin, David B Dunson, Aki Vehtari, and Donald B Rubin. *Bayesian data analysis*. Chapman and Hall/CRC, 2013.

Beliz Gokkaya, Jessica Ai, Michael Tingley, Yonglong Zhang, Ning Dong, Thomas Jiang, Anitha Kubendran, and Arun Kumar. Bayesian neural networks using hackppl with application to user location state prediction. 2018.

Noah Goodman, Vikash Mansinghka, Daniel M Roy, Keith Bonawitz, and Joshua B Tenenbaum. Church: a language for generative models. *arXiv preprint arXiv:1206.3255*, 2012.

Divya Gopinath, Guy Katz, Corina S Pasareanu, and Clark Barrett. Deepsafe: A data-driven approach for checking adversarial robustness in neural networks. *arXiv preprint arXiv:1710.00486*, 2017.

Andrew D Gordon, Thomas A Henzinger, Aditya V Nori, and Sriram K Rajamani. Probabilistic programming. In *FoSE*, 2014.

Maria I Gorinova, Dave Moore, and Matthew D Hoffman. Automatic reparameterisation of probabilistic programs. *arXiv preprint arXiv:1906.03028*, 2019.

Shixiang Gu and Luca Rigazio. Towards deep neural network architectures robust to adversarial examples. *arXiv preprint arXiv:1412.5068*, 2014.

Matthew D Hoffman and Andrew Gelman. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *Journal of Machine Learning Research*, 15 (1):1593–1623, 2014.

Zixin Huang, Zhenbang Wang, and Sasa Misailovic. Psense: Automatic sensitivity analysis for probabilistic programs. In *16th International Symposium on Automated Technology for Verification and Analysis*, ATVA, 2018.

Zixin Huang, Saikat Dutta, and Sasa Misailovic. Aqua: Automated quantized inference for probabilistic programs. In *Automated Technology for Verification and Analysis: 19th International Symposium, ATVA 2021, Gold Coast, QLD, Australia, October 18–22, 2021, Proceedings 19*, pages 229–246. Springer, 2021.

Peter Huber and Elvezio Ronchetti. *Robust statistics*. Wiley, New York, 1981.

Robert E Kass and Adrian E Raftery. Bayes factors. *Journal of the american statistical association*, 90(430):773–795, 1995.

Alp Kucukelbir, Rajesh Ranganath, Andrew Gelman, and David M. Blei. Automatic variational inference in stan. In *NIPS*, 2015.

Richard McElreath. *Statistical rethinking: A Bayesian course with examples in R and Stan*. Chapman and Hall/CRC, 2020.

Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 582–597. IEEE, 2016.

Christian Robert and George Casella. *Monte Carlo statistical methods*. Springer Science & Business Media, 2013.

Feras A Saad, Marco F Cusumano-Towner, Ulrich Schaechtle, Martin C Rinard, and Vikash K Mansinghka. Bayesian synthesis of probabilistic programs for automatic data modeling. *Proceedings of the ACM on Programming Languages*, 3(POPL):37, 2019.

Uri Shaham, Yutaro Yamada, and Sahand Negahban. Understanding adversarial training: Increasing local stability of supervised models through robust optimization. *Neurocomputing*, 307:195–204, 2018.

Stan User's Guide. Chapter 23.2. Label Switching in Mixture Models. `https://mc-stan.org/docs/stan-users-guide/label-switching-problematic.html`, 2023. Version 2.32.

Stan User's Guide. Chapter 25.7. Reparameterization. `https://mc-stan.org/docs/stan-users-guide/reparameterization.html`, 2023. Version 2.32.

StanExampleModels, 2018. `https://github.com/stan-dev/example-models`.

Mervyn Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the royal statistical society: Series B (Methodological)*, 36(2):111–133, 1974.

Sean J Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018.

Chong Wang, David M Blei, et al. A general method for robust bayesian modeling. *Bayesian Analysis*, 13(4):1159–1187, 2018.

Yixin Wang, Alp Kucukelbir, and David M. Blei. Robust probabilistic modeling with bayesian data reweighting. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML, 2017.

Sumio Watanabe and Manfred Opper. Asymptotic equivalence of bayes cross validation and widely applicable information criterion in singular learning theory. *Journal of machine learning research*, 11(12), 2010.