AquaSense: Automated Sensitivity Analysis of Probabilistic Programs via Quantized Inference

Zitong Zhou(✉), Zixin Huang(✉), and Sasa Misailovic(✉)

University of Illinois Urbana-Champaign, Urbana, IL, USA
{zitongz4,zixinh2,misailo}@illinois.edu

Abstract. We propose a novel tool, AquaSense, to automatically reason about the sensitivity analysis of probabilistic programs. In the context of probabilistic programs, sensitivity analysis investigates how the perturbation in the parameters of prior distributions affects the program's result, i.e., the program's posterior distribution. AquaSense leverages quantized inference, an efficient and accurate approximate inference algorithm that represents distributions of random variables with quantized intervals. AquaSense is the first tool to support sensitivity analysis of probabilistic programs that is at the same time symbolic, differentiable, and practical.

Our evaluation compares AquaSense with an existing system PSense (a system that relies on fully symbolic inference). AquaSense can compute the sensitivity of all 45 parameters from 12 programs, compared to 11/45 that PSense computes. AquaSense is particularly effective on programs with continuous distributions: it achieves an average speedup of 18.10× over PSense (which, in contrast, can solve only a handful of problems). Our evaluation shows that AquaSense computes exact results on discrete programs. On 91% of evaluated continuous parameters, AquaSense computed the sensitivity results within 40 s with high accuracy (below 5% error). The paper also discusses AquaSense's performance-accuracy trade-offs, which can enable different operational points for programs with different input data sizes.

Keywords: Probabilistic Programming · Sensitivity Analysis · Quantized Inference

1 Introduction

Probabilistic programming (PP) provides an intuitive way to encode statistical models in the form of programs. It is a quickly rising discipline that has seen applications in areas like computer vision [22], robotics [25], scientific simulation [4], and data science [28]. Probabilistic programming allows a developer to encode uncertainty in the program as random variables. When declaring random variables, the developer specifies the *prior* beliefs of the random variables using probability distributions and encodes the model in the program by relating the random variables to data observations. The developer then makes queries

É. André and J. Sun (Eds.): ATVA 2023, LNCS 14216, pp. 288–301, 2023.
https://doi.org/10.1007/978-3-031-45332-8_16

about the *posterior* distribution of these random variables after execution of the program.

When developing a probabilistic program, developers need to make assumptions regarding the model and the data on which the inference is performed (e.g., a common assumption is Gaussian distributions with a fixed variance). However, it is unknown how reliable these assumptions are. Many studies have reported that a wrong prior could lead to incorrect results [5,21,27]. Testing the sensitivity of the parameters of prior distributions is a way to identify such incorrectly-chosen priors and improve the underlying statistical model.

In this work, we focus on the sensitivity analysis of probabilistic programs, which addresses the question: *if we change the prior distribution, how will the posterior distribution of random variables change?*

AquaSense. We present AquaSense, an automated tool for efficient and accurate sensitivity analysis of probabilistic programs. AquaSense takes a probabilistic program as input, injects a symbolic perturbation ϵ for each prior parameter in the program, then simulates the change in the posterior distribution due to the ϵ values.

At its core, AquaSense leverages *quantized inference* of probabilistic programs. Quantized inference splits the values of continuous random variables into finite intervals and thus works around intractable integrals [17]. Our quantization-based sensitivity analysis can solve a significantly broader range of probabilistic programs than existing tools, while guaranteeing the point-wise convergence of the result sensitivity for continuous programs and small error in practice.

Results. We compare our approach to PSense [19], a system for exact sensitivity analysis, which uses PSI [14], an exact symbolic inference engine, together with a computer algebra system to computes a symbolic and exact sensitivity function.

We evaluated AquaSense on 12 probabilistic programs and analyzed the sensitivity of 45 prior parameters. Results show that AquaSense computes the sensitivity of all 45 parameters, compared to 11 by the baseline PSense. On all 11 discrete parameters, AquaSense produces exact results with comparable performance with PSense. On 34 continuous parameters, AquaSense achieves an average speedup of 18.10× over PSense. On 31 (91%) continuous parameters, AquaSense produces results within 5% of relative error in 40 s, averaging 5.89s. We also show that the time-accuracy trade-off of AquaSense is reasonable.

Contributions. We summarize our contributions as follows:

1. We design and build a quantization-based sensitivity analyzer AquaSense for real-world probabilistic programs. AquaSense supports multiple front-end languages and leverages quantized inference to analyze models that are out of reach of existing tools.
2. We formally prove the point-wise convergence of AquaSense analysis to the exact analysis results on continuous programs with bounded support. We present empirical evidence that AquaSense is exact on discrete programs.

Listing 1.1. Example Prob. Program

```
1  # Data observations
2  vector x[N] = [3.0,...]
3  vector y[N] = [0.6094,...]
4
5  # Model
6  b0 ~ uniform(-1, 1)
7  b1 ~ uniform(-1, 1)
8  sigma ~ uniform(0, 2)
9  for (i in 1:N)
10     y[i] ~ normal(b0+x[i]*b1, sigma)
```

3. We experimentally show AquaSense supports a broader set of continuous programs and achieves orders-of-magnitude speedup than the existing tool PSense, while having comparable capability and speed on discrete programs.

Availability. Latest source code and artifact is available at https://github.com/uiuc-arc/aquasense.

2 Example: Sensitivity-Driven Development

Probabilistic programming is an intuitive way to express a statistical model as a computer program. Listing 1.1 shows such a simple probabilistic program representing a regression model. Suppose we observed a dataset with pairs of `x` and `y`, and we want to fit a line `y = b0 + x * b1` to the dataset, where `b0` (the slope) and `b1` (the intercept) are unknowns. We write such a probabilistic program to solve the distributions of `b0` and `b1` in the program.

In the program, we first specify the prior distributions of intercept `b0`, slope `b1`, and standard deviation `sigma` as uniform distributions, indicating they are equally likely everywhere on their support `[-1,1]` and `[0,2]` (Lines 6–8). Next, we specify that each datum `y[i]` is drawn from a normal distribution with mean `b0 + x[i] * b1` and standard deviation `sigma` (Lines 9–10). In Bayesian terms, in each iteration, we update our belief (prior) about the slope, intercept, and error, upon learning that the datum `y[i]` follows the specified distribution. In the end, the program is represented by a joint posterior probability density $f(\texttt{b0}, \texttt{b1}, \texttt{sigma})$. Given a probabilistic program, probabilistic systems can automatically compute the joint probability density defined by the program.

Choosing Prior Parameters. In this program, the developer chose a uniform prior to reflect the lack of a prior knowledge of `b1`. However, when choosing the parameters of the uniform prior - the lower and upper bounds (marked in **brown** in Listing 1.1) - the developers are unaware of how such ad-hoc decisions would affect the final result. Given the program as input, AquaSense can automatically

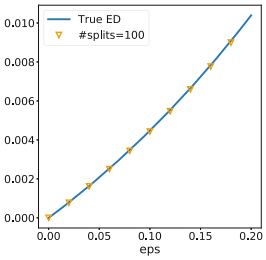


Fig. 1. AquaSense vs. True Results

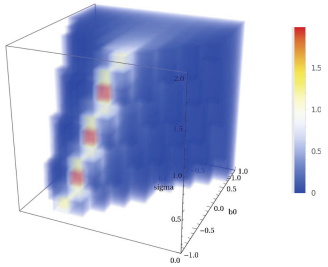


Fig. 2. Density Cube Visualization

test the sensitivity of these parameters, which guide developers to adjust the prior distributions/parameters so that the model's sensitivity is fitting. We detail the example at the end of this section.

Sensitivity Analysis with AquaSense. Given the program (Listing 1.1), AquaSense first performs a pre-analysis to identify the three random variables and their six prior parameters. AquaSense injects noise to test each parameter's sensitivity. For example, to test how the posterior of `b1` changes if its upper bound parameter of the prior `1` is perturbed, AquaSense injects a perturbation parameter ϵ and updates the prior to `b1~uniform(-1, 1+`ϵ`)`.

AquaSense measures sensitivity as the distance between the posteriors with and without perturbation, as in previous works [19]. For simplicity, we use the Expectation Distance (ED) that measures the absolute difference between the expectations of posteriors; AquaSense also supports standards such as Total Variation Distance, Kolmogorov-Smirnov distance [24], and user-defined metrics. The sensitivity of a random variable X measured in Expectation Distance is

$$ED_X(\epsilon) = |\mathbb{E}_{X\sim P(0)}[X] - \mathbb{E}_{X\sim P(\epsilon)}[X]|,$$

where $\mathbb{E}_{X\sim P(\epsilon)}[X]$ and $\mathbb{E}_{X\sim P(0)}[X]$ are the expectations of the posterior distribution of X with and without ϵ added to its prior parameters, respectively.

In the example above, AquaSense would sample evenly-distributed ϵs, whose range can be supplied by the user or inferred by AquaSense using heuristics. It calls AQUA [17], the quantized inference algorithm, to run the programs with and without noise (i.e., $\epsilon = 0$). AQUA would return the approximated posterior distribution density functions, $\hat{p}_{X\sim P(\epsilon)}(x)$ and $\hat{p}_{X\sim P(0)}(x)$, for the programs with and without noise. Next, AquaSense integrates the approximated density functions to get the approximated posterior expectation as $\hat{\mathbb{E}}_{X\sim P(\epsilon)}[X]$ and $\hat{\mathbb{E}}_{X\sim P(0)}[X]$. Because AQUA outputs the posterior densities $\hat{p}.(\cdot)$ as piecewise constant functions, AquaSense can get around integration with summation. In the end, AquaSense computes the approximated $\hat{ED}_X(\epsilon)$. We can show that $\hat{ED}_X(\epsilon)$ could converge pointwisely to the exact $ED_X(\epsilon)$ with more quantization splits (see Sect. 4).

AquaSense outputs the sensitivity of the program as an interpolated function of ϵ. AquaSense can also visualize the distance function by plotting distance against the noise like the yellow markers in Fig. 1. To demonstrate AquaSense's accuracy, we also show the true expectation distance computed manually with a solid blue line in Fig. 1. For this simple example, PSense fails to compute the sensitivity of `b1` (See Sect. 5).

Improving the Program Based on Sensitivity Results. As the function of difference between posterior expectations with respect to perturbation, a steep *ED* indicates the prior chosen is sensitive to perturbation. In the example above, as the developer supplies an upper bound parameter (`1`) to the uniform distribution, the probability will be truncated to zero when `b1` is larger than `1`. If the incoming data exhibit a probability distribution that is "substantial" on $[1, \infty]$, e.g., the part $[1, \infty]$ has more likelihood than the prior support [`-1`,`1`], then the computed posterior will "miss" this part of the likelihood due to the prior. In Fig. 1, AquaSense helps detect that the *ED* is 0.01 when ϵ is 0.2. This means if the developer has chosen a different prior `b1 ~ uniform(-1, 1.2)`, the result expectation of `b1` would change by 0.01, which is negligible for `uniform(-1, 1)`. This result indicates the chosen prior parameter is relatively insensitive to perturbation. In contrast, suppose the sensitivity at $\epsilon = 0.2$ is high, e.g. $ED = 1$, which means changing the prior from `b1 ~ uniform(-1, 1)` to `uniform(-1, 1.2)` would increase the expectation of the posterior of `b1` by 1, so the developer is advised to modify the prior to `b1 ~ uniform(-1, 1.2)` in order to capture the "missing" posterior density of `b1` on $[1, 1.2]$. Sensitivity analysis and prior updates can be applied iteratively this way until sensitivity is deemed suitable.

In conclusion, sensitivity analysis can help a) expose such misses of density outside of the prior support and, b) quantitatively measure its severity. Analogously, sensitivity analysis can also be used to identify other types of poorly-chosen prior parameters, e.g., mean, standard deviation, and degrees of freedom.

3 Background: Automated Inference Algorithms

The goal of probabilistic programming is to compute the joint probability density f. To this end, probabilistic programming languages (e.g., AQUA [17], PSI [15], Stan [6]) are coupled with automated inference algorithms that compute the density f either exactly or approximately. For example, PSI implements exact inference using computer algebra, computes the posterior symbolically via $p(\texttt{b0}, \texttt{b1}, \texttt{sigma}) = \frac{f(\texttt{b0},\texttt{b1},\texttt{sigma})}{\int f(\texttt{b0},\texttt{b1},\texttt{sigma}) d\texttt{b0},\texttt{b1},\texttt{sigma}}$, which requires integration that is often intractable. The prior work PSense uses PSI to compute posterior distributions of probabilistic programs, and thus also suffers from intractable integrals.

AquaSense implements sensitivity analysis on top of AQUA's quantized inference. AQUA approximates the symbolic joint probability density $f(\texttt{b0}, \texttt{b1}, \texttt{sigma})$ with quantized samples, by storing the quantization of f's domain and co-domain in multidimensional arrays. In the example above, AQUA

quantizes `b0`, `b1`, `sigma` into evenly spaced values, e.g., $[-1, -0.8, ..., 0.8, 1]$, when using 10 splits. Then AQUA computes $f(\texttt{b0}, \texttt{b1}, \texttt{sigma})$ at all combinations of the variable values, to obtain a three-dimensional array, called Density Cube. Figure 2 shows a visualization of the Density Cube, with each dimension representing a random variable. Among the 10^3 mini-cubes, a warmer color means higher probability. In AQUA, normalization is reduced to summation over the Density Cube. AQUA outputs the approximated joint posterior density function, denoted $\hat{p}(\texttt{b0}, \texttt{b1}, \texttt{sigma})$.

Alternatives to AQUA include computer-algebra-based exact inference like PSI and sampling-based inference like Stan. Intractability severely limits exact inference to simple models with few continuous distributions (see Sect. 5). Sampling-based inference are not accurate enough for sensitivity analysis, as studies have shown [19] [17]. For the particular task of sensitivity analysis, quantized inference is an ideal candidate as it can get around the intractability problem while being more accurate than sampling-based inference.

4 AquaSense Workflow

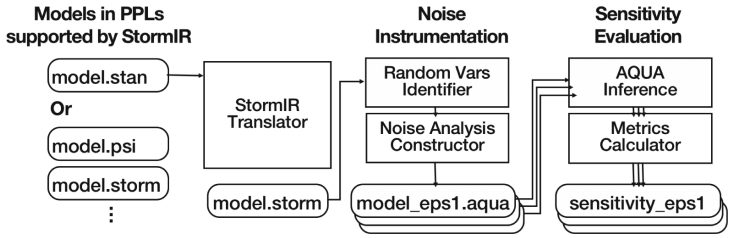


Fig. 3. AquaSense Workflow

Input: A Probabilistic Program. AquaSense takes a probabilistic program in any probabilistic programming language (PPL) supported by StormIR [13] (which is an intermediate probabilistic programming language), including Stan [16], PSI [15], Pyro [26], or StormIR itself. See Fig. 4 for its syntax.

Noise Instrumentation. Given a program P, AquaSense applies a pre-analysis to find the random variables and their prior parameters. The bound of noise of each parameter is user-supplied or computed using a heuristic. For each prior parameter, it generates a new program, as $P(\epsilon)$, by injecting a symbolic noise variable ϵ at the parameter. AquaSense evenly samples a set of values of ϵ from its bounds as $\mathcal{B}$.

$x \in$ Vars
$c \in$ Consts
$op \in \{+, -, *, >, \ldots\}$
$d \in$ {Normal,Uniform,..}

E := c | x | E[E*] | E op E | d(E*).pdf(E*) | f(E*)
S := x = E | $x \sim d$(E*) | factor(E) | observe(d(E*),x)
| if (E) S* else S* | for $x \in$ 1..N; {S*}
P := S+; return x^+

Fig. 4. Syntax of StormIR

AQUA Inference. AquaSense employs AQUA [17], the quantized inference engine, to solve a probabilistic program. AQUA takes a probabilistic program P and outputs the approximated posterior of a random variable x as a piece-wise constant function, denoted as $\hat{p}_{X \sim P}(x)$.

Figure 5 illustrates an example of AQUA analysis results. The red line represents the true density that PSI would calculate, and the gray bars represent AQUA's approximation. With AquaSense noise instrumentation, AquaSense runs AQUA on the program $P(\epsilon)$ with quantized values of ϵ, to simulate the program results due to different ϵ, as $\{\hat{p}_{X \sim P(\epsilon_i)}(x) | \epsilon_i \in \mathcal{B}\}$.

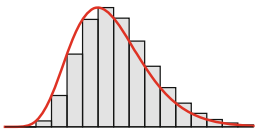


Fig. 5. AQUA Inference Example

Metrics Calculator. Next, AquaSense computes the sensitivity metrics based on inference results, e.g., the Expectation Distance (*ED*) [19], Kolmogorov-Smirnov statistic [24], Total Variation Distance (TVD) or other user-provided metrics. For simplicity, we use *ED* throughout this work. For each $\epsilon_i \in \mathcal{B}$, AquaSense first computes $\hat{\mathbb{E}}_{X \sim P(0)} = \int_x x \cdot \hat{p}_{X \sim P(0)}(x) dx$ and $\hat{\mathbb{E}}_{X \sim P(\epsilon_i)} = \int_x x \cdot \hat{p}_{X \sim P(\epsilon_i)}(x) dx$, and then computes $\hat{ED}_X(\epsilon_i) = |\hat{\mathbb{E}}_{X \sim P(0)} - \hat{\mathbb{E}}_{X \sim P(\epsilon_i)}|$.

Output: Program Sensitivity. Finally, AquaSense interpolates sensitivity as a function of ϵ. Using *ED*, it outputs $ED_X(\epsilon)$ by interpolating $\{\hat{ED}_X(\epsilon_i) | \epsilon_i \in \mathcal{B}\}$. AquaSense allows users to specify the number of ϵ samples and the number of quantization splits used in AQUA to control the analysis' time-accuracy trade-off. This design allows AquaSense to produce accurate sensitivity estimates on a much wider range of probabilistic programs than existing tools.

Formal Guarantee of AquaSense Accuracy. For continuous PPs with bounded support, we formally state the convergence of AquaSense's quantized sensitivity at any concrete $\epsilon \in \mathcal{B}$. For discrete PPs, we show in Sect. 5 with empirical experiments that AquaSense is exact up to machine imprecision. Without loss of generality, we assume AquaSense uses the *ED* metric; and one can show the convergence for other metrics (e.g. KS and TVD) analogously.

Theorem 1. *Given any $\epsilon \in \mathcal{B}$, denote AquaSense output as $\hat{ED}_X^{N,\mathbf{C}}(\epsilon)$, where N is number of quantization splits of each random variable and $\mathbf{C}$ is the bounded domain of all the random variables required by AQUA. Let $ED_X(\epsilon)$ be the exact sensitivity at ϵ. If the support of all the random variables is a subset of $\mathbf{C}$, then*

$$\lim_{N \to \infty} \hat{ED}_X^{N,\mathbf{C}}(\epsilon) = ED_X(\epsilon).$$

We can prove the theorem using the following lemma from [17].

Lemma 1. *Let the posterior density function of the program P computed by AQUA be $\hat{p}^{N,\mathbf{C}}_{X\sim P}(x)$, which defines the cumulative density function (CDF), $\hat{F}^{N,\mathbf{C}}_{X\sim P}(x) = \int \hat{p}^{N,\mathbf{C}}_{X\sim P}(x)dx$. Let the exact CDF of the program be $F_{X\sim P}(x)$. Then by Theorem 1 of AQUA algorithm [17], one can guarantee the convergence in distribution:*

$$\lim_{N\to\infty} \hat{F}^{N,\mathbf{C}}_{X\sim P}(x) = F_{X\sim P}(x).$$

Corollary 1. *Given that* $\mathbf{C}$ *is a* bounded *domain containing all the support of random variables in the program, we can apply the Portmanteau lemma [20] to get the convergence of approximated expectation to the exact one:*

$$\lim_{N\to\infty} \hat{\mathbb{E}}^{N,\mathbf{C}}_{X\sim P}[X] = \mathbb{E}_{X\sim P}[X].$$

Here, $\hat{\mathbb{E}}^{N,\mathbf{C}}_{X\sim P}[X] = \int_{x\in\mathbf{C}_X} x\cdot\hat{p}^{N,\mathbf{C}}_{X\sim P}(x)dx$ will be computed by AquaSense without additional approximation; $\hat{p}^{N,\mathbf{C}}_{X\sim P}(x)$ is a piecewise constant function (output of AQUA), and AquaSense can evaluate the integral with summation. The corollary also holds for $\hat{\mathbb{E}}^{N,\mathbf{C}}_{X\sim P(\epsilon)}[X]$ when we inject a constant value ϵ in the program.

Proof of Theorem 1. AquaSense employs AQUA to compute the posteriors and it sets a hyper-parameter N to be the quantization splits for each random variable. Given that the support of all the random variables is a subset of $\mathbf{C}$, by Corollary 1 and the definition of limits (i.e. the subtraction and absolute rules of limits),

$$\lim_{N\to\infty} |\hat{\mathbb{E}}^{N,\mathbf{C}}_{X\sim P(0)}[X] - \hat{\mathbb{E}}^{N,\mathbf{C}}_{X\sim P(\epsilon)}[X]| = |\mathbb{E}_{X\sim P(0)}[X] - \mathbb{E}_{X\sim P(\epsilon)}[X]|.$$

By definition of *ED*, we prove Theorem 1.

5 Evaluation

Benchmarks. We evaluate AquaSense on a benchmark suite consisted of 12 probabilistic programs: 7 from PSense [19] benchmarks, 3 from AQUA [17], and 2 new programs; they have a total of 11 discrete and 34 continuous parameters. We performed the experiments on AMD Ryzen 7 5800X 8-Core CPU @ 3.00 GHz with 32 GM RAM and one Nvidia Geforce RTX 3090 with 24 GB memory (running Ubuntu 20.04). AquaSense's tensor computation is performed on the GPU.

Accuracy Metrics. For each parameter, we evaluated two metrics: the average absolute error $|\text{Err}| = \frac{1}{|\mathcal{B}|}\sum_{\epsilon\in\mathcal{B}} |ED_X(\epsilon) - ED_{truth}(\epsilon)|$, and average relative error $\text{Err}\% = \sum_{\epsilon\in\mathcal{B}} \frac{|ED_X(\epsilon) - ED_{truth}(\epsilon)|}{|\mathcal{B}|\, ED_{truth}(\epsilon)}$, i.e., the average distance (and its ratio) between AquaSense interpolated *ED* and true *ED*. We consider $\mathcal{B}$ to be a valid set of noises with moderate sensitivity to evaluate both tools. The ground truth of sensitivity ED_{truth} is computed using two methods: a) PSense, b) manually computed with the assistance of Mathematica when PSense fails. Computing the true sensitivity may take hours or days, which adds to the necessity of an automated tool like AquaSense. We discard the sensitivity below the threshold 1e-6 when computing the errors to tolerate machine imprecision.

5.1 Performance and Accuracy of AquaSense

Table 1 presents AquaSense's accuracy and performance compared to PSense. Each row represents a parameter of which AquaSense evaluates the sensitivity. The first three columns shows the **Parameter Properties**: "Prog." shows the name of the program; "Dist." shows the distributions in the program, where prior distributions are underlined; "D/C" shows whether the program is discrete (D) or continuous (C); "Param" shows the parameter to analyze. For example, the program "expl_away" contains four discrete, Uniform Integer distributions ($\mathcal{U}_I$), where two of them are priors ($\underline{\mathcal{U}_I}$). Each Discrete Uniform Integer distribution has two parameters, i.e. the lower and upper bound (lb, ub), so AquaSense analyzed four parameters for this program.

We run AquaSense doubling #splits from 100 until Err% is below 5% or |Err| is below 1e-6 (colored in green), or AquaSense runs out of memory (in red). "#spl" (Column 5) shows the largest #splits that produces the corresponding Err% (Column 7) and |Err| (Column 8). On discrete programs with finite support, AquaSense uses the cardinality of the distribution support as #spl, denoted by *Sup*. The column "Acc." shows if AquaSense is accurate enough (Err% is below 5% or |Err| is below 1e-6). Column "**PSense Time(s)**" shows PSense execution time in seconds. We report a timeout (T.O.) if it exceeds 10 min, and an error (Err.) if the result finished but not solved to closed form. Column "**AquaSense Time(s)**" shows the total time, minus the time to initialize the GPU. Total time include the noise instrumentation time ("NI(s)") and sensitivity evaluation time ("SE(s)"). "**Speedup**" is AquaSense's speedup over PSense.

Capability. Our results show that AquaSense successfully computes the sensitivity of all parameters. In comparison, PSense is only able to solve the sensitivity of 8 out of 11 discrete parameters and 3 out of 34 continuous parameters. We observe that for most continuous program, PSense failed to solve integrals to the closed form, which is the fundamental problem of exact inference, meaning PSense's capability cannot be improved much by simply allocating more time.

Execution Time and Accuracy. Compared to PSense, AquaSense is on average faster by 18.10× on continuous models, and for discrete models has similar execution time (slower by 11%). The maximum speedup is 35.16× (for the "gamma" model). For all the discrete models, AquaSense results are exact (with error smaller than machine imprecision). For 31/34 (91%) continuous parameters, AquaSense has an average relative error less than 5% or an average absolute error less than 1e-6. Two parameters in "tug" show higher relative error as the sensitivities are close to zero (<1e–4), but the absolute error is already at around 1e−3. One parameter in "sgl_reg" has higher (6%) relative error for the same reason. *Overall, AquaSense works on many real-world models out of reach of PSense, and offers orders-of-magnitude speedup at a reasonable cost of accuracy.*

Table 1. Performance of AquaSense vs. PSense

Parameter Properties				AquaSense Accuracy				PSense	AquaSense Performance			
Prog.	Dist.	D/C	Param	#spl	Acc.	Err%	|Err|	Time(s)	Time(s)	NI(s)	SE(s)	Speedup
coins	$\underline{\mathcal{B}^2}$	D	$\mathcal{B}, p$	*Sup*	T	0.00	0.00	1.24	1.38	0.28	1.11	0.90
			$\mathcal{B}, p$	*Sup*	T	0.00	0.00	1.26	1.42	0.28	1.14	0.89
murder	$\underline{\mathcal{B}^3}\times\mathcal{B}^1$	D	$\mathcal{B}, p$	*Sup*	T	0.00	0.00	1.26	1.26	0.19	1.06	1.00
			$\mathcal{B}, p$	*Sup*	T	0.00	0.00	1.07	1.23	0.19	1.04	0.87
			$\mathcal{B}, p$	*Sup*	T	0.00	0.00	1.01	1.33	0.19	1.13	0.76
binomial	$\underline{\mathcal{B}^1_m}$	D	$\mathcal{B}_m, n$	*Sup*	T	0.00	0.00	T.O.	1.81	0.36	1.45	∞
			$\mathcal{B}_m, p$	*Sup*	T	0.00	0.00	1.76	1.81	0.36	1.45	0.97
expl_away	$\underline{\mathcal{U}^2_I}\times\mathcal{U}^2_I$	D	$\mathcal{U}_I, ub$	*Sup*	T	0.00	0.00	Err.	1.34	0.26	1.08	∞
			$\mathcal{U}_I, lb$	*Sup*	T	0.00	0.00	2.27	1.36	0.26	1.11	1.67
			$\mathcal{U}_I, lb$	*Sup*	T	0.00	0.00	2.46	1.35	0.26	1.10	1.82
			$\mathcal{U}_I, ub$	*Sup*	T	0.00	0.00	Err.	1.34	0.26	1.08	∞
gamma	$\underline{\Gamma^1}$	C	Γ, α	12800	T	3.62	0.01	158.63	8.64	0.09	8.55	18.36
			Γ, β	100	T	2.68	0.01	39.29	1.12	0.09	1.03	35.16
true_obs	$\underline{\mathcal{N}^2}\times\mathcal{N}^1$	C	$\mathcal{N}, \sigma$	200	T	2.81	0.00	T.O.	2.08	0.04	2.04	∞
			$\mathcal{N}, \mu$	200	T	0.07	0.00	1.72	2.17	0.04	2.12	0.79
rect_game	$\underline{\mathcal{U}^4}\times\mathcal{U}^{16}$	C	$\mathcal{U}, lb$	200	T	1.88	0.00	T.O.	38.84	0.02	38.81	∞
			$\mathcal{U}, ub$	100	T	3.06	0.01	T.O.	2.66	0.02	2.63	∞
			$\mathcal{U}, lb$	100	T	2.61	0.00	T.O.	2.64	0.02	2.62	∞
			$\mathcal{U}, ub$	200	T	4.20	0.00	T.O.	38.62	0.02	38.60	∞
			$\mathcal{U}, ub$	100	T	4.14	0.00	T.O.	2.63	0.02	2.61	∞
			$\mathcal{U}, lb$	200	T	1.88	0.00	T.O.	38.72	0.02	38.69	∞
			$\mathcal{U}, lb$	100	T	4.98	0.00	T.O.	2.67	0.02	2.64	∞
			$\mathcal{U}, ub$	100	T	4.14	0.00	T.O.	2.63	0.02	2.61	∞
sgl_reg	$\underline{\mathcal{U}^3}\times\mathcal{N}^1$	C	$\mathcal{U}, lb$	100	T	2.38	0.00	T.O.	1.21	0.03	1.18	∞
			$\mathcal{U}, lb$	800	F	6.21	0.01	T.O.	10.56	0.03	10.53	∞
			$\mathcal{U}, ub$	100	T	3.17	0.00	T.O.	1.14	0.03	1.11	∞
			$\mathcal{U}, ub$	100	T	2.20	0.00	T.O.	1.18	0.03	1.15	∞
			$\mathcal{U}, lb$	100	T	2.60	0.00	T.O.	1.12	0.03	1.09	∞
			$\mathcal{U}, ub$	100	T	2.84	0.00	T.O.	1.17	0.03	1.14	∞
post_pred	$\underline{\mathcal{U}^1}\times\mathcal{B}^2_m$	C	$\mathcal{U}, ub$	400	T	4.48	0.00	T.O.	3.24	0.09	3.15	∞
			$\mathcal{U}, lb$	200	T	4.03	0.00	T.O.	2.21	0.09	2.12	∞
altermu	$\underline{\mathcal{N}^3}\times\mathcal{N}^{40}$	C	$\mathcal{N}, \mu$	100	T	2.58	0.00	T.O.	1.55	0.17	1.38	∞
			$\mathcal{N}, \sigma$	100	T	N/A	0.00	T.O.	1.55	0.17	1.37	∞
			$\mathcal{N}, \mu$	100	T	2.66	0.00	T.O.	1.54	0.17	1.36	∞
			$\mathcal{N}, \sigma$	100	T	3.32	0.00	T.O.	1.55	0.17	1.38	∞
			$\mathcal{N}, \sigma$	100	T	N/A	0.00	T.O.	1.56	0.17	1.39	∞
			$\mathcal{N}, \mu$	100	T	2.58	0.00	T.O.	1.58	0.17	1.41	∞
tug	$\underline{\mathcal{U}^2}\times\mathcal{N}^4\times\mathcal{B}^{40}$	C	$\mathcal{U}, ub$	1600	T	3.80	0.00	T.O.	6.28	0.35	5.92	∞
			$\mathcal{U}, lb$	1600	T	4.74	0.00	T.O.	6.33	0.35	5.98	∞
			$\mathcal{U}, ub$	25600	F	77.08	0.00	T.O.	76.65	0.35	76.30	∞
			$\mathcal{U}, lb$	25600	F	11.27	0.00	T.O.	76.70	0.35	76.35	∞
neural	$\underline{\mathcal{U}^2}\times\mathcal{B}^{39}_{\log}$	C	$\mathcal{U}, lb$	100	T	2.79	0.00	T.O.	1.51	0.34	1.16	∞
			$\mathcal{U}, lb$	100	T	3.47	0.00	T.O.	1.50	0.34	1.16	∞
			$\mathcal{U}, ub$	100	T	2.51	0.00	T.O.	1.44	0.34	1.09	∞
			$\mathcal{U}, ub$	100	T	4.50	0.00	T.O.	1.46	0.34	1.11	∞

– Dist.: $\mathcal{B}$: Bernoulli, $B_{\log}$: Bernoulli-Logit, $\mathcal{B}_m$: Binomial, $\mathcal{U}$: Continuous Uniform, $\mathcal{U}_I$: Discrete Uniform (Integer), $\mathcal{N}$: Normal, β: Beta, Γ: Gamma
– Param: p: flip chance, n: No. of flips, lb/ub: lower/upper bound of Uniform, μ: mean, σ: standard deviation, α, β: shape parameters.

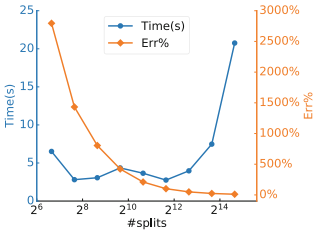
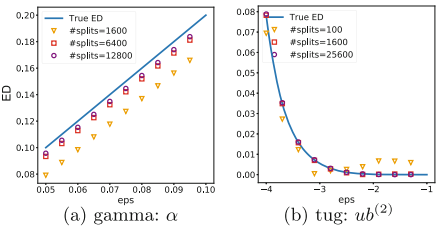


Fig. 6. Time-Err% Trade-off



(a) gamma: α (b) tug: $ub^{(2)}$

Fig. 7. Examples of AquaSense results

5.2 Trade-Off Between Accuracy and Performance

The number of quantization splits (#splits) controls the trade-off between performance and accuracy of AquaSense. Figure 6 shows AquaSense's relative error and execution time w.r.t. #splits. Error and time are averaged over all 34 continuous benchmarks. Execution time fluctuates when #splits is less than 12800 due to overhead, but grows exponentially afterward as expected. Relative error decreases exponentially as #splits increase. *Our key observation: on average, the relative error is already small when execution time starts growing exponentially.*

To illustrate the trade-off, we pick the two parameters that used the most #splits in Table 1, i.e. on which AquaSense performed the worst. We plot their True *ED* against AquaSense's interpolations with different #splits. In Fig. 7, the x-axis shows the values of ϵ and the y-axis shows *ED*. The True *ED* is shown in a blue line, and AquaSense results are shown in markers of different styles/colors. These plots demonstrate that AquaSense converges as #splits increases.

6 Related Work

Existing sensitivity analysis techniques suffer from scalability and/or precision problems. PSense [19] is the state-of-the-art sensitivity analysis tool for probabilistic programs. PSense symbolically evaluates integrals that represent the program's posterior distribution. This approach works only for small programs, and becomes intractable when the program has multiple continuous distributions (See Table 1). Sound logic frameworks for bounding the sensitivity of probabilistic programs [1–3,30] often yield a coarse over-approximation of sensitivity for soundness. Also, they are not fully automated and require developers' effort to implement the proof for general probabilistic programs. Chan and Darwiche [7] implemented the tool SamIam to compute the sensitivity of belief networks. However, it only supports discrete distributions.

Sensitivity analysis, as illustrated in our example (Sect. 2), can help developers debug anomalies in the model through an iterative process. The previous methods for debugging probabilistic programs targeted different challenges: [23] focuses on debugging probabilistic assertion failures, while [8] concentrates on addressing convergence issues of MCMC. Other approaches [9–12] focus on debugging the implementation of the probabilistic programming systems

or machine learning applications. Furthermore, through the lens of statistical modeling, researchers in statistics have proposed various strategies [5,29,31] to improve the model robustness. According to a recent study [18] that systematically evaluated these strategies, sensitivity analysis can aid developers select the most appropriate among these strategies.

7 Conclusion

We propose a new system, AquaSense, for sensitivity analysis on real-world probabilistic programs. AquaSense leverages quantized inference to interpolate parameter sensitivity. Our evaluation on 12 programs with 45 parameters shows that AquaSense achieved better efficiency and scalability than the baseline. AquaSense empowers software engineers and data scientists with the ability to understand and improve the reliability of their probabilistic programs.

Acknowledgements. This research was supported in part by NSF Grants No. CCF-1846354, CCF-1956374, CCF-200888, and CCF-2217144, and C3.ai DTI research award.

References

1. Aguirre, A., Barthe, G., Hsu, J., Kaminski, B.L., Katoen, J.P., Matheja, C.: Kantorovich continuity of probabilistic programs. arXiv preprint arXiv:1901.06540 (2019)
2. Aguirre, A., Barthe, G., Hsu, J., Kaminski, B.L., Katoen, J.P., Matheja, C.: A pre-expectation calculus for probabilistic sensitivity. Proc. ACM Program. Lang. **5**(POPL), 1–28 (2021)
3. Barthe, G., Espitau, T., Grégoire, B., Hsu, J., Strub, P.Y.: Proving expected sensitivity of probabilistic programs, vol. 2 (2017)
4. Baydin, A.G., et al.: Etalumis: bringing probabilistic programming to scientific simulators at scale. In: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. SC '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3295500.3356180
5. Berger, J.O., et al.: An overview of robust Bayesian analysis. TEST **3**(1), 5–124 (1994)
6. Blei, D., Lafferty, J.: Topic models. In: Text Mining: Classification, Clustering, and Applications (2009)
7. Chan, H., Darwiche, A.: Sensitivity analysis in Bayesian networks: from single to multiple parameters. In: Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence, pp. 67–75. UAI '04, AUAI Press (2004)
8. Dutta, S., Huang, Z., Misailovic, S.: SixthSense: debugging convergence problems in probabilistic programs via program representation learning. In: FASE 2022. LNCS, vol. 13241, pp. 123–144. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-99429-7_7
9. Dutta, S., Legunsen, O., Huang, Z., Misailovic, S.: Testing probabilistic programming systems. In: FSE (2018)

10. Dutta, S., Selvam, J., Jain, A., Misailovic, S.: Tera: optimizing stochastic regression tests in machine learning projects. In: ISSTA (2021)
11. Dutta, S., Shi, A., Choudhary, R., Zhang, Z., Jain, A., Misailovic, S.: Detecting flaky tests in probabilistic and machine learning applications. In: ISSTA (2020)
12. Dutta, S., Shi, A., Misailovic, S.: Flex: fixing flaky tests in machine learning projects by updating assertion bounds. In: FSE (2021)
13. Dutta, S., Zhang, W., Huang, Z., Misailovic, S.: 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, pp. 729–739. ACM (2019)
14. Gehr, T., Misailovic, S., Vechev, M.: PSI: exact symbolic inference for probabilistic programs. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9779, pp. 62–83. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41528-4_4
15. Gehr, T., Misailovic, S., Vechev, M.: PSI: exact symbolic inference for probabilistic programs. In: Computer Aided Verification, pp. 62–83 (2016)
16. Gelman, A., Lee, D., Guo, J.: Stan: A probabilistic programming language for Bayesian inference and optimization. J. Educ. Behav. Stat. **40**(5), 530–543 (2015)
17. Huang, Z., Dutta, S., Misailovic, S.: AQUA: automated quantized inference for probabilistic programs. In: Hou, Z., Ganesh, V. (eds.) ATVA 2021. LNCS, vol. 12971, pp. 229–246. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-88885-5_16
18. Huang, Z., Dutta, S., Misailovic, S.: Astra: understanding the practical impact of robustness for probabilistic programs. In: Uncertainty in Artificial Intelligence, pp. 900–910. PMLR (2023)
19. Huang, Z., Wang, Z., Misailovic, S.: PSense: automatic sensitivity analysis for probabilistic programs. In: 16th International Symposium on Automated Technology for Verification and Analysis. ATVA (2018)
20. Klenke, A.: Probability Theory, January 2008. https://doi.org/10.1007/3-540-33414-9
21. Lavine, M.: Sensitivity in Bayesian statistics: the prior and the likelihood. J. Am. Stat. Assoc. **86**(414), 396–399 (1991)
22. Mansinghka, V.K., Kulkarni, T.D., Perov, Y.N., Tenenbaum, J.: Approximate Bayesian image interpretation using generative probabilistic graphics programs. In: Burges, C., Bottou, L., Welling, M., Ghahramani, Z., Weinberger, K. (eds.) Advances in Neural Information Processing Systems, vol. 26. Curran Associates, Inc. (2013), https://proceedings.neurips.cc/paper/2013/file/fa14d4fe2f19414de3ebd9f63d5c0169-Paper.pdf
23. Nandi, C., Grossman, D., Sampson, A., Mytkowicz, T., McKinley, K.S.: Debugging probabilistic programs. In: Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, pp. 18–26. ACM (2017)
24. Nikulin, M.: Kolmogorov-Smirnov test (2011). https://encyclopediaofmath.org/wiki/Kolmogorov-Smirnov_test
25. Potapov, A., Rodionov, S., Potapova, V.: Real-time GA-based probabilistic programming in application to robot control. In: Steunebrink, B., Wang, P., Goertzel, B. (eds.) AGI -2016. LNCS (LNAI), vol. 9782, pp. 95–105. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41649-6_10
26. Pyro (2018). http://pyro.ai
27. Roos, M., Martins, T.G., Held, L., Rue, H.: Sensitivity analysis for Bayesian hierarchical models. Bayesian Anal. **10**(2), 321–349 (2015)

28. Saad, F., Mansinghka, V.K.: A probabilistic programming approach to probabilistic data analysis. In: Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 29. Curran Associates, Inc. (2016). https://proceedings.neurips.cc/paper/2016/file/46072631582fc240dd2674a7d063b040-Paper.pdf
29. Wang, C., Blei, D.M.: A general method for robust Bayesian modeling. Bayesian Anal. **13**(4), 1159–1187 (2018)
30. Wang, P., Fu, H., Chatterjee, K., Deng, Y., Xu, M.: Proving expected sensitivity of probabilistic programs with randomized variable-dependent termination time. arXiv preprint arXiv:1902.04744 (2019)
31. Wang, Y., Kucukelbir, A., Blei, D.M.: Robust probabilistic modeling with Bayesian data reweighting. In: Proceedings of the 34th International Conference on Machine Learning, vol. 70, pp. 3646–3655. ICML'17, JMLR.org (2017). http://dl.acm.org/citation.cfm?id=3305890.3306058