Large Language Bayes

Justin Domke
University of Massachusetts Amherst

Abstract

Many domain experts do not have the time or expertise to write formal Bayesian models. This paper takes an informal problem description as input, and combines a large language model and a probabilistic programming language to define a joint distribution over formal models, latent variables, and data. A posterior over latent variables follows by conditioning on observed data and integrating over formal models. This presents a challenging inference problem. We suggest an inference recipe that amounts to generating many formal models from the large language model, performing approximate inference on each, and then doing a weighted average. This is justified and analyzed as a combination of self-normalized importance sampling, MCMC, and importance-weighted variational inference. Experimentally, this produces sensible predictions from only data and an informal problem description, without the need to specify a formal model.

1 Introduction

Why isn't Bayesian inference more popular? Arguably, a major reason is simply that creating probabilistic models is hard. Most people interested in analyzing data are neither programmers nor experts in statistics. Yet, writing a probabilistic model requires learning a probabilistic programming language (PPL), fluency with a range of statistical distributions, and experience formalizing problem intuitions. Even for experts, this is difficult and error-prone.

A natural idea is to ask the user to describe their problem in plain language and then use a Large Language Model (LLM) to generate a formal probabilistic model. While LLMs can write serviceable models, in practice they struggle in the same way as humans—sometimes the models they create are good and sometimes they aren't [27].

Yet LLMs have one major advantage over humans: they don't mind being asked to create many different candidate models for the same problem.

The central idea of this paper is to mathematically "glue" an LLM to a PPL. Given an informal description of a problem, a joint distribution is defined over (1) formal probabilistic models, (2) observed data, and (3) unobserved target variables. We then condition on data and marginalize out the space of formal models to get a final posterior over target variables.

The main contributions of this paper are:

- A new problem definition, in which an informal description and a dataset define a posterior via an LLM and a PPL. (Sec. 2)
- A broad algorithmic recipe for solving the resulting inference problem. (Sec. 3)
- Experiments illustrating that the final approximated posterior captures user intent and is typically better than taking a naive average of formal models. (Sec. 4)
- Theory analyzing the expected accuracy of the suggested inference recipe. (Sec. 5)

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

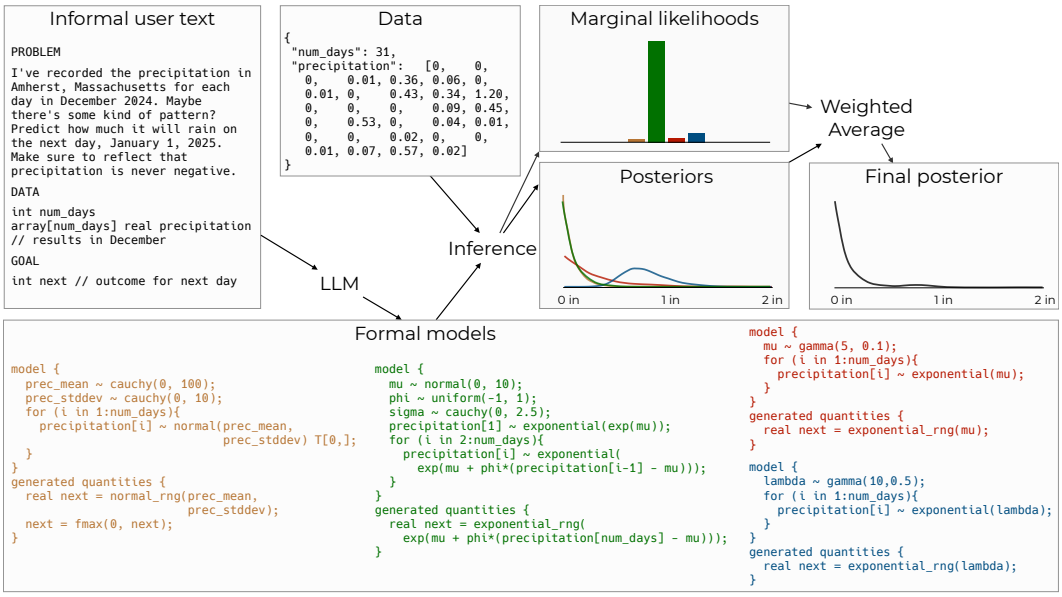

Figure 1: **The basic idea.** Given informal user text, an LLM generates a set of candidate formal models. Inference is performed on each and the posteriors are combined with weight proportional to the marginal likelihood. Here four (real) LLM-generated formal models in the Stan language are shown in different colors, with corresponding colors for marginal likelihoods and posteriors.

2 The basic idea

The user provides a plain-language description of the problem, including anything they know about the phenomena under study, the types and shapes of the input data, what assumptions should be used, and what target variables should be predicted. For example, a user might write the informal text shown at the top left of Fig. 1. Given such a string, and asked to create models in the Stan PPL [5], an LLM could output any of the formal models shown at the bottom of Fig. 1.

Denote the informal user text as t and a formal model as m. Since LLMs are stochastic, we can think of one (along with a system prompt, hyperparameters, etc.) as defining a distribution

$$p\left(\underbrace{m}_{\substack{\text{formal}\\\text{model}}} \mid \underbrace{t}_{\substack{\text{textual}\\\text{description}}}\right). \tag{1}$$

Meanwhile, we can think of a PPL as defining a distribution over target variables z and data x, conditional on the formal model. That is, we may think of a PPL as defining a distribution

$$p\left(\underbrace{z}_{\substack{\text{target}\\\text{variables}}}, \underbrace{x}_{\substack{\text{observed}\\\text{data}}} \mid \underbrace{m}_{\substack{\text{formal}\\\text{model}}}\right). \tag{2}$$

The core idea of this paper is to glue the above two distributions together to define the joint

$$p(z, x, m|t) = \underbrace{p(m|t)}_{\text{LLM}}\ \underbrace{p(z, x|m)}_{\text{PPL}}. \tag{3}$$

Our hypothesis is that the distribution defined in Eq. (3) is a good one. We are interested in the posterior over z, conditioning on x and t, but integrating out m. It is not hard to show that this is

$$\underbrace{p(z|x,t)}_{\text{final posterior}} = \sum_m \underbrace{p(m|x,t)}_{\substack{\text{posterior}\\\text{model weight}}}\ \underbrace{p(z|x,m)}_{\substack{\text{posterior of}\\\text{model } m \text{ (PPL)}}}. \tag{4}$$

Here, $p(z|x,m)$ is the posterior for model m (as defined by the PPL in Eq. (2)) and

$$\underbrace{p(m|x,t)}_{\substack{\text{posterior}\\\text{model weight}}} \propto \underbrace{p(m|t)}_{\substack{\text{prior model}\\\text{weight (LLM)}}}\ \underbrace{p(x|m)}_{\substack{\text{marginal likelihood}\\\text{of model } m \text{ (PPL)}}}. \tag{5}$$

Note that the final posterior in Eq. (4) can be seen as an instance of Bayesian model averaging [16, 32], just with a prior over models that's defined using an LLM.

Algorithm 1 Theoretical exact LLB algorithm (intractable)

1. Input textual description t and data x.
2. For all possible model strings m:
   (a) Compute model probability $p(m|t)$. // using LLM
   (b) Compute posterior $p(z|x,m)$ and marginal likelihood $p(x|m)$. // under PPL
3. Set $w^{(m)} \propto p(m|t)p(x|m)$, where $\sum_m w^{(m)} = 1$.
4. Return final posterior $p(z|x,t) = \sum_m w^{(m)} p(z|x,m)$.

2.1 Varying latent spaces

Note that the user is *not* expected to specify all latent variables, only the target variables to predict. This is crucial since, as illustrated in Fig. 1, different formal models typically have different latent spaces. This is fine. All that's needed is that each model produced by the LLM contains the target variables z specified in the user prompt t. Formally, suppose that model m defines some distribution

$$p(z, x, u^{(m)}|m), \tag{6}$$

where $u^{(m)}$ varies in meaning (and dimensionality) for different models m. Then Eqs. (1) to (5) are all still correct. The final posterior remains as in Eq. (4), just with $p(z,x|m)$ interpreted as Eq. (6) after marginalizing out $u^{(m)}$. The posterior weights in Eq. (5) make no reference to the latent space, and so need no modification.

2.2 Comparison to flat averaging

It is useful to contrast the final posterior $p(z|x,t)$ to the result of taking a "flat" average of of models sampled from the LLM, i.e.

$$p_{\text{flat}}(z|x,t) = \sum_m \underbrace{p(m|t)}_{\substack{\text{prior model} \\ \text{weight (LLM)}}} \underbrace{p(z|x,m)}_{\substack{\text{posterior of} \\ \text{model } m \text{ (PPL)}}}. \tag{7}$$

Clearly $p_{\text{flat}}(z|x,t)$ is not equal to $p(z|x,t)$ as defined by Eq. (4), but it can be seen as an "ensemble of posteriors". The difference is that p_{flat} gives model m weight equal to the prior $p(m|t)$ whereas $p(z|x,t)$ gives weight that is also proportional to the marginal likelihood $p(x|m)$ (See Eq. (5)). Thus, the fundamental difference between the true posterior and "flat" posterior is that the former gives more influence to models that are more consistent with the observed data.

3 Inference

While mathematically simple, the posterior $p(z|x,t)$ in Eq. (4) is computationally difficult. In principle, one might imagine computing it as in Alg. 1, but this will rarely be practical. One familiar issue is that for a given model m, it's usually difficult to compute the posterior $p(z|x,m)$ or the marginal likelihood $p(x|m)$. In addition, the space of models m is very large or possibly infinite.

A more subtle issue is that while we assume one can sample from $p(m|t)$, the probability $p(m|t)$ is often unavailable. Many commercial LLM providers decline to share this information. Further, it's often beneficial to ask LLMs to "think out loud" before producing a final answer. This too makes $p(m|t)$ intractable, since the same model could appear after many different "thinking" passages.

So, while the posterior in Eq. (4) can be seen as an instance of Bayesian model averaging, existing *algorithms* for Bayesian model averaging (e.g. reversible jump MCMC [8, §11]) seem difficult to apply, as they involve explicitly iterating over all models and/or using evaluations of $p(m|t)$. For these reasons, $p(z|x,t)$ appears to represent a novel inference challenge.

To approximate the final posterior, this paper will use the recipe in Alg. 2. The simplest interpretation of this recipe is as a heuristic approximation of Alg. 1 where the sum over all models is replaced by random sampling, and $\tilde{p}(z|x,m)$ and $\tilde{p}(x|m)$ denote approximations of the posterior and marginal likelihood for a given model m. Sec. 5 will give more a formal justification and analysis.

Algorithm 2 Suggested generic approximate LLB recipe.

1. Input textual description t and data x.
2. For $n = 1, 2, \cdots, N$:
   (a) Sample model $m^{(n)} \sim p(m|t)$. // using LLM
   (b) Approximate posterior $\tilde{p}(z|x, m^{(n)})$ and marginal likelihood $\tilde{p}(x|m^{(n)})$. // under PPL
3. Set $w^{(n)} \propto \tilde{p}(x|m^{(n)})$, where $\sum_{n=1}^{N} w^{(n)} = 1$.
4. Return final approximate posterior $p(z|x,t) \approx \sum_{n=1}^{N} w^{(n)} \tilde{p}(z|x, m^{(n)})$.

3.1 Preliminary observations

We experimented with using various LLMs to generate formal Bayesian models in various PPLs, including Stan [5], NumPyro [26] and PyMC [1]. LLMs seemed better at generating Stan code, perhaps since more Stan code is available and included in LLM datasets. We also found it was helpful to give LLMs a system prompt that instructed them to describe their modeling strategy in words before producing a formal model. Finally, to aid in-context learning, we found it was helpful to provide examples of user prompts along with "good" responses.

3.2 Generating models

These experiments used a system prompt (Appendix F.1) that told the LLM to first generate a "thoughts" block that would describe the modeling strategy in words, and then create the formal Stan model. A few additional instructions were used to avoid common mistakes and to encourage the LLM to create well-normalized distributions.

For in-context learning, we provided the LLM with six examples of ostensible user inputs, along with high quality outputs (Appendix G). Each input followed the format illustrated in Fig. 1 with `PROBLEM` block, describing the problem, a `DATA` block describing the data, and a `GOAL` block describing the target variable(s). Each output followed the instructions in the system prompt, with a `THOUGHTS` block describing the modeling strategy and a `MODEL` block with formal Stan code.

For the problems below, 1024 models were generated using Llama-3.3-70B [14, 21] with 4-bit AWQ quantization [19]. Since many models were generated from the same prompt, continuous batching (parallel inference) greatly increased inference speed. Using a single A100 GPU, generating 1024 models took 10-15 minutes, depending on the problem.

3.3 Inference

Outputs from the LLM were rejected if they did not contain `GOAL` and `MODEL` blocks or if the code in the model block did not compile. For models that compiled, 2 chains of Stan's implementation of NUTS [17] were run for 10,000 iterations each. These samples were then considered as representing the approximate posterior $\tilde{p}(z|x, m^{(n)})$.

It remains only to estimate the marginal likelihood $p(x|m^{(n)})$. While methods exist to estimate the marginal likelihood from a set of posterior samples [23], these are often considered unreliable [22]. Instead, we elected to use variational bounds. The typical approach is to create some variational distribution $q(z)$ (e.g. a Gaussian) and optimize it to maximize the lower-bound

$$\mathrm{ELBO}(q) = \mathop{\mathbb{E}}_{\mathsf{z} \sim q} \log \frac{p(\mathsf{z}, x|m^{(n)})}{q(\mathsf{z})} \leq \log p(x|m^{(n)}). \tag{8}$$

This bound will be loose if the true posterior $p(z|x, m^{(n)})$ is not in the variational family. For a tighter-bound, we pursued importance-weighted bounds [3] of the form

$$L(q) = \mathop{\mathbb{E}}_{\mathsf{z}^1, \cdots, \mathsf{z}^K \sim q} \log \frac{1}{K} \sum_{k=1}^{K} \frac{p(\mathsf{z}^k, x|m^{(n)})}{q(\mathsf{z}^k)} \leq \log p(x|m^{(n)}). \tag{9}$$

where $\mathsf{z}^1, \cdots, \mathsf{z}^K$ are independent samples from q.

Algorithm 3 The variant of Alg. 2 used in the experiments of this paper.

1. Input textual description t and data x.
2. For $n = 1, 2, \cdots, N$:
   (a) Sample model $m^{(n)} \sim p(m|t)$. // using LLM
   (b) Draw samples $z^{(n,1)}, \cdots, z^{(n,K)} \sim p(z|x, m^{(n)})$ using MCMC. // under PPL
   (c) Compute a lower bound $L^{(n)}$ on the marginal likelihood $\log p(x|m^{(n)})$ using Eq. (9).
3. Set $w^{(n)} \propto \exp L^{(n)}$, where $\sum_{n=1}^{N} w^{(n)} = \frac{1}{K}$.
4. Return the samples $\{z^{(n,k)}\}$ where $z^{(n,k)}$ is given weight $w^{(n)}$.

It remains to choose the distribution q. The typical approach with importance-weighted bounds [3, 2, 10, 11] is to explicitly optimize q to maximize Eq. (9). However, it is difficult to make stochastic optimization fully automatic and reliable across thousands of heterogenous models. Instead, we exploit the fact that we have already (approximately) sampled from $p(z|x, m^{(n)})$ using MCMC and simply set q to be a Gaussian distribution matching the empirical mean and variance of the MCMC samples.

This seemingly-heuristic choice is justified by the fact that Eq. (9) is asymptotically tight (as $K \to \infty$) with an asymptotic rate of $1/K$ and a constant determined by the χ^2 divergence between q and p [20, 11]. From the perspective of alpha-divergences, maximizing $\mathrm{ELBO}(q)$ is equivalent to minimizing $\mathrm{KL}(q\|p) = D_0(p\|q)$, an "exclusive" divergence. Eq. (9) would be (asymptotically) tightest if we could minimize the "inclusive" divergence $\chi^2(p\|q) = D_2(p\|q)$, but this is difficult to optimize [13]. Matching the mean and covariance of the posterior is equivalent to minimizing $\mathrm{KL}(q\|p) = D_1(p\|q)$, and so is a pragmatic compromise. Experimentally, this performed at least as well as explicitly optimizing Eq. (9), but was faster and more reliable.

The full inference recipe is summarized as Alg. 3. More details are in Appendix F.

4 Experiments

It is likely that all standard models and datasets are included in LLM training data. To avoid the risk that the LLM would simply "remember" human-written models, these experiments use all-new problems and datasets.

Since this paper will presumably also be included in future LLM datasets, any future work in this direction should not use these problems for evaluation with any LLM with a knowledge cutoff after the date this paper was first made public, namely April 21, 2025.

```
PROBLEM

I've been recording if it rains
each day, with a 1 for rain and 0
for no rain. Maybe there's some
kind of pattern? Predict if it will
 rain the next day.

DATA

int num_days
array[num_days] int rain // results
 so far, in order

GOAL

int next // outcome for next day
```

```
{"num_days":22,
 "rain":[1,1,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,1,1,1]}
```

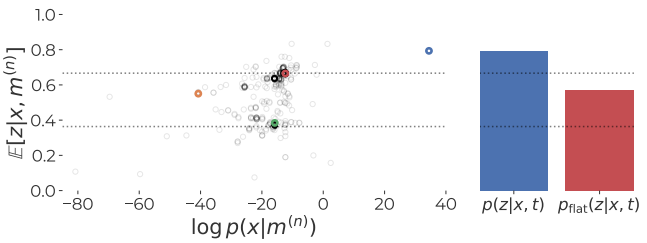

Figure 2: The **rain** problem. Left: Informal user text t. Top right: The given data x. Bottom center: Estimated marginal likelihoods $p(x|m^{(n)})$ and posterior means $\mathbb{E}[z|x, m^{(n)}]$ for each generated model $m^{(n)}$. Markers for the four models in Fig. 7 are colored. Bottom right: The final posterior mean, compared to a flat average.

$t^{(1)}$ `I just got the coin from the US mint, so I'm almost completely sure that it's a standard US penny.`

$t^{(2)}$ `At first glance, it appears to be a standard US penny, although I haven't examined it closely.`

$t^{(3)}$ `I can see that the penny is quite bent, so I'm pretty sure the bias is different from 0.5, though I can't tell in what direction.`

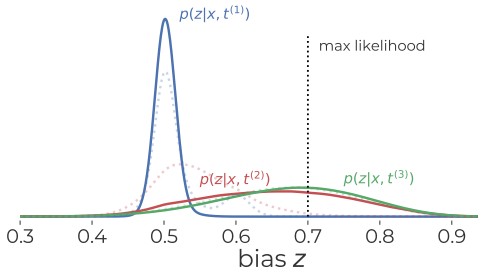

Figure 3: The **coin** problem. Left: Snippets from three different user prompts. Right: Resulting final posteriors, which appear to reflect user intent. Predictions from p_{flat} are shown as faint dotted lines.

4.1 Rain

In this problem (Fig. 2), a user recorded if it rained on a sequence of days and wishes to predict if it will rain on the following day. The data provides two contradictory signals. One the one hand, most days did not have rain. However, adjacent days are correlated and it rained on the last three days.

Out of 1024 generated models, 960 (93.8%) compiled and allowed for inference. Four example models are shown in Fig. 7 (supplement). Many models treat each day as independent and give predictions near the base rate of $8/22 \approx 0.364$. Others give predictions near the between-day consistency rate of $14/21 \approx 0.667$. (Both of these base rates are shown as dotted lines in Fig. 2.) A flat average includes many models of both types and predicts 0.573. However, the final posterior is dominated by a single model that models multi-day dependencies and predicts an even-higher value of 0.793. Detailed results are in Appendix B.1.

4.2 Coin

This problem (Fig. 3) tests the impact of different assumptions stated in the user text. Three prompts are given describing flipping a coin but with different assumptions about the true bias. In all cases, 20 flips were observed, out of which 14 were heads.

Out of 1024 generated models, between 848 (82.8%) and 989 (96.6%) compiled and allowed for inference, depending on the prompt variant. For all three variants, many generated models had marginal likelihoods high enough to contribute to the final estimated posterior. The final estimated posteriors are compared in Fig. 3, where the different assumptions seem reflected in the final posteriors. Detailed results are in Appendix B.2, Appendix B.3, and Appendix B.4.

4.3 Polling

In this problem (Fig. 4), a user describes a candidate with fluctuating popularity who is polled by three different pollsters at various times throughout a year, and asks to predict the true popularity on each day. See Appendix B.5 for the full prompt.

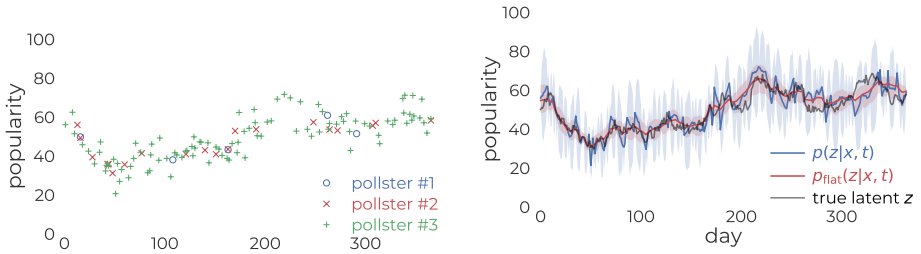

Figure 4: The **polling** problem. Left: Observed data x. Right: Final estimated posterior, compared to a flat average.

Out of 1024 generated models, 568 (55.5%) compiled and allowed for inference. For this problem, the estimated posterior shows little benefit over a flat average. A single model captured essentially all posterior weight, which may indicate inaccurate inference. Detailed results are in Appendix B.5.

4.4 City Temperature

In this problem (Fig. 5), the temperature is given for a set of random days in a handful of random cities around the world, along with the temperature on subsequent days. The user gives a set of test days and asks to predict the temperature on the following days.

Out of 1024 generated models, 736 (71.9%) compiled and allowed for inference. In this problem, again the final estimated posterior isn't obviously better than a flat average. Detailed results are in Appendix B.6.

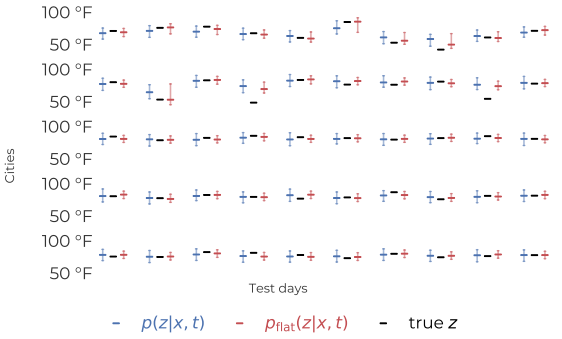

Figure 5: Medians and 90% credible intervals for each city and test day for the the **city temperature** problem.

4.5 Gold

In this problem (Fig. 6), the user describes a rod with a varying density of gold. Binary tests have been done at different positions and the goal is to infer the true density. Inference is done with two different datasets, one with 30 observations, and one with 150.

This problem proved quite challenging, with only 1-2% of generated models being syntactically valid and allowing inference to proceed. Thus, 16,384 models were generated, out of which only 192 (1.2%) compiled and allowed for inference with 30 data and 71 (0.4%) with 150 data. The marginal likelihoods result in almost all weight concentrating in just three models in the smaller dataset and just one model with the larger dataset. Using these gives sensible results, and clearly better than a "flat" average, at least with 150 data . Detailed results are in Appendix B.7 and Appendix B.8.

Finally, note that some of the above results may be impacted by the fact that if truncated distributions are created through the use of constraints in Stan [5], densities are not re-normalized to reflect the truncation. So, for example, if a model includes a standard half-normal, computed log-densities for that model will be lower they should be by a factor of $\log \frac{1}{2} \approx 0.693$ nats. Unfortunately, this seems difficult to address automatically. This is "safe" in the sense that it can decrease but not increase

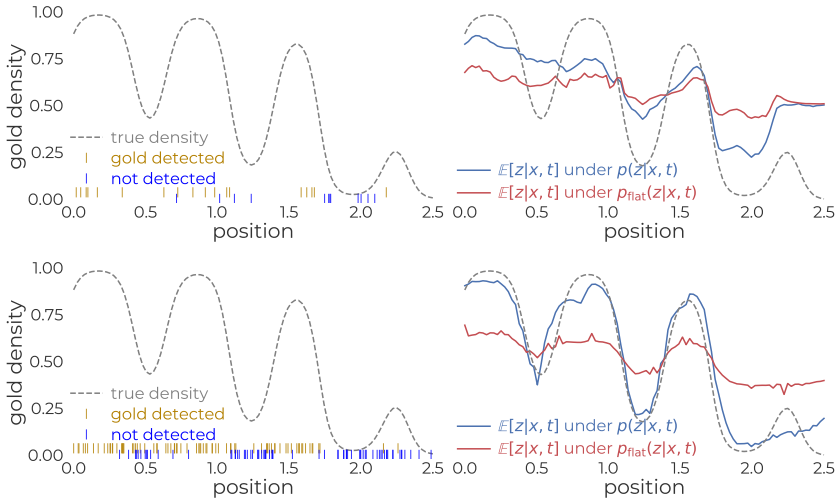

Figure 6: The **Gold** problem. Left column: Observed data. Right column: Final estimated posterior, compared to a flat average. Top row: Results with 30 observations. Bottom row: 150 observations.

estimated marginal likelihoods, but it means that some "good" models may have lower weights then they deserve in all the above experiments.

5 Theory

This section seeks to understand how accurately Alg. 2 will approximate the true posterior, in particular how the accuracy depends on the number of samples N, and the quality of the posterior and marginal likelihood approximations, and the LLM-defined prior over models $p(m|t)$.

5.1 Importance sampling in model space

Recall the final posterior $p(z|x,t) = \sum_m p(m|x,t)p(z|x,m)$ from Eq. (4). To cope with the large model space, one idea would be to perform importance sampling—to draw $m \sim p(m|t)$ and reweight $p(z|x,m)$ by the ratio $p(m|x,t)/p(m|t)$. Unfortunately, this is infeasible, since the normalizing constant for $p(m|x,t)$ in Eq. (5) is intractable and the probabilities $p(m|t)$ are unavailable (Sec. 3).

An alternative is *self-normalized* importance sampling (SNIS)—to draw a set of models from $p(m|t)$ and give each a weight proportional to $p(m|x,t)/p(m|t)$, but normalized to sum to one. It can be shown (Appendix C.1) that this estimator simplifies into

$$p(z|x,t) \approx \sum_{n=1}^{N} w^{(n)}\, p(z|x,m^{(n)}), \quad w^{(n)} = \frac{p(x|m^{(n)})}{\sum_{n'=1}^{N} p(x|m^{(n')})}, \tag{10}$$

where $m^{(1)}, \cdots, m^{(N)} \sim p(m|t)$. This is equivalent to Alg. 4 (Appendix D).

This will rarely be practical, since the posteriors $p(z|x,m^{(n)})$ and marginal likelihoods $p(x|m^{(n)})$ must be approximated. But it is useful to analyze it to isolate error due purely to SNIS. Suppose we are interested in the expectation of some function f with respect to the posterior, i.e. $\mu = \mathbb{E}_{p(z|x,t)} f(z)$. If $\tilde{p}(z|x,t)$ is the approximation in (Eq. (10)), one can estimate μ with

$$\hat{\mu}_N = \mathop{\mathbb{E}}_{\tilde{p}(z|x,t)} f(z) = \sum_{n=1}^{N} w^{(n)} \mathop{\mathbb{E}}_{p(\mathsf{z}|x,m^{(n)})} \left[p(\mathsf{z}|x,m^{(n)}) \right]. \tag{11}$$

Standard results (Appendix C.2) show that $\hat{\mu}_N$ is asymptotically unbiased with asymptotic variance

$$\mathbb{V}[\hat{\mu}_N] \approx V_N := \frac{1}{N} \mathop{\mathbb{E}}_{p(\mathsf{m}|t)} \left[\left(\frac{p(\mathsf{m}|x,t)}{p(\mathsf{m}|t)} \right)^2 (g(\mathsf{m}) - \mu)^2 \right].$$

Further, if $|g(m) - \mu| \le \delta$ (as would be true, for example, if $|f(z) - \mu| \le \delta$), then,

$$V_N \le \frac{\delta^2}{N} \Big(1 + \chi^2 \left(p(\mathsf{m}|x,t) \| p(\mathsf{m}|t) \right) \Big), \tag{12}$$

where $\chi^2(p\|q)$ again denotes the chi-squared divergence. Eq. (12) suggests that for Alg. 4 to achieve a given accuracy, N must be proportional to $\chi^2\left(p(\mathsf{m}|x,t)\|p(\mathsf{m}|t)\right)$. This is an "inclusive" divergence, meaning that if $p(m|t)$ is small but $p(m|x,t)$ is large for some model m, this greatly increases the divergence, while the reverse situation only causes a modest increase.

This suggests that it is important that the LLM-defined prior $p(m|t)$ be relatively broad: The accuracy of the estimated posterior is harmed more by the prior being "overconfident" than being "underconfident". In particular, since $p(m|x,t)$ only varies from $p(m|t)$ by a factor of $p(x|m)$ (and normalization), it is important that $p(m|t)$ should not give vanishingly small probability to models m where the marginal likelihood $p(x|m)$ is large.

5.2 Approximate inference

This section considers the impact of approximating the marginal likelihood and the posterior for a given model. Suppose temporarily that for each m, some approximation $q(z|x,m) \approx p(z|x,m)$ is known. Define the joint approximation

$$q(z,m|x) = q(m|x)q(z|x,m), \tag{13}$$

where $q(m|x)$ remains to be chosen. In principle, we would like to optimize $q(m|x)$ to make the marginal $q(z|x)$ close to $p(z|x,t)$, but this seems difficult, since even evaluating $p(z|x,t)$ is intractable (Eq. (4)). Instead we propose to choose $q(m|x)$ to minimize the joint divergence

$$\mathrm{KL}(q(\mathsf{z},\mathsf{m}|x)\|p(\mathsf{z},\mathsf{m}|x,t))\,, \tag{14}$$

which (by the chain rule of KL-divergence [9, Thm 2.5.3]) is an upper bound on the divergence from $q(z|x)$ to $p(z|x,t)$. The following result gives the the optimal $q(m|x)$. (Proof in Appendix E.1)

Theorem 1. *Suppose $p(z,x,m|t)$ and $q(z|x,m)$ are fixed. Then $\mathrm{KL}(q(\mathsf{z},\mathsf{m}|x)\|p(\mathsf{z},\mathsf{m}|x,t))$ is minimized by*

$$q(m|x) \propto p(m|t)p(x|m) \times \exp\Big(-\mathrm{KL}(q(\mathsf{z}|x,m)\|p(\mathsf{z}|x,m))\Big), \tag{15}$$

with a resulting joint divergence of

$$\mathrm{KL}(q(\mathsf{z},\mathsf{m}|x)\|p(\mathsf{z},\mathsf{m}|x,t)) = -\log \mathop{\mathbb{E}}_{p(\mathsf{m}|x,t)} \exp\Big(-\mathrm{KL}(q(\mathsf{z}|x,\mathsf{m})\|p(\mathsf{z}|x,\mathsf{m}))\Big). \tag{16}$$

Informally, the optimal $q(m|x)$ in Eq. (15) can be seen as taking $p(m|x,t) \propto p(m|t)p(x|m)$ from Eq. (5) and down-weighting models m where $q(z|x,m)$ is a worse approximation of $p(z|x,m)$. Eq. (16) is less intuitive. As KL-divergence is non-negative, the inner expectation is between 0 and 1, with larger values giving a smaller joint divergence. Thus, inaccuracy in $q(z|x,m)$ "hurts" more when $p(m|x,t)$ is larger. This same intuition follows from noting that Eq. (16) can be upper-bounded by $\mathbb{E}_{p(\mathsf{m}|x,t)}\,\mathrm{KL}(q(\mathsf{z}|x,\mathsf{m})\|p(\mathsf{z}|x,\mathsf{m}))$ (See Thm. 4 in Appendix C.5).

Thm. 1 is not immediately useful, since the marginal likelihood $p(x|m)$ and KL divergence for a given m are typically intractable. However, recall the ELBO decomposition

$$\log p(x|m) = \mathrm{ELBO}(m) + \mathrm{KL}(q(\mathsf{z}|x,m)\|p(\mathsf{z}|x,m))\,, \tag{17}$$

where the (tractable) evidence lower-bound (ELBO) (now written as a function of the model m rather than the distribution q as in Eq. (8)) is

$$\mathrm{ELBO}(m) := \mathop{\mathbb{E}}_{q(\mathsf{z}|x,m)} \log \frac{p(\mathsf{z},x|m)}{q(\mathsf{z}|x,m)}. \tag{18}$$

Variational inference (VI) algorithms maximize the ELBO which is equivalent (since $\log p(x|m)$ is fixed) to minimizing the KL-divergence. Using this decomposition, Thm. 1 can be given in the following alternate form. (Proof in Appendix E.2.)

Corollary 2. *Under the same assumptions as Thm. 1, the optimal $q(m|x)$ can also be written as*

$$q(m|x) \propto p(m|t)\exp\Big(\mathrm{ELBO}(m)\Big), \tag{19}$$

with a resulting joint divergence of

$$\mathrm{KL}(q(\mathsf{z},\mathsf{m}|x)\|p(\mathsf{z},\mathsf{m}|x,t)) = \log p(x|t) - \log \mathop{\mathbb{E}}_{p(\mathsf{m}|t)} \exp\left(\mathrm{ELBO}(\mathsf{m})\right). \tag{20}$$

Note that Eq. (19) was previously shown by [18, Sec. 2.2] and Ohn & Lin [24, Thm 2.1].

Taken literally, Thm. 2 suggests that to find the joint approximation $q(z,m|x)$ closest to $p(z,m|x,t)$, one should loop over all models m, independently do variational inference on each, and then average the variational distributions using the weights from Eq. (19). Such an algorithm is shown as Alg. 6 (Appendix D), but is not practical since again the space of models m is large and the LLM probabilities $p(m|t)$ are unavailable.

However, one can also incorporate the SNIS ideas from Sec. 5.1. Suppose we would like to estimate $q(z|x) = \mathbb{E}_{q(\mathsf{m}|x)}\, q(z|x,\mathsf{m})$, where $q(z|x,m)$ is an approximation of $p(z|x,m)$ and $q(m|x)$ are the weights from Eq. (19). If we use a proposal distribution $p(m|t)$, then it's easy to show (Appendix C.3) that the self-normalized weights are proportional to $\exp(\mathrm{ELBO}(m))$. This results in Alg. 5. This is an instance of the recipe in Alg. 2, since $\mathrm{ELBO}(m)$ is a lower-bound on $\log p(x|m)$.

5.3 Algorithmic variants

It can be beneficial to use approximating distributions $q(z|x,m)$ where only a *lower-bound* on the ELBO is available. One example of this would be to take advantage of Monte Carlo VI methods like importance-weighted VI [3, 12, 2, 10, 7]. Another example would be when one believes MCMC can efficiently sample from $p(z|x,m)$. If one thinks of those samples as representing an approximating distribution $q(z|x,m)$, one might believe it is essentially exact. But even if this is true, estimating the marginal likelihood from samples is famously difficult [23, 22], so one might bound the marginal likelihood using a different method (e.g. variational inference).

To justify this, imagine that one first finds a variational approximation $q(z|x,m)$ for each model m, and sets the model weights $q(m|x)$ using Eq. (19) Then, imagine *replacing* each distribution $q(z|x,m)$ with one with a higher ELBO (lower KL-divergence) while leaving $q(m|x)$ fixed. By the chain rule of KL-divergence, $\mathrm{KL}(q(\mathsf{z},\mathsf{m}|x)\|p(\mathsf{z},\mathsf{m}|x,t)) = \mathrm{KL}(q(\mathsf{m}|x)\|p(\mathsf{m}|x,t)) + \mathrm{KL}(q(\mathsf{z}|\mathsf{m},x)\|p(\mathsf{z}|\mathsf{m},x))$. Improving $q(z|m,x)$ will decrease the second KL-divergence on the right while leaving the first divergence unchanged, meaning the joint divergence can only decrease.

Appendix C.4 gives a generalized version of the above results when the model weights $q(m|t)$ are computed based on inexact ELBO values. A particularly interesting special case is when the distributions $p(z|x,m)$ can be computed exactly (e.g. using MCMC) but the weights $q(z|m)$ are computed based on some bound on the marginal likelihood. (See Alg. 3 in Appendix D.) Then the joint divergence result from Appendix C.4 reduces to $\mathrm{KL}(q(\mathsf{z},\mathsf{m}|x)\|p(\mathsf{z},\mathsf{m}|x,t)) = -\log \mathbb{E}_{p(\mathsf{m}|x,t)} \exp(-\delta^{(\mathsf{m})} + \bar{\delta})$, where $\delta^{(m)}$ is the gap between $\log p(x|m)$ and the lower bound used when computing $q(m|x)$ and $\bar{\delta} = \mathbb{E}_{q(\mathsf{m}|x)}\, \delta^{(\mathsf{m})}$. This result has a somewhat similar intuition as Eq. (16): Constant errors have no effect, and errors on models where $p(m|x,t)$ very is small have little effect. What really matters is if $\delta^{(m)}$ *varies* among models where $p(m|x,t)$ is large.

6 Discussion and limitations

This paper proposed the LLB scheme for defining a posterior from natural language (Sec. 2), suggested a broad approximate inference strategy for it (Sec. 3), tested it experimentally (Sec. 4) and analyzed the error of the inference strategy theoretically (Sec. 5).

One direction for future work would be to investigate better ways of using LLMs to define distributions over models—essentially better distributions $p(m|t)$. There are many obvious options, such as different system prompts, more/better examples, different formatting of user inputs, or training (or fine-tuning) an LLM specifically for this task.

Another direction is better inference methods. The suggested recipe involves doing inference independently on each sampled model. While fully parallelizable, this is expensive, and practical only on relatively small problems. We intend no claim of optimality. When doing inference on many models for the same task, many models are often quite similar, suggesting it might be possible to share work between models. It might also be possible to use constrained generation to guarantee that only valid models were created, eliminating the overhead of generating and then rejecting some syntactically invalid models.

6.1 Related work

In terms of usage of LLMs for Bayesian modeling, Wong et al. [33] suggest a system where an LLM translates natural language text into a PPL. This is related to our approach, but assumes that "facts" as provided to the LLM as text, rather than being *described* in text and then provided to inference as numerical data. Capstick et al. [4] consider using an LLM to take a natural language description and predict the parameters of Gaussian priors (to be used in concert with known likelihood models). Selby et al. [29] also consider a similar idea, and compare the results from several LLMs to priors elicited from human experts Stefan et al. [30]. Requeima et al. [28] suggest a scheme to make probabilistic prediction from natural language plus data. Other work considers the use of LLMs for time-series predictions [15, 34] or as regressors [31]. Choi et al. [6] suggest a general framework for using natural language information to inform learning procedures via an LLM.

Acknowledgement

This material is based upon work supported in part by the National Science Foundation under Grant No. 2045900.

References

[1] Abril-Pla, O., Andreani, V., Carroll, C., Dong, L., Fonnesbeck, C. J., Kochurov, M., Kumar, R., Lao, J., Luhmann, C. C., Martin, O. A., Osthege, M., Vieira, R., Wiecki, T., and Zinkov, R. PyMC: A modern, and comprehensive probabilistic programming framework in Python. *PeerJ Computer Science*, 9:e1516, September 2023.

[2] Bachman, P. and Precup, D. Training Deep Generative Models: Variations on a Theme. In *NIPS Workshop: Advances in Approximate Bayesian Inference*, 2015.

[3] Burda, Y., Grosse, R., and Salakhutdinov, R. Importance Weighted Autoencoders. In *ICLR*, 2015.

[4] Capstick, A., Krishnan, R. G., and Barnaghi, P. Using Large Language Models for Expert Prior Elicitation in Predictive Modelling, December 2024.

[5] Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., and Riddell, A. Stan: A Probabilistic Programming Language. *Journal of Statistical Software*, 76(1), 2017.

[6] Choi, K., Cundy, C., Srivastava, S., and Ermon, S. LMPriors: Pre-Trained Language Models as Task-Specific Priors. In *NeurIPS 2022 Foundation Models for Decision Making Workshop*, November 2022.

[7] Christian Andersson Naesseth. *Machine Learning Using Approximate Inference: Variational and Sequential Monte Carlo Methods*. PhD thesis, Linköping University, 2018.

[8] Christian Robert. *Monte Carlo Statistical Methods*. 2004.

[9] Cover, T. M. and Thomas, J. A. *Elements of Information Theory*. Wiley-Interscience, Hoboken, N.J, 2nd ed edition, 2006. ISBN 978-0-471-24195-9.

[10] Cremer, C., Morris, Q., and Duvenaud, D. Reinterpreting Importance-Weighted Autoencoders. *arXiv:1704.02916 [stat]*, April 2017.

[11] Domke, J. and Sheldon, D. Importance Weighting and Variational Inference. In *NeurIPS*, 2018.

[12] Domke, J. and Sheldon, D. Divide and Couple: Using Monte Carlo Variational Objectives for Posterior Approximation. In *NeurIPS*, pp. 11, 2019.

[13] Geffner, T. and Domke, J. On the difficulty of unbiased alpha divergence minimization. In *ICML*, pp. 3650–3659, July 2021.

[14] Grattafiori, A., Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Vaughan, A., Fan, A., Goyal, A., Hartshorn, A., Yang, A., Mitra, A., Sravankumar, A., Korenev, A., Hinsvark, A., Rao, A., Zhang, A., Rodriguez, A., Gregerson, A., Spataru, A., Roziere, B., Biron, B., Tang, B., Chern, B., Caucheteux, C., Nayak, C., Bi, C., Marra, C., McConnell, C., Keller, C., Touret, C., Wu, C., Wong, C., Ferrer, C. C., Nikolaidis, C., Allonsius, D., Song, D., Pintz, D., Livshits, D., Wyatt, D., Esiobu, D., Choudhary, D., Mahajan, D., Garcia-Olano, D., Perino, D., Hupkes, D., Lakomkin, E., AlBadawy, E., Lobanova, E., Dinan, E., Smith, E. M., Radenovic, F., Guzmán, F., Zhang, F., Synnaeve, G., Lee, G., Anderson, G. L., Thattai, G., Nail, G., Mialon, G., Pang, G., Cucurell, G., Nguyen, H., Korevaar, H., Xu, H., Touvron, H., Zarov, I., Ibarra, I. A., Kloumann, I., Misra, I., Evtimov, I., Zhang, J., Copet, J., Lee, J., Geffert, J., Vranes, J., Park, J., Mahadeokar, J., Shah, J., van der Linde, J., Billock, J., Hong, J., Lee, J., Fu, J., Chi, J., Huang, J., Liu, J., Wang, J., Yu, J., Bitton, J., Spisak, J., Park, J., Rocca, J., Johnstun, J., Saxe, J., Jia, J., Alwala, K. V., Prasad, K., Upasani, K., Plawiak, K., Li, K., Heafield, K., Stone, K., El-Arini, K., Iyer, K., Malik, K., Chiu, K., Bhalla, K., Lakhotia, K., Rantala-Yeary, L., van der Maaten, L., Chen, L., Tan, L., Jenkins, L., Martin, L., Madaan,

L., Malo, L., Blecher, L., Landzaat, L., de Oliveira, L., Muzzi, M., Pasupuleti, M., Singh, M., Paluri, M., Kardas, M., Tsimpoukelli, M., Oldham, M., Rita, M., Pavlova, M., Kambadur, M., Lewis, M., Si, M., Singh, M. K., Hassan, M., Goyal, N., Torabi, N., Bashlykov, N., Bogoychev, N., Chatterji, N., Zhang, N., Duchenne, O., Çelebi, O., Alrassy, P., Zhang, P., Li, P., Vasic, P., Weng, P., Bhargava, P., Dubal, P., Krishnan, P., Koura, P. S., Xu, P., He, Q., Dong, Q., Srinivasan, R., Ganapathy, R., Calderer, R., Cabral, R. S., Stojnic, R., Raileanu, R., Maheswari, R., Girdhar, R., Patel, R., Sauvestre, R., Polidoro, R., Sumbaly, R., Taylor, R., Silva, R., Hou, R., Wang, R., Hosseini, S., Chennabasappa, S., Singh, S., Bell, S., Kim, S. S., Edunov, S., Nie, S., Narang, S., Raparthy, S., Shen, S., Wan, S., Bhosale, S., Zhang, S., Vandenhende, S., Batra, S., Whitman, S., Sootla, S., Collot, S., Gururangan, S., Borodinsky, S., Herman, T., Fowler, T., Sheasha, T., Georgiou, T., Scialom, T., Speckbacher, T., Mihaylov, T., Xiao, T., Karn, U., Goswami, V., Gupta, V., Ramanathan, V., Kerkez, V., Gonguet, V., Do, V., Vogeti, V., Albiero, V., Petrovic, V., Chu, W., Xiong, W., Fu, W., Meers, W., Martinet, X., Wang, X., Wang, X., Tan, X. E., Xia, X., Xie, X., Jia, X., Wang, X., Goldschlag, Y., Gaur, Y., Babaei, Y., Wen, Y., Song, Y., Zhang, Y., Li, Y., Mao, Y., Coudert, Z. D., Yan, Z., Chen, Z., Papakipos, Z., Singh, A., Srivastava, A., Jain, A., Kelsey, A., Shajnfeld, A., Gangidi, A., Victoria, A., Goldstand, A., Menon, A., Sharma, A., Boesenberg, A., Baevski, A., Feinstein, A., Kallet, A., Sangani, A., Teo, A., Yunus, A., Lupu, A., Alvarado, A., Caples, A., Gu, A., Ho, A., Poulton, A., Ryan, A., Ramchandani, A., Dong, A., Franco, A., Goyal, A., Saraf, A., Chowdhury, A., Gabriel, A., Bharambe, A., Eisenman, A., Yazdan, A., James, B., Maurer, B., Leonhardi, B., Huang, B., Loyd, B., Paola, B. D., Paranjape, B., Liu, B., Wu, B., Ni, B., Hancock, B., Wasti, B., Spence, B., Stojkovic, B., Gamido, B., Montalvo, B., Parker, C., Burton, C., Mejia, C., Liu, C., Wang, C., Kim, C., Zhou, C., Hu, C., Chu, C.-H., Cai, C., Tindal, C., Feichtenhofer, C., Gao, C., Civin, D., Beaty, D., Kreymer, D., Li, D., Adkins, D., Xu, D., Testuggine, D., David, D., Parikh, D., Liskovich, D., Foss, D., Wang, D., Le, D., Holland, D., Dowling, E., Jamil, E., Montgomery, E., Presani, E., Hahn, E., Wood, E., Le, E.-T., Brinkman, E., Arcaute, E., Dunbar, E., Smothers, E., Sun, F., Kreuk, F., Tian, F., Kokkinos, F., Ozgenel, F., Caggioni, F., Kanayet, F., Seide, F., Florez, G. M., Schwarz, G., Badeer, G., Swee, G., Halpern, G., Herman, G., Sizov, G., Guangyi, Zhang, Lakshminarayanan, G., Inan, H., Shojanazeri, H., Zou, H., Wang, H., Zha, H., Habeeb, H., Rudolph, H., Suk, H., Aspegren, H., Goldman, H., Zhan, H., Damlaj, I., Molybog, I., Tufanov, I., Leontiadis, I., Veliche, I.-E., Gat, I., Weissman, J., Geboski, J., Kohli, J., Lam, J., Asher, J., Gaya, J.-B., Marcus, J., Tang, J., Chan, J., Zhen, J., Reizenstein, J., Teboul, J., Zhong, J., Jin, J., Yang, J., Cummings, J., Carvill, J., Shepard, J., McPhie, J., Torres, J., Ginsburg, J., Wang, J., Wu, K., U, K. H., Saxena, K., Khandelwal, K., Zand, K., Matosich, K., Veeraraghavan, K., Michelena, K., Li, K., Jagadeesh, K., Huang, K., Chawla, K., Huang, K., Chen, L., Garg, L., A, L., Silva, L., Bell, L., Zhang, L., Guo, L., Yu, L., Moshkovich, L., Wehrstedt, L., Khabsa, M., Avalani, M., Bhatt, M., Mankus, M., Hasson, M., Lennie, M., Reso, M., Groshev, M., Naumov, M., Lathi, M., Keneally, M., Liu, M., Seltzer, M. L., Valko, M., Restrepo, M., Patel, M., Vyatskov, M., Samvelyan, M., Clark, M., Macey, M., Wang, M., Hermoso, M. J., Metanat, M., Rastegari, M., Bansal, M., Santhanam, N., Parks, N., White, N., Bawa, N., Singhal, N., Egebo, N., Usunier, N., Mehta, N., Laptev, N. P., Dong, N., Cheng, N., Chernoguz, O., Hart, O., Salpekar, O., Kalinli, O., Kent, P., Parekh, P., Saab, P., Balaji, P., Rittner, P., Bontrager, P., Roux, P., Dollar, P., Zvyagina, P., Ratanchandani, P., Yuvraj, P., Liang, Q., Alao, R., Rodriguez, R., Ayub, R., Murthy, R., Nayani, R., Mitra, R., Parthasarathy, R., Li, R., Hogan, R., Battey, R., Wang, R., Howes, R., Rinott, R., Mehta, S., Siby, S., Bondu, S. J., Datta, S., Chugh, S., Hunt, S., Dhillon, S., Sidorov, S., Pan, S., Mahajan, S., Verma, S., Yamamoto, S., Ramaswamy, S., Lindsay, S., Lindsay, S., Feng, S., Lin, S., Zha, S. C., Patil, S., Shankar, S., Zhang, S., Zhang, S., Wang, S., Agarwal, S., Sajuyigbe, S., Chintala, S., Max, S., Chen, S., Kehoe, S., Satterfield, S., Govindaprasad, S., Gupta, S., Deng, S., Cho, S., Virk, S., Subramanian, S., Choudhury, S., Goldman, S., Remez, T., Glaser, T., Best, T., Koehler, T., Robinson, T., Li, T., Zhang, T., Matthews, T., Chou, T., Shaked, T., Vontimitta, V., Ajayi, V., Montanez, V., Mohan, V., Kumar, V. S., Mangla, V., Ionescu, V., Poenaru, V., Mihailescu, V. T., Ivanov, V., Li, W., Wang, W., Jiang, W., Bouaziz, W., Constable, W., Tang, X., Wu, X., Wang, X., Wu, X., Gao, X., Kleinman, Y., Chen, Y., Hu, Y., Jia, Y., Qi, Y., Li, Y., Zhang, Y., Zhang, Y., Adi, Y., Nam, Y., Yu, Wang, Zhao, Y., Hao, Y., Qian, Y., Li, Y., He, Y., Rait, Z., DeVito, Z., Rosnbrick, Z., Wen, Z., Yang, Z., Zhao, Z., Ma, Z., and and others. The Llama 3 Herd of Models, November 2024.

[15] Gruver, N., Finzi, M., Qiu, S., and Wilson, A. G. Large Language Models Are Zero-Shot Time Series Forecasters. In *NeurIPS*, 2023.

[16] Hoeting, J. A., Madigan, D., Raftery, A. E., and Volinsky, C. T. Bayesian Model Averaging: A Tutorial. *Statistical Science*, 14(4):382–401, 1999.

[17] Hoffman, M. D. and Gelman, A. The No-U-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15(1):1593–1623, 2014.

[18] Kejzlar, V., Bhattacharya, S., Son, M., and Maiti, T. Black Box Variational Bayesian Model Averaging. *The American Statistician*, 77(1):85–96, January 2023.

[19] Lin, J., Tang, J., Tang, H., Yang, S., Chen, W.-M., Wang, W.-C., Xiao, G., Dang, X., Gan, C., and Han, S. AWQ: Activation-aware weight quantization for LLM compression and acceleration. In *MLSys*, 2024.

[20] Maddison, C. J., Lawson, J., Tucker, G., Heess, N., Norouzi, M., Mnih, A., Doucet, A., and Teh, Y. Filtering Variational Objectives. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *NeurIPS*, 2017.

[21] Meta. Llama 3.3 | Model Cards and Prompt formats. https://www.llama.com/docs/model-cards-and-prompt-formats/llama3_3/, 2024.

[22] Neal, R. The Harmonic Mean of the Likelihood: Worst Monte Carlo Method Ever. https://radfordneal.wordpress.com/2008/08/17/the-harmonic-mean-of-the-likelihood-worst-monte-carlo-method-ever/, August 2008.

[23] Newton, M. A. and Raftery, A. E. Approximate Bayesian Inference with the Weighted Likelihood Bootstrap. *Journal of the Royal Statistical Society: Series B (Methodological)*, 56(1): 3–26, January 1994.

[24] Ohn, I. and Lin, L. Adaptive variational Bayes: Optimality, computation and applications. *The Annals of Statistics*, 52(1):335–363, February 2024.

[25] Owen, A. *Monte Carlo Theory, Methods and Examples*. 2013.

[26] Phan, D., Pradhan, N., and Jankowiak, M. Composable Effects for Flexible and Accelerated Probabilistic Programming in NumPyro. *arXiv:1912.11554 [cs, stat]*, December 2019.

[27] Price, P. ChatGPT4 writes Stan code so I don't have to. https://statmodeling.stat.columbia.edu/2023/04/18/chatgpt4-writes-stan-code-so-i-dont-have-to/, April 2023.

[28] Requeima, J., Bronskill, J. F., Choi, D., Turner, R. E., and Duvenaud, D. LLM Processes: Numerical Predictive Distributions Conditioned on Natural Language. In *NeurIPS*, November 2024.

[29] Selby, D. A., Spriestersbach, K., Iwashita, Y., Bappert, D., Warrier, A., Mukherjee, S., Asim, M. N., Kise, K., and Vollmer, S. J. Had enough of experts? Elicitation and evaluation of Bayesian priors from large language models. In *NeurIPS 2024 Workshop on Bayesian Decision-making and Uncertainty*, October 2024.

[30] Stefan, A. M., Katsimpokis, D., Gronau, Q. F., and Wagenmakers, E.-J. Expert agreement in prior elicitation and its effects on Bayesian inference. *Psychonomic Bulletin & Review*, 29(5): 1776–1794, October 2022.

[31] Vacareanu, R., Negru, V. A., Suciu, V., and Surdeanu, M. From Words to Numbers: Your Large Language Model Is Secretly A Capable Regressor When Given In-Context Examples. In *First Conference on Language Modeling*, August 2024.

[32] Wasserman, L. Bayesian Model Selection and Model Averaging. *Journal of Mathematical Psychology*, 44(1):92–107, March 2000.

[33] Wong, L., Grand, G., Lew, A. K., Goodman, N. D., Mansinghka, V. K., Andreas, J., and Tenenbaum, J. B. From Word Models to World Models: Translating from Natural Language to the Probabilistic Language of Thought, June 2023.

[34] Xue, H. and Salim, F. D. PromptCast: A New Prompt-Based Learning Paradigm for Time Series Forecasting. *IEEE Transactions on Knowledge and Data Engineering*, 36(11):6851–6864, November 2024.

A Acronyms and notation

Acronyms

- LLM: Large Language Model
- PPL: Probabilistic Programming Language
- BMA: Bayesian Model Averaging
- SNIS: Self-normalized importance sampling
- VI: Variational inference

Notation

- t: Plain-language description of an inference problem.
- m: Formal model definition (in a PPL)
- x: Input data
- z: Query variables / latent variables
- n: Index for sampled model
- $m^{(n)}$: nth sampled model
- $w^{(n)}$: weight given to nth sampled model

Terminology

- Single-model posterior: $p(z|x, m)$
- Posterior model weight: $p(m|x, t) = \frac{p(m|t)p(x|m)}{\sum_{m'} p(m'|t)p(x|m')} = \frac{p(m|t)p(x|m)}{p(x|t)}$
- Final posterior: $p(z|x, t)$

B Experimental details

This section gives detailed results for the experiments. For each problem, we describe the prompt and data, show example generated models, attempt to visualize the posterior of each model, show marginal likelihoods and weights, and the final estimated posterior, compared against a "flat" average as in Eq. (7).

B.1 Rain

In this problem, the user describes recording if it rained on not on a series of days, and wishes to predict if it will rain on the next day. Meanwhile, the given data x provides two contradictory signals: On the one hand, most days did not have rain. However, adjacent days are correlated and it rained on the final day.

```
PROBLEM

I've been recording if it rains each day, with a 1 for
rain and 0 for no rain. Maybe there's some kind of pattern
? Predict if it will rain the next day.

DATA

int num_days
array[num_days] int rain // results so far, in order

GOAL

int next // outcome for next day
```

```
{"num_days":22,
 "rain":[1,1,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,1,1,1]}
```

Out of 1024 models generated, 742 (72.5%) compiled and allowed for inference. Fig. 7 shows four example models. Fig. 8 shows the estimated marginal likelihoods and posteriors for each of the valid models. Finally, Fig. 9 shows the estimated marginal likelihoods and corresponding weights for the top-scoring 50 valid models, as well as the final estimated posterior $p(z|x,t)$ compared to a "flat" average $p_{\text{flat}}(z|x,t)$.

In this case, there are two common "clusters" of models. One consists of models similar to $m^{(247)}$ (Fig. 7c) that treat each day as independent and give predictions near the base rate of $\frac{8}{22} \approx .364$. A second consists of models similar to $m^{(463)}$ (Fig. 7b) that model the probability that rain is followed by no-rain or vice-versa and give predictions near the base rate of $\frac{14}{21} \approx 0.667$ for between-day consistency. Models from the first cluster tend to have lower-marginal likelihoods than models from the second cluster. However, the top-scoring model $m^{(742)}$ (Fig. 7a) has a still-higher marginal likelihood, and so receives essentially all weight in the estimated final posterior $p(z|x,t)$.

```
data {
  int num_days;
  array[num_days] int<lower=0, upper=1> rain;
}
parameters {
  real<lower=-10, upper=10> logit_initial;
  real autoregressive_coefficient;
  array[num_days-1] real noise;
}
transformed parameters{
  array[num_days] real logit_prob;
  logit_prob[1] = logit_initial;
  for(i in 2:num_days){
    logit_prob[i] = logit_prob[i-1] *
        autoregressive_coefficient + noise[i-1];
  }
}
model {
  logit_initial ~ uniform(-10,10);
  autoregressive_coefficient ~ normal(0,1);
  noise ~ cauchy(0,10);
  for (i in 1:num_days){
    rain[i] ~ bernoulli_logit(logit_prob[i]);
  }
}
generated quantities {
  real logit_next = logit_prob[num_days] *
      autoregressive_coefficient + cauchy_rng(0,10);
  int next = bernoulli_logit_rng(logit_next);
}
```

(a) $m^{(742)}$, the top-scoring model.

```
data {
  int num_days;
  array[num_days] int<lower=0, upper=1> rain;
}
parameters {
  real<lower=0, upper=1> p_rain_given_rain;
  real<lower=0, upper=1> p_rain_given_no_rain;
}
model {
  p_rain_given_rain ~ beta(1,1);
  p_rain_given_no_rain ~ beta(1,1);
  rain[1] ~ bernoulli(0.5); // uniform prior for first day
  for (i in 2:num_days){
    if (rain[i-1]==1){
      rain[i] ~ bernoulli(p_rain_given_rain);
    } else {
      rain[i] ~ bernoulli(p_rain_given_no_rain);
    }
  }
}
generated quantities {
  int next;
  if (rain[num_days]==1){
    next = bernoulli_rng(p_rain_given_rain);
  } else {
    next = bernoulli_rng(p_rain_given_no_rain);
  }
}
```

(b) $m^{(463)}$, typical of one "cluster" of models.

```
data {
  int num_days;
  array[num_days] int<lower=0, upper=1> rain;
}
parameters {
  real<lower=0, upper=1> p;
}
model {
  p ~ uniform(0,1);
  for (i in 1:num_days){
    rain[i] ~ bernoulli(p);
  }
}
generated quantities {
  int next = bernoulli_rng(p);
}
```

(c) $m^{(247)}$, typical of a second cluster.

```
data {
  int num_days;
  array[num_days] int<lower=0, upper=1> rain; // results so
      far
}
parameters {
  real<lower=0, upper=1> p01;
  real<lower=0, upper=1> p11;
}
model {
  p01 ~ beta(20, 20);
  p11 ~ beta(20, 20);
  rain[1] ~ bernoulli(0.5); // no information about the first
       day
  for (i in 2:num_days){
    rain[i] ~ bernoulli(rain[i-1] ? p11 : p01);
  }
}
generated quantities {
  int next;
  if(rain[num_days]){
    next = bernoulli_rng(p11);
  } else {
    next = bernoulli_rng(p01);
  }
}
```

(d) $m^{(6)}$, a low scoring model.

Figure 7: Example models for the **rain** problem, sorted by estimated marginal likelihoods.

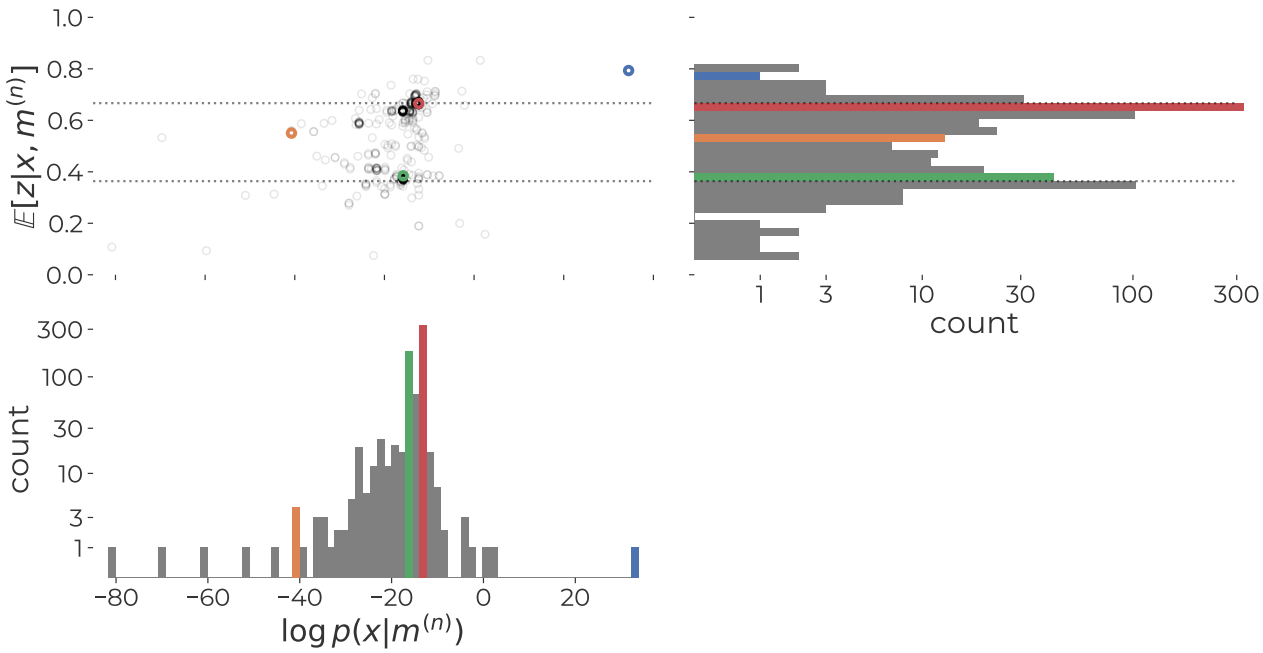

Figure 8: Top left: Estimated log marginal likelihoods $\log p(x|m^{(n)})$ and the estimated probability of rain on the next day $\mathbb{E}[z|x, m^{(n)}]$ for each of the valid models n for the **rain** problem. Markers are colored for four example models from Fig. 7. Top right: Histogram for estimated probability of rain. Bottom left: Histogram for estimated marginal likelihoods. The dotted lines at $\frac{8}{22} \approx 0.364$ and $\frac{14}{21} \approx .667$ show the base rates for rain and between-day consistency in rain, respectively.

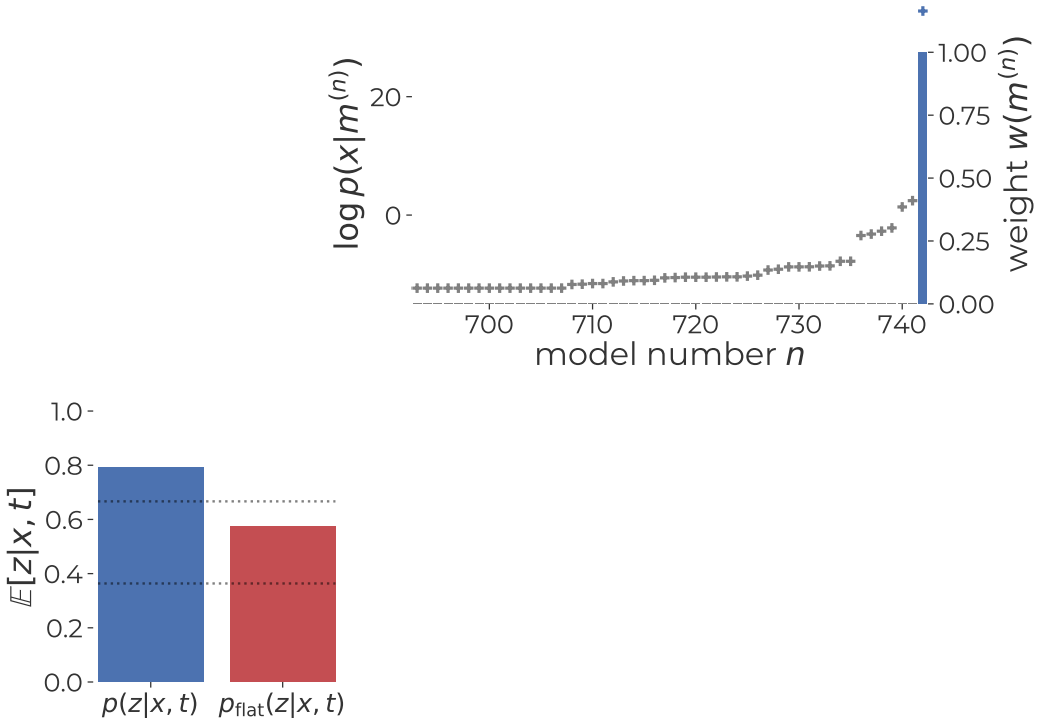

Figure 9: Left: Estimated log-marginal likelihoods $\log p(x|m^{(n)})$ ("+" symbols, left axis) and weights $w(m^{(n)})$ (bars, right axis) for the top 50 models n for the **rain** problem. Right: The final estimated posterior, compared to a "flat" average. The dotted lines at $\frac{8}{22} \approx 0.364$ and $\frac{14}{21} \approx .667$ show the base rates for rain and between-day consistency in rain, respectively.

B.2 Coin (standard)

In this problem, the user has flipped a coin several times and would like to predict the true bias. In this first version, the user strongly implies that it is a "standard" coin with a bias near 0.5. The observed data are 20 flips, out of which 14 were heads.

```
PROBLEM

I have a coin. I've flipped it a bunch of times. I'm
wondering what the true bias of the coin is.

I just got the coin from the US mint, so I'm almost
completely sure that it's a standard US penny.

DATA

int num_flips; // number of times the coin was flipped
int num_heads; // number of times heads was observed in
the flips

GOAL

real bias; // true bias of the coin
```

```
{"num_flips": 20, "num_heads": 14}
```

Out of 1024 models generated, 989 (96.6%) compiled and allowed for inference. Fig. 10 shows four models. Fig. 11 shows the estimated marginal likelihoods for each of the valid models. Fig. 12 compares the estimated posteriors for each valid models. Finally, Fig. 13 shows the estimated marginal likelihoods and corresponding weights for the top-scoring 50 valid models, as well as the final estimated posterior $p(z|x,t)$ compared to a "flat" average $p_{\text{flat}}(z|x,t)$.

In this case, while there are two high-scoring models similar to $m^{(968)}$ (Fig. 10a), they are essentially "outvoted" in the posterior by lower-scoring but more numerous models similar to $m^{(938)}$ (Fig. 10b). Thus, the final posterior in Fig. 13 is similar to the posterior from $m^{(938)}$.

```
data {
  int num_flips;
  int num_heads;
}
parameters {
  real logit_bias;
}
transformed parameters {
  real bias = inv_logit(logit_bias);
}
model {
  logit_bias ~ normal(0, 0.1);
  num_heads ~ binomial(num_flips, bias);
}
generated quantities {
  // Nothing to generate here, just inference on the bias
       parameter
}
```

(a) $m^{(968)}$, the highest-scoring model.

```
data {
  int num_flips;
  int num_heads;
}
parameters {
  real<lower=0, upper=1> bias;
}
model {
  bias ~ normal(0.5, 0.01);
  num_heads ~ binomial(num_flips, bias);
}
```

(b) $m^{(938)}$, a model from the most common "cluster".

```
data {
  int num_flips;
  int num_heads;
}
parameters {
  real<lower=0, upper=1> bias;
}
model {
  bias ~ beta(600, 400);
  num_heads ~ binomial(num_flips, bias);
}
```

(c) $m^{(385)}$, a low-scoring model

```
data {
  int num_flips;
  int num_heads;
}
parameters {
  real<lower=0, upper=1> bias;
}
model {
  bias ~ beta(10000, 10000);
  num_heads ~ binomial(num_flips, bias);
}
```

(d) $m^{(29)}$, a very low-scoring model

Figure 10: Example models for the **coin (standard)** problem, sorted by estimated marginal likelihoods.

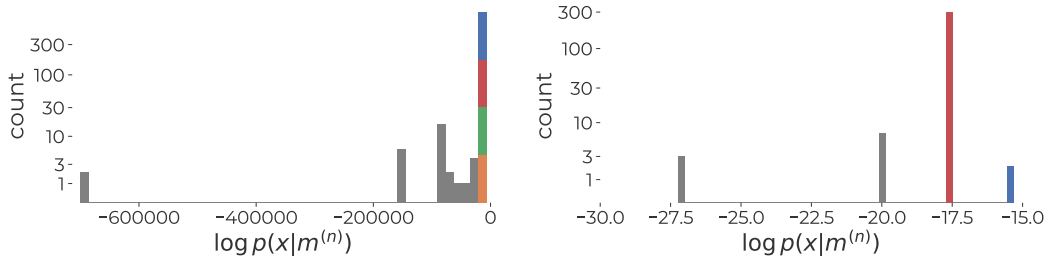

Figure 11: Estimated marginal likelihoods $p(x|m^{(n)})$ for each of the valid models n for the **coin (standard)** problem. Because of the many order of magnitude, two different ranges are shown. Bars are colored for four example models from Fig. 10. If multiple models map to the same bin, all colors are shown stacked.

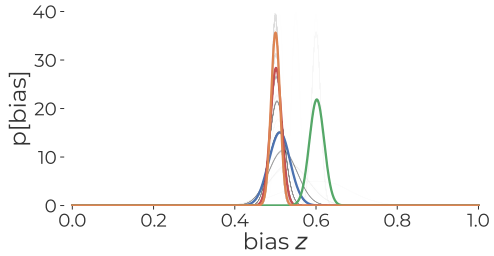

Figure 12: The estimated posteriors for each of the valid models n for the **coin (standard)** problem. Posteriors are colored for the four example models from Fig. 10.

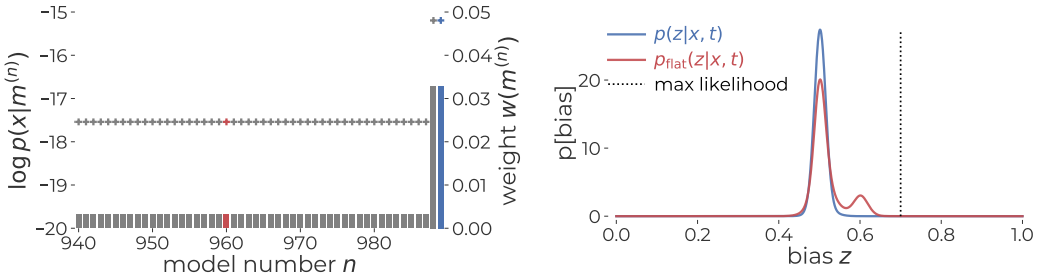

Figure 13: Left: Estimated log-marginal likelihoods $\log p(x|m^{(n)})$ ("+" symbols, left axis) and weights $w(m^{(n)})$ (bars, right axis) for the top 50 models n for the **coin (standard)** problem. There are two models similar to $m^{(968)}$, which each receive a weight of approximately 0.036, while there are several hundred models similar to $m^{(938)}$, which each receive a weight of approximately 0.0032. Right: The final estimated posterior, compared to a "flat" average. The maximum-likelihood estimator of $14/20 = 0.7$ is also shown for reference.

B.3 Coin (looks)

In this second version of the coin problem, the user states that the coin "looks" like a standard coin, but implies less confidence. As before, the observed data are 20 flips, out of which 14 were heads.

```
PROBLEM

I have a coin. I've flipped it a bunch of times. I'm
wondering what the true bias of the coin is.

At first glance, it appears to be a standard US penny,
although I haven't examined it closely.

DATA

int num_flips; // number of times the coin was flipped
int num_heads; // number of times heads was observed in
the flips

GOAL

real bias; // true bias of the coin
```

```
{"num_flips": 20, "num_heads": 14}
```

Out of 1024 models generated, 983 (96.0%) compiled and allowed for inference. Fig. 14 shows four models, while the results of inference are shown in Fig. 15, Fig. 16, and Fig. 17.

In this case, there are seven models similar to the top-scoring model $m^{(960)}$ (Fig. 14a) that each have weight of around 0.1, and seven models similar to $m^{(949)}$ (Fig. 14b) with a weight of around 0.03. All of these have priors with a reasonable amount of uncertainty. There are many models similar to $m^{(661)}$ that have prior with low variance, but these have a low marginal-likelihood and so get very low weight. The final estimated posterior in Fig. 17 thus resembles a mixture of the posteriors of $m^{(960)}$ and $m^{(949)}$.

```
data {
  int num_flips;
  int num_heads;
}
parameters {
  real<lower=0, upper=1> bias;
}
model {
  bias ~ uniform(0, 1);
  num_heads ~ binomial(num_flips, bias);
}
```

(a) $m^{(960)}$, the highest-scoring model.

```
data {
  int num_flips;
  int num_heads;
}
parameters {
  real<lower=0, upper=1> bias;
}
model {
  bias ~ normal(0.5, 0.1);
  num_heads ~ binomial(num_flips, bias);
}
```

(b) $m^{(949)}$, a slightly lower-scoring model.

```
data {
  int num_flips;
  int num_heads;
}
parameters {
  real<lower=0, upper=1> bias;
}
model {
  bias ~ beta(60,60);
  num_heads ~ binomial(num_flips, bias);
}
```

(c) $m^{(661)}$, from the most common "cluster" of models.

```
data {
  int num_flips;
  int num_heads;
}
parameters {
  real<lower=0, upper=1> bias;
}
model {
  bias ~ beta(600,600);
  num_heads ~ binomial(num_flips, bias);
}
```

(d) $m^{(1)}$, the lowest-scoring model

Figure 14: Example models for the **coin (looks)** problem, sorted by estimated marginal likelihoods.

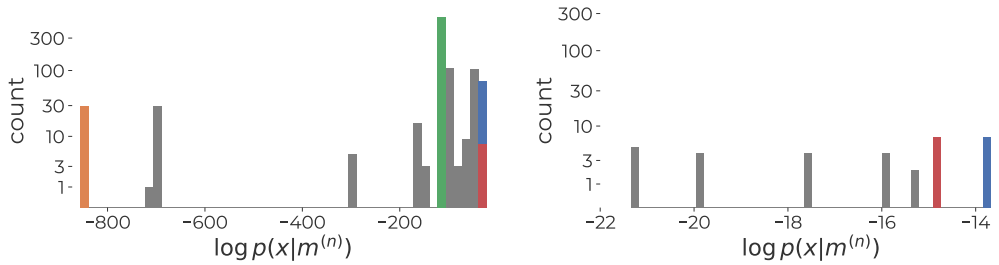

Figure 15: Estimated marginal likelihoods $p(x|m^{(n)})$ for each of the valid models n for the **coin (looks)** problem. Because of the many order of magnitude, two different ranges are shown. Bars are colored for four example models from Fig. 14. If multiple models map to the same bin, all colors are shown stacked.

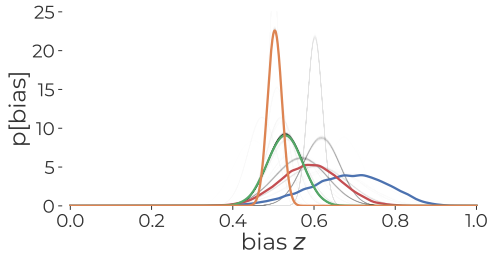

Figure 16: The estimated posteriors for each of the valid models n for the **coin (looks)** problem. Posteriors are colored for the four example models from Fig. 14.

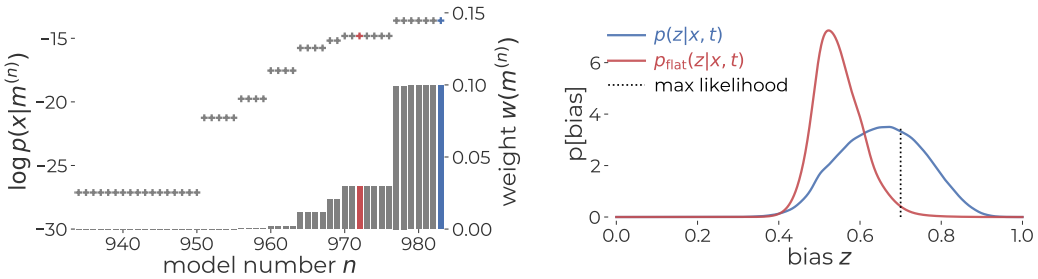

Figure 17: Left: Estimated log-marginal likelihoods $\log p(x|m^{(n)})$ ("+" symbols, left axis) and weights $w(m^{(n)})$ (bars, right axis) for the top 50 models n for the **coin (looks)** problem. Right: The final estimated posterior, compared to a "flat" average. The maximum-likelihood estimator of $14/20 = 0.7$ is also shown for reference.

B.4 Coin (bent)

In this third version of the coin problem, the user states that the coin looks "bent", implying great uncertainty about the true bias. As before, the observed data are 20 flips, out of which 14 were heads.

```
PROBLEM

I have a coin. I've flipped it a bunch of times. I'm
wondering what the true bias of the coin is.

I can see that the penny is quite bent, so I'm pretty sure
 the bias is different from 0.5, though I can't tell in
what direction.

DATA

int num_flips; // number of times the coin was flipped
int num_heads; // number of times heads was observed in
the flips

GOAL

real bias; // true bias of the coin
```

```
{"num_flips": 20, "num_heads": 14}
```

Out of 1024 models generated, 848 (82.8%) compiled and allowed for inference. Fig. 18 shows four models, while the results of inference are shown in Fig. 19, Fig. 20, and Fig. 21.

In this case, many models contribute substantially to the posterior. While $m^{(811)}$ has a slightly lower estimated marginal likelihood than the highest scoring model ($m^{(830)}$), models similar to it were generated many times, and so it wields the most influence in the posterior.

```
data {
  int num_flips;
  int num_heads;
}
parameters {
  real bias_logit;
}
transformed parameters {
  real<lower=0, upper=1> bias = inv_logit(bias_logit);
}
model {
  bias_logit ~ uniform(-10, 10); // diffuse prior that
       distributes mass fairly evenly over [0,1] after
       logistic transform
  num_heads ~ binomial(num_flips, bias);
}
generated quantities {
  // nothing
}
```

(a) $m^{(830)}$, the highest-scoring model

```
data {
  int num_flips;
  int num_heads;
}
parameters {
  real<lower=0, upper=1> bias;
}
model {
  bias ~ beta(0.5, 0.5); // a broad prior, but with more mass
        near 0 and 1
  num_heads ~ binomial(num_flips, bias);
}
```

(b) $m^{(811)}$, a typical model

```
data {
  int num_flips;
  int num_heads;
}
parameters {
  real<lower=0, upper=1> bias;
}
model {
  bias ~ beta(2,2); // symmetric around 0.5, low density
       at 0.5
  num_heads ~ binomial(num_flips, bias);
}
```

(c) $m^{(531)}$, from the most common "cluster" of models

```
data {
  int num_flips;
  int num_heads;
}
parameters {
  real<lower=0, upper=1> bias;
}
model {
  bias ~ beta(20,30);
  num_heads ~ binomial(num_flips, bias);
}
```

(d) $m^{(1)}$, the lowest-scoring model

Figure 18: Example models for the **coin (bent)** problem, sorted by estimated marginal likelihoods.

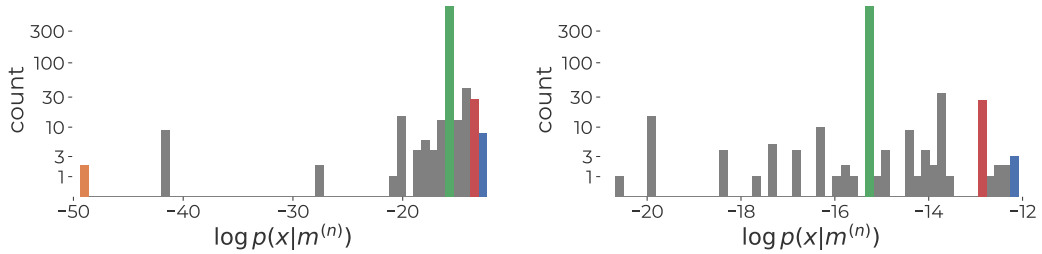

Figure 19: Estimated marginal likelihoods $p(x|m^{(n)})$ for each of the valid models n for the **coin (bent)** problem. Because of the many order of magnitude, two different ranges are shown. Bars are colored for four example models from Fig. 18. If multiple models map to the same bin, all colors are shown stacked.

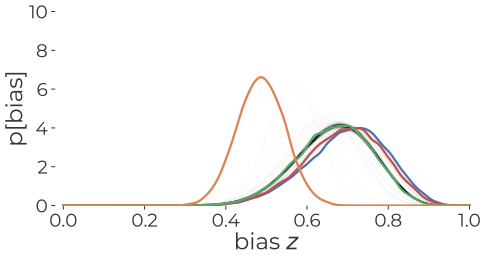

Figure 20: The estimated posteriors for each of the valid models n for the **coin (looks)** problem. Posteriors are colored for the four example models from Fig. 18.

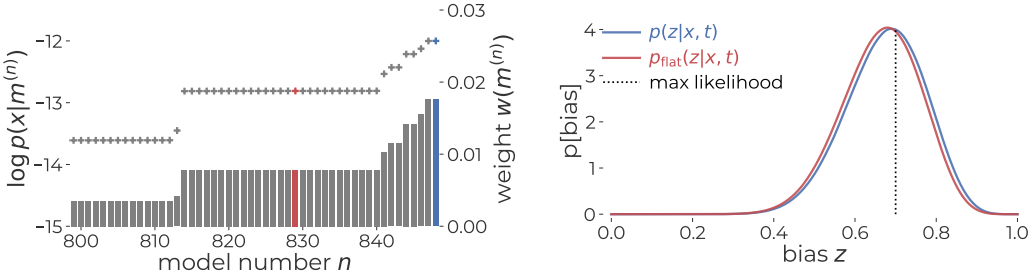

Figure 21: Left: Estimated log-marginal likelihoods $\log p(x|m^{(n)})$ ("+" symbols, left axis) and weights $w(m^{(n)})$ (bars, right axis) for the top 50 models n for the **coin (bent)** problem. Right: The final estimated posteriors, compared to a "flat" average. The maximum-likelihood estimator of $14/20 = 0.7$ is also shown for reference.

B.5 Polling

In this problem, the user describes a politician with fluctuating levels of popularity and different pollsters who take polls at different times. The goal is to predict the true popularity on each day.

```
PROBLEM

There is a politician. They have some true approval rating (between 0 and 100) that changes over time. There are some
pollsters that measure the popularity of the politician. Each pollster does polls on some different subset of days. Each
pollster does polls with some differing level of noise.

I have no idea what their popularity might be at the beginning of the year. Make sure to model the variability in the true
popularity, and make sure to model a distribution over the noise levels of different pollsters.
DATA

int num_days; // number of days over which popularity is modeled
int num_pollsters; // number of pollsters
int max_polls; // maximum number of polls performed by any pollster
array[num_pollsters] int num_polls; // number of polls done by each pollster
array[num_pollsters, max_polls] int day; // on what day was each poll done? (day[i,j] is only meaningful when j <=
num_polls[i])
array[num_pollsters, max_polls] real polls; // actual polling data (polls[i,j] is only meaningful when j <= num_polls[i])

GOAL

array[num_days] popularity; // true popularity of the politician on each day
```

A sample of the observed data is shown below, while the full data is plotted in Fig. 22.


```
{
    "num_days": 365,
    "num_pollsters": 3,
    "max_polls": 100,
    "num_polls": [5, 20, 100],
    "day": [[291, 16, ..., 1],
      [365, 151, ..., 1],
      [314, 66, ..., 57]],
    "polls": [[51.385352595786244, 49.741758128722864, ..., 0.0],
        [58.24149596144896, 40.825747460445996, ..., 0.0],
        [46.236574610204, 34.54862941696214, ..., 32.98157260195043]]
}
```


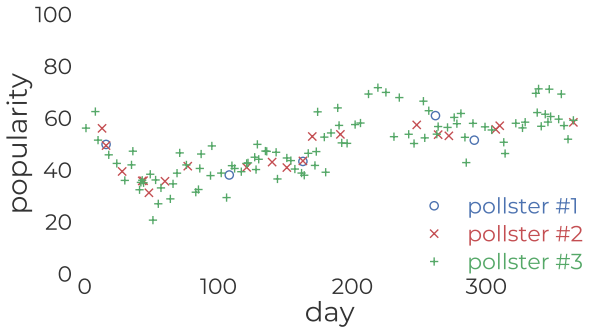

Figure 22: Observed data for the **polling** problem.

Out of 1024 models generated, 568 (55.5%) compiled and allowed for inference. Fig. 23 shows three models. Fig. 24 shows the estimated marginal likelihoods for each of the valid models. Fig. 25 compares the estimated posteriors for each valid models. Finally, Fig. 26 shows the estimated marginal likelihoods and corresponding weights for the top-scoring 50 valid models, as well as the final estimated posterior $p(z|x,t)$ compared to a "flat" average $p_{\text{flat}}(z|x,t)$.

In this case, the highest-scoring model $m^{(568)}$ (Fig. 23a) is very "simple" yet has a high enough marginal likelihood to take almost all weight in the final posterior in Fig. 26. The LLM did attempt to create many more sophisticated models, e.g. hierarchical models. But these often failed to compile or, as with $m^{(89)}$, simply had a lower marginal-likelihood.

```
data {
  int num_days;
  int num_pollsters;
  int max_polls;
  array[num_pollsters] int num_polls;
  array[num_pollsters, max_polls] int day;
  array[num_pollsters, max_polls] real polls;
}
transformed data {
  vector[num_days] zeros = rep_vector(0, num_days);
}
parameters {
  vector[num_days] popularity;
  array[num_pollsters] real<lower=0> sigma;
}
model {
  //_polls are modeled by a distribution with some
       underlying mean and noise sigma
  //which is pollster-dependent
  popularity[1] ~ uniform(0, 100);
  for (i in 2:num_days){
    popularity[i] ~ normal(popularity[i-1], 10);
  }
  for (i in 1:num_pollsters){
    sigma[i] ~ inv_gamma(5, 20);
    for (j in 1:num_polls[i]){
      polls[i,j] ~ normal(popularity[day[i,j]], sigma[i]);
    }
  }
}
```

(a) $m^{(568)}$, the highest-scoring model.

```
data { ... }
parameters {
    array[num_days] real popularity;
    array[num_pollsters] real<lower=0> pollster_noise;
}
model {
    // popularity changes over time according to a simple
         random walk
    popularity[1] ~ normal(50,20); // we have no idea what
         the initial popularity is
    for (i in 2:num_days){
        popularity[i] ~ normal(popularity[i-1], 5); // not
             too much change from one day to the next
    }
    // observed polls are noisy versions of the true
         popularity
    for (i in 1:num_pollsters){
        for (j in 1:num_polls[i]){
            polls[i,j] ~ normal(popularity[day[i,j]],
                 pollster_noise[i]);
        }
    }
    // prior over the noise levels of the pollsters
    for (i in 1:num_pollsters){
        pollster_noise[i] ~ cauchy(0, 5);
    }
}
generated quantities {
    array[num_days] real popularity_sample = popularity;
}
```

(b) $m^{(553)}$, a lower-scoring model.

```
data { ... }
parameters {
  real initial_popularity;
  real<lower=0> step_size;
  array[num_days-1] real steps;
  vector<lower=0>[num_pollsters] pollster_noise;
  real<lower=0> pollster_noise_mean;
  real<lower=0> pollster_noise_sigma;
}
transformed parameters{
  array[num_days] real popularity;
  popularity[1] = initial_popularity;
  for (i in 2:num_days){
    popularity[i] = popularity[i-1] + steps[i-1];
  }
}
model {
  initial_popularity ~ normal(50,20); // very vague prior over initial popularity
  step_size ~ normal(0,5);
  for (i in 1:(num_days-1)){
    steps[i] ~ normal(0,step_size); // random walk for the popularity
  }
  pollster_noise_mean ~ normal(5,10); // very vague prior over the mean of the noise
  pollster_noise_sigma ~ cauchy(0,2); // very vague prior over the sigma of the noise
  pollster_noise ~ normal(pollster_noise_mean, pollster_noise_sigma); // hierarchical prior for the noise of each pollster
  for (i in 1:num_pollsters){
    for (j in 1:num_polls[i]){
      polls[i,j] ~ normal(popularity[day[i,j]], pollster_noise[i]);
    }
  }
}
```

(c) $m^{(89)}$, a lower-scoring (hierarchical) model.

Figure 23: Example models for the **polling** problem, sorted by estimated marginal likelihoods. The `data` block was always identical, so is truncated for space for some models.

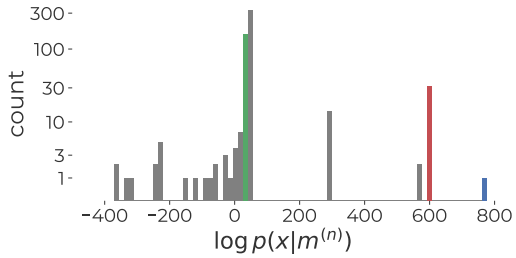

Figure 24: Estimated marginal likelihoods $p(x|m^{(n)})$ for each of the valid models n for the **polling** problem. Bars are colored for four example models from Fig. 23.

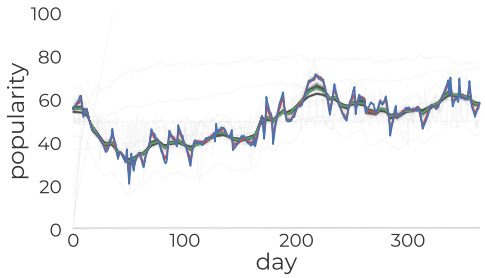

Figure 25: The estimated posteriors for each of the valid models n for the **polling** problem. Posteriors are colored for the four example models from Fig. 23.

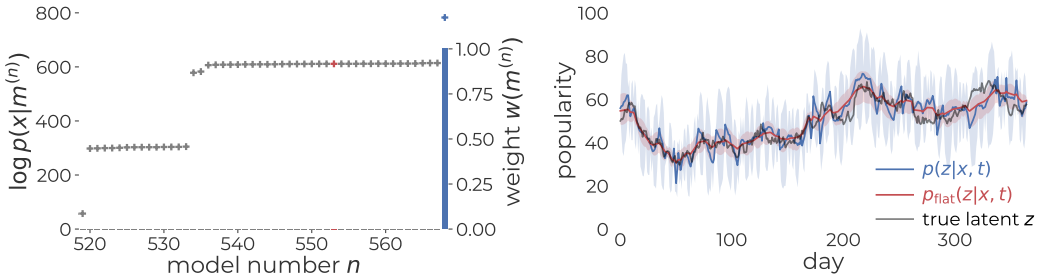

Figure 26: Left: Estimated log-marginal likelihoods $\log p(x|m^{(n)})$ ("+" symbols, left axis) and weights $w(m^{(n)})$ (bars, right axis) for the top 50 models n for the **polling** problem. Right: The final estimated posteriors, compared to a "flat" average.

B.6 City temperature

In this problem the user describes observing temperatures on neighboring pairs of days for a set of different cities. Then, after observing temperatures on a set of "test" days for each city, they wish to predict the temperature on the following days.

```
PROBLEM

There is training data for a bunch of different cities on random days, along with the temperature for the following data.
Then, there is test data for a bunch of different days in those same cities. Predict the temperature on the following days.

Assume that different cities tend to have different weather. And assume that the weather tends to be similar in neighboring
 days. So when guessing the weather in the next day, both the typical temperature in the city and the weather on the
previous day will matter.

All temperatures are in Fahrenheit.

Be sure to use informative priors for all quantities.

DATA

int num_cities; // number of cities included
int num_days_train; // number of training days in each city
int num_days_test; // number of training days in each city
array[num_cities, num_days_train] real day1_temp_train; // temperature on the first day, training set
array[num_cities, num_days_train] real day2_temp_train; // temperature on the following day, training set
array[num_cities, num_days_test] real day1_temp_test; // temperature on the first day, test set

GOAL

array[num_cities, num_days_test] real day2_temp_test; // temperature on the following day, test set
```

In the given data, there are 5 cities, with 10 training day pairs, and 10 test days.


```
{
    "num_cities": 5,
    "num_days_train": 10,
    "num_days_test": 10,
    "day1_temp_train":
    [[61.0, 74.9, 61.5, 73.2, 78.1, 69.8, 55.2, 74.2, 53.9, 75.0],
     [73.0, 78.5, 80.2, 78.9, 82.0, 84.2, 79.0, 81.3, 77.3, 76.0],
     [79.3, 82.1, 82.2, 76.0, 81.2, 84.9, 79.0, 79.0, 81.1, 85.7],
     [81.0, 76.5, 82.0, 78.8, 77.6, 77.1, 76.9, 82.2, 83.7, 78.5],
     [79.8, 78.8, 77.8, 79.6, 76.8, 75.4, 80.4, 75.6, 81.0, 76.2]],
    "day1_temp_test":
    [[69.3, 77.5, 74.8, 65.3, 58.5, 87.0, 54.9, 48.5, 59.4, 72.9],
     [76.5, 48.2, 84.1, 66.6, 85.1, 81.9, 81.0, 77.4, 71.9, 77.7],
     [80.6, 78.9, 79.4, 84.3, 80.5, 81.7, 80.6, 82.7, 82.2, 79.7],
     [82.7, 74.7, 82.1, 77.9, 82.5, 76.2, 81.8, 76.5, 79.1, 82.1],
     [78.4, 75.1, 81.0, 75.0, 74.5, 74.2, 80.5, 76.5, 79.0, 77.8]],
    "day2_temp_train":
    [[61.5, 69.9, 62.0, 73.4, 78.8, 68.8, 53.1, 74.5, 49.4, 76.2],
     [76.4, 78.0, 82.4, 78.9, 80.2, 82.8, 79.4, 79.1, 78.3, 80.0],
     [79.5, 83.8, 77.9, 76.8, 78.5, 85.2, 79.8, 81.6, 80.0, 85.5],
     [80.5, 77.1, 86.2, 78.2, 75.5, 79.1, 78.9, 81.8, 84.2, 82.7],
     [80.8, 78.5, 75.7, 76.1, 74.0, 80.3, 81.6, 77.8, 79.5, 72.8]]
}
```


Out of 1024 models generated, 736 (71.9%) compiled and allowed for inference. Four example models are shown in (Fig. 27) and (Fig. 28). A histogram of estimated marginal likelihoods is in (Fig. 29), while Fig. 30 shows the estimated marginal likelihoods and corresponding weights for the top-scoring 50 valid models. Finally, Fig. 31compares the final estimated posterior $p(z|x,t)$ to a "flat" average $p_{\text{flat}}(z|x,t)$.

In this case, the top scoring model $m^{(736)}$ Fig. 27a models per-city mean temperatures and shared regression coefficients to predict test temperatures. Many generated models used more shared parameters. For example, $m^{(688)}$ (Fig. 28a) has global parameters for the distribution of temperatures and $m^{(487)}$ (Fig. 28b) also has a shared autocorrelation parameter. However, these more complex models had lower marginal likelihoods. Thus the final posterior is essentially the same as that of $m^{(736)}$.

```
data {
  int num_cities;
  int num_days_train;
  int num_days_test;
  array[num_cities, num_days_train] real day1_temp_train;
  array[num_cities, num_days_train] real day2_temp_train;
  array[num_cities, num_days_test] real day1_temp_test;
}
parameters {
  array[num_cities] real city_mean;
  array[num_cities, num_days_train] real day1_temp_adj;
  real beta;
  real alpha;
}
model {
  beta ~ normal(0.8, 0.2);
  alpha ~ normal(0, 5);
  for (i in 1:num_cities){
    city_mean[i] ~ normal(50, 20);
    for (j in 1:num_days_train){
      day1_temp_adj[i, j] ~ normal(0, 10);
      day2_temp_train[i, j] ~ normal(city_mean[i] + alpha + beta * (day1_temp_train[i, j] - city_mean[i] + day1_temp_adj[i
           , j]), 5);
    }
  }
}
generated quantities {
  array[num_cities, num_days_test] real day2_temp_test;
  for (i in 1:num_cities){
    for (j in 1:num_days_test){
      day2_temp_test[i, j] = normal_rng(city_mean[i] + alpha + beta * (day1_temp_test[i, j] - city_mean[i]), 5);
    }
  }
}
```

(a) $m^{(736)}$, the highest-scoring model.

```
data {
  int num_cities;
  int num_days_train;
  int num_days_test;
  array[num_cities, num_days_train] real day1_temp_train;
  array[num_cities, num_days_train] real day2_temp_train;
  array[num_cities, num_days_test] real day1_temp_test;
}
parameters {
  array[num_cities] real city_mean;
  array[num_cities, num_days_train] real day_effect;
  real<lower=-1, upper=1> temp_change;
  real<lower=0> temp_sd;
}
model {
  for(i in 1:num_cities){
    city_mean[i] ~ normal(50,20);
    for(j in 1:num_days_train){
      day_effect[i,j] ~ normal(0,10);
    }
  }
  temp_change ~ uniform(-1, 1);
  temp_sd ~ cauchy(0, 5);
  for(i in 1:num_cities){
    for(j in 1:num_days_train){
      day2_temp_train[i,j] ~ normal(city_mean[i] + day_effect[i,j] + temp_change * (day1_temp_train[i,j] - city_mean[i] -
           day_effect[i,j]), temp_sd);
    }
  }
}
generated quantities {
  array[num_cities, num_days_test] real day2_temp_test;
  for(i in 1:num_cities){
    for(j in 1:num_days_test){
      day2_temp_test[i,j] = normal_rng(city_mean[i] + temp_change * (day1_temp_test[i,j] - city_mean[i]), temp_sd);
    }
  }
}
```

(b) $m^{(735)}$, the second-highest scoring model

Figure 27: Example models for the **city temperature** problem.

```
data {
  int num_cities;
  int num_days_train;
  int num_days_test;
  array[num_cities, num_days_train] real day1_temp_train;
  array[num_cities, num_days_train] real day2_temp_train;
  array[num_cities, num_days_test] real day1_temp_test;
}
parameters {
  real mu;
  real<lower=0> sigma;
  array[num_cities] real city_mu;
  array[num_cities] real<lower=-1, upper=1> city_ar;
}
model {
  mu ~ normal(50, 20);
  sigma ~ normal(0, 20);
  for (c in 1:num_cities){
    city_mu[c] ~ normal(mu, sigma);
    city_ar[c] ~ uniform(-1, 1);
    for (d in 1:num_days_train){
      day2_temp_train[c, d] ~ normal(city_mu[c] + city_ar[c] * (day1_temp_train[c, d] - city_mu[c]), 5);
    }
  }
}
generated quantities {
  array[num_cities, num_days_test] real day2_temp_test;
  for (c in 1:num_cities){
    for (d in 1:num_days_test){
      day2_temp_test[c, d] = normal_rng(city_mu[c] + city_ar[c] * (day1_temp_test[c, d] - city_mu[c]), 5);
    }
  }
}
```

(a) $m^{(688)}$, a lower-scoring model.

```
data {
  int num_cities;
  int num_days_train;
  int num_days_test;
  array[num_cities, num_days_train] real day1_temp_train;
  array[num_cities, num_days_train] real day2_temp_train;
  array[num_cities, num_days_test] real day1_temp_test;
}
parameters {
  vector[num_cities] city_mean;
  vector<lower=0, upper=1>[num_cities] city_autocorrelation;
  real global_mean;
  real<lower=0, upper=1> global_autocorrelation;
  real<lower=0> sigma;
}
model {
  global_mean ~ normal(60, 10);
  global_autocorrelation ~ uniform(0, 1);
  for (i in 1:num_cities){
    city_mean[i] ~ normal(global_mean, 10);
    city_autocorrelation[i] ~ uniform(0, 1);
    for (j in 1:num_days_train){
      day2_temp_train[i, j] ~ normal(day1_temp_train[i, j] * city_autocorrelation[i] + city_mean[i] * (1 -
          city_autocorrelation[i]), sigma);
    }
  }
  sigma ~ cauchy(0, 5);
}
generated quantities {
  array[num_cities, num_days_test] real day2_temp_test;
  for (i in 1:num_cities){
    for (j in 1:num_days_test){
      day2_temp_test[i, j] = normal_rng(day1_temp_test[i, j] * city_autocorrelation[i] + city_mean[i] * (1 -
          city_autocorrelation[i]), sigma);
    }
  }
}
```

(b) $m^{(487)}$, a typical but lower-scoring model.

Figure 28: Example models for the **city temperature** problem.

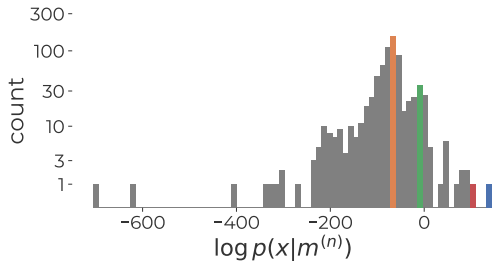

Figure 29: Estimated marginal likelihoods $p(x|m^{(n)})$ for each of the valid models n for the **city temperature** problem. Bars are colored for four example models from (Fig. 27) and (Fig. 28).

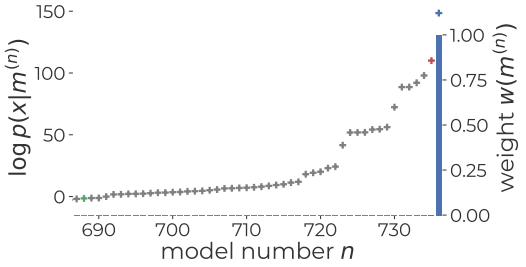

Figure 30: Estimated log-marginal likelihoods $p(x|m^{(n)})$ and weights $w(m^{(n)})$ for the top 50 models n for the **city temperature** problem. markers are colored for four example models from Fig. 27 and Fig. 28.

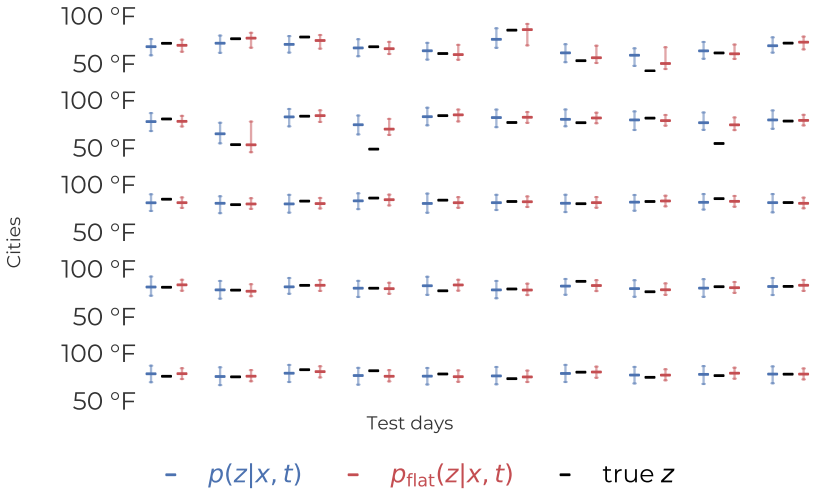

Figure 31: Medians and 90% credible intervals computed for each city on each test day for the **city temperature** problem. In this case, both LLB and p_{flat} have coverage of 44/50=88% of the true values, indicating reasonable calibration, though the posteriors under LLB are slightly more narrow.

B.7 Gold (small)

In this problem, the user describes a "rod" of some length, with a varying density of gold atoms. Samples were taken at various positions and tested to be gold or not. Finally, a set of future test locations are given. The user "forgot" to give types for some variables.

```
PROBLEM

I have a metal rod. I've gone to various positions along the rod and sampled single atoms and tested if they were gold. (
Yes or no.) I'd like to predict if gold would be detected if I tried measurements at various test locations. I assume the
density of gold changes smoothly, but the actual density could be quite complicated.

DATA

real rod_length; // length of rod in meters
int num_train; // number of times rod was sampled
array[num_train] train_locs; // where samples taken in training data
array[num_train] gold_train; // was gold detected at each location (0 if no, 1 if yes)
int num_test; // hypothetical test locations
array[num_test] test_locs; // possible future locations for sampling

GOAL

array[num_test] gold_test; // would gold be detected at future locations
```

Synthetic data was generated by taking training locations x uniformly from 0 to 2.5. The true density of gold was $\sigma\left(1+\sin(9x)-\frac{1}{2}x^2\right)$ where $\sigma(s)=1/(1+e^{-s})$ denotes a sigmoid function. The test locations were 100 points equally spaced from 0 to 2.5. A sample of the data is shown below, while the full data is plotted in Fig. 32.


```
{
    "rod_length": 2.5,
    "num_train": 30,
    "train_locs": [2.000547344419736, 0.9166378396087462, 0.16507930158778522, ..., 1.5834750504530812],
    "gold_train": [0, 1, 1, ..., 1],
    "num_test": 100,
    "test_locs": [0.0, 0.025252525252525252, 0.050505050505050504, ..., 2.5],
}
```


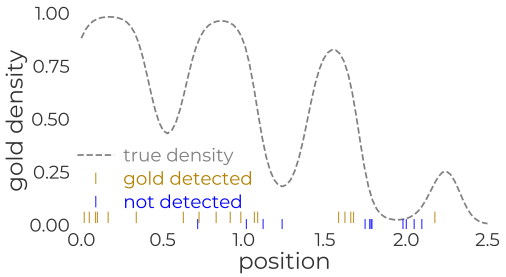

Figure 32: Observed data for the **gold (small)** problem.

In preliminary testing, it was difficult for the LLM to generate valid models for this problem. Thus, instead of 1000, $2^{14}=16384$ candidate models were generated. Of these, 192 (1.2%) compiled and allowed for inference. The top three models and a low scoring model are shown in Fig. 33, Fig. 34, Fig. 35, and Fig. 36. Histograms of estimated marginal likelihoods are in Fig. 37, while Fig. 38 shows the estimated posterior for each model, and Fig. 39 shows the weights for the top 50 models and the final estimated posterior.

In this case, essentially all posterior weight is given to the top-three scoring models, $m^{(192)}$ Fig. 33, $m^{(191)}$ Fig. 34, and $m^{(190)}$ Fig. 35. The final posterior appears a bit better than a flat average.

```
data {
  real rod_length;
  int num_train;
  array[num_train] real train_locs;
  array[num_train] int<lower=0, upper=1> gold_train;
  int num_test;
  array[num_test] real test_locs;
}
transformed data {
  int<lower=1> N = num_train + num_test;
  array[N] real all_locs = append_array(train_locs, test_locs);
}
parameters {
  vector[N] eta;
  real<lower=0> rho;
}
transformed parameters {
  vector[N] f;
  {
    matrix[N, N] K;
    for (i in 1:N) {
      K[i, i] = 1 + 0.01; // diagonal element
      for (j in 1:(i-1)) {
        real dist = sqrt(square(all_locs[i] - all_locs[j]));
        K[i, j] = sqrt(5)/sqrt(3*rho)*pow(1+sqrt(5*dist/rho), -1.5 - 1/2)*exp(-sqrt(5*dist/rho/3)*1.2);
        K[j, i] = K[i, j]; // fill in lower triangle
      }
    }
    f = K * eta;
  }
}
model {
  eta ~ normal(0, 1);
  rho ~ normal(0, rod_length); // use rod length to specify the input scale of the function

  for (i in 1:num_train) {
    gold_train[i] ~ bernoulli_logit(f[i]);
  }
}
generated quantities {
  array[num_test] int<lower=0, upper=1> gold_test;
  for (i in 1:num_test) {
    gold_test[i] = bernoulli_logit_rng(f[num_train + i]);
  }
}
```

Figure 33: $m^{(192)}$, the highest-scoring model for the **gold(small)** problem.

```
data {
  real rod_length;
  int num_train;
  array[num_train] real train_locs;
  array[num_train] int<lower=0, upper=1> gold_train;
  int num_test;
  array[num_test] real test_locs;
}
transformed data {
  // creating a grid of locations to compute covariance matrix
  int num_latent = num_train + num_test;
  array[num_latent] real all_locs;
  for(i in 1:num_train){
    all_locs[i] = train_locs[i];
  }
  for(i in 1:num_test){
    all_locs[i+num_train] = test_locs[i];
  }
}
parameters {
  vector[num_latent] z;
}
transformed parameters {
  vector[num_latent] f;
  {
    matrix[num_latent, num_latent] K;
    for (i in 1:num_latent) {
      for (j in 1:num_latent) {
        real dist = sqrt((all_locs[i] - all_locs[j])^2);
        K[i, j] = (1 + sqrt(3 * dist)) * exp(-sqrt(3 * dist));
      }
    }
    f = rep_vector(0.5, num_latent) + cholesky_decompose(K) * z;
  }
}
model {
  z ~ std_normal();
  for(i in 1:num_train){
    gold_train[i] ~ bernoulli(inv_logit(f[i]));
  }
}
generated quantities {
  array[num_test] int<lower=0, upper=1> gold_test;
  for(i in 1:num_test){
    real true_gold_density = inv_logit(f[i+num_train]);
    gold_test[i] = bernoulli_rng(true_gold_density);
  }
}
```

Figure 34: $m^{(191)}$, the second-highest-scoring model for the **gold(small)** problem.

```
data {
  real rod_length;
  int num_train;
  array[num_train] real train_locs;
  array[num_train] int<lower=0, upper=1> gold_train;
  int num_test;
  array[num_test] real test_locs;
}
transformed data {
  int num_locs = num_train + num_test;
  array[num_locs] real all_locs;
  for(i in 1:num_train){
    all_locs[i] = train_locs[i];
  }
  for(i in 1:num_test){
    all_locs[i+num_train] = test_locs[i];
  }
}
parameters {
  real<lower=0> length;
  vector[num_locs] z;
}
transformed parameters {
  vector[num_locs] p;
  {
    matrix[num_locs, num_locs] K;
    for (i in 1:num_locs){
      for (j in 1:num_locs){
        K[i,j] = exp(-((all_locs[i]-all_locs[j])/length)^2);
      }
    }
    p = K * z;
  }
}
model {
  z ~ normal(0,1);
  length ~ normal(0.1,0.1);
  for (i in 1:num_train){
    gold_train[i] ~ bernoulli(1/(1+exp(-p[i])));
  }
}
generated quantities {
  array[num_test] int<lower=0, upper=1> gold_test;
  vector[num_locs] p_smooth;
  p_smooth = inv_logit(p);
  for (i in 1:num_test){
    gold_test[i] = bernoulli_rng(p_smooth[i+num_train]);
  }
}
```

Figure 35: $m^{(190)}$, the third-highest-scoring model for the **gold(small)** problem.

```
data {
  real rod_length;
  int num_train;
  array[num_train] real train_locs;
  array[num_train] int<lower=0, upper=1> gold_train;
  int num_test;
  array[num_test] real test_locs;
}
parameters {
  real<lower=0> rho; // length scale for the kernel
  real<lower=0> alpha; // scale for the kernel
  vector[num_train] eta; // latent variables
}
transformed parameters {
  vector[num_train] mu;
  {
    matrix[num_train, num_train] K;
    for (i in 1:num_train) {
      K[i, i] = 1 + alpha;
      for (j in 1:num_train) {
        K[i, j] = alpha * exp(-((train_locs[i] - train_locs[j])^2) / (2 * rho^2));
      }
    }
    mu = K * eta;
  }
}
model {
  rho ~ cauchy(0, 5);
  alpha ~ cauchy(0, 5);
  eta ~ normal(0, 1);
  gold_train ~ bernoulli_logit(mu);
}
generated quantities {
  vector[num_test] test_mu;
  {
    matrix[num_test, num_train] K_test_train;
    for (i in 1:num_test) {
      for (j in 1:num_train) {
        K_test_train[i, j] = alpha * exp(-((test_locs[i] - train_locs[j])^2) / (2 * rho^2));
      }
    }
    test_mu = (K_test_train * eta);
  }
  array[num_test] int<lower=0, upper=1> gold_test;
  for (i in 1:num_test) {
    gold_test[i] = bernoulli_logit_rng(test_mu[i]);
  }
}
```

Figure 36: $m^{(171)}$, a low-scoring model for the **gold(small)** problem.

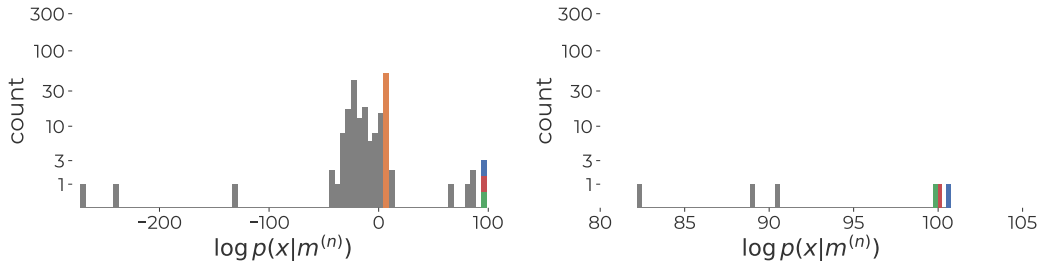

Figure 37: Estimated marginal likelihoods $p(x|m^{(n)})$ for each of the valid models n for the **gold (small)** problem. Because of the many order of magnitude, two different ranges are shown. Bars are colored for four example models from Fig. 33, Fig. 34, Fig. 35, and Fig. 36. If multiple models map to the same bin, all colors are shown stacked.

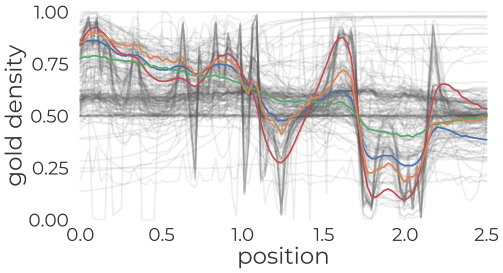

Figure 38: The estimated posteriors $\mathbb{E}[z|m^{(n)}, x]$ for each of the valid models n for the **gold (small)** problem. Lines are colored for four example models from Fig. 33, Fig. 34, Fig. 35, and Fig. 36.

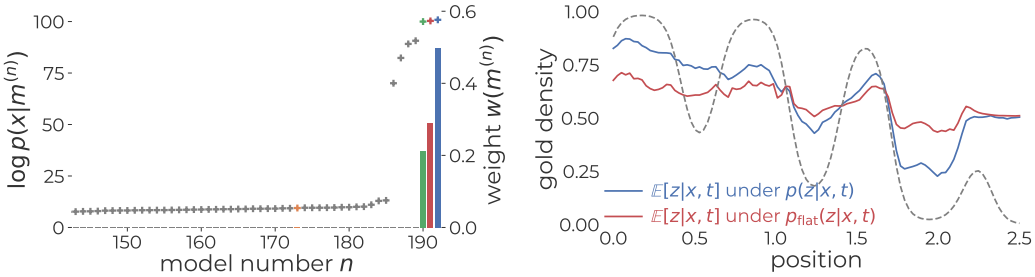

Figure 39: Left: The log-marginal likelihoods and weights for the top 50 models for the **gold (small)** problem. Right: The final estimated posteriors, compared to a "flat" average. The true density is shown for reference.

B.8 Gold (large)

This problem concerns the same gold problem, except with a larger dataset with more observations. Synthetic data was generated by taking 150 training locations x uniformly from 0 to 2.5. The true density of gold was $\sigma\left(1+\sin(9x)-\frac{1}{2}x^2\right)$ where $\sigma(s)=1/(1+e^{-s})$ denotes a sigmoid function. The test locations were 100 points equally spaced from 0 to 2.5. A sample of the data is shown below, while the full data is plotted in Fig. 40.


```
{
    "rod_length": 2.5,
    "num_train": 150,
    "train_locs": [0.6563454972396541, 0.4490753033133052, ..., 1.3381547009467185],
    "gold_train": [1, 0, ..., 0],
    "num_test": 100,
    "test_locs": [0.0, 0.025252525252525252, ..., 2.5],
}
```


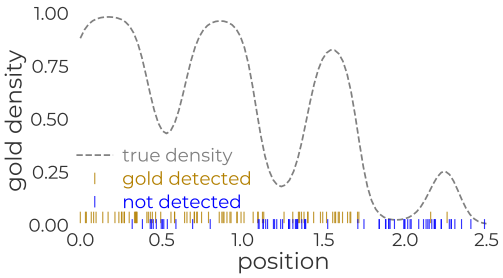

Figure 40: Observed data for the **gold (large)** problem.

The same $2^{14}=16384$ candidate models were considered as in the previous problem. Of these, only 71 (0.4%) compiled and allowed for inference with this larger dataset. The top three models and a low scoring model are shown in Fig. 41, Fig. 42, Fig. 43, and Fig. 44. Histograms of estimated marginal likelihoods are in Fig. 45, while Fig. 46 shows the estimated posterior for each model, and Fig. 47 shows the weights for the top 50 models and the final estimated posterior.

In this case, essentially all posterior weight is given to the top-three scoring models, $m^{(71)}$ Fig. 41. This produces a final posterior quite close to the true density, and much better than a flat average.

```
data {
  real rod_length;
  int num_train;
  array[num_train] real train_locs;
  array[num_train] int<lower=0, upper=1> gold_train;
  int num_test;
  array[num_test] real test_locs;
}
transformed data {
  int<lower=1> N = num_train + num_test;
  array[N] real all_locs = append_array(train_locs, test_locs);
}
parameters {
  vector[N] eta;
  real<lower=0> rho;
}
transformed parameters {
  vector[N] f;
  {
    matrix[N, N] K;
    for (i in 1:N) {
      K[i, i] = 1 + 0.01; // diagonal element
      for (j in 1:(i-1)) {
        real dist = sqrt(square(all_locs[i] - all_locs[j]));
        K[i, j] = sqrt(5)/sqrt(3*rho)*pow(1+sqrt(5*dist/rho), -1.5 - 1/2)*exp(-sqrt(5*dist/rho/3)*1.2);
        K[j, i] = K[i, j]; // fill in lower triangle
      }
    }
    f = K * eta;
  }
}
model {
  eta ~ normal(0, 1);
  rho ~ normal(0, rod_length); // use rod length to specify the input scale of the function

  for (i in 1:num_train) {
    gold_train[i] ~ bernoulli_logit(f[i]);
  }
}
generated quantities {
  array[num_test] int<lower=0, upper=1> gold_test;
  for (i in 1:num_test) {
    gold_test[i] = bernoulli_logit_rng(f[num_train + i]);
  }
}
```

Figure 41: $m^{(71)}$, the highest-scoring model for the **gold(large)** problem.

```
data {
  real rod_length;
  int num_train;
  array[num_train] real train_locs;
  array[num_train] int<lower=0, upper=1> gold_train;
  int num_test;
  array[num_test] real test_locs;
}
transformed data {
  int<lower=1> N = num_train + num_test;
  array[N] real all_locs = append_array(train_locs, test_locs);
}
parameters {
  real<lower=0> rho; // length scale for Matern covariance
  real<lower=0> alpha; // scale for Matern covariance
  vector[N] eta; // latent GP value at each location
}
model {
  rho ~ cauchy(0, 2); // prior for length scale
  alpha ~ cauchy(0, 2); // prior for scale
  eta ~ multi_normal(rep_vector(0, N),
    gp_matern32_cov(all_locs, alpha, rho)); // sample from GP prior
  for (i in 1:num_train){
    gold_train[i] ~ bernoulli_logit(eta[i]);
  }
}
generated quantities {
  array[num_test] int<lower=0, upper=1> gold_test;
  for (i in 1:num_test){
    gold_test[i] = bernoulli_logit_rng(eta[i + num_train]);
  }
}
```

Figure 42: $m^{(70)}$, the second-highest-scoring model for the **gold(large)** problem.

```
data {
  real rod_length;
  int num_train;
  array[num_train] real train_locs;
  array[num_train] int<lower=0, upper=1> gold_train;
  int num_test;
  array[num_test] real test_locs;
}
parameters {
  vector[num_train] latent;
  real<lower=0> sigma;
  real<lower=0> rho;
}
model {
  sigma ~ cauchy(0, 2);
  rho ~ cauchy(0, 2);
  {
    matrix[num_train, num_train] K;
    for (i in 1:num_train) {
      K[i, i] = 1 + sigma^2;
      for (j in 1:(i - 1)) {
        K[i, j] = sigma^2 * exp(-((train_locs[i] - train_locs[j])^2) / (2 * rho^2));
        K[j, i] = K[i, j];
      }
    }
    latent ~ multi_normal(rep_vector(0, num_train), K);
  }
  gold_train ~ bernoulli_logit(latent);
}
generated quantities {
  vector[num_test] test_latent;
  {
    matrix[num_test, num_train] K_test;
    matrix[num_train, num_train] K_train;
    for (i in 1:num_train) {
      K_train[i, i] = 1 + sigma^2;
      for (j in 1:(i - 1)) {
        K_train[i, j] = sigma^2 * exp(-((train_locs[i] - train_locs[j])^2) / (2 * rho^2));
        K_train[j, i] = K_train[i, j];
      }
    }
    for (i in 1:num_test) {
      for (j in 1:num_train) {
        K_test[i, j] = sigma^2 * exp(-((test_locs[i] - train_locs[j])^2) / (2 * rho^2));
      }
    }
    test_latent = (K_test * inverse(K_train)) * latent;
  }
  array[num_test] int<lower=0, upper=1> gold_test;
  for (i in 1:num_test) {
    gold_test[i] = bernoulli_logit_rng(test_latent[i]);
  }
}
```

Figure 43: $m^{(69)}$, the third-highest-scoring model for the **gold(large)** problem.

```
data {
  real rod_length;
  int num_train;
  array[num_train] real train_locs;
  array[num_train] int<lower=0, upper=1> gold_train;
  int num_test;
  array[num_test] real test_locs;
}
parameters {
  real<lower=0> gp_sigma;
  real<lower=0> gp_rho;
  vector[num_train] gp_z;
}
transformed parameters {
  vector[num_train] eta;
  eta = gp_z;
  for (i in 1:num_train) {
    real sum = 0;
    for (j in 1:num_train) {
      sum += gp_sigma * exp(-((train_locs[i] - train_locs[j]) / gp_rho)^2) * gp_z[j];
    }
    eta[i] = sum;
  }
}
model {
  gp_sigma ~ inv_gamma(1,1);
  gp_rho ~ inv_gamma(1,1);
  gp_z ~ normal(0,1);
  gold_train ~ bernoulli_logit(eta);
}
generated quantities {
  vector[num_test] eta_test;
  for (i in 1:num_test) {
    real sum = 0;
    for (j in 1:num_train) {
      sum += gp_sigma * exp(-((test_locs[i] - train_locs[j]) / gp_rho)^2) * gp_z[j];
    }
    eta_test[i] = sum;
  }
  array[num_test] int<lower=0, upper=1> gold_test;
  for (i in 1:num_test) {
    gold_test[i] = bernoulli_logit_rng(eta_test[i]);
  }
}
```

Figure 44: $m^{(51)}$, a low-scoring model for the **gold(large)** problem.

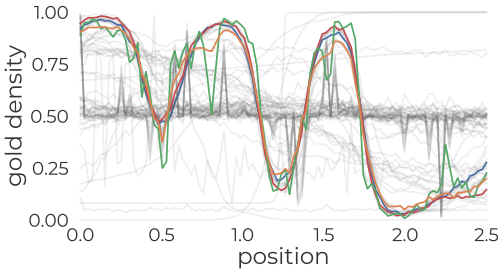

Figure 46: The estimated posteriors $\mathbb{E}[z|m^{(n)}, x]$ for each of the valid models n for the **gold (small)** problem. Lines are colored for four example models from Fig. 41, Fig. 42, Fig. 43, and Fig. 44.

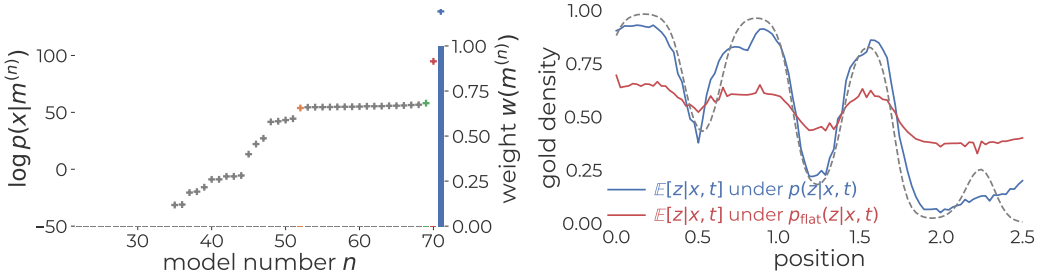

Figure 47: Left: The log-marginal likelihoods and weights for the top 50 models for the **gold (small)** problem. Right: The final estimated posteriors, compared to a "flat" average. The true density is shown for reference.

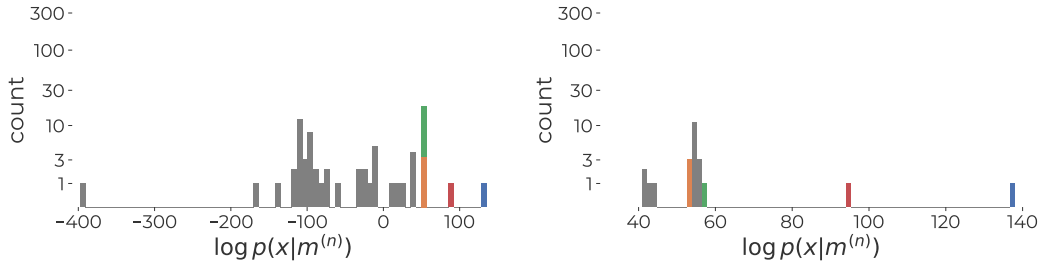

Figure 45: Estimated marginal likelihoods $p(x|m^{(n)})$ for each of the valid models n for the **gold (large)** problem. Because of the many order of magnitude, two different ranges are shown. Bars are colored for four example models from Fig. 41, Fig. 42, Fig. 43, and Fig. 44. If multiple models map to the same bin, all colors are shown stacked.

C Additional analysis

C.1 Form of the self-normalized importance sampling weights

Consider self-normalized importance sampling with a target distribution of $p(m|x,t)$ and a proposal distribution of $p(m|t)$. The naive importance weights would thus be

$$w^{(n)} = \frac{\frac{p(m^{(n)}|x,t)}{p(m^{(n)}|t)}}{\sum_{n'=1}^{N} \frac{p(m^{(n')}|x,t)}{p(m^{(n')}|t)}}.$$

However, we know from Eq. (5) that $p(m|x,t) \propto p(m|t)p(x|m)$ or, more precisely

$$p(m|x,t) = \frac{p(m|t)p(x|m)}{\sum_{m'} p(m'|t)p(x|m')} = \frac{p(m|t)p(x|m)}{p(x|t)}.$$

From this, we obtain that

$$\frac{p(m|x,t)}{p(m|t)} = \frac{p(x|m)}{p(x|t)}.$$

Substituting this into the above form for $w^{(n)}$, we obtain that

$$\begin{aligned} w^{(n)} &= \frac{\frac{p(x|m^{(n)})}{p(x|t)}}{\sum_{n'=1}^{N} \frac{p(x|m^{(n')})}{p(x|t)}} \\ &= \frac{p(x|m^{(n)})}{\sum_{n'=1}^{N} p(x|m^{(n')})}. \end{aligned}$$

C.2 Variance of the self-normalized importance sampling estimator

Consider the importance sampling estimator from Eq. (11), i.e.

$$\hat{\mu} = \sum_{n=1}^{N} \mathsf{w}^{(n)} g(\mathsf{m}^{(n)}).$$

Standard theory [e.g. 25, Eq. 9.8] says that the variance of $\hat{\mu}$ is asymptotically

$$\lim_{N\to\infty} N\mathbb{V}[\hat{\mu}] = \frac{\mathbb{E}_{p(\mathsf{m}|t)}\left[w(\mathsf{m})^2 \left(g(\mathsf{m}) - \mu\right)^2\right]}{\mathbb{E}_{p(\mathsf{m}|t)}\left[w(\mathsf{m})\right]^2},$$

where $w(m) = \frac{p(m|x,t)}{p(m|t)}$. This simplifies into

$$\begin{aligned} \lim_{N\to\infty} N\mathbb{V}[\hat{\mu}] &= \frac{\mathbb{E}_{p(\mathsf{m}|t)}\left[\frac{p(\mathsf{m}|x,t)^2}{p(\mathsf{m}|t)^2} \left(g(\mathsf{m}) - \mu\right)^2\right]}{\mathbb{E}_{p(\mathsf{m}|t)}\left[\frac{p(\mathsf{m}|x,t)}{p(\mathsf{m}|t)}\right]^2} \\ &= \mathop{\mathbb{E}}_{p(\mathsf{m}|t)}\left[\frac{p(\mathsf{m}|x,t)^2}{p(\mathsf{m}|t)^2} \left(g(\mathsf{m}) - \mu\right)^2\right]. \end{aligned}$$

Now, if we assume that $|g(m) - \mu| \leq \delta$ then this can be bounded by

$$\begin{aligned} \lim_{N\to\infty} N\mathbb{V}[\hat{\mu}] &\leq \delta^2 \mathop{\mathbb{E}}_{p(\mathsf{m}|t)}\left[\frac{p(\mathsf{m}|x,t)^2}{p(\mathsf{m}|t)^2}\right] \\ &= \delta^2 \left(1 + \chi^2\left(p(\mathsf{m}|x,t) \| p(\mathsf{m}|t)\right)\right) \end{aligned}$$

C.3 Form of the self-normalized importance sampling weights with variational inference

Consider trying to estimate

$$q(z|x) = \mathop{\mathbb{E}}_{q(\mathsf{m}|x)} q(z|x, \mathsf{m}),$$

where $q(z|x, m)$ is some distribution and $q(m|x)$ is as defined in Thm. 2. We are interested in estimating this expectation using SNIS with the proposal distribution $p(m|t)$. It follows that we should use an estimator

$$\tilde{q}(z|x) = \sum_{n=1}^{N} w^{(n)} q(z|x, m^{(n)}),$$

where $m^{(1)}, \cdots, m^{(N)} \sim p(m|t)$, and the self-normalized weights are

$$w^{(m)} \quad = \quad \frac{\frac{q(m^{(n)}|x)}{p(m^{(n)}|t)}}{\sum_{n'=1}^{N} \frac{q(m^{(n')}|x)}{p(m^{(n')}|t)}}.$$

Now, observe from Eq. (19) that

$$q(m|x) = \frac{p(m|t) \exp\Big(\mathrm{ELBO}(q(\mathsf{z}|x, m) \| p(\mathsf{z}, m|x, t))\Big)}{\sum_{m'} p(m'|t) \exp\Big(\mathrm{ELBO}(q(\mathsf{z}|x, m') \| p(\mathsf{z}, m'|x, t))\Big)},$$

where the denominator does not depend on m. It follows that the weights are

$$w^{(m)} \quad = \quad \frac{\exp\Big(\mathrm{ELBO}(q(\mathsf{z}|x, m^{(n)}) \| p(\mathsf{z}, m^{(n)}|x, t))\Big)}{\sum_{n'=1}^{N} \exp\Big(\mathrm{ELBO}(q(\mathsf{z}|x, m^{(n')}) \| p(\mathsf{z}, m^{(n')}|x, t))\Big)}.$$

C.4 Analysis with inexact ELBO estimates

Theorem 3. *Suppose that $p(z, x, m|t)$ and $q(z|x, m)$ are fixed and we choose*

$$q(m|x) \propto p(m|x) \times \exp\Big(\mathrm{ELBO}(q(\mathsf{z}|x, m) \| p(\mathsf{z}, x|m)) - \delta^{(\mathsf{m})}\Big)$$

where $\delta^{(m)} \geq 0$ represents the "slack" in between the true ELBO for model m and the bound used for computing $q(m|x)$. Then the resulting joint divergence is

$$\begin{aligned}
&\mathrm{KL}(q(\mathsf{z}, \mathsf{m}|x) \| p(\mathsf{z}, \mathsf{m}|x, t)) \\
&= -\log \mathop{\mathbb{E}}_{p(\mathsf{m}|x,t)} \exp\Big(-\mathrm{KL}^{(\mathsf{m})} - \delta^{(\mathsf{m})} + \bar{\delta}\Big) && (21) \\
&= \log p(x|t) - \log \mathop{\mathbb{E}}_{p(\mathsf{m}|t)} \exp\Big(\mathrm{ELBO}^{(\mathsf{m})} - \delta^{(\mathsf{m})} + \bar{\delta}\Big) && (22)
\end{aligned}$$

where $\bar{\delta} = \mathbb{E}_{q(\mathsf{m}|x)} \delta^{(\mathsf{m})}$, and $\mathrm{KL}^{(m)}$ and $\mathrm{ELBO}^{(m)}$ represent the KL-divergence and ELBO from Eq. (17).

(Proof in Appendix E.3.)

To interpret this result, first note that if all ELBOs are exact, meaning $\delta^{(m)} = 0$, then Eq. (21) is equivalent to Eq. (16) while Eq. (22) is equivalent to Eq. (20). This also shows that using "improved" distributions and tolerating "slack" in the ELBO can only help, compared to using distributions where the exact ELBO has the same value. That is, no matter the value of the slack variables $\delta^{(m)}$, Eq. (21) is never worse than Eq. (16) and Eq. (22) is never worse than Eq. (20).

The obvious implementation of Thm. 3 would be be an algorithm that loops over all models. This is given explicitly as Alg. 7 in Appendix D. Again, such an algorithm is intractable, but can be made practical by combining it with SNIS. This is given as Alg. 8.

An interesting special case is when MCMC is used to approximate each posterior $p(z|x, m)$ and we assume that these samples in fact come from $p(z|x, m)$. (See Alg. 3 in Appendix D.) If this case, the

best strategy is simply to use the best possible lower-bound on the marginal likelihood, computed by any method. It's see in this case that the joint divergence reduces to

$$-\log \sum_m p(m|x,t) \exp\Big(-\delta^{(m)} + \bar{\delta}\Big).$$

From this, we can see that if all errors $\delta^{(m)}$ are equal, then $\delta^{(m)} = \bar{\delta}$ and the joint KL-divergence remains zero. Also, if there are errors on models where $p(m|x,t)$ is negligible, these also have negligible impact on the joint divergence. However, if there are different levels of error on models where $p(m|x,t)$ is large, this could impact the joint divergence.

C.5 Relaxed bound

Corollary 4. *Under the assumptions of Thm. 1,*

$$\mathrm{KL}(q(z,m|x)\|p(z,m|x,t)) \leq \mathop{\mathbb{E}}_{p(m|x,t)} \mathrm{KL}(q(z|x,m)\|p(z|x,m))\Big).$$

Proof. We have that

$$\begin{aligned}
\mathop{\mathbb{E}}_{p(m|x,t)} \mathrm{KL}(q(z|x,m)\|p(z|x,m))\Big) &= -\log \exp\left(-\mathop{\mathbb{E}}_{p(m|x,t)} \mathrm{KL}(q(z|x,m)\|p(z|x,m))\Big)\right) \\
&\geq -\log \mathop{\mathbb{E}}_{p(m|x,t)} \exp\left(-\mathrm{KL}(q(z|x,m)\|p(z|x,m))\Big)\right) \\
&= \mathrm{KL}(q(z,m|x)\|p(z,m|x,t))\,.
\end{aligned}$$

The first line is obvious. The second line follows from using Jensen's inequality to see that

$$\mathop{\mathbb{E}}_{p(m|x,t)} \exp\Big(-\mathrm{KL}(q(z|x,m)\|p(z|x,m))\Big) \geq \exp\Big(-\mathop{\mathbb{E}}_{p(m|x,t)} \mathrm{KL}(q(z|x,m)\|p(z|x,m))\Big).$$

The third line is the form of $\mathrm{KL}(q(z,m|x)\|p(z,m|x,t))$ from Thm. 1. □

Algorithm 4 LLB with exact inference and SNIS.

1. Input textual description t and data x.
2. For $n = 1, 2, \cdots, N$:
   (a) Sample $m^{(n)} \sim p(m|t)$. // using LLM
   (b) Compute marginal evidence $p(x|m^{(n)})$ and posterior $p(z|x, m^{(n)})$. // under PPL
3. Set $w^{(n)} \propto p(x|m^{(n)})$, where $\sum_{n=1}^{N} w^{(n)} = 1$.
4. Use final posterior approximation

$$p(z|x,t) \approx \sum_{n=1}^{N} w^{(n)} p(z|x, m^{(n)}).$$

Algorithm 5 LLB with variational inference and SNIS.

1. Input textual description t and data x.
2. For $n = 1, 2, \cdots, N$
   (a) Sample $m^{(n)} \sim p(m|t)$. // using LLM
   (b) Maximize $\mathrm{ELBO}(m^{(n)})$ over some variational family $q(z|x, m^{(n)})$. // under PPL
3. Set $w^{(n)} \propto \exp \mathrm{ELBO}(m^{(n)})$, where $\sum_{n=1}^{N} w^{(n)} = 1$.
4. Use final posterior approximation

$$p(z|x,t) \approx \sum_{n=1}^{N} w^{(n)} q(z|x, m^{(n)}).$$

D Other example algorithms

Algorithm 6 Variational LLB with exact enumeration of models (theoretical)

1. Input textual description t and data x.
2. For all possible model strings m:
   (a) Compute model probability $p(m|t)$ (using LLM)
   (b) Maximize $\mathrm{ELBO}(q(\mathsf{z}|x, m^{(n)}) \| p(\mathsf{z}, x|m^{(n)}))$ over some variational family q.
3. Set $w^{(m)} \propto p(m|t) \exp\Big(\mathrm{ELBO}(q(\mathsf{z}|x, m^{(n)}) \| p(\mathsf{z}, x|m^{(n)}))\Big)$.
4. Use final posterior approximation

$$p(z|x,t) \approx \sum_{m} w^{(m)} q(z|x, m^{(n)}).$$

Algorithm 7 LLB with variational inference with inexact ELBOs (theoretical)

1. Input textual description t and data x.
2. For all possible model strings m:
   (a) Compute model probability $p(m|t)$ (using LLM)
   (b) Compute some approximation to the posterior $q(z|x,m) \approx p(z|x,m)$.
   (c) Compute some bound on the ELBO

$$L^{(m)} \leq \mathop{\mathbb{E}}_{q(\mathsf{z}|x,m)} \log \frac{p(\mathsf{z},x|m)}{q(\mathsf{z}|x,m)} \leq \log p(x|m).$$

3. Set $w^{(m)} \propto p(m|t) \exp\left(L^{(m)}\right)$.
4. Use final posterior approximation

$$p(z|x,t) \approx \sum_m w^{(m)} q(z|x,m).$$

Algorithm 8 LLB with ELBO bounds and SNIS.

1. Input textual description t and data x.
2. For $n = 1, 2, \cdots, N$:
   (a) Sample $m^{(n)} \sim p(m|t)$ (using an LLM).
   (b) Compute some approximation to the posterior $q(z|x,m^{(n)}) \approx p(z|x,m^{(n)})$.
   (c) Compute some bound on the ELBO

$$L^{(n)} \leq \mathop{\mathbb{E}}_{q(z|x,m^{(n)})} \log \frac{p(z,x|m^{(n)})}{q(z|x,m^{(n)})} \leq \log p(x|m^{(n)}).$$

3. Set $w^{(n)} \propto \exp L^{(n)}$.
4. Use final posterior approximation

$$p(z|x,t) \approx \sum_{n=1}^{N} w^{(n)} p(z|x,m^{(n)}).$$

E Proofs

E.1 Proof of Thm. 1

Proof. First, notice that minimizing

$$\textsc{kl}(q(z, m|x)\|p(z, m|x, t))$$

is equivalent to maximizing

$$\text{ELBO}(q(z, m|x)\|p(z, x, m|t)) = \mathop{\mathbb{E}}_{q(z|x,m)} \log \frac{p(z, x, m, |t)}{q(z|x, m)}.$$

We will therefore attempt to minimize the latter. Set up Lagrangian to find optimal $q(m|x)$.

$$\begin{aligned}
\mathcal{L} &= \sum_m \int q(m|t)q(z|m, x) \log \frac{p(z, x, m|t)}{q(m|t)q(z|m, x)} dz - \lambda \left(\sum_m q(m|t) - 1 \right) \\
\frac{d\mathcal{L}}{dq(m|t)} &= \int q(z|m) \log \frac{p(z, x, m|t)}{q(m|t)q(z|m, x)} dz - \int q(m|t)q(z|m, x) \frac{d}{dq(m|t)} \log q(m|t) dz - \lambda \\
&= \int q(z|m) \log \frac{p(z, x, m|t)}{q(m|t)q(z|m, x)} dz - 1 - \lambda \\
&= \int q(z|m) \log \frac{p(m|t)p(x|m)p(z|x, m)}{q(m|t)q(z|m, x)} dz - 1 - \lambda \\
&= \log \frac{p(m|t)p(x|m)}{q(m|t)} - \textsc{kl}(q(z|x, m)\|p(z|x, m)) - 1 - \lambda
\end{aligned}$$

Solving for $\frac{d\mathcal{L}}{dq(m)} = 0$ we obtain that

$$q(m|t) \propto p(m|t)p(x|m) \exp\left(-\textsc{kl}(q(z|x, m)\|p(z|x, m))\right).$$

Now, for simplicity, define $\textsc{kl}(q(z|x, m)\|p(z|x, m)) = \Delta_m$. Then, we can write

$$\begin{aligned}
q(m|x) &= \frac{p(m|t)p(x|m) \exp\Big(-\Delta_m\Big)}{\sum_{m'} p(m'|t)p(x|m') \exp\Big(-\Delta_{m'}\Big)}, \\
p(m|x, t) &= \frac{p(m|t)p(x|m)}{\sum_{m'} p(m'|t)p(x|m')}.
\end{aligned}$$

Now, note that

$$\begin{aligned}
\frac{q(m|x)}{p(m|x, t)} &= \frac{p(m|t)p(x|m) \exp\Big(-\Delta_m\Big)}{\sum_{m'} p(m'|t)p(x|m') \exp\Big(-\Delta_{m'}\Big)} \frac{\sum_{m'} p(m'|t)p(x|m')}{p(m|t)p(x|m)} \\
&= \frac{\exp\Big(-\Delta_m\Big) \sum_{m'} p(m'|t)p(x|m')}{\sum_{m'} p(m'|t)p(x|m') \exp\Big(-\Delta_{m'}\Big)} \\
&= \frac{\exp\Big(-\Delta_m\Big)}{\sum_{m'} p(m'|x, t) \exp\Big(-\Delta_{m'}\Big)}
\end{aligned}$$

Thus we have that

$$
\begin{aligned}
\mathrm{KL}(q(z,m|x)\|p(z,m|x,t)) &= \mathop{\mathbb{E}}_{q(m|x)} \log \frac{q(m|x)}{p(m|x,t)} + \mathop{\mathbb{E}}_{q(z,m|x)} \log \frac{q(z|m,x)}{p(z|m,x)} \\
&= \mathop{\mathbb{E}}_{q(m|x)} \log \frac{\exp\Big(-\Delta_m\Big)}{\sum_{m'} p(m'|x,t)\exp\Big(-\Delta_{m'}\Big)} + \mathop{\mathbb{E}}_{q(m|x)} \Delta_m \\
&= \log \frac{1}{\sum_{m'} p(m'|x,t)\exp\Big(-\Delta_{m'}\Big)} \\
&= -\log \mathop{\mathbb{E}}_{p(m|x,t)} \exp(-\Delta_m)
\end{aligned}
$$

□

Note that since $\Delta_m \geq 0$, it follows that $\exp(-\Delta_m) \leq 1$ and thus $\mathbb{E}_{p(m|x,t)} \exp(-\Delta_m) \leq 1$ and so $\log \mathbb{E}_{p(m|x,t)} \exp(-\Delta_m) \leq 0$. Thus $\mathrm{KL}(q(z,m|x)\|p(z,m|x,t))$ is non-negative, as required.

E.2 Proof of Thm. 2

Proof. Recall that the Thm. 1 found that the optimal $q(m|x)$ is

$$
q(m|x) \propto p(m|t)p(x|m) \times \exp\Big(-\mathrm{KL}(q(\mathsf{z}|x,m)\|p(\mathsf{z}|x,m))\Big). \tag{23}
$$

And recall the "ELBO decomposition", i.e. the fact that

$$
\log p(x|m) = \mathrm{ELBO}(q(\mathsf{z}|x,m)\|p(\mathsf{z},x|m)) + \mathrm{KL}(q(\mathsf{z}|x,m)\|p(\mathsf{z}|x,m)) \,. \tag{24}
$$

From which it follows that

$$
p(x|m) \times \exp\left(-\mathrm{KL}(q(\mathsf{z}|x,m)\|p(\mathsf{z}|x,m))\right) = \mathrm{ELBO}(q(\mathsf{z}|x,m)\|p(\mathsf{z},x|m))
$$

Substituting into Eq. (23) gives that

$$
q(m|x) \propto p(m|t)\exp\Big(\mathrm{ELBO}(q(\mathsf{z}|x,m)\|p(\mathsf{z},m|x,t))\Big).
$$

Recall also that Thm. 1 found the joint divergence resulting from using the optimal $q(m|x)$ is

$$
\begin{aligned}
\mathrm{KL}(q(\mathsf{z},\mathsf{m}|x)\|p(\mathsf{z},\mathsf{m}|x,t)) &= -\log \mathop{\mathbb{E}}_{p(\mathsf{m}|x,t)} \exp\Big(-\mathrm{KL}(q(\mathsf{z}|x,\mathsf{m})\|p(\mathsf{z}|x,\mathsf{m}))\Big) \\
&= -\log \mathop{\mathbb{E}}_{p(\mathsf{m}|x,t)} \exp\left(\mathrm{ELBO}(q(\mathsf{z}|x,m)\|p(\mathsf{z},x|m)) - \log p(x|m)\right) \\
&= -\log \sum_m \frac{p(m|x,t)}{p(x|m)} \exp\left(\mathrm{ELBO}(q(\mathsf{z}|x,m)\|p(\mathsf{z},x|m))\right) \\
&= -\log \sum_m \frac{p(m|t)}{p(x|t)} \exp\left(\mathrm{ELBO}(q(\mathsf{z}|x,m)\|p(\mathsf{z},x|m))\right) \\
&= \log p(x|t) - \log \mathop{\mathbb{E}}_{p(\mathsf{m}|t)} \exp\left(\mathrm{ELBO}(q(\mathsf{z}|x,\mathsf{m})\|p(\mathsf{z},x|\mathsf{m}))\right)
\end{aligned}
$$

Where we have used that

$$
p(m|x,t) = \frac{p(m|t)p(x|m)}{p(x|t)}
$$

and so

$$
\frac{p(m|x,t)}{p(x|m)} = \frac{p(m|t)}{p(x|t)}.
$$

□

To understand this, suppose each q was exact so that

$$\mathrm{ELBO}(q(\mathsf{z}|x,m)\|p(\mathsf{z},x|m)) = \log p(x|m).$$

Then

$$\begin{aligned}\mathrm{KL}(q(\mathsf{z},\mathsf{m}|x)\|p(\mathsf{z},\mathsf{m}|x,t)) &= \log p(x|t) - \log \mathop{\mathbb{E}}_{p(\mathsf{m}|t)} p(x|\mathsf{m}) \\ &= \log p(x|t) - \log p(x|t) \\ &= 0.\end{aligned}$$

E.3 Proof with inexact ELBOS (Thm. 3)

Proof. □

Theorem 5. *Suppose that $p(z,x,m|t)$ is fixed and $q(z|x,m) = p(z|x,m)$ is exact but $q(m|x) \propto p(m|t)p(x|t)\exp(-\delta^{(m)})$ is not exact (e.g. because it is chosen based on ELBOs rather than true marginal likelihoods). Then,*

$$\begin{aligned}\mathrm{KL}(q(z|x)\|p(z|x,t)) &\leq \mathrm{KL}(q(m|x)\|p(m|x,t)) \\ &= -\log \mathop{\mathbb{E}}_{p(m|x,t)} \exp(-\delta^{(m)}) - \mathop{\mathbb{E}}_{q(m|x)} \delta^{(m)}.\end{aligned}$$

Proof.

$$\begin{aligned}\mathrm{KL}(q(m|x)\|p(m|x,t)) &= \mathop{\mathbb{E}}_{q(m|x)} \log \frac{q(m|x)}{p(m|x,t)} \\ &= \mathop{\mathbb{E}}_{q(m|x)} \log \frac{p(m|t)p(x|m)\exp(-\delta^{(m)})}{p(m|t)p(x|m)} + \log \frac{\sum_m p(m|t)p(x|m)}{\sum_m p(m|t)p(x|m)\exp(-\delta^{(m)})} \\ &= \mathop{\mathbb{E}}_{q(m|x)} \left[-\delta^{(m)}\right] + \log \frac{\sum_m p(m|t)p(x|m)}{\sum_m p(m|t)p(x|m)\exp(-\delta^{(m)})} \\ &= \log \frac{\sum_m p(m|t)p(x|m)}{\sum_m p(m|t)p(x|m)\exp(\bar{\delta} - \delta^{(m)})} \\ &= \log \frac{p(x|t)}{\sum_m p(m|t)p(x|m)\exp(\bar{\delta} - \delta^{(m)})} \\ &= -\log \mathop{\mathbb{E}}_{p(m|x,t)} \exp(\bar{\delta} - \delta^{(m)}) \\ &= -\log \mathop{\mathbb{E}}_{p(m|x,t)} \exp(-\delta^{(m)}) - \bar{\delta}\end{aligned}$$

□

Jensen's inequality ($-\log$ is convex) gives us the easy upper bound

$$\begin{aligned}\mathrm{KL}(q(m|x)\|p(m|x,t)) &= -\log \mathop{\mathbb{E}}_{p(m|x,t)} \exp(\bar{\delta} - \delta^{(m)}) \\ &\leq -\mathop{\mathbb{E}}_{p(m|x,t)} \log \exp(\bar{\delta} - \delta^{(m)}) \\ &= \mathop{\mathbb{E}}_{p(m|x,t)} \delta^{(m)} - \mathop{\mathbb{E}}_{q(m|x)} \delta^{(m)}\end{aligned}$$

To understand this, note that the difference of$p(m|x,t)$ and $q(m|x)$ is precisely that $q(m|x)$ is smaller when $\delta^{(m)}$ is larger.

F Experimental details

Models were generated using Llama-3.3-70B [14, 21] with 4-bit AWQ quantization [19]. In testing, LLMs seemed much better at writing Stan code [5] than other PPLs like NumPyro [26] or PyMC [1], possibly due to more code being available. We thus used Stan, though the possibility of creating unnormalized distributions in Stan poses some difficulties (Appendix F.2).

The LLM system prompt (Appendix F.1) explains the PROBLEM / DATA / GOAL format illustrated in the experiments and asks the LLM to first write a THOUGHTS block explaining how it plans to model the problem, followed by a MODEL block of Stan code. For in-context learning, we provided six example inputs along with high quality outputs (Appendix G). We rejected any models that did not compile or that used certain language constructs that might lead to unnormalized models (Appendix F.2).

Models were generated using a single A100. With continuous batching (parallel inference), model generation was reasonably fast—for example, it took around 14 minutes to generate 1000 models for the city temperature problem and around 11 minutes for the polling problem.

For inference, it is necessary to reliably and automatically approximate posteriors and marginal likelihoods for thousands of models. To approximate the posterior, we simply used Stan's default NUTS [17] sampler with 10,000 iterations. To approximate the marginal likelihood, we used importance-weighted variational bounds [3], with a Gaussian proposal distribution. In general, it is known that such bounds are improved when the proposal distribution would minimize the χ^2 divergence to the target. However, given the difficulty of minimizing such divergences Geffner & Domke [13] we elected instead to use a proposal distribution $q(z|x,m)$ minimizing $\mathrm{KL}(p(\mathsf{z}|x,m)\|q(\mathsf{z}|x,m))$, which, given samples from p, amounts to moment matching, i.e. matching the empirical mean and variance of the samples from MCMC. The full inference strategy is given explicitly as Alg. 9 .

Algorithm 9 The hybrid MCMC / VI / SNIS algorithm used in the experiments.

1. Input textual description t and data x.
2. For $n = 1, 2, \cdots, N$:
   (a) Sample $m^{(n)} \sim p(m|t)$ using an LLM.
   (b) Draw samples $z^{(n,1)}, \cdots z^{(n,K)} \sim p(z|x, m^{(n)})$ using MCMC.
   (c) Set $q(z|x, m^{(n)})$ to be a Gaussian with mean and covariance matching the sample $z^{(n,1)}, \cdots, z^{(n,K)}$.
   (d) Estimate the importance-weighted ELBO $L^{(n)} = \hat{\mathbb{E}} \log \hat{\mathbb{E}} \frac{p(\mathsf{z}|x,m^{(n)})}{q(\mathsf{z}|x,m^{(n)})}$, where the inner expectation is estimated using 25 samples $\mathsf{z} \sim q$ and the outer approximate expectation uses 10,000 repetitions.
3. Set $w^{(n,k)} \propto \exp L^{(n)}$.
4. Return the set of samples $\{z^{(n,k)}\}$ where $z^{(n,k)}$ is given weight $w^{(n)}$.

Measuring the amount of compute used for inference is difficult since it was run on a heterogeneous cluster. However, running MCMC inference on 1000 models is of course quite expensive and required tens of hours of CPU time for each model. Simply compiling 1000 Stan models (on CPU) is more time consuming than generating 1000 models using an LLM, though of course both of these steps are embarrassingly parallel.

F.1 System prompt

The following system prompt was used when generating all models:

```
You are StanWriter. Given a description of a problem, you write Stan code.

You always first write THOUGHTS and then describe your model in words. Then you write MODEL and
write the Stan code.

Be creative when coming up with your model!

When you declare arrays, ALWAYS use the syntax "array[2,5] int x" NEVER the old syntax "int x[2,5]
".

In the generated quantities block ALWAYS use the _rng syntax like "x ~ normal_rng(mu,sigma)" NEVER
 "x ~ normal(mu,sigma)".

NEVER write ‘‘‘stan before your code.

ALWAYS give priors for all variables. NEVER use implicit/improper priors.

NEVER use "target += ..." syntax.
```

F.2 Validating models

We rejected any model that did not contain the string `MODEL`, or where the text after model failed to compile.

Using Stan for the purpose of this algorithm, has one significant disadvantage: It is easy in Stan to create models that are not normalized. This of course poses no issue when doing MCMC sampling, but is significant here since it can affect the marginal likelihood. Unnormalized models can be created in several ways:

1. By default, Stan uses flat / unnormalized / improper priors.
2. One can use the same variable in a sampling statement more than once, e.g. one may write "`x ~ normal(0,10)`" and then later write "`x ~ normal(3,5)`". Both of these statement increment the density, meaning the result is no longer normalized.
3. One can directly manipulate the target using the "`target +=` " construct, essentially leading to arbitrary changes with no probabilistic interpretation.
4. One can transform variables on the "left-hand side" e.g. write "`log(x) ~ normal(0,1)`". This again leads to an unnormalized density.
5. If variables are constrained in Stan, this constraint is not reflected in the normalization.

Except for the #5, none of these are too critical, and one can use a system prompt (and in-context examples) that discourages use. This works quite well, but (probably due to frequent use in example models) #3 remains common. We thus reject all models that contain the string "`target +=` ", as if they did not compile. Checking usage of #1, #2, and #4 is not trivial to do programmatically, and thus we relied instead on spot checks in models given high weight. Only a handful of cases were found.

This leaves issue #5, which unfortunately is difficult to avoid due to frequent usage of constraints in practice, e.g. to sample from truncated normal distributions. We simply accept this bias in practice, though note the bias is always downwards, e.g. towards a lower marginal likelihood.

G Examples

This section provides the example inputs and outputs given to the LLM to aid in-context learning.

G.1 Weight

INPUT

```
PROBLEM

Given the weight of a bunch of students, infer true mean of all students. Use reasonably informative priors.

DATA

int num_students; // number of students
array[num_students] real student_weight; // weight of each student in kilograms

GOAL

real weight_mean; // true mean weight for all students
```

OUTPUT

```
THOUGHTS

I will model this problem by creating two latent variables. ‘weight_mean‘ will reflect the true mean weight of all students,
and ‘weight_sigma‘ will reflect the standard deviation. Since I was instructed to use reasonably informative priors, I’ll
assume a normal prior for ‘weight_mean‘ with a mean of 80kg and a standard deviation of 20kg. For the ‘weight_sigma‘, I’ll
choose a lognormal prior with mu=log(10) and sigma=log(20). This reflects that my best guess for the standard deviation is
around 10 kg, but could be significantly larger or smaller.

MODEL

data {
        int num_students;
        array[num_students] real student_weight;
}
parameters {
        real weight_mean;
        real<lower=0> weight_sigma;
}
model {
        weight_mean ~ normal(80,20);
        weight_sigma ~ lognormal(log(10),log(20));
        for (i in 1:num_students){
                student_weight[i] ~ normal(weight_mean, weight_sigma);
        }
}
```

G.2 Future weight

INPUT

```
PROBLEM

Given the weight of a bunch of students, infer weight of a new student drawn from the same distribution. Use reasonably
informative priors.

DATA

int num_students; // number of students
array[num_students] real student_weight; // weight of each student in kilograms

GOAL

real new_student_weight; // weight for a future student, sampled from the same distribution
```

OUTPUT

```
data {
        int num_students;
        array[num_students] real student_weight;
}
parameters {
        real weight_mean;
        real<lower=0> weight_sigma;
}
model {
        weight_mean ~ normal(70,30);
        weight_sigma ~ student_t(2.5,0,5); // heavy tails
        for (i in 1:num_students){
                student_weight[i] ~ normal(weight_mean, weight_sigma);
        }
}
generated quantities {
        real new_student_weight = normal_rng(weight_mean, weight_sigma);
}
```

G.3 Movies

INPUT

```
PROBLEM

Given a bunch of binary features of a movie, predict if a user will like it or not.

DATA

int num_train; // number of movies to train on
int num_test; // number of movies to test on
int num_features; // number of binary features for each movie
array[num_train, num_features] int train_features; // binary features
array[num_train] int train_like; // did the user like the movie
array[num_train, num_features] int test_features; // binary features

GOAL

array[num_test] int test_like; // does the user like each movie in the test set?
```

OUTPUT

```
THOUGHTS

I will model this problem by creating a vector ‘beta‘ of size ‘num_features‘. When the inner-product of this vector is taken
with the features of a movie, it gives the logit for how likely the user is to like that movie. I’ll sample each entry of that
 vector from a standard normal, which seems like a reasonable scale when being combined with binary features. I’ll then sample
 both ‘train_like‘ and ‘test_like‘ by taking the inner-product of ‘beta‘ with each row of ‘train_features‘ and ‘test_features
‘, respectively.

MODEL

data {
        int num_train;
        int num_test;
        int num_features;
        array[num_train, num_features] int<lower=0, upper=1> train_features;
        array[num_train] int<lower=0, upper=1> train_like;
        array[num_test, num_features] int<lower=0, upper=1> test_features;
}
parameters {
        vector[num_features] beta;
}
model {
        beta ~ normal(0,1);
        for (n in 1:num_train){
            train_like[n] ~ bernoulli_logit(to_row_vector(train_features[n]) * beta);
        }
}
generated quantities {
        array[num_test] int<lower=0, upper=1> test_like;
        for (n in 1:num_test){
                test_like[n] = bernoulli_logit_rng(to_row_vector(test_features[n]) * beta);
        }
}
```

G.4 Collaborative filtering

INPUT

```
PROBLEM

Given a bunch of like ratings of various users of various movies, predict if users will like future movies. Do this by
inferring underlying features of users and movies.

DATA

int num_users; // number of users
int num_movies; // number of movies
int num_ratings; // number of observed ratings
array[num_ratings] int user; // what user did rating
array[num_ratings] int movie; // what movie was rated
array[num_train, num_features] int like; // did user like the movie (1 if yes, 0 if no)

GOAL

int array[num_users, num_movies] all_ratings; // would each user like each movie?
```

OUTPUT

```
THOUGHTS

I will model this problem by assuming that both users and movies can be described in terms of ‘num_features‘=10 features. I’ll
 create a latent variable ‘user_features‘ that describes the features for each user and a latent variable ‘movie_features‘
that describes the features for each movie. I will assume standard cauchy priors for all these features, indicating that most
features are small, but some might be quite large. Finally, I assume that the probability of a given user liking a given movie
 is given by a bernoulli_logit distribution with a score consisting of the inner-product of the user and movie features. To
create the desired output ‘all_ratings‘, I will create a generating quantities block in which I loop over all pairs of users
and movies.

MODEL

data {
        int num_users; // number of users
        int num_movies; // number of movies
        int num_ratings; // number of observed ratings
        array[num_ratings] int user; // what user did rating
        array[num_ratings] int movie; // what movie was rated
        array[num_ratings] int like; // did user like the movie (1 if yes, 0 if no)
}
transformed data{
        int num_features = 10;
}
parameters {
        array[num_users, num_features] real user_features;
        array[num_movies, num_features] real movie_features;
}
model {
        for(k in 1:num_features){
                for(i in 1:num_users){
                        user_features[i,k] ~ cauchy(0,1);
                }
                for(j in 1:num_movies){
                        movie_features[j,k] ~ cauchy(0,1);
                }
        }
        for (n in 1:num_ratings){
                array[num_features] real x = user_features[user[n]];
                array[num_features] real y = movie_features[movie[n]];
                real score = dot_product(x,y);
                like[n] ~ bernoulli_logit(score);
        }
}
generated quantities {
        array[num_users, num_movies] int all_ratings;
        for(i in 1:num_users){
                for(j in 1:num_movies){
                        array[num_features] real x = user_features[i];
                        array[num_features] real y = movie_features[j];
                        real score = dot_product(x,y);
                        all_ratings[i,j] = bernoulli_logit_rng(score);
                }
        }
}
```

G.5 Vaccines

INPUT

```
PROBLEM

There is data for a bunch of different vaccine trials. There are different types of vaccines. Given this data, infer the mean
response rate for each trial type. Assume that different vaccines of the same type tend to have similar response rates and
that different types of vaccines also usually have response rates that aren't that different.

DATA

int num_types; // number of different types of vaccines
int num_trials; // number of trials
array[num_trials] int type; // what type of vaccine tested in each trial
array[num_trials] int num_subjects; // number of subjects in each trial
array[num_trials] int responders; // number of subjects in each trial that responded

GOAL

array[num_types] response_rate; // mean response rate for a new vaccine of each type
```

OUTPUT

```
THOUGHTS

I'll create a hierarchical model. The idea is that each trial has some response rate that is drawn from a per-trial-type
distribution. Then the parameters for the per-trial-type distributions are all drawn from some global distribution. By
sampling the three levels at once (individual trial response rates, per-trial-type response rates, and global rates) all
information can be optimally shared.

In more detail, I'll first create parameters `a` and `b` for each trial type. These determine the distribution over true
response rates for different trials of each type. The true response rate for trial `i`, `per_trial_response_rate[i]`, is drawn
 from a beta distribution with parameters `a[type[i]]` and `b[type[i]]`. Then the (observed) number of responders in each
trial will be drawn from a binomial distribution with `num_subjects[i]` repeats and a `per_trial_response_rate[i]` success
rate. To create a prior over `a` and `b`, I will draw a 2-dimensional vector `z[n]` for each trial type from some multi normal
 distribution parameterized by `mu` and `L`. This will be mapped to `a` and `b` in the transformed parameters block through an
 inverse logit transformation. I'll sample `mu` from a broad multivariate normal distribution and `L` from the distribution
that results from sampling from a broad inverse Wishart distribution and then computing the Cholesky factorization.

MODEL

data {
        int num_types;
        int num_trials;
        array[num_trials] int type;
        array[num_trials] int num_subjects;
        array[num_trials] int responders;
}
parameters {
        cholesky_factor_cov[2] L;
        vector[2] mu;
        array[num_types] vector[2] z; // parameters for response rate dist for each type
        array[num_trials] real<lower=0, upper=1> per_trial_response_rate;
}
transformed parameters{
        // map unconstrained parameters for each dist type to pair on 0-1 interval
        array[num_types] real a;
        array[num_types] real b;
        array[num_types] real response_rate;
        for(n in 1:num_types){
                a[n] = inv_logit(z[n,1]);
                b[n] = inv_logit(z[n,2]);
                response_rate[n] = a[n]/(a[n]+b[n]); // mean of beta distribution is a/(a+b)
        }
}
model {
        mu ~ multi_normal([0,0],[[5,0],[0,5]]);
        L ~ inv_wishart_cholesky(3.0, [[1,0],[0,1]]);
        for(n in 1:num_types){
                z[n] ~ multi_normal_cholesky(mu, L);
        }
        for(i in 1:num_trials){
                per_trial_response_rate[i] ~ beta(a[type[i]], b[type[i]]);
                responders[i] ~ binomial(num_subjects[i], per_trial_response_rate[i]);
        }
}
```

G.6 Water

INPUT

```
PROBLEM

I start with 1 kg of water. Every day, some fraction of the water (between 0 and 1) evaporates, and I measure how much is
remaining. My measurements have a noise of something around 10 g. Predict how much water evaporates every day. It could be
anything between 0 and 1, but I suspect it's something around 0.25. The first day with 1kg isn't measured.

DATA

int num_days;
array[num_days] measurements; // measurements in kg

GOAL

real evaporation; // what fraction evaporates each day?
```

OUTPUT

```
THOUGHTS

I will model this problem by creating a latent variable 'evaporation' which measures what fraction of the water disappears
each day. I will sample this from a beta distribution with alpha=1 and beta=3 meaning a mean of around 0.25. Given the
evaporation, I will use a for loop to define an array 'current_water' which is the true (unobserved) amount of water remaining
 after each day. Finally, I will sample 'measurements[i]' for each day i from a normal distribution with a mean of water[i]
and a standard deviation of 0.01, chosen because 0.01 kg = 10g.

MODEL

data {
        int num_days;
        array[num_days] real measurements;
}
parameters {
        real<lower=0, upper=1> evaporation;
}
transformed parameters{
        array[num_days] real<lower=0, upper=1> water;
        real current_water = 1.0;
        for(i in 1:num_days){
                current_water = current_water * (1-evaporation);
                water[i] = current_water;
        }
}
model {
        evaporation ~ beta(1,3);
        for(i in 1:num_days){
                measurements[i] ~ normal(water[i], 0.01);
        }
}
```

NeurIPS Paper Checklist

1. Claims

Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

Answer: [Yes]

Justification: The abstract and introduction reflect the contribution and scope of the paper, and do not include aspirational goals.

2. Limitations

Question: Does the paper discuss the limitations of the work performed by the authors?

Answer: [Yes]

Justification:

3. Theory assumptions and proofs

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [NA]

Justification: Yes, typically in appendix.

4 Experimental result reproducibility

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: All details disclosed.

5. Open access to data and code

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: It is difficult to provide code runnable by a third party given the usage of a local cluster and many non-portable modifications. Every effort has been made to make the results reproducible from the given description.

6. Experimental setting/details

Question: Does the paper specify all the training and test details (e.g., data splits, hyper-parameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

Answer: [Yes]

Justification:

7. Experiment statistical significance

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification:

8. Experiments compute resources

Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

Answer: [Yes]

Justification:

9. Code of ethics

Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics?

Answer: [Yes]

Justification:

10. Broader impacts

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [NA]

Justification: This paper does not appear to present significant social impacts beyond the core goal (which is discussed) of making probabilistic models easier to write.

11. Safeguards

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

Answer: [NA]

Justification:

12. Licenses for existing assets

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification:

13. New assets

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [NA]

Justification:

14. Crowdsourcing and research with human subjects

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [NA]

Justification:

15. Institutional review board (IRB) approvals or equivalent for research with human subjects

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [NA]

Justification:

16. Declaration of LLM usage

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing,

editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required

Answer: [Yes]

Justification: It was used only for generating probabilistic models as described clearly in the text. LLMs were not used in any other fashion, e.g. for generating code or writing.