Online and Interactive Bayesian Inference Debugging

Nathanael Nussbaumer[*]
nathanael.nussbaumer@tuwien.ac.at
TU Wien
Vienna, Austria

Markus Böck[*]
markus.h.boeck@tuwien.ac.at
TU Wien
Vienna, Austria

Jürgen Cito
juergen.cito@tuwien.ac.at
TU Wien
Vienna, Austria

Abstract

Probabilistic programming is a rapidly developing programming paradigm which enables the formulation of Bayesian models as programs and the automation of posterior inference. It facilitates the development of models and conducting Bayesian inference, which makes these techniques available to practitioners from multiple fields. Nevertheless, probabilistic programming is notoriously difficult as identifying and repairing issues with inference requires a lot of time and deep knowledge. Through this work, we introduce a novel approach to debugging Bayesian inference that reduces time and required knowledge significantly. We discuss several requirements a Bayesian inference debugging framework has to fulfill, and propose a new tool that meets these key requirements directly within the development environment. We evaluate our results in a study with 18 experienced participants and show that our approach to online and interactive debugging of Bayesian inference significantly reduces time and difficulty on inference debugging tasks.

CCS Concepts

• Software and its engineering → Software testing and debugging; **• Mathematics of computing → Bayesian computation**; **• Human-centered computing → Interactive systems and tools**.

Keywords

Probabilistic programming, Bayesian inference, debuggging

ACM Reference Format:
Nathanael Nussbaumer, Markus Böck, and Jürgen Cito. 2026. Online and Interactive Bayesian Inference Debugging. In *2026 IEEE/ACM 48th International Conference on Software Engineering (ICSE '26), April 12–18, 2026, Rio de Janeiro, Brazil.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3744916.3773215

1 Introduction

Probabilistic programming is a rapidly developing programming paradigm which enables the formulation of Bayesian models as programs and the automation of posterior inference. In Bayesian modeling we specify relationships between observable variables and unobservable *latent* variables. Knowledge about the latent variables before collecting data can be incorporated by defining a *prior distribution* over these variables. Once data has been collected, we are interested in the distribution over the latent variables *conditioned* on observing the data – the *posterior distribution.*

By abstracting away much of the inference machinery and letting the user focus on model building, probabilistic programming aims to make Bayesian inference more accessible to practitioners who are not experts in inference algorithms. However, this separation often makes it difficult to resolve issues in probabilistic programming, because its output – the approximation to the posterior – heavily depends on the interplay of model and inference. Furthermore, even if a probabilistic program runs without raising exceptions and is bug-free in the traditional programming sense, its output may still be poor. Currently, practitioners require deep knowledge of probabilistic programming systems to resolve issues. Furthermore, due to the heavy computational demands of Bayesian inference algorithms [5, 22, 25, 37], development cycles are slow, as one has to wait for a potentially long inference time before being able to analyzing the output.

In this work, we address the aforementioned challenges by designing, implementing, and evaluating a development environment for debugging probabilistic programs. We derive six design requirements for such an environment with the most central ones being *online debugging* and *interactive debugging*. The former refers to being able to analyze the output of a probabilistic program *during* its execution, while the latter requires a graphical user interface for diagnosing issues. We realized these requirements in the implementation of InferLog Holmes - our debugger specialized for workflows using so-called MCMC inference algorithms. In a user-study we found that practitioners are more effective at detecting and resolving issues with our environment.

In summary, the contributions of this work are

- the design of an online and interactive debugger for probabilistic programs,
- the open-source implementation of InferLog Holmes – a debugger specialized for MCMC workflows (https://github.com/ipa-lab/InferlogHolmes-Appendix or [33]),
- a user-study with 18 participants evaluating the debugging experience with InferLog Holmes,
- a quantitative and qualitative analysis of the effectiveness of InferLog Holmes.

2 Debugging in Probabilistic Programming

We studied the plethora of literature on diagnosing problems in Bayesian inference [7, 14, 18, 20, 21, 38], as well as the provided tooling of modern probabilistic programming systems [1, 4, 8, 17], and summarize our findings in three statements *DI1-DI3* termed *debugging insights.* We illustrate the process of debugging in probabilistic programming and introduce our insights at the hand of an example presented in Rubin [35].

[*]Both authors contributed equally to this research.

ACM ISBN 979-8-4007-2025-3/26/04
https://doi.org/10.1145/3744916.3773215

Namely, we consider a model where researchers investigated whether short-term coaching can lead to increasing SAT scores (examinations designed to be resistant against such short-term efforts). For each school i, the researchers have gathered data on the differences in test scores y_i along with a standard error estimate σ_i. To account for differences and similarities of the effect between schools, we model the experiment mathematically as follows:

$$\begin{aligned} \mu &\sim \text{Normal}(0, 10) && \ldots \quad \text{overall treatment effect} \\ \tau &\sim \text{HalfCauchy}(5) && \ldots \quad \text{effect variance between schools} \\ \theta_i &\sim \text{Normal}(\mu, \tau) && \ldots \quad \text{effect for school } i \\ y_i &\sim \text{Normal}(\theta_i, \sigma_i) && \ldots \quad \text{observed effect for school } i \end{aligned}$$

The appeal of probabilistic programming is that we can translate this mathematical description almost one-to-one to a programmatic formulation, for instance, to PyMC [1], as below:

```
with pm.Model() as eight_schools:                          1
  mu = pm.Normal("mu",mu=0,sigma=10)                        2
  tau = pm.HalfCauchy("tau",beta=5)                         3
  for i in range(8):                                        4
    theta = pm.Normal(f"theta{i}",mu=mu,sigma=tau)          5
    pm.Normal(f"y{i}",mu=theta,sigma=sigma,observed=y)      6
```

Separate from the model definition, we specify our inference routine, where we use the general-purpose HMC algorithm:

```
with eight_schools:
  step = pm.HamiltonianMC(step_scale=0.2)
  idata = pm.sample(3000,tune=100,step=step,chains=4)
```

The drawback of probabilistic programming systems is the fact that, while the programs look fairly simple, a lot of complex inference machinery is abstracted away and when inference does not work out-of-the-box, then finding the underlying issues is very challenging. In fact, the presented code would not produce a good approximation of the true posterior.

To improve the result, first, we would need to replace the for loop in lines 4-6 with

```
Z = pm.Normal("Z",mu=0,sigma=1,shape=(8,))
theta = pm.Deterministic("theta", mu + tau * Z)
pm.Normal(f"ys",mu=theta,sigma=sigmas,observed=ys)
```

This change accelerates inference by batching the school effects θ_i and observed data y_i, σ_i into vectors. But more importantly, it reparametrizes the model such that the scale of all variables is independent of other latent variables. Without going into too much detail, HMC cannot explore the posterior distribution well if the scale of θ depends on τ via `sigma=tau` and it is better to derive theta from a the latent variable `Z` with fixed scale.

Coming up with this improvement requires deep knowledge about HMC, how it is implemented in the internals of PyMC, and how our model definition affects its behavior. Such issues are not unique to HMC nor PyMC, and in general, to find these issues we state following debugging insight:

> *DI1. Deep understanding of the interplay between inference routine and model required.*

The second issue with the presented code lies within the suboptimal choice of the inference hyper-parameters. The `step_scale` is set slightly too small and in conjunction with a small number of tuning steps, leads to inefficient posterior exploration. But how would a practitioner identify these issues?

Results produced by Markov-Chain-Monte-Carlo (MCMC) type algorithms like HMC are typically analyzed through means of diagnostic statistics and plots. For instance, we may find that the so-called trace plot in Figure 1, which shows the value of a latent variable during inference, exhibits suspicious "flat" areas. This is an indication that the sampling algorithm got "stuck" at some point and that it proposed many values that are not from the true posterior. Still, deep knowledge is required to interpret these plots and make changes to the model and inference setup to mitigate the issues.

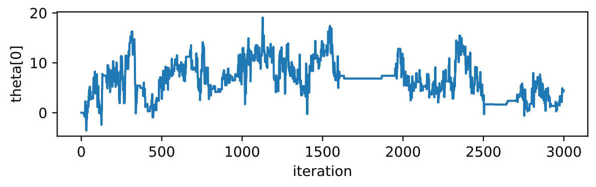


Figure 1: Example of a suspicious trace plot.

Furthermore, it is important to highlight that we can perform this analysis only when the inference routine has finished. As MCMC algorithms are only asymptotically correct, they have to be run for a large number of iterations and sample sizes between tens of thousands and millions are common. This leads to long inference time and slow model/inference development cycles. We summarize these insights into following fact:

> *DI2. Diagnosing inference results requires deep knowledge of the interplay between modeling and the underlying inference machinery, and is only possible after potentially long inference times.*

Lastly, we note that in Figure 1, we conveniently showed a specific diagnostic plot that is indicative of issues in the presented code. However, there is a large number of diagnostic statistics and plots that should be considered for each latent variable individually. Typically, the same inference routine is run in parallel several times such that issues may manifest in one run but not in others. With the configuration `chains=4`, we run inference four times and our model defines ten latent variables. If there are N suitable diagnostic plots to consider, the total number of plots to analyze is $40 \times N$. This even neglects the class of diagnostics tools designed to analyze *pairs* of random variables. With our last debugging insight we summarize that the number of diagnostics statistics and plots to consider grows quickly.

> *DI3. The debugging exploration space for probabilistic programs is very large.*

The challenges for practitioners discussed in this section are not unique to PyMC, the presented model or chosen inference algorithm. Instead, they are universally encountered in probabilistic programming and in the next sections, we will introduce our approach to improve the debugging experience.

3 Debugger Design

In this section, we derive six *design requirements REQ1-REQ6* for PPL debuggers derived from the debugging insights *DI1-DI3* introduced in Section 2. In this context, debugging refers to locating and fixing issues in the model definition or inference result. While these requirements are not tailored to any specific PPL nor inference algorithm, we implemented a tool specifically for debugging MCMC inference in PyMC based on them, named InferLog Holmes. We describe the implementation of InferLog Holmes alongside the design requirements and present the evaluation of InferLog Holmes in the subsequent sections.

3.1 REQ1: Multiple-View Integrated Development Environment

In contrast to a traditional debugger, which allows practitioners to set breakpoints and step through the program line by line, locating and debugging Bayesian inference issues comes down to analyzing the quality of the posterior distribution approximation.

The quality of this approximation is tightly coupled to the source code of a probabilistic program via the model definition, choice of inference method and various hyper-parameters. The user is required to understand the interplay of these components for fixing issues as stated in *DI1*. Having a secondary view for these analysis tools next to the primary source code editor gives the practitioner fast feedback on how a change in the source code is reflected in the inference result.

We have implemented InferLog Holmes as a VSCode extension which splits the screen in half. Fig. 4 shows that in addition to the source code view on the left, we have a secondary panel on the right for displaying debugging information. This secondary panel provides three tabs: 1) a *Model View* which visualizes the dependencies between random variables in the model graphically; 2) a *Live Debugging View* which displays intermediate results while inference is running, see Section 3.3; 3) a *Warnings View* which lists potential issues detected by analyzing the incoming inference results on the fly, see Section 3.5.

3.2 REQ2: Contextualized Workflows and Knowledge Base

There is a lot of information about recommended workflows for specific inference approaches throughout literature and educational resources. As noted in *DI1*, for fixing inference issues it is crucial to understand the interplay of the employed inference algorithm and the implemented model. Thus, a debugger has to provide analysis tools and workflows in the context of the chosen inference algorithm and supplies the practitioner only with information relevant to their current implementation.

To fulfill *REQ2*, our tool detects the inference algorithm specified in the source code and adapts its visualizations and background analysis accordingly. For instance, while for MCMC algorithms like HMC trace plots are suitable (see Figure 1), they do not make sense in a *Variational Inference* [22] context, where the convergence of the so-called ELBO loss is of interest. Differences may also be more subtle: The optimal acceptance rate[1], which is used as reference in the background analyses, differs among MCMC algorithms. As part of its modular design, InferLog Holmes allows us to easily add analyses and configure existing analyses for given inference algorithms. For instance, we may configure our analyses to the theoretically optimal acceptance rates of the MCMC algorithm at hand. Additionally, this modularity also enables InferLog Holmes to generalize to other classes of inference algorithms like variational inference, e.g. by adding plots of the ELBO loss.

3.3 REQ3: Online Diagnostic Tools

Executing a probabilistic program amounts to running an inference routine for thousands of iterations. As stated in *DI2*, for complex models this takes a long time. State-of-the-art probabilistic programming systems only allow you to analyze the inference result after the routine has finished, leading to slow development cycles. An effective debugger enables such analysis *online*, i.e. before, during, and after running the inference routine.

In general, this can be achieved by hooking into the inference loop of any PPL. Depending on the language design, this could be done in an external package or through direct integration. For instance, to collect the inference state after each iteration for PyMC, we implemented a simple class that wraps a trace object and makes use of functionality intended for storing traces on disk during inference. Through batching on both the language and tool side, we keep the performance overhead from communication and analysis negligible. This enables efficient, real-time updates of all visualizations and analyses as results become available on consumer-grade hardware.

In InferLog Holmes, the *Debugging View* shows the trace plot and histogram of variables up to the current iteration and the *Warnings View* displays the latest available insights from the background analysis. This way we enable practitioners to spot issues quickly and allow them to abort inference early if necessary.

3.4 REQ4: Interactive Diagnostic Tools

As stated in *DI2* and *DI3*, to analyze the inference result, the practitioner needs to know which diagnostic methods are applicable, typically write a lot of boilerplate code to use them, and repeatedly run the code on a selection of the variables. To make this process less cumbersome, a debugging tool must integrate such diagnostic tools as a core component and allow the user to interactively perform the same analysis without the need to write code.

To this end, with InferLog Holmes we enable the user to navigate the interface primarily by mouse clicks. For instance, in the *Debugging View* variables are selectable similar to browser tabs, the particular MCMC chain can be chosen from a drop-down menu, and the user can jump to more detailed plots by the press of a button. In the *Warnings View*, initially a condensed summary of the warnings is given, each of which can be expanded to a detailed description.

[1]The acceptance rate measures the proportion of proposed samples that are accepted during inference. Although not a strong standalone indicator, high acceptance rates can signal slow exploration of the posterior and waste of compute resources, while low acceptance rates often hint that the algorithm has trouble staying within regions of high posterior probability.

Lastly, we include links to relevant source code sections throughout the interface to strengthen the connection between code and analysis.

3.5 REQ5: Heuristic-Guided Exploration of the Debugging Space

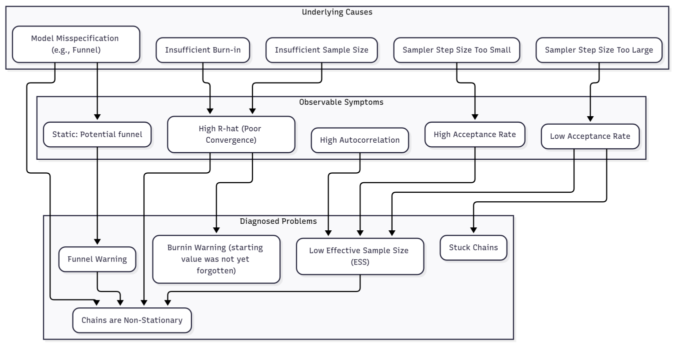

Underlying Causes
Model Misspecification (e.g., Funnel)
Insufficient Burn-in
Insufficient Sample Size
Sampler Step Size Too Small
Sampler Step Size Too Large
Observable Symptoms
Static: Potential funnel
High R-hat (Poor Convergence)
High Autocorrelation
High Acceptance Rate
Low Acceptance Rate
Diagnosed Problems
Funnel Warning
Burnin Warning (starting value was not yet forgotten)
Low Effective Sample Size (ESS)
Stuck Chains
Chains are Non-Stationary

Figure 2: Warnings Relation. Showing the causes, observable values and triggered warnings.

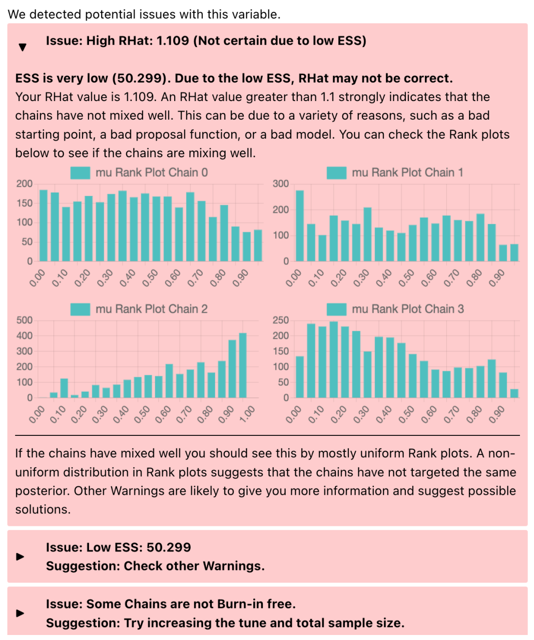
While interactivity already leads to a less cumbersome analysis experience, it still does not reduce the amount of steps taken to detect an inference issue. As stated in *DI3*, the number of configurations for a single analysis step (selection of variables, chain, and diagnostic tool) is very large even for simple models. Therefore, it is critical for a debugger to implement heuristics and automated checks that guide the user to subsets of this debugging space which are most likely to reveal issues in the inference result.

We realized this requirement in InferLog Holmes in the *Warnings View*. In the background, InferLog Holmes monitors a set of metrics relevant to MCMC inference including acceptance rate, effective sample size, and the rank-normalized $\hat{r}$. Then, a number of analyses are run on these metrics and the *Warnings View* is populated with messages that warn the user about instances indicative of faulty inference along with suggestions on how to fix them. These checks include static analyses that run on the source code of the program, which we have implemented with the LASAPP framework [6]. For a complete overview of the implemented warnings see Table 1. Fig 2 further shows the relationship between symptoms, analysis and warnings.

3.6 REQ6: Easy Integration with Existing PPLs

Lastly, it is important that our debugger can easily be applied to existing PPLs. This requires a light-weight API to send data during inference to the debugging and analysis backend.

After sending initial information, InferLog Holmes only requires the PPL backend to make regular HTTP requests, either batched or after each MCMC iteration, sending the current inference state as JSON. Since PyMC provides an API for making such callbacks, the integration was particularly easy. Some other PPLs offer similar APIs, but even if a PPL does not, the HTTP calls could be directly implemented in the language without significant challenges. By implementing the HTTP interface within a language,



We detected potential issues with this variable.

▼ **Issue: High RHat: 1.109 (Not certain due to low ESS)**

ESS is very low (50.299). Due to the low ESS, RHat may not be correct.
Your RHat value is 1.109. An RHat value greater than 1.1 strongly indicates that the chains have not mixed well. This can be due to a variety of reasons, such as a bad starting point, a bad proposal function, or a bad model. You can check the Rank plots below to see if the chains are mixing well.

If the chains have mixed well you should see this by mostly uniform Rank plots. A non-uniform distribution in Rank plots suggests that the chains have not targeted the same posterior. Other Warnings are likely to give you more information and suggest possible solutions.

► **Issue: Low ESS: 50.299**
Suggestion: Check other Warnings.

► **Issue: Some Chains are not Burn-in free.**
Suggestion: Try increasing the tune and total sample size.

Figure 3: Example of warnings raised by the debugger. This example shows one of the debuggers warnings in detail and 2 others collapsed. The warnings try to highlight probable issues and aim to inform the user to a point were they can make a decision if an action is needed or not. Whenever possible warnings show a suggestion that hopefully fixes the underlying issue.

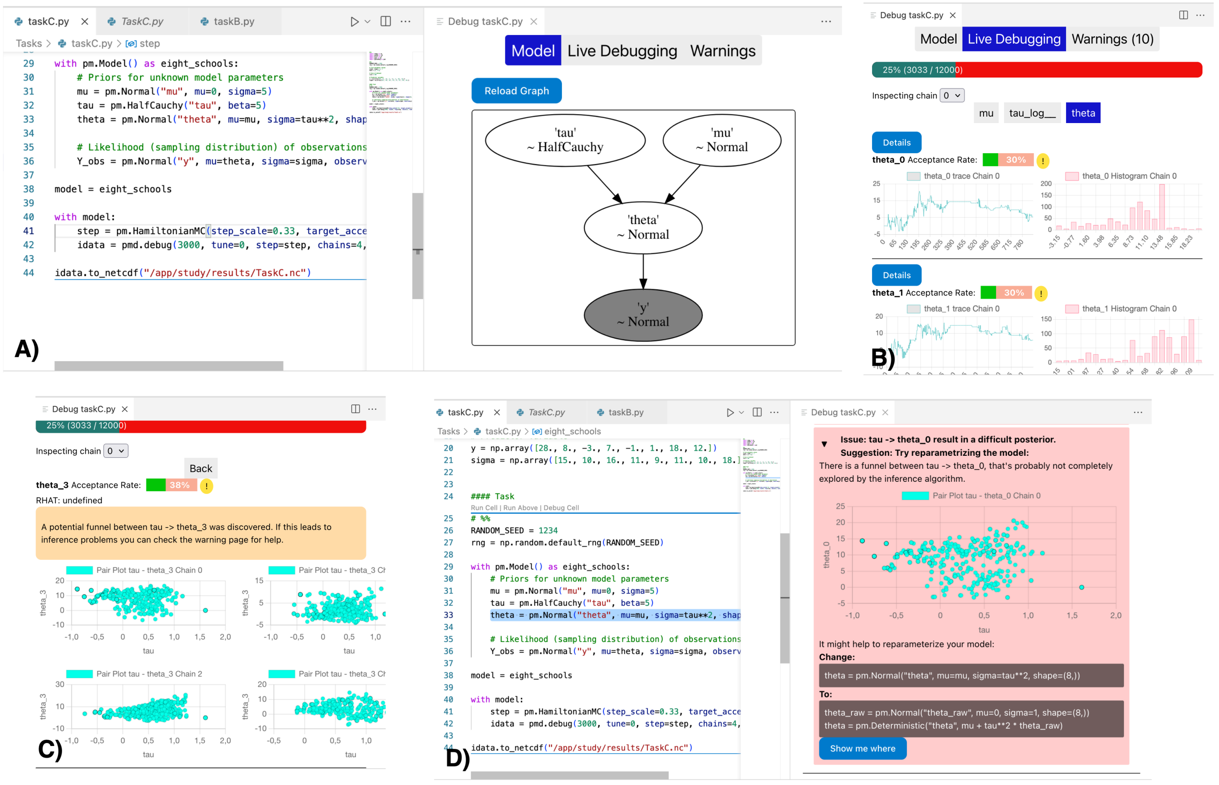
most features of InferLog Holmes are immediately available. To get full support for contextualized workflows, the *ConcretePPL* interface inside InferLog Holmes must be implemented for a new language. This allows InferLog Holmes to add additional information to the warnings that are specific to the given PPL.

4 Workflows in InferLog Holmes Exemplified

To illustrate the accelerated workflows enabled by InferLog Holmes, let us revisit the SAT coaching example from Section 2 from the users perspective.

Modeling. Suppose a user wants to investigate whether short-term coaching improves SAT scores, using a Bayesian model. They begin implementing a PyMC model, informed by their prior beliefs. At some point, they want to see how their model is structured. This is when they open InferLog Holmes, which shows the model code side-by-side with a visual representation of the model graph (Fig. 4 Step **A**).

Live Feedback During Inference. When ready, they launch inference. Immediately, live diagnostics appear (Fig. 4 Step **B**). As posterior samples stream in, the user can inspect evolving distributions, trace plots, and summary statistics in real-time. The user starts inspecting the progression of the different variables. When they look at an overview of the θ variables the user notices a warning icon next to the acceptance rate, which is at only 30%.



```
with pm.Model() as eight_schools:
    # Priors for unknown model parameters
    mu = pm.Normal("mu", mu=0, sigma=5)
    tau = pm.HalfCauchy("tau", beta=5)
    theta = pm.Normal("theta", mu=mu, sigma=tau**2, shap

    # Likelihood (sampling distribution) of observations
    Y_obs = pm.Normal("y", mu=theta, sigma=sigma, observ

model = eight_schools

with model:
    step = pm.HamiltonianMC(step_scale=0.33, target_acce
    idata = pmd.debug(3000, tune=0, step=step, chains=4,

idata.to_netcdf("/app/study/results/TaskC.nc")
```

25% (3033 / 12000)

theta_0 Acceptance Rate: 30%

theta_1 Acceptance Rate: 30%

theta_3 Acceptance Rate: 38%

RHAT: undefined

A potential funnel between tau -> theta_3 was discovered. If this leads to inference problems you can check the warning page for help.

```
y = np.array([28., 8., -3., 7., -1., 1., 18., 12.])
sigma = np.array([15., 10., 16., 11., 9., 11., 10., 18.]


#### Task
# %%
RANDOM_SEED = 1234
rng = np.random.default_rng(RANDOM_SEED)

with pm.Model() as eight_schools:
    # Priors for unknown model parameters
    mu = pm.Normal("mu", mu=0, sigma=5)
    tau = pm.HalfCauchy("tau", beta=5)
    theta = pm.Normal("theta", mu=mu, sigma=tau**2, shap

    # Likelihood (sampling distribution) of observations
    Y_obs = pm.Normal("y", mu=theta, sigma=sigma, observ

model = eight_schools

with model:
    step = pm.HamiltonianMC(step_scale=0.33, target_acce
    idata = pmd.debug(3000, tune=0, step=step, chains=4,

idata.to_netcdf("/app/study/results/TaskC.nc")
```

Issue: tau -> theta_0 result in a difficult posterior.
Suggestion: Try reparametrizing the model:
There is a funnel between tau -> theta_0, that's probably not completely explored by the inference algorithm.

It might help to reparameterize your model:
Change:

```
theta = pm.Normal("theta", mu=mu, sigma=tau**2, shape=(8,))
```

To:

```
theta_raw = pm.Normal("theta_raw", mu=0, sigma=1, shape=(8,))
theta = pm.Deterministic("theta", mu + tau**2 * theta_raw)
```

Figure 4: Exemplified workflows. A) shows the model graph view of the tool right next to the source code. **B)** is the *Live Debugging View*, which gives insights into the developing posteriors. **C)** shows the *Details View* for variable *theta_3* with included pair plots due to the statically identified variance dependence between *tau* and *theta*. In **D)** we see a warning with explanation, graph and the identified line in the source code that causes the issue, together with a suggestion for a solution.

Focused Inspection. Since the acceptance rate is very low for *HMC* based algorithms and the traceplots are indicative of suboptimal convergence, the user opens the details view for one of the θ variables (Fig. 4 Step **C**). The user immediately notices the orange text box that tells them that there is a potential funnel between τ and θ, a shape that is often difficult to explore for MCMC based algorithms. This leads them to inspecting the shapes in the conditionally displayed pairplots between these two variables, which indeed look funnel-shaped.

Warnings and Potential Solutions. While the user is inspecting the details page, they notice that there are 10 warnings for potential problems discovered by InferLog Holmes. So the user heads over to the warnings view and opens the topmost warning (Fig. 4 Step **D**) for the θ_0 variable. This warning, again, raises the issue of the funnel between τ and θ and shows that the resulting posterior is difficult for *HMC* algorithms to explore. It further includes the problematic line of code together with a suggested change. The user accepts this suggestion and clicks on the button "Show me where" to jump to the exact line of code causing the issue and replaces it.

Iterations. Since continuing inference after identifying a fault wastes time, the user stops inference early and restarts it with the applied changes in place. From there the user keeps iterating together with InferLog Holmes until all warnings are resolved and the inference results show no indication of non-convergence.

5 Study Design

To evaluate our framework we conducted a study with 18 participants. We formulate following research questions:

(RQ1) Does our online and interactive framework effectively decrease time spent debugging inference?
(RQ2) Are online visualizations of MCMC posteriors considered useful by PPL users?
(RQ3) Are the warnings raised at inference time considered useful by PPL users?

To answer these questions each participant was asked to solve three inference debugging tasks in PyMC and to fill out two questionnaires: one pre-study questionnaire and one post-study questionnaire.

RQ1 is answered quantitatively by evaluating the hypotheses stated in Table 2 with statistical tests.

For *RQ2* and *RQ3* we adopt a qualitative methodology, basing our analysis on participants responses to the questionnaires and verbal statements they made while performing the tasks.

Warning	Key Symptoms	Likely Cause(s)	Suggested Solution (from code)
High R^	$\hat{r} > 1.01$	Chains have not converged to the same distribution / Multiple	"See other warnings." Check rank plots.
Burn-in Warning	High $\hat{r}$ in the first part of a chain.	Initial samples are not from the posterior distribution.	"Increase the burn-in period."
Funnel Detected & Low/High Acceptance	A parameter's variance is controlled by another parameter.	Model parameterization causes posterior that is difficult to sample from.	"Reparameterize the model." (Provides code)
Low ESS & High Acceptance	ESS is low AND acceptanceRate is high.	Sampler step size is too small, leading to high autocorrelation.	"Increase the proposer's step size."
Low ESS & Low Acceptance	ESS is low AND acceptanceRate is low.	Sampler step size is too large, leading to high rejection rates.	"Lower the proposer's step size."
Stuck Chain	A chain has not accepted a new sample in many iterations.	Extremely low acceptance rate.	"Check your proposal functions and step size."
Low ESS (Isolated)	ESS is low.	High sample autocorrelation, reason unclear.	"Check other warnings, they might be indicative."
Low/High Acceptance (Isolated)	acceptanceRate is outside the optimal range.	Step size is likely miscalibrated, but effect on efficiency is not yet clear.	"Maybe change the step size."

Table 1: Warnings and their triggers. Shows all the possible warnings raised by InferLog Holmes together with a brief overview of triggers, likely causes and suggestions.

5.1 Study Setup

The study was conducted as a controlled experiment with a duration of 1h:15m to 2 hours for each participant.

Environment: The study was conducted fully online via a video conferencing tool. To avoid differences in inference runtime depending on hardware differences, we provided an online version of VSCode. The participants accessed the online IDE via https://vscode.dev and all program code was executed on the same lab-server for every participant.

Design & Baseline Condition: The study follows a within-subject design, where the task order was randomized and the first task

	Null hypothesis		Alternative hypothesis
$H1_0$:	The tool does not impact how many issues PPL user can solve	$H1$:	PPL users are able to resolve more issues with the tool
$H2_0$:	The tool does not impact how fast PPL user can identify issues	$H2$:	PPL users are able to identify issues faster with the tool
$H3_0$:	The tool does not impact how fast PPL users iterate over the problem	$H3$:	PPL users have faster iteration cycles with the tool
$H4_0$:	The tool does not impact how effective PPL users spend their time	$H4$:	PPL users spend their time more effectively with the tool

Table 2: Null and alternative hypothesis

serves as the *baseline* condition for each participant. For this baseline, participants were allowed to use any existing tool or resource of their liking, *except* InferLog Holmes, to closely reflect real-world workflows. With Task 2 and 3 we evaluated the *tool* condition, where the participants were allowed to use InferLog Holmes *in addition* to existing tools or resources.

We observed participants opting to use the following existing tools and resources for solving the first task: prior and posterior predictive checks, ArviZ (plots + summary diagnostics), search engines, PyMC + ArviZ documentation and tutorials, ChatGPT. In general, we observed that participants had different workflows but often relied heavily on ArviZ and search engines. Interestingly, only one participant asked a large language model for help. Across all participants, InferLog Holmes became the primary tool during Tasks 2 and 3, with many relying on it so extensively that they stopped using other tools and resources.

Protocol: Before the experiment, participants were asked to fill-out a pre-study questionnaire to assess their background and familiarity with probabilistic programming. Afterwards the participants connected to the online environment and were given an introduction notebook. Its main purpose was introducing participants to PyMC and relevant tooling around PyMC. Participants were told to take as much time as they needed and that they could always refer back to the notebook or use any other resource they wish.

Once a participant gave the signal that they were ready, they could proceed to the first task. The participants were informed that they have to solve the tasks in order and cannot go back and forth between them. The task order was randomly assigned to each participant, and every participant had to solve the first task without InferLog Holmes. However, they could use any tool or resource they had access to for all tasks. The time limit for each task was set to 30 minutes (subtracting setup issues), and participants had to decide for themselves when a task is finished. They did not receive any feedback on their progress during the study. After participants deemed the first task done (or reached the time limit) they received a ~5 minutes introduction to InferLog Holmes. Participants were now free to use InferLog Holmes, if they wanted, for the following two tasks. Aside from the new tool, everything else stayed the same. Once the participant finished all three tasks, they were asked to fill-out a post-study questionnaire. Afterwards, the study concluded and participants received feedback on their task performance if they wished.

5.2 Tasks

A task in our study is defined by a probabilistic program given as a PyMC source file with pre-implemented model and inference routine. In these programs we have introduced sub-optimal or faulty implementation decisions. The objective for the participants is to determine whether there is an issue with the model and/or inference implementation or whether there is no issues at all.

We have prepared three tasks in total:

(A) Linear regression model with HMC inference routine. *Fault:* bad choice of step-size hyper-parameter of HMC.
(B) Signal detection theory model [28] with HMC inference routine. *Fault:* Unused random variables, sub-optimal hyper-parameter choice for step-size and too short burn-in period and too small sample size.
(C) Eight schools model [35] with HMC inference routine. *Fault:* Sub-optimal model parameterization, too short burn-in period and depending on burn-in period setting potentially auto-tune needs to be disabled.

The tasks vary in difficulty: the simple model and poor inference result of task A makes it easier to identify and solve in comparison to the other more challenging tasks B and C.

5.3 Participants

Probabilistic programming is a relatively small community and since our study targets *PPL* users and requires prior knowledge of Bayesian inference preferably in the context of probabilistic programming, recruiting participants became a challenging task.

As our efforts for recruiting study participants in the forums of the three popular PPLs PyMC, Pyro, and Stan was unfortunately unsuccessful, we opted for a *snowball sampling* [2] strategy instead. Here, we considered three major pools:

(1) Students who took a university course on Probabilistic Programming *(8 participants recruited)*
(2) Probabilistic Programming and Bayesian modeling researchers *(7 participants recruited)*
(3) Practitioners, e.g., other scientists, data analysts, etc. *(3 participants recruited)*

5.4 Result Validation and Statistical Tests

To make our results verifiable, we conducted a series of statistical tests, where we set the significance level at 5%.

5.4.1 Ordinal Measurements of Uncensored Data. Due to the non-normality of our ordinal variables (confirmed with the Shapiro-Wilk test), we performed the Mann-Whitney U Test on these measurements. The Mann-Whitney U Test is non-parametric and thus makes no assumption on the distribution of our data. Following common practice, we further used Cliff's delta for estimating the effect size [9, 36].

5.4.2 Censored Data. Some of our data variables are *right censored*, which means that there may be events that did not occur within the study period. For instance, the *Time to first issue detected* is right-censored, because some participants were unable to detect the first issue within the given time limit. For these kinds of variables, statistical survival analysis can be adopted [3, 27]. We performed *Cox Proportional Hazards Regression* [10] and a log-rank test [30] to estimate the effect of our framework on the "survival rate". In this context, survival refers to the event that a bug "survives" the detection efforts of the participant.

Since our study measures a positive outcome (e.g. task solved) in contrast to many other studies using these methods, we will focus our attention on the cumulative densities $cd(T)$, instead of the survival function $sf(T)$, which is defined as $cd(T) = 1 - sf(T)$. So instead of investigating the probability of a task remaining incomplete beyond time T (e.g., survival), we focus on the probability of the task being completed (commonly interpreted as death in survival analysis) prior to time T.

5.5 Threats to Validity

5.5.1 External Validity. Given that participation in a study is entirely voluntary and the participants received no material benefit from participation, it is possible that our sample is not representative of the general population of *PPL* users. Furthermore, the use of snowball sampling may lead to a more homogeneous group of participants [9]. We mitigated these issues by reaching out to possible participants broadly and ensuring that our participants come from different backgrounds. We were especially careful to not only recruit students but also researchers and professionals to have participants from the full spectrum of *PPL* users. Furthermore, we ensured that the recruited participants had no dependency on the researchers (e.g., all students had been graded before participating).

5.5.2 Internal Validity. We identified two main threats to internal validity for this study. First, due to the necessarily artificial nature of the tasks and setup, it is possible that the tasks do not represent real world issues well. Second, the tasks may overly favor the usage of InferLog Holmes, through aligning with its feature set. We mitigated both these threats through basing our examples on existing literature and known examples. Task C, for example, is a well known problem in probabilistic programming research, Task B is based on a tutorial task for the *Pyro PPL* [4] with the main issue being a copy&paste / typo, which was an actual mistake made during task creation and evaluation and Task A is the prototypical linear regression example with a common issue of bad hyper-parameters. To further mitigate the effect of task alignment, we introduced multiple issues into Task B and Task C. For instance, while the funnel warning in InferLog Holmes was designed to detect funnel relationships as constructed by the centered eight school model in Task C, we introduced two additional issues in the program to reduce alignment. Nevertheless, the effects measured in this study may be somewhat larger than they would be in real world scenarios with more subtle bugs.

6 Quantitative Results

In this section, we present our quantitative analysis to answer the question *RQ1: Does our online and interactive framework effectively decrease time spent debugging inference?* In the following subsections, we show that we can reject each of the four null hypotheses of Table 2.

6.1 H1: Users are able to resolve more issues

Table 3 shows the mean and standard deviation of the percentage of resolved issues across all tasks, together with p-values from Mann-Whitney U test and Cliff's delta.

Task	Tool	Baseline	Cliff's δ	p-value
A	91.67 ± 28.87	66.67 ± 51.64	0.250	0.218
B	69.45 ± 26.43	5.55 ± 13.61	0.958	**0.001**
C	75.00 ± 32.18	5.55 ± 13.61	0.889	**0.002**

Table 3: Average Percent of solved issues (Mann-Whitney U).

The results show a significant difference between the groups for the tasks *B* and *C*. For the simplest task *A*, we could not find a significant difference between the groups. This suggests that our framework provides greater help as tasks grow in complexity and potentially involve multiple issues. Considering that task *A* only involves 3 latent variables whereas task *C* involves 10 and task *B* 20, this shows that our tool is likely to reduce the exploration space for larger problems. This further suggests that our tool successfully tackles the previously examined debugging insight *DI3*, where we discussed the exploration space quickly growing as the number of variables to consider increases.

For hypothesis H1, we further conducted a Bayesian analysis of the results, which allows us to quantify effects in relation to each other, in contrast to the standard statistical tests. We implemented a Bayesian logistic regression model, which represents the number of issues identified as a Bernoulli distribution. The effects are modeled as latent random variables and are summed up and passed through a sigmoid function, which outputs the probability p of the Bernoulli distribution.

Fig 5 shows a forest plot of the posteriors for *tool effect*, *participant skill*, *learning effect*, and *task effects*. From the plot we see a clear indication of InferLog Holmes having a large positive effect on the probability of solving an issue.

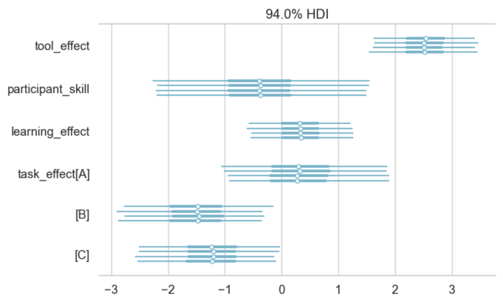


Figure 5: Forest plot of effect posteriors from Bayesian analysis.

We see that on average, the usage of the tool increases the probability of solving an issue by 39% points, 47% points, and 49% points for tasks A, B, and C, respectively. This means the average user has a 20% chance of solving any given issue in Task C without the tool and a 69% chance with InferLog Holmes. The distributions for these increased probabilities can be seen in Fig. 6.

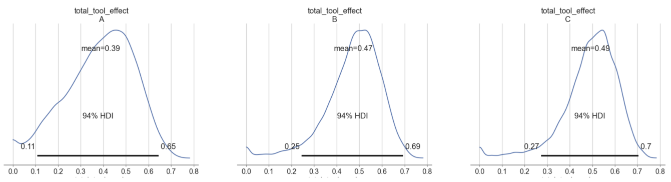


Figure 6: Distribution over effective increase in percentage points for solving any issue if the tool is used, compared to the tool not used.

More detailed results and descriptions of the implemented model can be found in the online appendix to this paper (https://github.com/ipa-lab/InferlogHolmes-Appendix).

6.2 H2: Users are able to identify issues faster

Table 4 shows the results from Cox-Proportional-Hazard regression analysis depending on tool use and for comparison p-values from a Log-Rank test for each task individually and for all tasks pooled. To interpret the results of the Cox Proportional-Hazards regression analysis, we examine the *exp(coef)*, which represents the estimated hazard ratio produced by the model. For a more comprehensive discussion of the interpretation of survival analysis methods, we refer the interested reader to [16]. Due to the small number of occurred events during the task time, in some groups, the results should be taken with care. Nevertheless, the large *exp(coef)* coefficient together with small p-values suggests that these differences would be highly unlikely under the null-hypothesis of equality between the groups. Furthermore, when we pool all tasks together, we see an even stronger effect, and the data suggests that a user has a **30x** higher chance of discovering an issue at any given point in time with our tool than without.

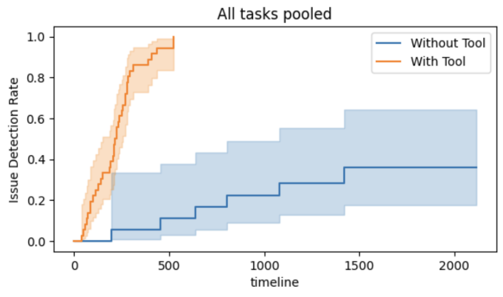


Figure 7: Time to First Issue Detected (seconds) Kaplan-Meier-Estimator Cumulative Densities Curves for all tasks pooled.

These results are supported by visual inspection of the Kaplan-Meier-Estimator's Cumulative Density Curve [24] in Fig. 7. With our developed framework, there is an **83.3%** chance for a user having detected an issue within the first 5 minutes. Without our tool the same chance is only **5.6**%.

covariate	Task	coef	exp(coef)	coef lower 95%	coef upper 95%	Concordance	Log-Likelihood ratio test	z	CoxPH p	Log-Rank p
tool	A	1.9640	7.1278	0.5353	3.3927	0.70	9.65 on 1 df	2.6944	**0.0071**	**0.0016**
tool	B	2.2097	9.1134	0.7922	3.6273	0.74	12.31 on 1 df	3.0553	**0.0022**	**0.0003**
tool	C	2.3902	10.9160	0.6806	4.0998	0.74	11.19 on 1 df	2.7403	**0.0061**	**0.0007**
tool	All Tasks pooled	3.4129	30.3523	1.9264	4.8993	0.72	45.93 on 1 df	4.5000	**<0.0000**	**<0.0000**

Table 4: Time to First Issue Detected depending on tool used Cox-Proportional-Hazard and Log-Rank test results. Results from the CoxPH test. *exp(coef)* shows how much the tool influences the hazard rate.

6.3 H3: Users have faster iteration cycles

Table 5 shows the mean and standard deviation of three measurements together with p-values from Mann-Whitney U Test and Cliff's delta. **Inference Canceled** describes the number of times a participant ran inference but actively canceled it before it finished. **Inference Started** is the number of times a participant ran inference on the task, and **Changes Made** represents the number of times a participant changed either parts of the model or of the inference algorithm.

Task	Measurement	Tool	Baseline	Cliff's δ	p-value
A	# Inference Canceled	1.33 ± 1.44	0.17 ± 0.41	0.625	**0.026**
	# Inference Started	4.33 ± 1.72	2.50 ± 0.84	0.667	**0.024**
	# Changes Made	3.08 ± 1.88	2.17 ± 1.94	0.292	0.341
B	# Inference Canceled	1.08 ± 1.00	0.17 ± 0.41	0.556	**0.048**
	# Inference Started	4.25 ± 1.06	2.33 ± 1.03	0.792	**0.007**
	# Changes Made	3.92 ± 1.31	1.50 ± 1.38	0.778	**0.009**
C	# Inference Canceled	2.42 ± 2.39	0.17 ± 0.41	0.736	**0.010**
	# Inference Started	5.75 ± 2.42	2.33 ± 1.37	0.875	**0.003**
	# Changes Made	4.75 ± 2.09	1.67 ± 1.21	0.847	**0.004**

Table 5: Comparing iterations with and without tool (Mann-Whitney U). The table compares results from the Mann-Whitney U test for all 3 tasks on 3 different indicators for iteration speed. *Inference Canceled* refers to how often participants stopped inference early. *Inference Started* refers to how often participants started the inference routine, and *Changes Made* shows the number of changes made to the model or the inference algorithm.

With the exception of *number of changes* performed at task *A*, we see significant differences across all measurements. Especially the number of times participants ran inference and the number of times they canceled inference early suggest that our framework accelerates the probabilistic programming workflow by facilitating more and faster iterations. For *B* and *C*, we also see a significant increase in changes made by participants. One possible explanation why we do not see this increase for task *A* is that task *A* is solvable in at least three different ways with only a single change. Task *B* and *C* in contrast have no single change solution (to our knowledge) and both require multiple changes to be fully solved.

6.4 H4: Users spend their time more effectively

Table 6 shows the mean and standard deviation of the percentage of time spent in inference relative to the total time spent across all tasks, together with p-values from Mann-Whitney U Test and Cliff's delta. A high percentage is indicative of a participant being able to debug issues while inference was running instead of having to wait for inference to complete and debug afterwards – the typical debugging experience in probabilistic programming before InferLog Holmes.

Task	Tool	Baseline	Cliff's δ	p-value
A	0.54 ± 0.11	0.23 ± 0.10	0.944	**<0.001**
B	0.56 ± 0.11	0.23 ± 0.09	1.000	**<0.001**
C	0.61 ± 0.13	0.27 ± 0.15	0.917	**0.001**

Table 6: Average percent of time spent in inference relative to total time spent on task(Mann-Whitney U).

The results suggest that, indeed, participants have reduced waiting times when using InferLog Holmes and that they can use the time spent in inference effectively for analyzing and debugging problems. Tasks they would otherwise do after inference completed, can now be done at inference time, which suggests better time utilization.

7 Qualitative Analysis and Participants' Perception

In this section, we present a qualitative analysis of participants' perceptions to answer *RQ2* and *RQ3*. We base our analysis on responses to the post-study survey and quotes from participants recorded during the study.

7.1 RQ2: Are online visualizations of MCMC posteriors considered useful by PPL users?

To answer RQ2, we tried to separate the effect of online visualizations and visualizations in general. We designed our questionnaire to differentiate between the selection of visualizations in the tool and the general idea of online visualizations during inference.

7.1.1 The tool provides a good selection of visualizations. As can be seen in Fig. 8, participants considered the selection of visualizations overwhelmingly positive and only a few participants were missing certain visualizations.

One participant mentioned that they missed a clean overview of posterior statistics like Arviz [26] provides through the *summary* function. Another participant related their statement about missing visualizations towards the static model graph. They mentioned that they are "[...] always missing the formula how parameters are added or mutiplied in the graph view as well in the tools case also shape information".

In another section of the questionnaire one participant made the remark that they "[...] would like for each visualization to have an explanation of on how to read it in detail, maybe via a link to another page".

We consider these points valid feedback and incorporating them into a future version of the tool should be straight forward without negatively impacting the tools usefulness.

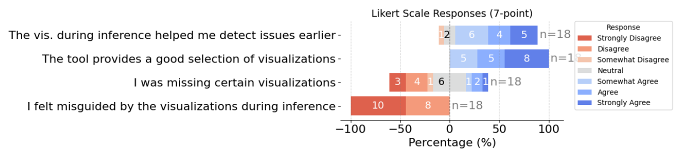


Figure 8: Visualization responses

7.1.2 Online visualizations help detect issues faster. We further see in Fig 8 that almost all participants found the visualizations during inference helpful for issue detection. With only 1 participant somewhat disagreeing and 2 answering neutral. One participant said during their third task: "It is nice to see how the chain develops" and another exclaimed at their second task: "It's actually really nice to see it happening". A third participant compared the tool to tensorboard [31] and called it "very comfortable". One specifically highlighted at the post-study survey, that they "[...] really like not having to wait for the model to be finished especially as pymc model[s] have long runtimes". The same participant also pointed out that the tool helped them "[...] learn new debugging visualizations [they] wouldnt use otherwise as [they] also have to create them which takes time".

7.2 RQ3: Are the warnings raised at inference time considered useful by PPL users?

As before, we tried to separate the perception of the warnings in general and the perception of the warning during inference.

7.2.1 In general, warnings are considered helpful. From the responses presented in Fig. 9, we see that a clear majority of participants strongly agreed that warnings helped them detect issues and only 2 participants "just" agreed and 1 participant somewhat agreed with the statement. While the statement "The warnings helped me solve issues" drew less strong agrees to it, the overall tendency is that the warnings are helpful in that regard as well.

This sentiment is further supported when participants were asked, if the warnings helped them understand issues and if the warnings made issues clearer. In both cases the majority of participants either agreed or strongly agreed with the statement. One participant highlighted in the optional free form input field of the post study questionnaire that "they[the warnings] were quite informative [...]". Another mentioned that the warnings are "very helpful with nice explanations!" and one participant pointed out that "[...] they were very helpfull[sic!], as it is hard to find from the docs, which parameters might influence the behaviour in a certain way". During their third task (task C) one participant exclaimed: "oh that's cool. Well let's try it out, when the tool says that". This was subjected at the funnel/reparametrization warning described in Table 1.

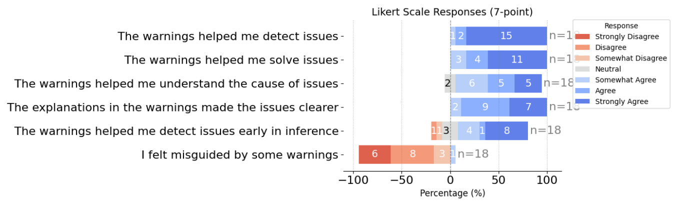


Figure 9: Warning responses

7.2.2 Warnings are considered mostly helpful during inference. While the helpfulness of warnings during inference is still perceived positive overall, participants agreed less strongly to it than to the general helpfulness of the warnings. One participant even disagreed, and another one somewhat disagreed to the statement that "the warnings helped me detect issues early in inference". 3 participants remained neutral on the question, and 4 somewhat agreed. The participant who disagreed with this, however, pointed out that they "[...] only really look[ed] at the warnings after inference".

In contrast, another participant stated that, "[...] their dynamicity really reflects the changing nature of the sampling process and how it should be interpreted".

We take from these results that the overall perception of warnings during inference is positive, but that their usefulness during inference is less clear than after inference. This could potentially be improved upon in future framework instantiations through better heuristics for raising warnings during inference.

7.2.3 Some users felt misguided or were missing information. Fig. 9 shows that most participants disagreed with the statement that they "[...] felt misguided by some warnings" and only one participant somewhat agreed to this. Nevertheless when asked whether they felt misguided by warnings during or after inference, specifically (multiple choice), 4 participants mentioned that they felt misguided by warnings during inference, 1 after inference and 14 that they never felt misguided. This suggests that the warnings system might need adjustments in some areas.

This is also reflected by statements made by two participants. One participant expressed that "in task three it told [them] to reformulate but it was missing what this would change" and another mentioned that "[...] it is somewhat counter intuitive, that e.g. the same warning appears 8 times, if the variable is of dim=8 [...]". Both statements express potential improvements we agree with. More detailed warnings could further improve usability and interpretability and reducing the amounts of almost duplicate warnings displayed could reduce clutter and improve clarity.

8 Related Work

Debugging Probabilistic Programs: Gorinova et al. proposed a live, multiple-representation editor for probabilistic programs targeting novices. The editor features a code editor, a model graph and prior as well as posterior distribution charts [20]. They further showed effectiveness of multiple representations for probabilistic program comprehension in their study. Our work draws from their work and also features multiple representations. While in their work, *live* refers to dynamically updating the representations when source code changes, our work features live updates in the context of inference. Furthermore, our work enables deep insights into the inference process and offers visual debugging aids as well as contextualized warnings and information on resolving potential issues that is model, inference algorithm and PPL specific. Hoppen et al.[23] proposed a debugger for probabilistic programs that, in contrast to our work, resembles classical step-by-step debugging workflows more closely. It allows setting breakpoints in a probabilistic program and shows current values as well as the distribution from which the value was drawn. Their work aims to improve comprehension of probabilistic code execution and debugging classical logic errors stemming from e.g. random numbers of loop iterations. *DEPP* [32] is another debugger for probabilistic programs that is closer to our work which features a combination of static and dynamic analysis. In contrast to our work, their work does not describe a *PPL* independent approach, but relies on a custom *PPL* implemented by them. Furthermore, they do not provide online inference insights in their described systems, which is a key innovation in our approach. SixthSense [13] is another work targeting probabilistic program debugging. In contrast to our work, SixthSense aims to predict model convergence within a specified number of steps ahead-of-run. Similar to our work they offer warnings for likely causes. In contrast to our work, they aim at prediction. Our work on the other hand gives deep insights into the real inference process by providing information and analysis tools at- and after-run-time. ShinyStan [15] is an interactive diagnostic tool with a graphical user interface for analyzing Bayesian posteriors after inference. It features interactive plots and tables commonly used for identifying MCMC inference problems. In contrast to our work it does not feature online analysis of Bayesian inference and does not incorporate details from the model or inference algorithm. Our framework further suggests potential problems based on these informations to reduce the debugging exploration space.

Bug Patterns in Probabilistic Programs and MCMC Diagnostic Work: Lieberman et al. [29] explain the necessity for a dedicated debugging framework for probabilistic programming. Hamada et al. [21] explores bug patterns in PyMC programs and extracts four common bug categories. Donovan et al. [12] gives general guidelines with clear questions and answers targeted to novices for diagnosing MCMC inference. Zitzmann et al. identifies the effective sample size as a possible stopping criterion for MCMC algorithms [39]. Vats et al. [38] presents important MCMC diagnostics, many of which are used in our approach, and an exemplary workflow incorporating these diagnostics. Gabry et al. show a visualization based debugging workflow [14], which was inspirational for the visualization selection in our work. [19] showed how funnels in probabilistic models can be identified and proposed an automatic reparameterisation. Before that, it was shown that there is no general optimal parameterisation and centered, as well as non-centered, have both potential case dependent benefits and drawbacks [34].

Other Work in Online Diagnostics and Visualizations: Tensorboard [31] is a well known tool for online diagnostics and visualizations in deep learning and was an inspiration to the live inference analysis presented in this paper. Glinda [11] is a complete system for supporting data science that features a domain specific language, live programming and a GUI. They showed the benefits of interactive plots and live updates in the domain of data science.

9 Conclusion

In this work, we propose a novel approach to debugging *probabilistic programs*. To the best of our knowledge, we are the first to realize online analysis of the Bayesian inference process. We further describe a holistic debugging approach combining model visualizations, statistics, and analysis plots with warnings and directions. We implemented this approach in InferLog Holmes for MCMC methods and evaluated its effectiveness in a user study with 18 participants. The study showed that our approach marks a significant advancement for debugging probabilistic models that future work can build upon.

Acknowledgments

The authors gratefully acknowledge the financial support under the scope of the COMET program within the K2 Center "Integrated Computational Material, Process and Product Engineering (IC-MPPE)" (Project No 886385). This program is supported by the Austrian Federal Ministries for Economy, Energy and Tourism (BMWET) and for Innovation, Mobility and Infrastructure (BMIMI), represented by the Austrian Research Promotion Agency (FFG), and the federal states of Styria, Upper Austria and Tyrol.

Additionally this research has been partially funded by the European Union through the CLOUDSTARS project (101086248).

References

[1] Oriol Abril-Pla, Virgile Andreani, Colin Carroll, Larry Dong, Christopher J Fonnesbeck, Maxim Kochurov, Ravin Kumar, Junpeng Lao, Christian C Luhmann, Osvaldo A Martin, et al. 2023. PyMC: a modern, and comprehensive probabilistic programming framework in Python. *PeerJ Computer Science* 9 (2023), e1516.

[2] R. Atkinson and J. Flint. 2001. Accessing hidden and hard-to-reach populations: Snowball research strategies. *Social Research Update* 33 (2001). https://pure.york.ac.uk/portal/en/publications/accessing-hidden-and-hard-to-reach-populations-snowball-research-

[3] Viv Bewick, Liz Cheek, and Jonathan Ball. 2004. Statistics review 12: Survival analysis. *Critical Care* 8, 5 (10 2004), 389. doi:10.1186/CC2955

[4] Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. 2019. Pyro: Deep universal probabilistic programming. *Journal of machine learning research* 20, 28 (2019), 1–6.

[5] Christopher M Bishop and Nasser M Nasrabadi. 2006. *Pattern recognition and machine learning*. Vol. 4. Springer, 462.

[6] Markus Böck, Michael Schröder, and Jürgen Cito. 2024. Language-Agnostic Static Analysis of Probabilistic Programs. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*. 78–90.

[7] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. 2011. *Handbook of markov chain monte carlo*. CRC press.

[8] Bob Carpenter, Andrew Gelman, Matthew D Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. 2017. Stan: A probabilistic programming language. *Journal of statistical software* 76 (2017), 1–32.

[9] Jürgen Cito, Philipp Leitner, Martin Rinard, and Harald C. Gall. 2019. Interactive Production Performance Feedback in the IDE. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. 971–981. doi:10.1109/ICSE.2019.00102

[10] D. R. Cox. 1972. Regression Models and Life-Tables. *Journal of the Royal Statistical Society: Series B (Methodological)* 34, 2 (1972), 187–202. arXiv:https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.2517-6161.1972.tb00899.x doi:10.1111/j.2517-6161.1972.tb00899.x

[11] Robert DeLine. 2021. Glinda: Supporting Data Science with Live Programming, GUIs and a Domain-specific Language. In *CHI '21: CHI Conference on Human Factors in Computing Systems,Virtual Event Yokohama, Japan, May 8-13, 2021*, Yoshifumi Kitamura, Aaron Quigley, Katherine Isbister, Takeo Igarashi, Pernille

Bjørn, and Steven Mark Drucker (Eds.). ACM, 309:1–309:11. doi:10.1145/3411764.3445267

[12] Therese M. Donovan and Ruth M. Mickey. 2019. MCMC Diagnostic Approaches. *Bayesian Statistics for Beginners* (7 2019), 212–223. doi:10.1093/OSO/9780198841296.003.0014

[13] Saikat Dutta, Zixin Huang, and Sasa Misailovic. 2022. SixthSense: Debugging Convergence Problems in Probabilistic Programs via Program Representation Learning. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 13241 LNCS (2022), 123–144. doi:10.1007/978-3-030-99429-7{_}7

[14] Jonah Gabry, Daniel Simpson, Aki Vehtari, Michael Betancourt, and Andrew Gelman. [n. d.]. Visualization in Bayesian workflow. ([n. d.]). https://github.com/jgabry/bayes-vis-paper

[15] Jonah Gabry, Stan Development Team, Michael Andreae, Michael Betancourt, Bob Carpenter, Yuanjun Gao, Andrew Gelman, Ben Goodrich, Daniel Lee, Dongying Song, and Rob Trangucci. 2022. ShinyStan. https://mc-stan.org/shinystan/

[16] James Gareth, Daniela Witten, Trevor Hastie, Robert Tibshirani, and Jonathan Taylor. 2023. *An introduction to statistical learning: With applications in python.* Springer International Publishing: Cham, Switzerland, 469–502.

[17] Hong Ge, Kai Xu, and Zoubin Ghahramani. 2018. Turing: a language for flexible probabilistic inference. In *International conference on artificial intelligence and statistics.* PMLR, 1682–1690.

[18] Andrew Gelman, John B Carlin, Hal S Stern, and Donald B Rubin. 1995. *Bayesian data analysis.* Chapman and Hall/CRC.

[19] Maria I Gorinova, Dave Moore, and Matthew D Hoffman. 2020. Automatic Reparameterisation of Probabilistic Programs. (2020). https://proceedings.mlr.press/v119/gorinova20a/gorinova20a.pdf

[20] Maria I. Gorinova, Advait Sarkar, Alan F. Blackwell, and Don Syme. 2016. A live, multiple-representation probabilistic programming environment for novices. *Conference on Human Factors in Computing Systems - Proceedings* (5 2016), 2533–2537. doi:10.1145/2858036.2858221/SUPPL{_}FILE/P2533-GORINOVA.MP4

[21] Shoma Hamada, Haibo Yu, Vo Dai Trinh, Yuri Nishimura, and Jianjun Zhao. 2022. Bug Patterns in Probabilistic Programming Systems. *Proceedings - 2022 IEEE 22nd International Conference on Software Quality, Reliability and Security Companion, QRS-C 2022* (2022), 384–391. doi:10.1109/QRS-C57518.2022.00063

[22] Matthew D. Hoffman, David M. Blei, Chong Wang, and John Paisley. 2013. Stochastic variational inference. *J. Mach. Learn. Res.* 14, 1 (May 2013), 1303–1347.

[23] Alexander Hoppen and Thomas Noll. 2021. A Debugger for Probabilistic Programs. In *Software Engineering and Formal Methods*, Radu Calinescu and Corina S. Păsăreanu (Eds.). Springer International Publishing, Cham, 282–289.

[24] E. L. Kaplan and Paul Meier. 1958. Nonparametric Estimation from Incomplete Observations. *J. Amer. Statist. Assoc.* 53, 282 (1958), 457–481. arXiv:https://www.tandfonline.com/doi/pdf/10.1080/01621459.1958.10501452 doi:10.1080/01621459.1958.10501452

[25] Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M. Blei. 2017. Automatic differentiation variational inference. *J. Mach. Learn. Res.* 18, 1 (Jan. 2017), 430–474.

[26] Ravin Kumar, Colin Carroll, Ari Hartikainen, and Osvaldo Martin. 2019. ArviZ a unified library for exploratory analysis of Bayesian models in Python. *Journal of Open Source Software* 4, 33 (2019), 1143. doi:10.21105/joss.01143

[27] Elisa T Lee and John Wang. 2003. *Statistical methods for survival data analysis.* Vol. 476. John Wiley & Sons.

[28] Michael D Lee and Eric-Jan Wagenmakers. 2014. *Bayesian cognitive modeling: A practical course.* Cambridge university press.

[29] Henry Lieberman, Yen-Ling Kuo, and Valeria Staneva. [n. d.]. Debugging Probabilistic Programs: Lessons from Debugging Research. ([n. d.]). https://web.media.mit.edu/~lieber/Publications/Debugging-Prob-Prog.pdf

[30] Nathan Mantel et al. 1966. Evaluation of survival data and two new rank order statistics arising in its consideration. *Cancer Chemother Rep* 50, 3 (1966), 163–170.

[31] Martín~Abadi, Ashish~Agarwal, Paul~Barham, Eugene~Brevdo, Zhifeng~Chen, Craig~Citro, Greg~S.~Corrado, Andy~Davis, Jeffrey~Dean, Matthieu~Devin, Sanjay~Ghemawat, Ian~Goodfellow, Andrew~Harp, Geoffrey~Irving, Michael~Isard, Yangqing Jia, Rafal~Jozefowicz, Lukasz~Kaiser, Manjunath~Kudlur, Josh~Levenberg, Dandelion~Mané, Rajat~Monga, Sherry~Moore, Derek~Murray, Chris~Olah, Mike~Schuster, Jonathon~Shlens, Benoit~Steiner, Ilya~Sutskever, Kunal~Talwar, Paul~Tucker, Vincent~Vanhoucke, Vijay~Vasudevan, Fernanda~Viégas, Oriol~Vinyals, Pete~Warden, Martin~Wattenberg, Martin~Wicke, Yuan~Yu, and Xiaoqiang~Zheng. 2015. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. https://www.tensorflow.org/

[32] Chandrakana Nandi, Dan Grossman, Adrian Sampson, Todd Mytkowicz, and Kathryn S. McKinley. 2017. Debugging probabilistic programs. *MAPL 2017 - Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, co-located with PLDI 2017* (6 2017), 18–26. doi:10.1145/3088525.3088564

[33] Nathanael Nussbaumer, Markus Böck, and Cito Jürgen. 2026. *Online and Interactive Bayesian Inference Debugging.* doi:10.5281/zenodo.18262423

[34] Omiros Papaspiliopoulos, Gareth O. Roberts, and Martin Sköld. 2007. A General Framework for the Parametrization of Hierarchical Models. *https://doi.org/10.1214/088342307000000014* 22, 1 (2 2007), 59–73. doi:10.1214/088342307000000014

[35] Donald B. Rubin. 1981. Estimation in Parallel Randomized Experiments. *http://dx.doi.org/10.3102/10769986006004377* 6, 4 (12 1981), 377–401. doi:10.3102/10769986006004377

[36] Guido Salvaneschi, Sebastian Proksch, Sven Amann, Sarah Nadi, and Mira Mezini. 2017. On the Positive Effect of Reactive Programming on Software Comprehension: An Empirical Study. *IEEE Transactions on Software Engineering* 43, 12 (2017), 1125–1143. doi:10.1109/TSE.2017.2655524

[37] Pavel Sountsov, Colin Carroll, and Matthew D. Hoffman. 2024. Running Markov Chain Monte Carlo on Modern Hardware and Software. arXiv:2411.04260 [stat.CO] https://arxiv.org/abs/2411.04260

[38] Dootika Vats, Nathan Robertson, James M. Flegal, and Galin L. Jones. 2020. Analyzing Markov chain Monte Carlo output. *Wiley Interdisciplinary Reviews: Computational Statistics* 12, 4 (7 2020), e1501. doi:10.1002/WICS.1501

[39] Steffen Zitzmann, Sebastian Weirich, and Martin Hecht. 2021. Using the Effective Sample Size as the Stopping Criterion in Markov Chain Monte Carlo with the Bayes Module in Mplus. *Psych 2021, Vol. 3, Pages 336-347* 3, 3 (7 2021), 336–347. doi:10.3390/PSYCH3030025

Received 18 July 2025; accepted 17 October 2025