Chapter 1

Accelerating Metropolis-hastings with lightweight inference compilation

While the subsequent chapters depart from a previously common theme of DPPs, it continues our study of statistical applications of randomized methods. In this chapter, we are concerned with the problem of sampling intractable posteriors of Bayesian graphical models. In order to construct accurate proposers for Metropolis-Hastings Markov Chain Monte Carlo, we integrate ideas from probabilistic graphical models and neural networks in a framework we call Lightweight Inference Compilation (LIC). LIC implements amortized inference within an open-universe declarative probabilistic programming language (PPL). Graph neural networks are used to parameterize proposal distributions as functions of Markov blankets, which during "compilation" are optimized to approximate single-site Gibbs sampling distributions. Unlike prior work in inference compilation (IC), LIC forgoes importance sampling of linear execution traces in favor of operating directly on Bayesian networks. Through using a declarative PPL, the Markov blankets of nodes (which may be non-static) are queried at inference-time to produce proposers. Experimental results show LIC can produce proposers which have less parameters, greater robustness to nuisance random variables, and improved posterior sampling in a Bayesian logistic regression and n-schools inference application. Parts of this chapter were originally published in **lic**.

1.1 Background

Deriving and implementing samplers has traditionally been a high-effort and application-specific endeavour [**porteous2008fast**; **murray2010elliptical**], motivating the development of general-purpose probabilistic programming languages (PPLs) where a non-expert can specify a generative model (i.e. joint distribution) $p(\boldsymbol{x}, \boldsymbol{y})$ and the software automatically performs inference to sample latent variables $\boldsymbol{x}$ from the posterior $p(\boldsymbol{x} \mid \boldsymbol{y})$ conditioned on observations $\boldsymbol{y}$. While exceptions exist, modern general-purpose PPLs typically implement variational inference [**bingham2019pyro**], importance sampling [**wood2014new**;

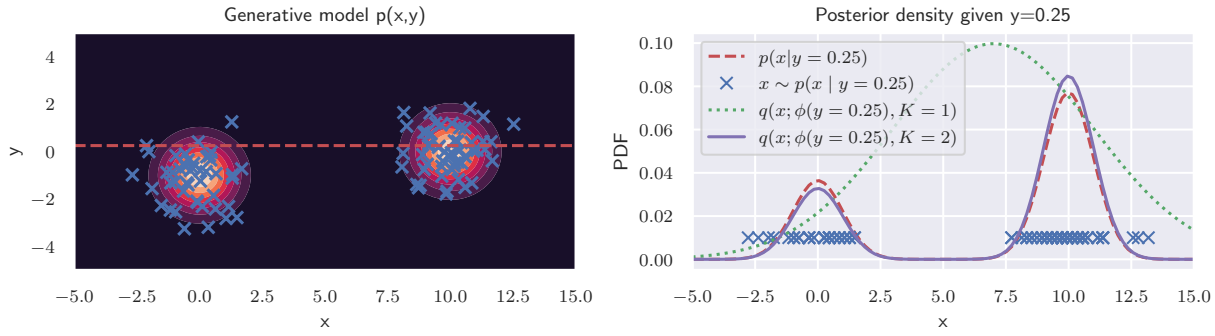


Figure 1.1: Intuition for Lightweight Inference Compilation (LIC). LIC uses samples $(x_i, y_i) \overset{\text{iid}}{\sim} p$ (blue "x" in left) drawn from the joint density $p(x, y)$ to approximate the expected inclusive KL-divergence $\mathbb{E}_{p(y)} D_{\text{KL}}(p(x \mid y) \parallel q(x; \phi(y)))$ between the posterior $p(x \mid y)$ and the LIC proposal distribution $q(x; \phi(y))$. For an observation $y = 0.25$ (dashed red line in left), the posterior $p(x \mid y = 0.25)$ (dashed red line in right) is "approximated" by samples "close" to y (blue "x" in right) to form an empirical inclusive KL-divergence minimized by LIC. As inclusive KL-divergence encourages a mass-covering / mean-seeking fit, the resulting proposal distribution $q(x; \phi(y = 0.25), K = 1)$ (green dotted line in right) when using a single $(K = 1)$ Gaussian proposal density covers both modes and can successfully propose moves which cross between the two mixture components. Using a 2-component $(K = 2)$ GMM proposal density results in $q(x; \phi(y = 0.25), K = 2)$ (purple solid line in right) which captures both the bi-modality of the posterior as well as the low probability region between the two modes. As a result of sampling the generative model, LIC can discover both posterior modes and their appropriate mixture weights (whereas other state of the art MCMC samplers fail, see fig. 1.3).

le2017inference], or Monte Carlo Markov Chain (MCMC, [**wingate2011lightweight**; **tehrani2020beanmachine**]).

Our work focuses on MCMC. More specifically, we target lightweight Metropolis-Hastings (LMH, [**wingate2011lightweight**]) within a recently developed declarative PPL called `beanmachine` [**tehrani2020beanmachine**]. The performance of Metropolis-Hastings critically depends on the quality of the proposal distribution used, which is the primary goal of LIC. Broadly speaking, LIC amortizes MCMC by constructing function approximators (parameterized by graph neural networks) from Markov blankets to proposal distribution parameters which are learned via forward simulation and training to match the full conditionals (equivalently, to minimize the inclusive KL divergence). In doing so, LIC makes the following contributions:

1. We present a novel implementation of inference compilation (IC) within an open-universe declarative PPL which combines Markov blanket structure with graph neural network

architectures. Our primary innovation is the use of a graph neural network to aggregate Markov blankets, which enables proposers to ignore irrelevant variables by construction.

2. We demonstrate LIC's ability to escape local modes, improved robustness to nuisance random variables, and improvements over state-of-the-art methods across a number of metrics in two industry-relevant applications.

Declarative Probabilistic Programming

To make Bayesian inference accessible to non-experts, PPLs provide user-friendly primitives in high-level programming languages for abstraction and composition in model representation [**goodman2013principles**; **ghahramani2015probabilistic**]. Existing PPLs can be broadly classified based on the representation inference is performed over, with declarative PPLs [**lunn2000winbugs**; **plummer2003jags**; **milch20071**; **tehrani2020beanmachine**] performing inference over Bayesian graphical networks and imperative PPLs conducting importance sampling [**wood2014new**] or MCMC [**wingate2011lightweight**] on linearized execution traces. Because an execution trace is a topological sort of an instantiated Bayesian network, declarative PPLs naturally preserve additional model structure such as Markov blanket relationships while imperative PPLs require additional dependency tracking instrumentation [**mansinghka2014venture**] or analysis tooling [**gorinova2020conditional**; **cusumano2019gen**] to achieve similar functionality.

Definition 1 *The* Markov Blanket $\mathrm{MB}(x_i)$ *of a node* x_i *is the minimal set of random variables such that*

$$p(x_i \mid \boldsymbol{x}_{-i}, \boldsymbol{y}) = p(x_i \mid \mathrm{MB}(x_i)) \tag{1.1}$$

In a Bayesian network, $\mathrm{MB}(x_i)$ *consists of the parents, children, and children's parents of* x_i *[***pearl1987evidential***].*

Inference Compilation

Amortized inference [**gershman2014amortized**] refers to the re-use of initial up-front learning to improve future queries. In context of Bayesian inference [**marino2018iterative**; **zhang2018advances**] and IC [**paige2016inference**; **weilbach2019efficient**; **harvey2019attention**], this means using acceleration performing multiple inferences over different observations $\boldsymbol{y}$ to amortize a one-time "compilation time." While compilation in both trace-based IC [**paige2016inference**; **le2017inference**; **harvey2019attention**] and LIC consists of drawing forward samples from the generative model $p(\boldsymbol{x}, \boldsymbol{y})$ and training neural networks to minimize inclusive KL-divergence, trace-based IC uses the resulting neural network artifacts to parameterize proposal distributions for importance sampling while LIC uses them for MCMC proposers.

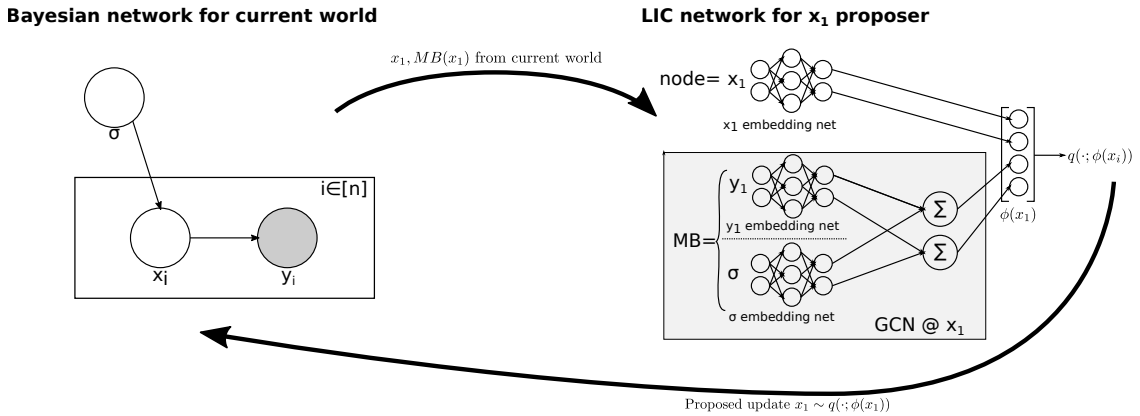


Figure 1.2: The Markov blankets in the Bayesian network for eq. (1.3) (left, expressed in plate notation) are available in a declarative PPL, and are used as inputs to LIC. The LIC proposer for node x_1 (right) is obtained by first performing neural network embedding of x_1 and every node in its Markov blanket, followed by a graph convolutional network aggregation over the Markov blanket of x_1. The resulting vectors are then combined to yield a parameter vector $\phi(x_1)$ for a proposal distribution $q(\cdot; \phi(x_1))$ which is then sampled for proposing an update within Metropolis-Hastings.

Lightweight Metropolis Hastings

Lightweight Metropolis Hastings (LMH, [**wingate2011lightweight**; **ritchie2016c3**]) is a simple but general method of implementing a PPL. In LMH, random variables are assigned string identifiers and their values and likelihoods are stored in a database (`World` in our implementation). MCMC is performed in a Metropolis-within-Gibbs manner where: (1) a single random variable is modified according to a proposal distribution while all others remain fixed, (2) the computations dependant upon the modified random variable (e.g. the continuation in a continuation-passing-style implementation) is re-executed to generate the remaining trace (re-using the database values for all other random variables) and the new trace's likelihood, and finally (3) a Metropolis-Hastings accept/reject correction is performed.

While a number of choices for proposal distribution exist, the single-site Gibbs sampler which proposes from eq. (1.1) enjoys a 100% acceptance probability [**pearl1987evidential**] and provides a good choice when available [**lunn2000winbugs**; **plummer2003jags**]. Unfortunately, outside of discrete models they are oftentimes intractable to directly sample so another proposal distribution must be used. LIC seeks to approximate these single-site Gibbs distributions using tractable neural network approximations.

Related Works

Prior work on IC in imperative PPLs can be broadly classified based on the order in which nodes are sampled. "Backwards" methods approximate an inverse factorization, starting at

observations and using IC artifacts to propose propose parent random variables. Along these lines, [**paige2016inference**] use neural autoregressive density estimators but heuristically invert the model by expanding parent sets. [**webb2018faithful**] proposes a more principled approach utilizing minimal I-maps and demonstrate that minimality of inputs to IC neural networks can be beneficial; an insight also exploited through LIC's usage of Markov blankets. Unfortunately, model inversion is not possible in universal PPLs [**le2017inference**].

The other group of "forwards" methods operate in the same direction as the probabilistic model's dependency graph. Starting at root nodes, these methods produce inference compilation artifacts which map an execution trace's prefix to a proposal distribution. In [**ritchie2016deep**], a user-specified model-specific guide program parameterizes the proposer's architecture and results in more interpretable IC artifacts at the expense of increased user effort. [**le2017inference**] automates this by using a recurrent neural network (RNN) to summarize the execution prefix while constructing a node's proposal distribution. This approach suffers from well-documented RNN limitations learning long distance dependencies [**hochreiter1998vanishing**], requiring mechanisms like attention [**harvey2019attention**] to avoid degradation in the presence of long execution trace prefixes (e.g. when nuisance random variables are present).

With respect to prior work, LIC is most similar to the attention-based extension [**harvey2019attention**] of [**le2017inference**]. Both methods minimize inclusive KL-divergence empirically approximated by samples from the generative model $p(\boldsymbol{x}, \boldsymbol{y})$, and both methods use neural networks to produce a parametric proposal distribution from a set of inputs sufficient for determining a node's posterior distribution. However, important distinctions include (1) LIC's use of a declarative PPL implies Markov blanket relationships are directly queryable and ameliorates the need for also learning an attention mechanism, and (2) LIC uses a graph neural network to summarize the Markov blanket rather than a RNN over the execution trace prefix. [**wang2017meta**] is also closely related to LIC as both are methods for amortizing MCMC by learning Markov blanket parameterized neural Gibbs proposers, but a key difference is that LIC exploits permutation-invariance of graph neural networks to address the issue where "Markov blankets... might not be consistent across all instantiations" whereas [**wang2017meta**] restricts "focus on hand-identified common structur[al motifs]" for constructing proposers.

1.2 Lightweight Inference Compilation

Architecture

Figure 1.2 shows a sketch of LIC's architecture. For every latent node x_i, LIC constructs a mapping $(x_i, \mathrm{MB}(x_i)) \mapsto \phi(x_i)$ parameterized by feedforward and graph neural networks to produce a parameter vector $\phi(x_i)$ for a parametric density estimator $q(\cdot; \phi(x_i))$. Every node x_i has feedforward "node embedding network" used to map the value of the underlying random variable into a vector space of common dimensionality. The set of nodes in the

Markov blanket are then summarized to a fixed-length vector following section 1.2, and a feedforward neural network ultimately maps the concatenation of the node's embedding with its Markov blanket summary to proposal distribution parameters $\phi(x_i)$.

Dynamic Markov Blanket embeddings

Because a node's Markov blanket may vary in both its size and elements (e.g. in a GMM, a data point's component membership may change during MCMC), $\text{MB}(x_i)$ is a non-static set of vectors (albeit all of the same dimension after node embeddings are applied) and a feed-forward network with fixed input dimension is unsuitable for computing a fixed-length proposal parameter vector $\phi(x_i)$. Furthermore, Markov blankets (unlike execution trace prefixes) are unordered sets and lack a natural ordering hence use of a RNN as done in [**le2017inference**] is inappropriate. Instead, LIC draws motivation from graph neural networks [**scarselli2008graph**; **dai2016discriminative**] which have demonstrated superior results in representation learning [**bruna2013spectral**] and performs summarization of Markov Blankets following [**kipf2016semi**] by defining

$$\phi(x_i) = \sigma\left(\boldsymbol{W} \underset{x_j \in \text{MB}(x_i)}{\square} \frac{1}{\sqrt{|\text{MB}(x_i)||\text{MB}(x_j)|}} f_j(x_j)\right)$$

where $f_j(x_j)$ denotes the output of the node embedding network for node x_j when provided its current value as an input, $\square$ is any differentiable permutation-invariant function (summation in LIC's case), and σ is an activation function. This technique is analogous to the "Deep sets trick" [**zaheer2017deep**], and we expect it to perform well when elements of the Markov blanket are conditionally exchangeable. However, we note this assumption of permutation invariance may not always hold hence additional investigation into more sophisticated aggregation schemes (e.g. identifying exchangeable elements, using permutation-dependent aggregators like RNNs) is important future work.

Parameterized density estimation

The resulting parameter vectors $\phi(x_i)$ of LIC are ultimately used to parameterize proposal distributions $q(x_i; \phi(\text{MB}(x_i)))$ for MCMC sampling. For discrete x_i, LIC directly estimates logit scores over the support. For continuous x_i, LIC transforms continuous x_i to unconstrained space following [**carpenter2017stan**] and models the density using a Gaussian mixture model (GMM). Note that although more sophisticated density estimators such as masked autoregressive flows [**kingma2016improved**; **papamakarios2017masked**] can equally be used.

Objective Function

To "compile" LIC, parameters are optimized to minimize the inclusive KL-divergence between the posterior distributions and inference compilation proposers: $D_{\text{KL}}(p(\boldsymbol{x} \mid \boldsymbol{y}) \parallel q(\boldsymbol{x} \mid \boldsymbol{y}; \phi))$.

Consistent with [**le2017inference**], observations $\boldsymbol{y}$ are sampled from the marginal distribution $p(\boldsymbol{y})$, but note that this may not be representative of observations $\boldsymbol{y}$ encountered at inference. The resulting objective function is given by

$$\begin{aligned}
&\mathbb{E}_{p(\boldsymbol{y})}\left[D_{\mathrm{KL}}(p(\boldsymbol{x} \mid \boldsymbol{y}) \parallel q(\boldsymbol{x} \mid \boldsymbol{y}; \phi))\right] \\
&= \mathbb{E}_{p(\boldsymbol{x},\boldsymbol{y})}\left[\log \frac{p(\boldsymbol{x} \mid \boldsymbol{y})}{q(\boldsymbol{x} \mid \boldsymbol{y}; \phi)}\right] \\
&\propto \mathbb{E}_{p(\boldsymbol{x},\boldsymbol{y})}\left[-\log q(\boldsymbol{x} \mid \boldsymbol{y}; \phi)\right] \\
&\approx \sum_{i=1}^{N} -\log q(\boldsymbol{x} \mid \boldsymbol{y}; \phi), \qquad (\boldsymbol{x}, \boldsymbol{y}) \overset{\text{i.i.d.}}{\sim} p(\boldsymbol{x}, \boldsymbol{y}) \\
&=: \mathcal{L}(\phi)
\end{aligned} \tag{1.2}$$

where we have neglected a conditional entropy term independent of ϕ and performed Monte-Carlo approximation to an expectation. The intuition for this objective is shown in Figure 1.1, which shows how samples from $p(\boldsymbol{x}, \boldsymbol{y})$ (left) form an empirical joint approximation where "slices" (at $y = 0.25$ in fig. 1.1) yield posterior approximations which the objective is computed over (right).

1.3 Experiments

To validate LIC's competitiveness, we conducted experiments benchmarking a variety of desired behaviors and relevant applications. In particular:

- Training on samples from the joint distribution $p(\boldsymbol{x}, \boldsymbol{y})$ should enable discovery of distant modes, so LIC samplers should be less likely to to get "stuck" in a local mode. We validate this in section 1.3 using a GMM mode escape experiment, where we see LIC escape not only escape a local mode but also yield accurate mixture component weights.

- When there is no approximation error (i.e. the true posterior density is within the family of parametric densities representable by LIC), we expect LIC to closely approximate the posterior at least for the range of observations $\boldsymbol{y}$ sampled during compilation (eq. (1.2)) with high probability under the prior $p(\boldsymbol{y})$. Section 1.3 shows this is indeed the case in a conjugate Gaussian-Gaussian model where a closed form expression for the posterior is available.

- Because Markov blankets can be explicitly queried, we expect LIC's performance to be unaffected by the presence of nuisance random variables (i.e. random variables which extend the execution trace but are statistically independent from the observations and queried latent variables). This is confirmed in section 1.3 using the probabilistic program from [**harvey2019attention**], where we see trace-based IC suffering an order

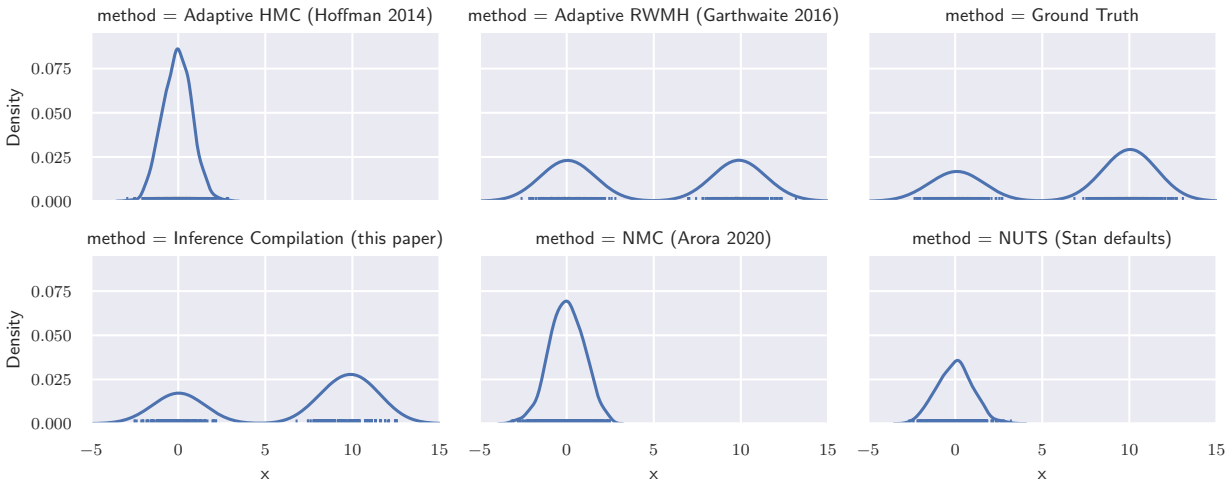


Figure 1.3: When sampling the bi-modal posterior density from fig. 1.1, only inference compilation (IC, this paper) and adaptive step-size random walk Metropolis-Hastings (Adaptive RWMH, [**garthwaite2016adaptive**]) are able to recover both posterior modes. Whereas RWMH's posterior samples erroneously assign approximately equal probability to both modes, IC's samples faithfully reproduce the ground truth and yields higher probability for the mode at $x = 10$ than the mode at $x = 0$.

of magnitude increase in model parameters and compilation time while yielding an effective sample size almost 5× smaller (Figure 1.5).

- To verify LIC yields competitive performance in applications of interest, we benchmark LIC against other state-of-the-art MCMC methods on a Bayesian logistic regression problem (section 1.3) and on a generalization of the classical eight schools problem [**rubin1981estimation**] called n-schools (section 1.3) which is used in production at a large internet company for Bayesian meta-analysis [**sutton2001bayesian**]. We find that LIC exceeds the performance of adaptive random walk Metropolis-Hastings [**garthwaite2016adaptive**] and Newtonian Monte Carlo [**arora2020newtonian**] and yields comparable performance to NUTS [**hoffman2014no**] despite being implemented in an interpreted (Python) versus compiled (C++) language.

A reference implementation for LIC and code to reproduce our experiments have been made publically available [**lic2020**].

GMM mode escape

Consider the multi-modal posterior resulting from conditioning on $y = 0.25$ in the 2-dimensional GMM in fig. 1.1, which is comprised of two Gaussian components with greater mixture probability on the right-hand component and a large energy barrier separating the

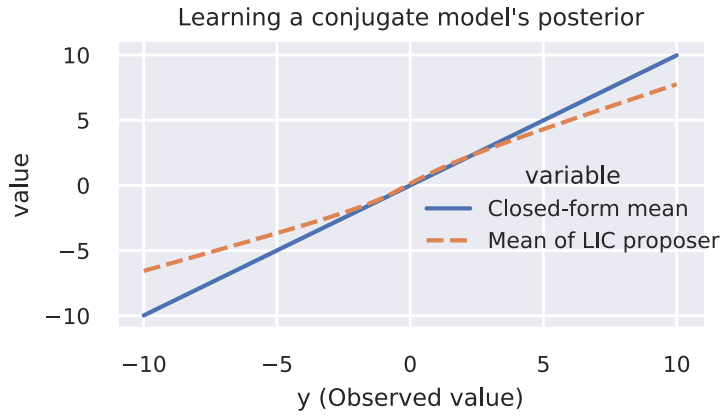


Figure 1.4: In a conjugate normal-normal model, LIC's proposal distribution mean (dashed orange line) closely follows the closed-form posterior mean (blue solid line) across a wide range of observed values y.

two components. Because LIC is compiled by training on samples from the joint distribution $p(x, y)$, it is reasonable to expect LIC's proposers to assign high probability to values for the latent variable x from both modes. In contrast, uncompiled methods such as random walk Metropolis-Hastings (RWMH) and NUTS may encounter difficulty crossing the low-probability energy barrier and be unable to escape the basin of attraction of the mode closest to their initialization.

This intuition is confirmed in fig. 1.3, which illustrates kernel density estimates of 1,000 posterior samples obtained by a variety of MCMC algorithms as well as ground truth samples. HMC with adaptive step size [**hoffman2014no**], NMC, and NUTS with the default settings as implemented in Stan [**carpenter2017stan**] are all unable to escape the mode they are initialized in. While both LIC and RWMH with adaptive step size escape the local mode, RWMH's samples erroneously imply equal mixture component probabilities whereas LIC's samples faithfully reproduce a higher component probability for the right-hand mode.

Conjugate Gaussian-Gaussian Model

We next consider a Gaussian likelihood with a Gaussian mean prior, a conjugate model with closed-form posterior given by:

$$\begin{aligned} x &\sim \mathcal{N}(0, \sigma_x), \qquad y \sim \mathcal{N}(x, \sigma_y) \\ \Pr[x \mid y, \sigma_x, \sigma_y] &\sim \mathcal{N}\left(\frac{\sigma_y^{-2}}{\sigma_x^{-2} + \sigma_y^{-2}} y, \frac{1}{\sigma_x^{-2} + \sigma_y^{-2}}\right) \end{aligned} \tag{1.3}$$

There is minimal approximation error because the posterior density is in the same family as LIC's GMM proposal distributions and the relationship between the Markov blanket

$\text{MB}(x) = \{y\}$ and the posterior mean is a linear function easily approximated (locally) by neural networks. As a result, we expect LIC's proposal distribution to provide a good approximation to the true posterior and LIC to approximately implement a direct posterior sampler.

To confirm this behavior, we trained LIC with a $K = 1$ component GMM proposal density on 1,000 samples and show the resulting LIC proposer's mean as the observed value y varies in fig. 1.4. Here $\sigma_x = 2$ and $\sigma_y = 0.1$, so the marginal distribution of y (i.e. the observations sampled during compilation in eq. (1.2)) is Gaussian with mean 0 and standard deviation $\sqrt{\sigma_x^2 + \sigma_y^2} \approx 2.0025$. Consistent with our expectations, LIC provides a good approximation to the true posterior for observed values y well-represented during training (i.e. with high probability under the marginal $p(y)$). While LIC also provides a reasonable proposer by extrapolation to less represented observed values y, it is clear from fig. 1.4 that the approximation is less accurate. This motivates future work into modifying the forward sampling distribution used to approximate eq. (1.2) (e.g. "inflating" the prior) as well as adapting LIC towards the distribution of observations y used at inference time.

Robustness to Nuisance Variables

An important innovation of LIC is its use of a declarative PPL and ability to query for Markov blanket relationships so that only statistically relevant inputs are utilized when constructing proposal distributions. To validate this yields significant improvement over prior work in IC, we reproduced an experiment from [**harvey2019attention**] where nuisance random variables (i.e. random variables which are statistically independent from the queried latent variables/observations whose only purpose is to extend the execution trace) are introduced and the impact on system performance is measured. As trace-based inference compilation utilizes the execution trace prefix to summarize program state, extending the trace of the program with nuisance random variables typically results in degradation of performance due to difficulties encountered by RNN in capturing long range dependencies as well as the production of irrelevant neural network embedding artifacts.

We reproduce trace-based IC as described in [**le2017inference**] using the author-provided software package [**pyprob2020**], and implement Program 1 from [**harvey2019attention**] with the source code illustrated in listing 1.1 where 100 nuisance random variables are added. Note that although `nuisance` has no relationship to the remainder of the program, the line number where they are instantiated has a dramatic impact on performance. By extending the trace between where `x` and `y` are defined, trace-based IC's RNNs degrade due to difficulty learning a long-range dependency[**hochreiter1998vanishing**] between the two variables. For LIC, the equivalent program expressed in the `beanmachine` declarative PPL [**tehrani2020beanmachine**] is shown in listing 1.2. In this case, the order in which random variable declarations appear is irrelevant as all permutations describe the same probabilistic graphical model.

Listing 1.1: A version of Program 1 from [**harvey2019attention**] to illustrate nuisance random variables

```
def magnitude(obs):
  x = sample(Normal(0, 10))
  for _ in range(100):
    nuisance = sample(Normal(0, 10))
  y = sample(Normal(0, 10))
  observe(
    obs**2,
    likelihood=Normal(
      x**2 + y**2,
      0.1))
  return x
```

Listing 1.2: The equivalent program in beanmachine, where independencies are explicit in program specification

```
class NuisanceModel:
  @random_variable
  def x(self):
      return dist.Normal(0, 10)
  @random_variable
  def nuisance(self, i):
      return dist.Normal(0, 10)
  @random_variable
  def y(self):
      return dist.Normal(0, 10)
  @random_variable
  def noisy_sq_length(self):
      return dist.Normal(
        self.x()**2 + self.y()**2,
        0.1)
```

Figure 1.5 compares the results between LIC and trace-based IC [**le2017inference**] for this nuisance variable model. Both `pyprob`'s defaults (1 layer 4 dimension sample embedding, 64 dimension address embedding, 8 dimension distribution type embedding, 10 component GMM proposer, 1 layer 512 dimension LSTM) and LIC's defaults (used for all experiments in this paper, 1 layer 4 dimension node embedding, 3 layer 8 dimension Markov blanket embedding, 1 layer node proposal network) with a 10 component GMM proposer are trained on 10,000 samples and subsequently used to draw 100 posterior samples. Although model size is not directly comparable due differences in model architecture, `pyprob`'s resulting models were over 7× larger than those of LIC. Furthermore, despite requiring more than 10× longer time to train, the resulting sampler produced by `pyprob` yields an effective sample size almost

	# params	compile time	ESS
LIC	3,358	44 sec.	49.75
PyProb	21,952	472 sec.	10.99

Figure 1.5: Number of parameters, compilation time (10,000 samples), and effective sample size (100 samples) for inference compilation in LIC (this work) versus [**pyprob2020**]

5× smaller than that produced by LIC.

Bayesian Logistic Regression

Consider a Bayesian logistic regression model over d covariates with prior distribution $\boldsymbol{\beta} \sim \mathcal{N}_{d+1}(\mathbf{0}_{d+1}, \text{diag}(10, 2.5\mathbf{1}_d))$ and likelihood $y_i \mid \boldsymbol{x}_i \overset{\text{i.i.d.}}{\sim} \text{Bernoulli}(\sigma(\boldsymbol{\beta}^\top \boldsymbol{x}_i))$ where $\sigma(t) = (1 + e^{-t})^{-1}$ is the logistic function. This model is appropriate in a wide range of classification problems where prior knowledge about the regression coefficients $\boldsymbol{\beta}$ are available.

Figure 1.6a shows the results of performing inference using LIC compared against other MCMC inference methods. Results for existing IC approaches [**le2017inference**] are omitted because they are not comparable due to lack of support for vector-valued random variables in the publically available reference implementation [**pyprob2020**]. All methods yield similar predictive log-likelihoods on held-out test data, but LIC and NUTS yield significantly higher ESS and $\widehat{R}$s closer to 1.0 suggesting better mixing and more effective sampling.

n-Schools

The eight schools model [**rubin1981estimation**] is a Bayesian hierarchical model originally used to model the effectiveness of schools at improving SAT scores. n-schools is a generalization of this model from 8 to n possible treatments, and is used at a large internet company for performing Bayesian meta-analysis [**sutton2001bayesian**] to estimate (fixed) effect sizes. Let K denote the total number of schools, n_j the number of districts/states/types, and j_k the district/state/type of school k.

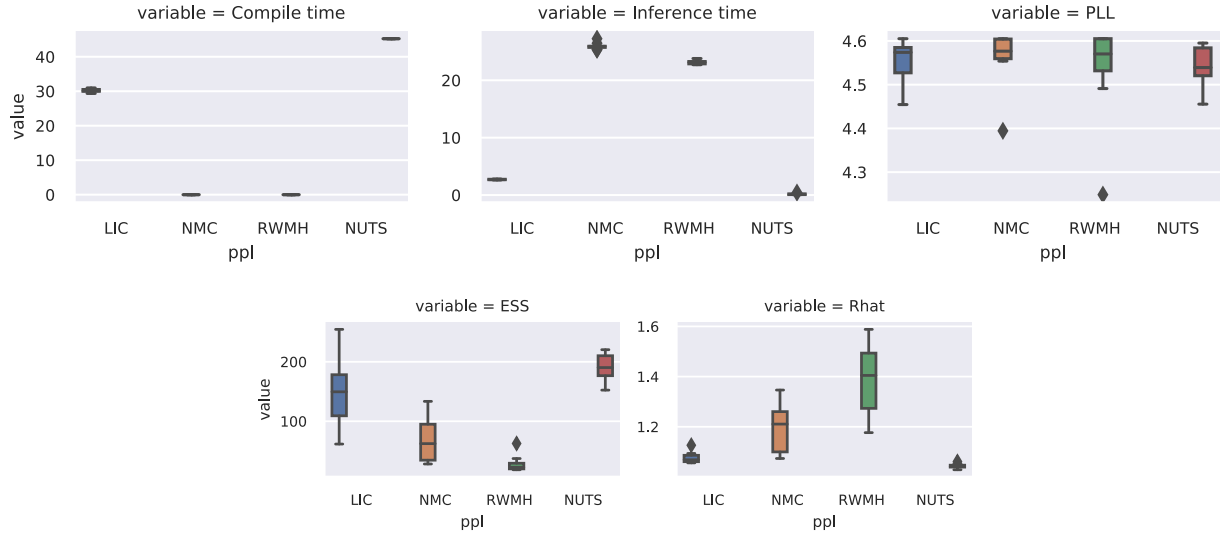
$$
\begin{aligned}
\beta_0 &\sim \text{StudentT}(3, 0, 10) \\
\tau_i &\sim \text{HalfCauchy}(\sigma_i) \qquad \text{for } i \in [\text{district}, \text{state}, \text{type}] \\
\beta_{i,j} &\sim \mathcal{N}(0, \tau_i) \qquad \text{for } i \in [\text{district}, \text{state}, \text{type}], j \in [n_i] \\
y_k &\sim \mathcal{N}(\beta_0 + \sum_i \beta_{i,j_k}, \sigma_k)
\end{aligned}
$$

The treatment effects y_k and standard errors σ_i and σ_k are observed.

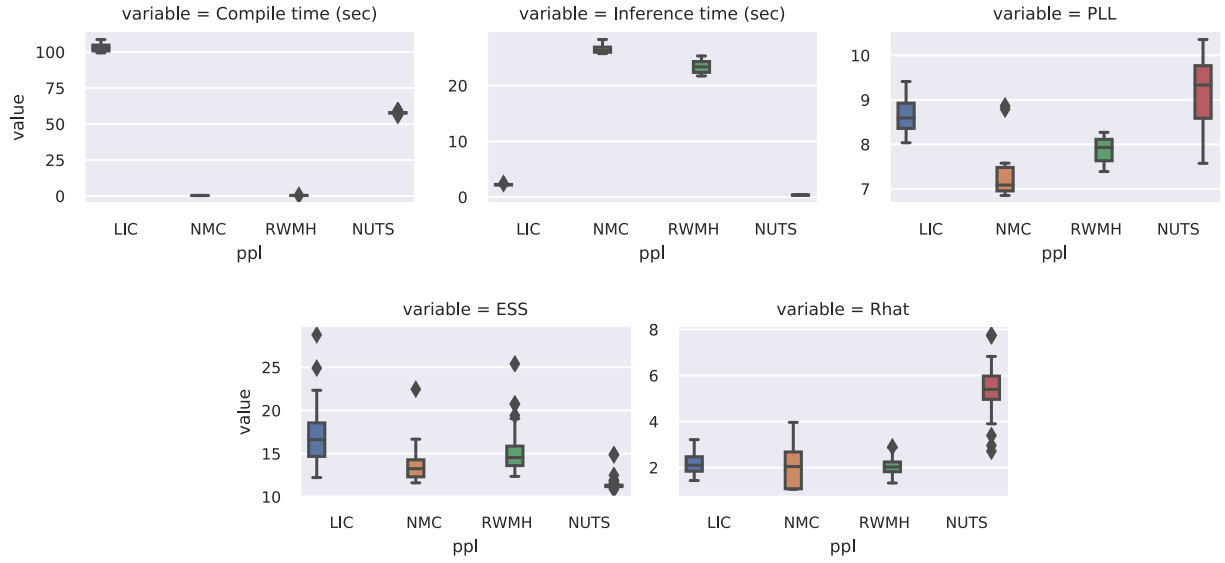
Intuitively, each "school" corresponds to a set of treatment parameters (here a tuple of district, state, and type) and $\beta_{i,j}$ measures the average change in response y when treatment

Figure 1.6: Results of MCMC on two Bayesian inference tasks where we compare the compilation time (neural network training for LIC and Stan's C++ codegen / compilation for NUTS), inference time, predictive log-likelihood (PLL) on hold-out data, expected sample size (ESS, higher is better, [**geyer2011introduction**]) and the rank normalized $\widehat{R}$ diagnostic (Rhat, closer to 1 is better, [**vehtari2020rank**]).

(a) Bayesian logistic regression (2000 rows, 10 features). Both LIC and Stan (NUTS) amortize the upfront compilation cost with accelerated inference times. LIC achieves comparable PLL to NUTS [**hoffman2014no**] and other methods, and yields ESS comparable to NUTS and higher than any other method. The $\widehat{R}$ of LIC is close to 1 (similar to NUTS and lower than all other methods).



(b) n-schools (1000 schools, 8 states, 5 districts, 5 types). Again, increased compilation times are offset by accelerated inference times. In this case, LIC achieves comparable PLL to NUTS while simultaneously producing higher ESS and lower $\widehat{R}$, suggesting that the resulting posterior samples are less autocorrelated and provide a more accurate posterior approximation.



parameter i is set equal to j (e.g. $\beta_{\text{state,CA}}$ measures the change in SAT scores when a school is located in California).

Figure 1.6b presents results in a format analogous to section 1.3. Here, we see that while both LIC and NUTS yield higher PLLs (with NUTS outperforming LIC in this case), LICs ESS is significantly higher than other compared methods. Additionally, the $\widehat{R}$ of NUTS is also larger than the other methods which suggests that even after 1,000 burn-in samples NUTS has still not properly mixed.

1.4 Conclusion

We introduced Lightweight Inference Compilation (LIC) for building high quality single-site proposal distributions to accelerate Metropolis-Hastings MCMC. LIC utilizes declarative probabilistic programming to retain graphical model structure and graph neural networks to approximate single-site Gibbs sampling distributions. To our knowledge, LIC is the first proposed method for inference compilation within an open-universe declarative probabilistic programming language and an open-source implementation will be released in early 2021. Compared to prior work, LIC's use of Markov blankets resolves the need for attention to handle nuisance random variances and yields posterior sampling comparable to state-of-the-art MCMC samplers such as NUTS and adaptive RWMH.