Bit Blasting Probabilistic Programs

POORVA GARG, University of California, Los Angeles, USA
STEVEN HOLTZEN, Northeastern University, USA
GUY VAN DEN BROECK, University of California, Los Angeles, USA
TODD MILLSTEIN, University of California, Los Angeles, USA

Probabilistic programming languages (PPLs) are an expressive means for creating and reasoning about probabilistic models. Unfortunately *hybrid* probabilistic programs that involve both continuous and discrete structures are not well supported by today's PPLs. In this paper we develop a new approximate inference algorithm for hybrid probabilistic programs that first discretizes the continuous distributions and then performs discrete inference on the resulting program. The key novelty is a form of discretization that we call *bit blasting*, which uses a binary representation of numbers such that a domain of 2^b discretized points can be succinctly represented as a discrete probabilistic program over $poly(b)$ Boolean random variables. Surprisingly, we prove that many common continuous distributions can be bit blasted in a manner that incurs no loss of accuracy over an explicit discretization and supports efficient probabilistic inference. We have built a probabilistic programming system for hybrid programs called `HyBit`, which employs bit blasting followed by discrete probabilistic inference. We empirically demonstrate the benefits of our approach over existing sampling-based and symbolic inference approaches

CCS Concepts: • **Mathematics of computing → Probabilistic representations**; **Probabilistic inference problems**.

Additional Key Words and Phrases: discretization, bit blasting, probabilistic inference

ACM Reference Format:
Poorva Garg, Steven Holtzen, Guy Van den Broeck, and Todd Millstein. 2024. Bit Blasting Probabilistic Programs. *Proc. ACM Program. Lang.* 8, PLDI, Article 182 (June 2024), 24 pages. https://doi.org/10.1145/3656412

1 INTRODUCTION

Probabilistic programming languages (PPLs) are an expressive means for creating and reasoning about probabilistic models. Many such models are naturally *hybrid*, involving both continuous (e.g., Gaussian distributions) and discrete structures (e.g., Bernoulli random variables, `if` statements and other control flow). For example, hybrid models arise in applications such as medical diagnosis, gene expression and cyber-physical systems [Chen et al. 2020; Lee and Seshia 2017].

Unfortunately, hybrid programs are not well supported by today's PPLs. The primary analysis task in probabilistic programming languages is *probabilistic inference*, computing the probability that an event occurs according to the distribution defined by the program. Existing inference algorithms employ forms of sampling to perform approximate inference. Some approaches, notably Hamiltonian Monte Carlo, used in the PPLs Pyro and Stan [Bingham et al. 2019; Gorinova et al. 2021], do not support discrete random variables, instead requiring them to be (manually or automatically) marginalized out. However, this approach has numerous fatal cases that explode exponentially

Authors' addresses: Poorva Garg, University of California, Los Angeles, USA, poorvagarg@cs.ucla.edu; Steven Holtzen, Northeastern University, Boston, USA, s.holtzen@northeastern.edu; Guy Van den Broeck, University of California, Los Angeles, USA, guyvdb@cs.ucla.edu; Todd Millstein, University of California, Los Angeles, USA, todd@cs.ucla.edu.

ACM 2475-1421/2024/6-ART182
https://doi.org/10.1145/3656412

Table 1. Distributions with sound and efficient bit blasting subsumed by mixed-gamma densities.

Distribution	Density	Distribution	Density
Uniform	1	Gamma	$\frac{\beta^\alpha x^{\alpha-1} e^{-\beta x}}{\Gamma(\alpha)}$
Linear	x	Laplace	$\frac{1}{2b} e^{\frac{-\lvert x-\mu\rvert}{b}}$
Polynomial	x^n	Chi-squared	$\frac{1}{2^{\frac{k}{2}}\Gamma(\frac{k}{2})} x^{\frac{k}{2}-1} e^{\frac{-x}{2}}$
Exponential	$\lambda e^{-\lambda x}$	Student-T	$c(1+\frac{x^2}{\nu})^{-\frac{\nu+1}{2}}$

in the number of discrete variables.[1] Other sampling-based approaches are universal and so can handle discreteness, such as importance sampling, Markov Chain Monte Carlo and Sequential Monte Carlo, etc. [Koller and Friedman 2009]. However, these algorithms are known to struggle with multimodal distributions [Yao et al. 2021], which arise through discreteness, as well as with programs that condition on low-probability events.

In this paper we develop a new inference algorithm for hybrid probabilistic programs via *discretization*: we convert the continuous distributions in a hybrid program to discrete distributions. This yields a fully discrete probabilistic program on which existing algorithms for discrete inference can be used. Discretization approximates a continuous distribution as a sequence of *intervals*, with each interval associated with the probability of the value falling in that interval. Forms of discretization have been used in prior work [Albarghouthi et al. 2017; Beutner et al. 2022; Claret et al. 2013; Huang et al. 2021] but they all scale linearly in the number of intervals. This imposes a clear tradeoff: one needs many small intervals in order to avoid losing too much precision, but the cost of inference quickly becomes prohibitive as the number of intervals grows.

We introduce a new approach to discretization that we call *bit blasting*, by analogy with the technique of the same name in verification [Bruttomesso and Sharygina 2009]. The key property of a bit blasted discretization is that it uses only $poly(\log n)$ Boolean random variables to represent a discretization on n intervals. This is achieved by employing a binary representation of numbers and representing discretizations as discrete probabilistic programs over this binary representation. At first blush, this succinct representation would appear to lose too much accuracy to be a viable strategy, but we present both theoretical and empirical results to the contrary.

First, we prove that a large class of common continuous densities can be bit blasted *soundly*, that is with no loss of accuracy versus a naïve discretization. Table 1 lists example distributions that are in this class; we refer to the entire class as *mixed-gamma* distributions.

For instance, consider discretizing a continuous uniform distribution between 0 and 1. Naïve discretization to 2^{32} intervals requires enumeration of 2^{32} values. Instead, this distribution can be represented in binary as a tuple of 32 Bernoulli random variables of the form $\texttt{flip}(0.5)$, i.e., coin tosses that are equally likely to have the value 0 or 1. This observation is not new, and a similar result holds for exponential distributions as well [Marsaglia 1971]. However, the bit blasted discretizations of other mixed-gamma densities are novel. Further, unlike the case for uniforms and exponentials, these discretizations are not defined as simply a tuple of independent Bernoulli random variables but rather require full-fledged discrete probabilistic programs over such variables.

A succinct representation does not necessarily imply efficient inference, which is hard in general. As our second contribution, however, we prove that bit blasted mixed-gamma distributions are not only sound and succinct, but they also support polynomial-time inference in the number of bits of precision. Specifically, we prove that the *knowledge compilation* approach to discrete probabilistic

[1]Indeed, the Pyro documentation states that it cannot support more than 25 discrete variables in CUDA and 64 discrete variables on a CPU [Bingham et al. 2018].

inference [Chavira and Darwiche 2008; Chavira et al. 2006; De Raedt et al. 2007; Fierens et al. 2015; Holtzen et al. 2020], which reduces inference to weighted model counting on a boolean formula, has this property for bit blasted mixed-gamma distributions. Therefore, the process of bit blasting a mixed-gamma distribution followed by discrete inference via knowledge compilation is guaranteed to take time that is polynomial in the bitwidth used for discretization.

Third, we have used the above theoretical results to design a new probabilistic programming system called `HyBit` that performs non-stochastic approximate inference for hybrid probabilistic programs via bit blasting. `HyBit` is a discrete probabilistic language that includes support for fixed-point binary numbers and associated arithmetic operations, and it is implemented as an embedded domain specific language in Julia [Bezanson et al. 2017]. The `HyBit` API allows users to produce bit blasted fixed-point approximations of arbitrary continuous distributions. Mixed-gamma distributions can be represented by their sound bit blasted discrete distributions. For other distributions the API enables users to employ piece-wise discrete approximations, where each piece is itself a bit blasted mixed-gamma distribution.

`HyBit` leverages knowledge compilation to perform exact inference on the given discrete probabilistic program. In the worst case, the inference for an arbitrary hybrid probabilistic program via bit blasting can be exponential in the number of bits, but we empirically demonstrate the benefits of our approach. We show that `HyBit` performs better than existing sampling-based and symbolic inference approaches on a comprehensive benchmark suite of hybrid probabilistic programs.

Overall, we present the following contributions in this paper:

(1) We motivate the challenges for inference on hybrid probabilistic programs in Section 2.
(2) We present a new form of discretization called bit blasting that is characterized by its succinctness in the number of discrete intervals in Section 3.
(3) We present a class of continuous distributions, namely mixed-gamma distributions, for which a sound bit blasted representation exists. We formalize this construction and prove its properties in Section 3. We further prove that knowledge compilation based inference scales polynomially in the bitwidth for these distributions.
(4) We describe the `HyBit` PPL and its new inference algorithm via bit blasting in Section 4.
(5) In Section 5, we empirically compare `HyBit` with other PPLs on benchmarks obtained from the existing literature. We also characterize the behavior of `HyBit` with respect to its hyperparameters, i.e. number of bits and pieces.

HyBit is available at https://github.com/Tractables/Dice.jl/tree/hybit. The full version of this paper is available on arXiv as [Garg et al. 2023], which contains full proofs.

2 MOTIVATING EXAMPLES

This section motivates the challenge of inference for hybrid probabilistic programs using three examples. First, we present an example from computational biology with inherent logical structure. Next, we show an example from the literature with a multimodal posterior arising due to discrete control flow. Finally, we show an example of low probability observations through conjugate Gaussians. We then investigate the performance of various inference algorithms, including `HyBit`. The examples demonstrate the advantages of `HyBit` over other approximate inference algorithms. We provide a detailed comparison with several approximate and exact baselines in Section 5.

2.1 Logical Structure

We present a simplified example from computational biology which relates genetic expression with blood sugar levels. Figure 1 shows the probabilistic program, where the task is to get the updated belief of a gene's occurrence in a patient given their blood sugar levels.

The first four lines of the probabilistic program in Figure 1 use a beta distribution as the prior probability of each of T genes occurring in the general population. The syntax `flip`(θ) denotes a Bernoulli random variable with success probability θ. On line 5, the program uses the syntax `reduce(|, gene)` to denote the expression $\bigvee_{i=1}^{T}$ `gene[i]`. In other words, the patient is considered to have diabetes if at least one of the genes is expressed. What follows is multiple readings of the patient's blood sugar level. For each reading a random variable first defines the blood sugar depending on whether they have diabetes. Then we use the syntax `observe` (y, v) to condition on the random variable y having the value v — in the program this is used to condition on actual blood-sugar readings from the patient. Finally, on line 12, the program queries for the expectation of the posterior distribution of the occurrence of the first gene.

```
1  for i in 1:T
2      gene_occurrence[i] = beta(1, 1)
3      gene[i] = flip(gene_occurrence[i])
4  end
5  diabetes = reduce(|, gene)
6  blood_sugar1 = if diabetes normal(80, 2)
7                 else normal(135, 2) end
8  observe(blood_sugar1, 79)
9  blood_sugar2 = if diabetes normal(80, 2)
10                else normal(135, 2) end
11 observe(blood_sugar2, 136)
12 return Expectation(gene_occurrence[1])
```

Figure 1. Example 1: Gene Expression

Figure 2 shows the results of inference with a timeout of 20 minutes on this program using different inference algorithms, as the number of genes (T) increases. Stan uses Hamiltonian Monte Carlo, which does not directly support discrete random variables. Instead, they are marginalized out, either manually or automatically using variable elimination [Gorinova et al. 2020]. As shown in the figure, Stan times out when there are more than 15 genes — as its strategy scales exponentially in T. The same issue of exponential blowup plagues GuBPI [Beutner et al. 2022], which employs a combination of symbolic evaluation and discretization to find upper and lower bounds. As the figure shows, the universal sampling methods MCMC with a Metropolis Hastings kernel (WebPPL MH) and Sequential Monte Carlo (WebPPL SMC) can scale and provide reasonable accuracy. Exact inference strategy Psi [Gehr et al. 2016] also scales well with the number of discrete random variables when the program is written to avoid a large discrete state space. AQUA discretizes hybrid programs [Huang et al. 2021] but does not support this program.

Our system and approach, `HyBit`, scales to 50 genes and has the least absolute error among approximate inference algorithms. The program, when written in `HyBit`, is represented by its discrete bit-level abstraction. The user, while writing the program can employ the `HyBit` API to replace all continuous distributions (specifically on Lines 2, 6, 9 in Figure 1) with their bit blasted discrete approximations. As a result, we now have a discrete program with distributions over Boolean and fixed-point numbers. Note that now `observe` on Line 8 conditions on the fact that the value of the discrete distribution for `blood_sugar1` lies in the interval corresponding to the value 79.

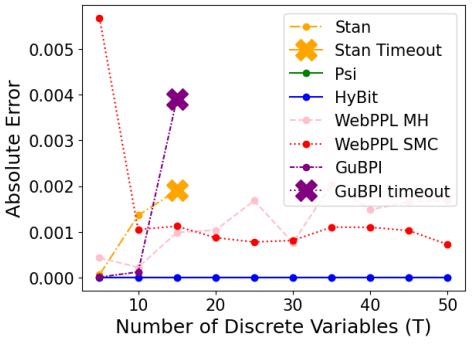


Figure 2. Scaling on Logical Constraints. `HyBit` scales to 50 genes with the least absolute error. The graph for Psi and HyBit overlap closely.

In the experiment shown we use a bitwidth of 25 — each continuous distribution is discretized as a

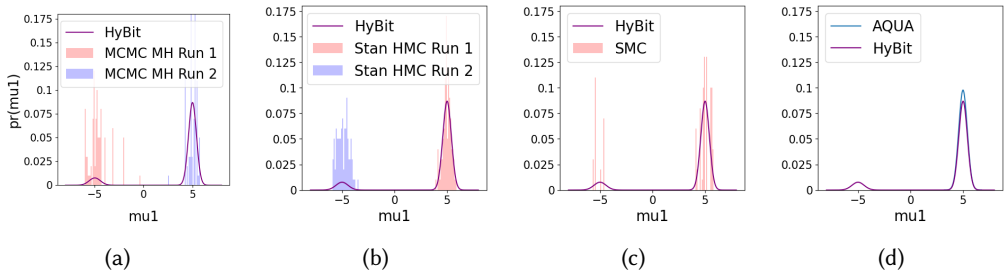


Figure 4. Posterior from different baselines compared with that of `HyBit` for Figure 3. `HyBit` and WebPPL SMC are able to identify both the modes in the posterior distribution. (a) Different runs of WebPPL MCMC with a Metropolis Hastings kernel converge to different modes. (b) Different runs of Stan HMC converge to different modes. (c) Different runs of WebPPL SMC are able to find both the modes. (d) AQUA with its adaptive interval strategy only finds the more probable mode

program over a tuple of 25 bits interpreted as a fixed-point number. Further, we approximate these continuous distribution using 4096 pieces, each of which is a bit blasted exponential distribution. Naive discretization with 25 bits would be prohibitively slow, as it yields 2^{25} intervals i.e. 134M intervals. However, our bit blasted program only uses 53K coin flips (Boolean random variables) to represent them. Moreover, knowledge compilation based inference [Fierens et al. 2015; Holtzen et al. 2020] automatically identifies and exploits conditional independences in the program's logical structure and helps to scale inference. More details of this experiment can be found in the appendix.

2.2 Handling Multimodality

This section presents an example of a multimodal distribution to highlight another challenge for inference on hybrid probabilistic programs. Multimodal distributions have multiple peaks separated by low probability regions. These distributions commonly emerge in various applications such as sensor network localization, cosmology and many more [Shaw et al. 2007; Tak et al. 2018]. We adapt an example from the existing literature [Yao et al. 2021], as shown in Figure 3.

The probabilistic program shown in Figure 3 is hard for existing probabilistic inference approaches. The `datapts` on Line 3 has nine entries, with two-thirds being 5 and the other one-third being -5. This leads to the posterior for (μ_1, μ_2) being bimodal around $(5, -5)$ and $(-5, 5)$. As the number of data points increases with the same proportion, the posterior of (μ_1, μ_2) converges to $(5, -5)$. However, in the presence of limited data points, the posterior for μ_1 is bimodal around 5 and -5.

```
1 mu1 = uniform(-16, 16)
2 mu2 = uniform(-16, 16)
3 datapts = [5, 5, 5, 5, 5, 5, -5, -5, -5]
4 for data in datapts
5    y = if flip(2/3) normal(mu1, 1)
6         else normal(mu2, 1) end
7    observe(y, data)
8 end
9 return mu1
```

Figure 3. Example 2: Yao-Vehtari-Gelman model

The existence of multiple modes challenges sampling-based algorithms as they tend to get stuck in one of the modes. Specifically, WebPPL using MCMC with a Metropolis Hastings kernel and both Stan and WebPPL using HMC end up arbitrarily in one of the modes and fail to explore the other mode. Figures 4a and 4b show the results obtained using WebPPL MCMC and Stan HMC respectively, where two different runs get stuck in two different modes. On the other

hand, HyBit performs exact inference on its discrete abstraction and so explores the distribution globally, allowing it to identify both modes. Sequential Monte Carlo (SMC) (Figure 4c), Psi and GuBPI also explore the distribution globally and thus, are able to find both modes. Finally, to address the computational challenge of direct discretization, AQUA adapts its discretizing intervals to focus on high probability regions, and ends up only identifying the higher probability mode (Figure 4d).

2.3 Handling Low Probability Observations

This section presents conjugate Gaussians (Figure 5) with low probability observations. In Figure 5, the posterior distribution for random variable mu is queried after conditioning on low probability data on Lines 2 and 3.

```
1  mu = normal(0, 1)
2  observe(normal(mu, 1), 8)
3  observe(normal(mu, 1), 9)
4  return mu
```

Figure 5. Example 3: Conjugate Gaussians

Why are low probability observations hard? Intuitively, general-purpose sampling algorithms begin sampling from the prior distribution and struggle to find samples with considerable weight. Only after a very large number of samples do these algorithms manage to sample from the true posterior. On the other hand, HyBit explores the domain of the posterior distribution globally, via exact inference on a bit blasted abstraction, and so it is insensitive to this issue.

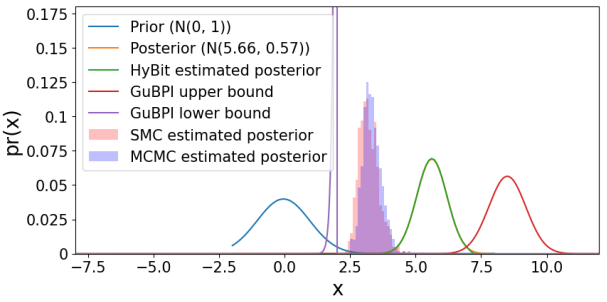


Figure 6. Results for Example 3. The HyBit estimated posterior overlaps closely with the true posterior distribution.

Figure 6 plots the true prior and posterior distributions along with results from different inference algorithms. For the sampling-based algorithms MCMC with a Metropolis Hastings kernel and SMC, we obtained and plotted 1000 samples after running the corresponding WebPPL [Goodman and Stuhlmüller 2014] program. The importance sampling algorithm was not able to obtain any sample with non-negligible weight for this program. The samplers are shifting the posterior towards the true posterior distribution but require many more samples to achieve that. Even after sampling about 16M and 65K samples respectively, the expectation of the samples obtained from MCMC and SMC have absolute error of 0.549798 and 1.520776. GuBPI reports upper and lower bounds on the probability for each interval and incurs an absolute error of 2.33. On the other hand, the posterior distribution from HyBit overlaps perfectly with the ground truth. Stan HMC handles low probability observations well and obtains high accuracy. Psi also obtains the exact symbolic expression for the posterior distribution. Finally, the mean of AQUA's reported posterior had an error of 5.66 as it fails to make any update to the prior of μ.

3 BIT BLASTING: KEY INSIGHTS

To scale inference on hybrid probabilistic programs with respect to discrete structure, we need an algorithm that treats discreteness as first class, and that discretizes away continuous structure. This section defines the semantic notion of bit blasting and sets it up as a special case of discretization with desirable properties. Then, we provide bit blasting functions for common classes of continuous

distributions. We provide discretization techniques that are sound (accurate up to b bits), succinct, and amenable to efficient inference.

3.1 Discretization and Bit Blasting

In the standard terminology of probability theory [Rosenthal 2006], a probability space (Ω, Σ, μ) consists of a sample space Ω, a σ-algebra on Ω denoted Σ, and a probability measure on Σ denoted μ. In a general sense, a discretization function takes as input such a probability space (Ω, Σ, μ) and outputs a discrete probability space $(\Omega_D, \Sigma_D, \mu_D)$ where Ω_D is a countable set.

We will study a more specific notion of discretization: one that takes as input a continuous distribution over a finite interval and outputs a discrete distribution over 2^b points for some number of bits b. Formally, let $[l, u)$ be an interval with $l, u \in \mathrm{R}$. For the input probability space, we use $\mathcal{B}([l, u))$ to denote the Borel σ-algebra of subsets of interval $[l, u)$ [Rosenthal 2006]. For the output probability space, we write $\mathcal{P}(S)$ to refer to the power set (a σ-algebra) of set S. Moreover, we will assume the sample space to be discretized as follows.

Definition 1 (b-bit interval). *A b-bit interval $[l, u]_b$ is the set of points obtained by dividing $[l, u)$ into 2^b intervals: $[l, u]_b = \{r \mid r2^b \in \mathrm{Z}, r \in [l, u)\}$*

We are now ready to define the notion of discretization function used in this paper.

Definition 2 (b-bit discretization function). *A b-bit discretization function takes as input a probability space $([l, u), \mathcal{B}([l, u)), \mu)$, a bit width $b \in \mathrm{Z}^+$ and outputs a discrete probability space $([l, u]_b, \mathcal{P}([l, u]_b), \mu_D)$ for some measure μ_D.*

Example 1. *Let π be a uniform distribution on the unit interval, described by the probability space $([0, 1), \mathcal{B}([0, 1)), \mu)$ such that the probability density function specified by μ is 1 on the unit interval. Consider a function d_1 that takes π as input and outputs the probability space $(S, \mathcal{P}(S), \mu_D)$ where $S = \{0, 0.25, 0.5, 0.75\}$ and μ_D is a probability measure on $\mathcal{P}(S)$. Then d_1 is a 2-bit discretization function. As a concrete example μ_D can be defined as $\{0 \to 0.1, 0.25 \to 0.2, 0.5 \to 0.3, 0.75 \to 0.4\}$[2].*

As the example shows, one can come up with any arbitrary b-bit discretization function. To qualify them further, we need a notion of accuracy — soundness up to b bits defined as follows.

Definition 3 (soundness of b-bit discretization function). *For any integer $b > 0$, a b-bit discretization function is b-sound for a particular input probability space $([l, u), \mathcal{B}([l, u)), \mu)$ if it outputs a discrete probability space $([l, u)_b, \mathcal{P}([l, u]_b), \mu_D)$ such that the following holds:*

$$\forall x \in [l, u]_b \qquad \int_x^{x+\frac{1}{2^b}} d\mu(y) = \mu_D(\{x\})$$

Example 2. *Let d_2 be a b-bit discretization function that takes π (as defined in Example 1) as input and outputs the probability space $(S, \mathcal{P}(S), \widetilde{\mu_D})$ where $S = \{0, 0.25, 0.5, 0.75\}$ and $\widetilde{\mu_D}$ can be defined as $\{0 \to 0.25, 0.25 \to 0.25, 0.5 \to 0.25, 0.75 \to 0.25\}$. Then d_2 is a sound 2-bit discretization function for the probability space π.*

Before we define a b-bit blasting function, we need to fix a generic representation of discrete probability distributions. To that purpose, we define the concept of a discrete probabilistic closure, akin to probabilistic Turing machines [Arora and Barak 2006].

Each probabilistic closure is a deterministic function from a set of biased coin flips to a discrete set. This induces a probability distribution on the output of the function through probabilities

[2]We define discrete probability measures μ_D using a mapping from single elements in the sample space S to their probabilities, which can then be used to compute μ_D for any set in $\mathcal{P}(S)$.

associated with coin flips. It also consists of an accepting Boolean formula that handles observations and limits the set of values that input flips can take. We provide a formal definition below:

Definition 4 (discrete probabilistic closure). *A discrete probabilistic closure is defined as the tuple (φ, γ, w) where w is a vector of Boolean random variables (or biased coin flips), φ is a deterministic function from $\{\mathsf{T}, \mathsf{F}\}^{|w|}$ to a finitely countable set S and γ is a deterministic Boolean formula over variables in w.*

The semantics of a discrete probabilistic closure, i.e. $\langle(\varphi, \gamma, w)\rangle$ defines a probability space $(S, \mathcal{P}(S), \mu)$ such that

$$\forall T \subseteq S, \mu(T) = \frac{\mathrm{E}_w((\varphi(w) \in T) \wedge \gamma)}{\mathrm{E}_w(\gamma)} \qquad \text{if } \mathrm{E}_w(\gamma) \neq 0$$

Example 3. *From Example 2, consider $\widetilde{\mu_D} = \{0 \to 0.25, 0.25 \to 0.25, 0.5 \to 0.25, 0.75 \to 0.25\}$. $\widetilde{\mu_D}$ can be represented using a discrete probabilistic closure using the function* `naïve_unif`*, the weight function w and the accepting Boolean formula γ as follows:*

$$\gamma = \mathsf{T} \qquad w = [f_0 \to \texttt{flip}\ \frac{1}{4}, f_1 \to \texttt{flip}\ \frac{1}{3}, f_2 \to \texttt{flip}\ \frac{1}{2}]$$

Now, one can calculate the probability that `naïve_unif` *returns 0.5 as follows:*

$$Pr(\texttt{val} = 0.5) = Pr(\neg f_0)Pr(\neg f_1)Pr(f_2) = (1 - 0.25)(1 - 0.33)0.5 = 0.25$$

```
1 function naïve_unif(f0, f1, f2)
2   val = if f0 then 0
3   else if f1 then 0.25
4   else if f2 then 0.5
5   else 0.75
6   return val
```

Note that as explained by Example 3, any discrete distribution μ_D over 2^b values can be represented as a discrete probabilistic closure (φ, γ, w) where $|w| = 2^b - 1$.

Dice [Holtzen et al. 2020] and Problog [Fierens et al. 2015] are examples of PPLs that directly fit into the paradigm of a discrete probabilistic closure.

We want a discrete probabilistic closure to be more succinct — of size polynomial in the number of bits of precision, b. To this purpose, we define a bit blasting function.

Definition 5 (b-bit blasting function). *A b-bit blasting function $[.]_b$ is a b-bit discretization function that outputs a discrete probabilistic closure (φ, γ, w) that uses a number of Boolean random variables that is polynomial in the number of bits b, that is, $|w| \in O(poly(b))$*

It follows from Definitions 3, 4 and 5 that for an integer $b > 0$, a b-bit blasting function is sound for a given probability space $([l, u), \mathcal{B}([l, u)), \mu)$ if

$$\forall x \in [l, u]_b \qquad \int_x^{x + \frac{1}{2^b}} d\mu(y) = \langle [\mu]_b \rangle,$$

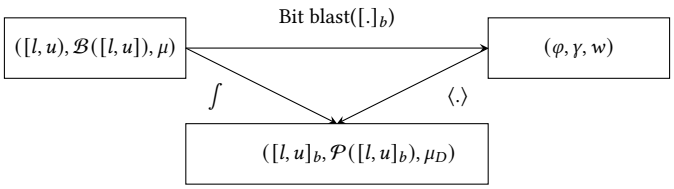


Figure 7. Commutative diagram for a b-sound bit blasting function

Example 4. *Again, consider the result of the sound 2-bit discretization function from Example 2, that is* $\widetilde{\mu_D} = \{0 \to 0.25, 0.25 \to 0.25, 0.5 \to 0.25, 0.75 \to 0.25\}$. *Measure* $\widetilde{\mu_D}$ *can alternatively be represented using a discrete probabilistic closure using the function* `bitblast_unif`, *the weight function* w *and the accepting Boolean formula* γ *as follows:*

$$\gamma = \mathsf{T} \qquad\qquad w = [f_0 \to \texttt{flip}\ \frac{1}{2}, f_1 \to \texttt{flip}\ \frac{1}{2}]$$

Note that in `bitblast_unif`, f_0 *and* f_1 *are being used as bits in the binary representation of the variable* `val`. *Now, one can calculate the probability that* `bitblast_unif` *returns 0.5 as follows:*

$$Pr(\texttt{val} = 0.5) = Pr(f_0)Pr(\neg f_1) = (0.5)(1 - 0.5) = 0.25$$

We would like to point out that `bitblast_unif` uses only 2 coin flips to represent a distribution over 4 values. It can be generalized to use b coin flips to represent a uniform distribution over 2^b values. On the other hand, `naïve_unif` uses 3 coin flips and would generalize to use $2^b - 1$ coin flips. Hence, `bitblast_unif` is a viable output for a b-bit blasting function. In the next section, we provide details for the b-bit blasting functions for exponential and mixed-gamma distributions.

```
1 function bitblast_unif(f0, f1)
2   bit0 = if f0 then 1 else 0
3   bit1 = if f1 then 1 else 0
4   val = bit0/2 + bit1/4
5   return val
```

3.2 Concrete Bit Blasting Function: Preliminaries

Next, our goal is to provide a concrete instantiation of a sound bit blasting function for mixed gamma distributions, which have probability density functions as defined below.

Definition 6 (generalized-gamma density). *Given parameters* $\alpha \in \mathrm{Z}^+$ *and* $\beta \in \mathrm{R}$, *a generalized-gamma density* $\pi_{\alpha,\beta}$ *is a probability density function over the interval* $[0, 1)$ *of the form*

$$\pi_{\alpha,\beta}(x) = \frac{x^\alpha e^{\beta x}}{\int_{[0,1)} y^\alpha e^{\beta y}\, dy}.$$

Definition 7 (mixed-gamma density). *Given a collection of* $N \in \mathrm{Z}^+$ *generalized-gamma densities* π_{α_i,β_i} *with their associated weights* $a_i \in [0, 1]$ *such that* $\sum_{i=1}^{N} a_i = 1$, *a mixed-gamma density* Υ *is a probability density function over the interval* $[0, 1)$ *of the form*

$$\Upsilon(x) = \sum_{i=1}^{N} a_i \pi_{\alpha_i,\beta_i}(x).$$

For notational convenience, we confine the continuous distributions to the unit interval to get discrete distributions over a b-bit unit interval. We generalize our approach to any finite interval for building the probabilistic programming system `HyBit` based on bit blasting.

To describe our construction of the bit blasting function, we make use of Dice [Holtzen et al. 2020]. Dice already compiles its programs to weighted Boolean formulas (via the $\rightsquigarrow$ judgement) that fit the definition of a discrete probabilistic closure.[3] This allows us to only define a $\rightsquigarrow_b$ judgment from probability density functions to Dice programs to specify a bit blasting function. Dice also defines a distributional semantics function $[\![.]\!]_D : \mathsf{p} \to V \to [0, 1]$ that takes as input a Dice program p and outputs a normalized probability distribution (described as a function from the set of values V to probabilities). We use the function $[\![.]\!]_D$ to argue about soundness of our construction later. More details about syntax and semantics of Dice can be found in the appendix.

[3]Dice compiles to weighted Boolean formulas (φ, γ, w) where φ outputs (tuples of) Boolean values, γ represents observations in a Dice program and w consists of weights associated with Boolean variables (biased coin flips)

The compilation judgement for mixed-gamma densities are of the form $\Upsilon \rightsquigarrow_b \mathsf{p}$ where p are Dice programs. We further provide the following definitions for b-equivalence between densities and Dice programs and b-succinctness of Dice programs.

Definition 8 (binarizing function). *The binarizing function for b bits, $(\!|.|\!)_b$ takes as input a number $r \in [0,1]_b$ and returns a b-bit tuple $(v_1, (v_2, (\ldots v_b)\ldots))$ such that $r = \sum_{i=1}^{b} \frac{v_i}{2^i}$*

Definition 9 (b-equivalence). *A mixed-gamma density Υ and a Dice program p are b-equivalent for some $b \in \mathbb{Z}^+$ if for all $r \in [0,1]_b$*

$$\int_r^{r+\frac{1}{2^b}} \Upsilon(y)\, dy = [\![\mathsf{p}]\!]_D\, ((\!|r|\!)_b)$$

Note that b-equivalence is analogous to b-soundness but is specialized for Dice programs.

Definition 10. *The flip count function* `flip_count` $(.)$ *takes as input a Dice program* p *and outputs the number of Boolean random variables (coin flips).*

Definition 11. *Compilation $\rightsquigarrow_b$ is b-succinct for Υ if $\exists k > 0, \forall b \in \mathbb{Z}^+$ such that if $\Upsilon \rightsquigarrow_b \mathsf{p}$, then* `flip_count` $(\mathsf{p}) \leq kb$

We define b-succinctness such that the Dice program p employs coin flips linear in the number of bits b. Observe that b-succinctness imposes a stricter condition than that required by a b-bit blasting function (which requires $poly(b)$ coin flips). This implies that if we have a b-succinct judgment for a mixed-gamma density, then we can have a b-bit blasting function for that distribution. We describe this in more detail later.

3.3 Judgment $\rightsquigarrow_b$ and bit blasting

This section describes the rules for the judgement $\rightsquigarrow_b$. We first describe the rules for an exponential distribution and then move on to generalized gamma distributions. Finally, we describe the rule for mixed-gamma densities. For each of the rules $\Upsilon \rightsquigarrow_b \mathsf{p}$, we prove the b-equivalence of the density Υ and the Dice program p and the b-succinctness of p. Detailed proofs can be found in the appendix. We also describe how $\rightsquigarrow_b$ allows us to construct a bit blasting function for mixed-gamma densities.

3.3.1 Exponential Distribution, $\pi_{0,\beta}$. Let us first consider the uniform distribution ($\pi_{0,0}$), a special case of an exponential distribution. If we bit blast a uniform distribution using b bits into 2^b intervals, we end up with a discrete distribution D_b over $[0,1]_b$ with 2^b discrete points each having probability $\frac{1}{2^b}$. A straightforward discretization strategy enumerates 2^b values using $2^b - 1$ coin flips. But the same can be achieved using a tuple of b bits, where each bit is an unbiased coin `flip(0.5)`.

The strategy to bit blast a uniform distribution using its binary representation works because of the independence between binary digits. The same strategy can be extended to general exponential distributions as well. This fact was shown in a classic paper in the statistics literature [Marsaglia 1971]. We formalize that idea using the following rule.

Definition 12. *The function* `flip_param`: $\mathbb{R} \times \mathbb{Z}^+ \to [0,1]$ *is defined* `flip_param`$(\beta, b) = \frac{e^{\frac{\beta}{2^b}}}{1+e^{\frac{\beta}{2^b}}}$

$$\frac{\texttt{flip\_param}(\beta,1)=\theta_1 \quad \texttt{flip\_param}(\beta,2)=\theta_2 \quad \ldots \quad \texttt{flip\_param}(\beta,b)=\theta_b}{\pi_{0,\beta} \rightsquigarrow_b \begin{array}{l} \texttt{fresh } y_1, y_2, \ldots y_b \\ \\ \texttt{let } y_1 = \texttt{flip}(\theta_1) \texttt{ in} \\ \texttt{let } y_2 = \texttt{flip}(\theta_2) \texttt{ in} \\ \ldots \\ \texttt{let } y_b = \texttt{flip}(\theta_b) \texttt{ in} \\ (y_1, (y_2, (\ldots, y_b)) \ldots) \end{array} \quad \text{(Expo0)}$$

We prove the b-equivalence of the density $\pi_{0,\beta}$ and the Dice program in rule Expo0 and b-succinctness of the latter in the following lemmas. It is more straightforward to see the succinctness of the Dice program — it uses only b coin flips.

Lemma 1. $\forall b \in Z^+, \beta \in R$, p, *if* $\pi_{0,\beta} \rightsquigarrow_b$ p, *then* $\pi_{0,\beta}$ *and* p *are b-equivalent.*

Lemma 2. $\forall \beta \in R, \rightsquigarrow_b$ *is b-succinct for* $\pi_{0,\beta}$

We provide detailed proofs of the above lemmas in the appendix. For rest of the paper, we consider exponential distributions as primitives to build other distributions, since these are the only ones that enjoy the property of independent bits. But, sound bit blasting functions are still possible for other distributions, as we show next.

3.3.2 Gamma Distribution $\pi_{1,\beta}$. To come up with a sound bit blasting function for $\pi_{1,\beta}$, we present a key mathematical insight. Consider the program in Figure 8a. Continuous random variables `X` and `Y` have a uniform ($\pi_{0,0}$) and exponential ($\pi_{0,\beta}$) distribution respectively. It returns the new distribution of `X` after conditioning on the inequality `Y < X`. It turns out that the posterior distribution is a specific gamma distribution $\pi_{1,\beta}$. We show the resulting calculation below, where *pdf* refers to the probability density function.

$$pdf(X|Y<X) \propto \int_{y=0}^{1} pdf(Y)\cdot pdf(X)\cdot \mathbb{1}(Y<X)\, dy = \int_{y=0}^{x} 1\cdot e^{\beta x}\, dy \propto x e^{\beta x} \tag{1}$$

What happens if we discretize the program in Figure 8a using b bits? We get the program in Figure 8b where each continuous random variable has been replaced with its bit blasted counterpart (`X` replace by X_b and so on). We have already seen that for uniform and exponential distributions, b-equivalent Dice programs exist. But what about the other constructs? As Figure 8d demonstrates, `observe (`Y_b `<` X_b`)` incurs error over its continuous counterpart `observe (X < Y)`. The good news is that we can account for the error as shown by the following equations:

$$\begin{aligned}\Pr(X_b \mid Y<X) &= \Pr(X_b, Y_b<X_b \mid Y<X) + \Pr(X_b, Y_b == X_b \mid Y<X) \\ &= \underbrace{\Pr(X_b \mid Y_b < X_b)}_{\text{Output of the discrete program}} \cdot \Pr(Y_b < X_b \mid Y<X) + \underbrace{\Pr(X_b \mid Y_b == X_b, Y<X)}_{\text{Correction} \propto [\pi_{0,\beta}]_b} \cdot \Pr(Y_b == X_b \mid Y<X).\end{aligned}$$

The correction term in the above equations when computed algebraically turns out to be proportional to the exponential density ($\pi_{0,\beta}$). So now, the resulting discrete probabilistic program after correction looks as shown in Figure 8c where $\theta = Pr(Y_b == X_b | Y < X)$.

```
1 Y ~ uniform(0, 1)
2 X ~ exponential(β)
3 observe(Y < X)
4 return X
```

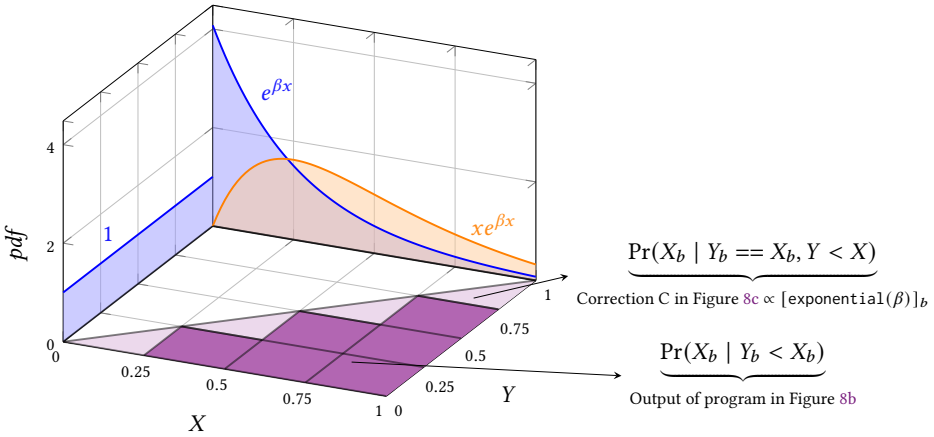
(a) Continuous probabilistic program for density $xe^{\beta x}$

```
1 Y_b ~ [uniform(0, 1)]_b
2 X_b ~ [exponential(β)]_b
3 observe(Y_b < X_b)
4 return X_b
```

(b) bit blasted probabilistic program analogous to program 8a

```
1 Y_b ~ [uniform(0, 1)]_b
2 X_b ~ [exponential(β)]_b
3 C ~ [exponential(β)]_b
4 observe(Y_b < X_b)
5 Z = if flip(θ) then C
      else X_b
6 return Z
```

(c) Sound bit blasted probabilistic program for density $xe^{\beta x}$



(d) Sound b-bit blasting of $xe^{\beta x}$. Prior densities for X and Y (shown in blue) when conditioned on $Y < X$ (shown in violet) returns the posterior for X (shown in orange).

Figure 8. Key insight in bit blasting $xe^{\beta x}$ probability density. $[.]_b$ refers to discretization of a continuous density into 2^b intervals and X_b refers to the discretization of X.

The rule Expo1 and Trans-expo1zero (in the appendix) captures the above intuition. Here, `unifObs(y, b) = p` is a helper judgment where p constructs a uniform distribution that it conditions through observe on being less than y.

$$\frac{\pi_{0,\beta} \rightsquigarrow_b \mathtt{p}_1 \qquad \mathtt{unifObs}(\mathtt{y}_1, \mathtt{b}) = \mathtt{p}_3 \qquad \begin{array}{c}\mathtt{fresh}\ \mathtt{y}_1, \mathtt{y}_2, \mathtt{y}_3 \qquad \beta \neq 0 \\ \theta = \frac{(e^{\beta\cdot 2^{-b}}(\beta\cdot 2^{-b}-1)+1)(1-e^{\beta})}{(1-e^{\beta\cdot 2^{-b}})(e^{\beta}(\beta-1)+1)}\end{array}}{\begin{array}{l}\mathtt{let}\ \mathtt{y}_1 = \mathtt{p}_1\ \mathtt{in} \\ \mathtt{let}\ \_ = \mathtt{p}_3\ \mathtt{in} \\ \mathtt{let}\ \mathtt{y}_2 = \mathtt{p}_1\ \mathtt{in} \\ \mathtt{let}\ \mathtt{y}_3 = \mathtt{flip}(\theta)\ \mathtt{in} \\ \mathtt{if}\ \mathtt{y}_3\ \mathtt{then}\ \mathtt{y}_2\ \mathtt{else}\ \mathtt{y}_1\end{array} \pi_{1,\beta} \rightsquigarrow_b} \quad \text{(Expo1)}$$

We prove the b-equivalence and b-succinctnesss. The proof for b-equivalence is much more involved but it is easy to see that the Dice program in the above rule uses $3b+1$ coin flips: b coin flips in each occurrence of $\mathtt{p}_1$, b coin flips in $\mathtt{p}_3$ and 1 coin flip in the `if then else` guard to create a mixture.

Lemma 3. $\forall b \in \mathbb{Z}^+, \beta \neq 0 \in \mathbb{R}, \mathtt{p}$, *if* $\pi_{1,\beta} \rightsquigarrow_b \mathtt{p}$, *then* $\pi_{1,\beta}$ *and* p *are* b*-equivalent.*

Lemma 4. $\forall \beta \in \mathbb{R}, \rightsquigarrow_b$ *is* b*-succinct for* $\pi_{1,\beta}$

3.3.3 Generalized Gamma Distribution $\pi_{\alpha,\beta}$. The previous subsection shows how conditioning on the inequality (`Y < X`) introduces a linear factor to $\pi_{0,\beta}$ to obtain $\pi_{1,\beta}$. Note that conditioning on (`Y < X`) introduces a linear factor regardless of what initial probability density X had. That is, if X's initial probability density was $f(x)$, the resulting density of the following probabilistic program would be $xf(x)$. This implies that if we can bit blast $f(x)$, we can bit blast $xf(x)$. We still need to account for the error incurred by `observe (Y_b < X_b)` i.e. $\Pr(X_b | Y_b == X_b, Y < X)$. It turns out that the correction term is a mixture of gamma distributions which can be bit blasted soundly as well. We provide the judgement rule and proof of the following lemma in the appendix.

```
1 Y ~ uniform(0, 1)
2 X ~ f(x)
3 observe(Y < X)
4 return X
```

Lemma 5. *$\forall b, \alpha \in \mathrm{Z}^+, \beta \in \mathrm{R}$, p, if $\pi_{\alpha,\beta} \rightsquigarrow_b \mathsf{p}$, then $\pi_{\alpha,\beta}$ and p are b-equivalent.*

Lemma 6. *$\forall \beta \in \mathrm{R}, \alpha \in \mathrm{Z}^+, \rightsquigarrow_b$ is b-succinct for $\pi_{\alpha,\beta}$*

3.3.4 Mixture of Gamma Distributions $\sum_i a_i \pi_{\alpha_i,\beta_i}$. Since generalized gamma densities $\pi_{\alpha,\beta}$ can be bit blasted, mixed gamma densities can be bit blasted as well. One bit blasts each individual generalized gamma density and creates their mixture using `if then else` constructs as follows.

$$\frac{N > 1 \qquad \pi_{\alpha_N,\beta_N} \rightsquigarrow_b \mathsf{p}_1 \qquad \text{fresh } \mathsf{y}_1, \mathsf{y}_2, \mathsf{y}_3 \qquad \sum_{i=1}^{N-1} \frac{a_i}{1-a_N} \pi_{\alpha_i,\beta_i} \rightsquigarrow_b \mathsf{p}_2 \qquad \forall i, a_i \in [0,1] \qquad \sum_{i=1}^{N} a_i = 1}{\sum_{i=1}^{N} a_i \pi_{\alpha_i,\beta_i} \rightsquigarrow_b \begin{array}{l} \texttt{let } \mathsf{y}_1 \texttt{ = flip}(a_N) \texttt{ in} \\ \texttt{let } \mathsf{y}_2 \texttt{ = } \mathsf{p}_1 \texttt{ in} \\ \texttt{let } \mathsf{y}_3 \texttt{ = } \mathsf{p}_2 \texttt{ in} \\ \texttt{if } \mathsf{y}_1 \texttt{ then } \mathsf{y}_2 \texttt{ else } \mathsf{y}_3 \end{array}} \quad \text{(Trans-mix)}$$

We prove the following theorems with details in the appendix.

Theorem 7. *$\forall \Upsilon, b \in \mathrm{Z}^+$, p, if $\Upsilon \rightsquigarrow_b \mathsf{p}$ then Υ and p are b-equivalent.*

Theorem 8. *$\rightsquigarrow_b$ is b-succinct for all mixed-gamma densities Υ*

Now that we have specified the rules for judgment $\rightsquigarrow_b$, we specifically define a sound bit blasting function for all mixed-gamma densities Υ, that is $\rightsquigarrow \circ \rightsquigarrow_b$.

Theorem 9. *$\rightsquigarrow \circ \rightsquigarrow_b$ is a sound b-bit blasting function*

Earlier work [Holtzen et al. 2020] defines the judgment $\rightsquigarrow$ that takes as input a Dice program p and outputs a weighted Boolean formula (φ, γ, w) that aligns with Definition 4 of a discrete probabilistic closure. And since $\rightsquigarrow_b$ is b-succinct for all mixed-gamma densities by Theorem 8, $\rightsquigarrow$ always outputs a w with $poly(b)$ coin flips. Thus, $\rightsquigarrow \circ \rightsquigarrow_b$ is a b-bit blasting function. Earlier work [Holtzen et al. 2020] also proves the correctness of compilation to weighted Boolean formula with respect to the semantics of the Dice program. This fact combined with Theorem 7 concludes that $\rightsquigarrow \circ \rightsquigarrow_b$ is a sound b-bit discretization function. Detailed proofs can be found in the appendix.

3.3.5 Example: Laplace Distribution. Previous sections described how mixed gamma densities can be bit blasted when they are confined to a unit interval. But how can distributions that are shifted or scaled to other finite intervals be bit blasted? We explain it through the example of a Laplace distribution. A Laplace distribution has two parameters: location (μ) and scale (b) and has the probability density function as described below where $x \in \mathrm{R}$.

$$Laplace(x|\mu, b) = \frac{1}{2b} e^{-\frac{|x-\mu|}{b}} = \begin{cases} \frac{1}{2b} e^{\frac{\mu}{b}} e^{-\frac{x}{b}} & x \geq \mu \\ \frac{1}{2b} e^{-\frac{\mu}{b}} e^{\frac{x}{b}} & x < \mu \end{cases}$$

Let us consider the Laplace distribution truncated at the interval $[\mu - r, \mu + r)$. We assume that r would be a suitable power of 2 allowing product with r (denoted by $r \times \mathtt{p}$) to be just a decimal shift and μ to be a b-bit representable number allowing precise shifting of $\mathtt{p}$. First we generate the exponentials scaled for an interval of width r instead of width 1:

$$\pi_{0,-\frac{r}{b}} \rightsquigarrow_b \mathtt{p}_1 \qquad \pi_{0,\frac{r}{b}} \rightsquigarrow_b \mathtt{p}_2$$

And then we create a mixture of them:

$$Laplace(x|\mu, b) \rightsquigarrow_b \mathtt{if\ flip\ 0.5\ then}\ \mu + r \times \mathtt{p}_1\ \mathtt{else}\ \mu - r + r \times \mathtt{p}_2$$

Since $\mathtt{p}_1$ and $\mathtt{p}_2$ use b coin flips each, the program shown above uses $2b$ coin flips and $\rightsquigarrow \circ \rightsquigarrow_b$ is a sound bit blasting function for Laplace distributions as well.

3.4 How does bit blasting help in inference?

We have demonstrated sound bit blasting functions for mixed gamma distributions. But how does that help in inference for probabilistic programs with these distributions? To answer this question, we focus on a particular inference strategy - that is knowledge compilation. We first describe the necessary preliminaries about knowledge compilation and then argue about how the programs obtained through $\rightsquigarrow_b$ are efficient for knowledge compilation.

Knowledge compilation based approaches [Fierens et al. 2015; Holtzen et al. 2020] for exact discrete probabilistic inference compile discrete probabilistic programs into weighted Boolean formulas that are represented using (reduced) ordered binary decision diagrams (OBDDs). These OBDDs are single rooted in case of a single Boolean random variable being returned and multi-rooted in case of a tuple of Boolean random variables being returned. By fixing a value for all the flips (biased coin flips with associated weights) in the program, and by traversing the OBDD following those values (solid line for true, dashed for false), we reach the terminal corresponding to the value of each bit. The operation of weighted model counting computes the probability of reaching the 1-terminal for each bit in the returned value. It is a dynamic programming algorithm that runs in time linear in the OBDD size. The size of an OBDD for a Boolean formula ϕ, denoted $OBDD(\phi)$, is the number of nodes in the OBDD. Thus, if we can obtain a smaller OBDD representation for a distribution, we can efficiently compose it with other constructs in a discrete probabilistic program.

We discuss and prove formally how every program obtained through the judgement $\rightsquigarrow_b$ compiles into a weighted Boolean formula that compiles to a multi-rooted OBDD that grows linearly in the number of bits as opposed to the worst case exponentially. Recall that Dice programs compile to weighted Boolean formula (φ, γ, w) where φ is the (tuple of) Boolean formulae corresponding to the return value of the program, γ is the accepting Boolean formula to encode observations and w is the weight function with flip probabilities.

Theorem 10. *$\forall \Upsilon, \mathtt{p}, \varphi, \gamma, w, \exists k, \forall b$, if $\Upsilon \rightsquigarrow_b \mathtt{p}$ and $\mathtt{p} \rightsquigarrow (\varphi, \gamma, w)$, then there exists a variable order Π of Boolean random variables in w such that OBDD(φ) + OBDD(γ) $\leq kb$*

We now provide intuition for the proof of the above theorem. Note that in the programs obtained through the judgement $\rightsquigarrow_b$, there are only two constructs that depend on the number of bits: (1) construction of exponential distribution $\pi_{0,\beta}$, and (2) conditioning on inequality between an exponential and a uniform distribution through `unifObs(y, b)`. We provide intuition how the *OBDD* size for these constructs increase linearly in the number of bits, b.

3.4.1 Exponential Distribution. In Figure 9, we 3-bit blast an exponential distribution, i.e. we have a discrete exponential distribution over 8 values (0, 0.125, 0.25, . . ., 0.875). The figure shows a 3-rooted OBDD where each root labeled as b_1, b_2 and b_3 represents a bit in the returned value of 3 bits. Consider that we fix the value of the flip corresponding to the node f_1 to be true, then for b_1, we would

reach the terminal 1 and its value would be assigned 1. The operation of weighted model counting (WMC) would calculate the probability of b_1 to be 1 as $6.14e^{-6}$ as node f_1 has probability $6.14e^{-6}$ to be true. Similarly WMC can be used for other roots of this OBDD. Since each bit needs one OBDD node, the overall OBDD size grows linearly with the number of bits. Since WMC runs in time linear in the OBDD size, probabilistic inference for an exponential distribution would run linear in the number of bits $O(b)$. Another example of OBDD for a uniform distribution can be found in the appendix.

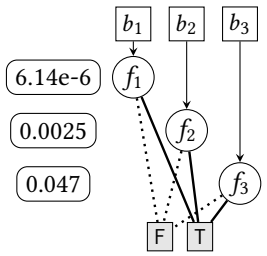


Figure 9. Compiled BDD for exponential (-3) e^{-3x}.

For all programs p obtained through the judgement $\rightsquigarrow_b$, if p $\rightsquigarrow (\varphi, \gamma, w)$, then φ is the tuple of Boolean formulae representing the return value of the program. We argue that the return value of p is always a mixture of exponential distributions making its OBDD size linear in the number of bits.

3.4.2 Conditioning on inequality between an exponential and a uniform distribution. The rules for judgment $\rightsquigarrow_b$ use the helper judgment `unifObs(y, b) = p` to condition on an inequality between binary representations of a uniform distribution and an exponential distribution.

Definition 13 (inequality function). *A b-bit inequality function, $LT_b : \{0,1\}^b \times \{0,1\}^b \to \{0,1\}$ takes as input two b-bit numbers x, y and outputs 1 if $x < y$ and 0 otherwise. Thus,*

$$LT_b((x_1, \ldots x_b), (y_1, \ldots y_b)) = \begin{cases} \neg x_1 y_1 & b = 1 \\ \neg x_1 y_1 + (\neg x_1 \neg y_1 + x_1 y_1) LT_{b-1}((x_2, \ldots, x_b), (y_2, \ldots, y_b)) & b > 1 \end{cases}$$

We prove the following lemma which states that the OBDD size for the inequality function grows linearly with the number of bits.

Lemma 11. *$\exists k, \forall b$, for the variable order $x_1, y_1, x_2, y_2, \ldots, x_b, y_b$, the size of the OBDD, that is $OBDD(LT((x_1, x_2, \ldots, x_b), (y_1, y_2, \ldots, y_b))) \leq kb$.*

Since the only constructs that depend on the number of bits (exponential distributions and inequalities) grow linearly with the number of bits, Theorem 10 holds intuitively. We provide a formal proof in the appendix.

4 HYBIT: A PROBABILISTIC PROGRAMMING SYSTEM

The previous section described how to bit blast mixed-gamma distributions. We further use it to build a probabilistic program system `HyBit` for hybrid probabilistic programs. This section describes its syntax and implementation and elaborates on two important aspects: piece-wise approximations of continuous distributions and advantages of a binary representation.

```
1  τ ::= Bool | DistFix{n, n}
2  v ::= T | F | DistFix{n, n}(r)
3  e ::= x | v | flip θ | general_gamma(n, n, n, r, r, r) | bitblast(n, pdf, n, b, r, r)
4        | if e then e else e | observe e | op^n(e_1, ..., e_n)
```

Figure 10. Syntax for the core `HyBit` expressions. `DistFix{n, n}` refers to the type of fixed-point numbers. The metavariable r ranges over real numbers, n over integers, b over Booleans, x over variable names, and θ over real numbers in the range $[0, 1]$. The metavariable pdf ranges over continuous density functions $pdf : \mathrm{R} \to \mathrm{R}$. op^n ranges over arithemtic operations with n-arity. We explain the API `DistFix`, `general_gamma` and `bitblast` and their arguments in Figure 11.

`DistFix{W, F}`

Parameters: `W`: total number of bits being used; `F`: number of bits after the binary point

---

`DistFix{W, F} general_gamma(int W, int F, int alpha, float beta, float ll, float ul)`

Parameters: `W, F`: number of bits for bit blasting; `alpha, beta`: parameters of the density $\pi_{\alpha,\beta}$
`ll, ul`: range of the continuous density
Returns: Sound bit blasted distribution of type `DistFix{W, F}`

`DistFix{W, F} bitblast(int W, int F, function pdf, int pieces, bool dist, float ll, float ul)`

Parameters: `W, F`: number of bits for bit blasting; `pdf`: continuous density function
`pieces`: number of pieces; `ll, ul`: range of the continuous density
`dist`: Boolean indicating whether to use linear or exponential pieces
Returns: Bit blasted distribution of type `DistFix{W, F}`

```
Dict{float, float} pr(DistFix{W, F} var)
float expectation(DistFix{W, F} var)
float variance(DistFix{W, F} var)
```

Parameters: `var`: random variable
Returns: Probability distribution / expectation / variance of `var`.

Figure 11. API for HyBit.

4.1 HyBit — Syntax and Implementation Details

We build a probabilistic programming system HyBit around sound bit blasting of mixed-gamma densities and approximate bit blasting of other continuous distributions. HyBit has been implemented as a shallow embedded domain specific language in Julia [Bezanson et al. 2017].

The core syntax of HyBit expressions is given in Figure 10. It provides support for distributions over Booleans (`flip` θ) and fixed-point numbers (`general_gamma` and `bitblast`). It supports Boolean operations ($\neg, \wedge, \vee$) and arithmetic operations (+, -, *, /, %, <, ==) over these distributions as well as hard observations for probabilistic conditioning (`observe`). For all the constructs in Figure 10, HyBit performs a non-standard execution and compiles them to OBDDs to perform probabilistic inference. Since HyBit has been implemented as a library in Julia, HyBit programmers can also make use of Julia constructs such as (bounded) loops, tuples and functions. As an example, a `for` loop from Julia can be used with HyBit constructs in the loop body to build a probabilistic model. HyBit is available as an open source repository with a comprehensive set of examples. [4].

Figure 11 contains more details on the API of HyBit. `DistFix{W, F}` is the type of fixed point numbers of bitwidth `W` with `F` bits after the binary point. The function `general-gamma` performs sound bit blasting of a specified generalized gamma density to the given fixed-point `W` and `F`. Sound bit blasting of mixed-gamma densities is achieved by using the `if-then-else` construct over generalized gamma densities. The function `bitblast` is used for bit blasting any arbitrary continuous distribution using piece-wise approximation, employing the bits and pieces specified using the parameters `W`, `F`, and `pieces`. The API also allows the user to choose the type of discrete distribution – linear or exponential – for the piece-wise approximation. The parameters of linear

[4] https://github.com/Tractables/Dice.jl/tree/hybit

(slope) and exponential (β) pieces are automatically chosen such that the ratio of probabilities of the first and last interval is the same as that for the naïve discretization. Finally, the API also provides functions for querying the probability distribution, expectation and the variance of a random variable. The next subsections describe piece-wise approximations and the computation of expectation and variance in more detail.

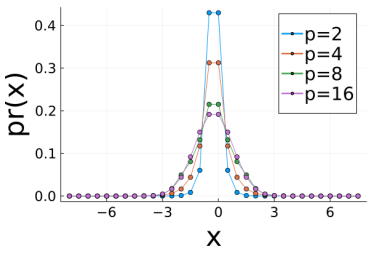


Figure 12. Bit blasted Gaussian distribution using 5 bits in the range $[-8, 8)$ using exponential pieces p — for 2, 4, 8 and 16 pieces.

4.2 Piece-wise Approximations

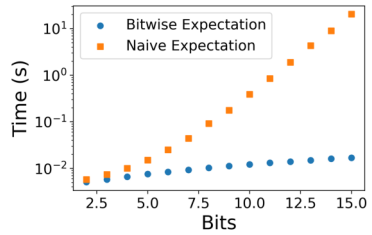
Even though mixed-gamma distributions capture many natural distributions, there are other commonly occurring ones, such as the Gaussian. It is an open problem as to whether Gaussians admit a sound bit blasting function, let alone one that compiles to a compact OBDD. For such distributions, one can instead use a piece-wise approximation.

Let C be an arbitrary continuous probability distribution over the interval $[l, u)$. To bit blast C using a piece-wise distribution with t pieces, C is approximated using a mixture of t discrete probability distributions over disjoint intervals. For every piece, one creates a shifted and scaled instance of a bit blasted mixed-gamma density and then creates a mixture of them. Note that since each piece uses $O(b)$ coin flips, a piece-wise distribution with t pieces uses $O(tb)$ coin flips. Section 5 shows empirical advantages of this approach. This piece-wise approximation using linear or exponential pieces can be easily built using the `bitblast` API available in `HyBit` (Figure 11). Figure 12 shows bit blasting of a Gaussian distribution using 2, 4, 8 and 16 pieces, where each piece is a bit blasted exponential. This provides the user with the conventional trade off between accuracy and performance. We elaborate more on this in Section 5.2.

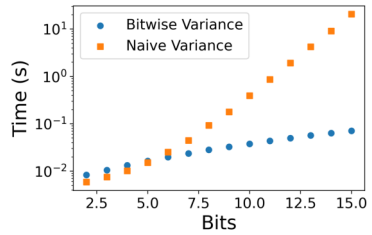
4.3 Advantages of the Binary Representation

The binary representation has important advantages for probabilistic reasoning beyond the succinctness that bit blasting provides. First, many hybrid probabilistic models involve arithmetic operation on continuous random variables. Since we use a binary representation of fixed point numbers, arithmetic operations such as +, *, /, < are compiled as Boolean formulas over binary numbers (similar to ALU circuits in architecture). This representation allows probabilistic inference (specifically the knowledge compilation approach that we employ) to identify and exploit the structure that exists in arithmetic, such as conditional independences among the resulting bits in a computation. Recent work [Cao et al. 2023] described this compilation and showed its advantage empirically for integers; `HyBit` leverages these advantages for computations over fixed-point numbers.

The binary representation also enables an optimized computation of expectation and variance. Naïve computation of expectation and variance for a distribution over 2^b values requires one to compute probability of 2^b values. Bitwise representation allows one to achieve this computation by only



(a) Expectation



(b) Variance

Figure 13. Speedup in computing expectation and variance

computing probability of b bits which gives an exponential improvement. Note that in the worst case for an arbitrary hybrid probabilistic program, getting the corresponding OBDD for the binary representation can itself be exponential in the number of bits. But for the class of mixed-gamma distributions, this conversion is linear in the number of bits (Theorem 10). We formalize the computation of expectation and variance in the following two theorems and provide proofs in the appendix.

Theorem 12. *Let D be a discrete probability distribution over the interval $[0, 2^n)$ represented as a distribution over n bits as $(b_n, b_{n-1}, \ldots, b_1)$, then the expectation of D can be computed using linearity of expectation as follows:*

$$\mathrm{E}[D] = \sum_{i=0}^{2^n-1} i \cdot pr(i) = \mathrm{E}(\sum_{j=1}^{n} 2^{j-1} b_j) = \sum_{j=1}^{n} 2^{j-1} \cdot pr(b_j)$$

Theorem 13. *Let D be a discrete probability distribution over the interval $[0, 2^n)$ represented as a distribution over n bits as $(b_n, b_{n-1}, \ldots, b_1)$, then the variance of D can be computed as follows:*

$$Var[D] = \sum_{i=0}^{2^n-1} i^2 pr(i) - (\mathrm{E}(D))^2 = Var(\sum_{j=1}^{n} 2^{j-1} b_j) = \sum_{k=1}^{n} \sum_{l=1}^{n} 2^{l+k-2} [pr(b_l \wedge b_k) - pr(b_l) pr(b_k)]$$

Example. Consider a discrete uniform distribution U_4 over the integers $\{0, 1, 2, 3\}$ represented using two bits, (X_2, X_1). The direct way of calculating the expectation and variance of this uniform distribution requires inferring the probability of all the integers in the domain. But Theorems 12 and 13 allow us to compute these quantities by using only the probabilities of the individual bits.

$$\mathrm{E}[U_4] = \sum_{j=1}^{2} 2^{j-1} \cdot pr(b_j) = 1.5 \qquad Var[U_4] = \sum_{k=1}^{2} \sum_{l=1}^{2} 2^{l+k-2} [pr(b_l \wedge b_k) - pr(b_l) pr(b_k)] = 1.25$$

Figure 13 empirically shows the performance benefits in computing expectation and variance of a distribution as we increase the number of bits in bit blasting a standard normal distribution.

5 EMPIRICAL EVALUATION

We evaluate the practicality of bit blasting on real-life probabilistic programs. We have carried out relevant experiments to investigate the following questions:

Q1: How does `HyBit` perform in comparison to existing inference algorithms? Section 5.1
Q2: How effective is the piece-wise approximation? Section 5.2

5.1 Comparison with existing inference algorithms

5.1.1 Approximate Inference Algorithms. We evaluate `HyBit` against two classes of approximate inference algorithms.

Sampling Methods We compare against WebPPL rejection sampling, MCMC sampling (with a Metropolis Hastings kernel), SMC sampling and Stan HMC as representatives of this class.

Discretization Methods We compare against AQUA and GuBPI in this class of inference algorithms [Beutner et al. 2022; Huang et al. 2021].

Comparing performance of different probabilistic programming systems is a challenging task since performance is directly affected by the structure of the program. We write equivalent programs for these benchmarks in each system and put in our best effort to optimize them. The tables in this section and subsequent sections report the mean value of absolute error over 10 runs for stochastic algorithms. For other inference algorithms, we report output of a single run. All experiments were single-threaded and were carried out on a server with 2.4 GHz CPU and 512 GB RAM.

Table 2 reports the results of performance evaluation of HyBit against other approximate inference algorithms. We take all the hybrid and continuous benchmarks on which *Psi* [Gehr et al. 2016] was evaluated and a few more relevant benchmarks from existing work [Huang et al. 2021]. We put in our best effort to compute ground truth for these benchmarks either analytically or using computer algebra systems. We include only those benchmarks in our evaluation for which we were able to compute the ground truth reliably. We report the absolute error with respect to the ground truth for all the benchmarks. For benchmarks that returned a non-boolean value, we compute the absolute error of expectation for each of the approaches. We report the minimum error achieved by an inference algorithm within a timeout of 20 minutes.

For all the benchmarks, HyBit replaced mixed-gamma distributions with their sound bit blasted distribution and all other distributions with their linear piece-wise approximation $\pi_{1,0}$. The employed bits and pieces for each benchmark are reported in Table 2. To run Stan on these benchmarks, we make use of SlicStan [Gorinova et al. 2020] to get the Stan program with marginalized discrete random variables. For all WebPPL baselines, default settings were used for all the sampling algorithms with maximum number of samples within 20 minutes.

Table 3. Comparison of HyBit with Psi, an exact inference PPL. 'ϕ' denotes a timeout, '✗' denotes result unsolved by Mathematica

Benchmarks	HyBit	Psi
Pi	✓	✗
weekend	✓	✓
spacex	✓	✗
GPA	✓	✓
Tug of war	✓	✗
altermu2	✓	✓
conjugate gaussians	✓	✓
normal_mixture (θ)	✓	ϕ
normal_mixture (μ_1)	✓	ϕ
normal_mixture (μ_2)	✓	ϕ
zeroone (w1)	✓	ϕ
zeroone (w2)	✓	ϕ
coinBias	✓	✓
Addfun/sum	✓	✓
ClickGraph	✓	✓
trueskill	✓	✗
clinicaltrial1	✓	ϕ
clinicaltrial2	✓	✓
addfun/max	✓	✓

Table 2. *Comparison of HyBit against other approximate inference algorithms.* Each row consists of one entry in bold indicating the lowest absolute error achieved among all inference algorithms. A '✗' denotes that the baseline does not support inference for the benchmark. A 'ϕ' denotes timeout. A '∞' denotes infinite bounds.

Benchmarks	HyBit			AQUA	WebPPL			Stan	GuBPI
		Bit	Pieces		rejection	MCMC	SMC		
Pi [@10kdiver [n. d.]]	1.05E-04	14	–	✗	8.30E-05	9.66E-05	1.38E-03	**4.84E-05**	✗
weekend [Gehr et al. 2016]	**2.08E-08**	24	4096	✗	1.57E-02	1.57E-02	1.66E-02	✗	2.50E-05
spacex [canyon289 2022]	6.94E-04	19	32	✗	9.06E-04	3.24E-03	1.88E-02	**1.15E-04**	ϕ
GPA [Wu et al. 2018]	**2.22E-16**	25	4096	3.62E-01	1.70E-02	9.39E-03	1.51E-02	✗	3.88E-01
Tug of war [Huang et al. 2021]	**4.50E-07**	22	16	✗	6.93E-04	6.94E-04	2.35E-03	4.51E-05	ϕ
altermu2 [Nishihara et al. 2013]	3.48E-06	17	256	**3.41E-07**	ϕ	4.61E-01	4.38E-01	1.68E-03	1.57E-02
conjugate gaussians [Jordan 2010]	**4.92E-06**	23	16	0.99	2.19E-04	3.53E-04	3.18E-03	1.06E-04	1.09E-03
normal_mix (θ) [Huang et al. 2021]	5.49E-05	9	64	**4.13E-07**	ϕ	3.90E-04	5.30E-03	4.29E-01	∞
normal_mix (μ_1) [Huang et al. 2021]	5.20E-03	9	16	**7.55E-06**	ϕ	1.36E-03	2.00E-02	1.87E+01	9.21E+00
normal_mix (μ_2) [Huang et al. 2021]	3.92E-03	9	32	**8.65E-06**	ϕ	7.11E-04	1.15E-02	1.77E+01	9.44E+00
zeroone (w1) [Bissiri et al. 2016]	**9.40E-05**	16	–	5.66E-02	ϕ	ϕ	ϕ	1.73E-01	∞
zeroone (w2) [Bissiri et al. 2016]	**4.51E-04**	19	–	3.69E+00	1.64E+00	1.64E+00	1.66E+00	2.38E-01	∞
coinBias [Gehr et al. 2016]	**2.02E-07**	22	4096	6.25E-02	1.69E-05	7.73E-05	1.22E-03	1.18E-05	4.01E-03
Addfun/sum [Gehr et al. 2016]	**3.81E-06**	23	16	✗	4.41E-04	1.69E-03	5.41E-03	8.45E-05	3.12E-02
ClickGraph [Gehr et al. 2016]	1.75E-03	10	–	✗	7.29E-04	1.22E-03	3.41E-03	**2.80E-05**	ϕ
trueskill [Gehr et al. 2016]	3.05E-03	10	16	✗	1.81E-04	4.22E-04	1.68E-03	**6.88E-05**	ϕ
clinicaltrial1 [Gehr et al. 2016]	**5.27E-16**	8	–	✗	1.51E-01	1.53E-01	1.49E-01	9.27E-04	ϕ
clinicaltrial2 [Gehr et al. 2016]	**6.81E-07**	12	–	✗	1.42E-01	1.43E-01	1.42E-01	4.54E-05	2.86E-01
addfun/max [Gehr et al. 2016]	**2.93E-07**	23	128	✗	4.38E-04	6.11E-04	3.26E-03	1.19E-04	8.56E-01

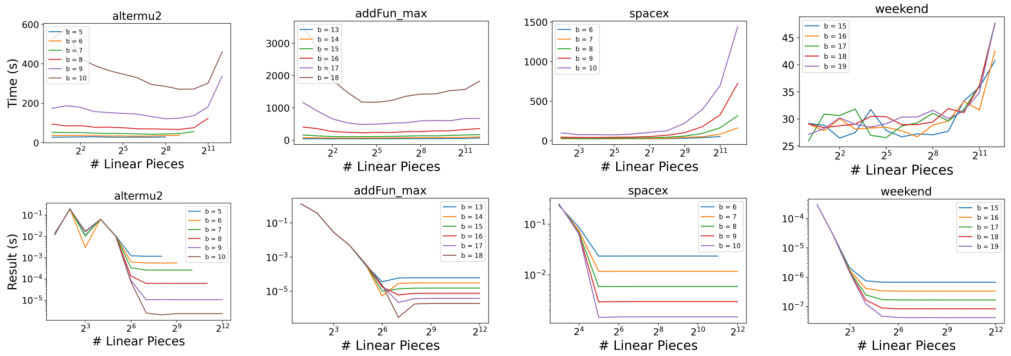


Figure 14. Runtime and Accuracy trends with respect to linear pieces for different bitwidths

As Table 2 shows, HyBit with bit blasting is comparable with the existing approaches on all the benchmarks, even better on 11/19 of them. For the other 8 benchmarks, HyBit achieves a very close accuracy. AQUA performs better on only four of the benchmarks and GuBPI fails to obtain good accuracy. This is primarily because their enumerative discretization does not scale well for higher precision. WebPPL and Stan (equipped with automated marginalization through SlicStan) support most of the benchmarks but do not achieve good accuracy within the threshold time. This is because sampling based algorithms are stochastic and cannot obtain sufficiently many samples from the true posterior in limited time.

5.1.2 Exact Inference Algorithms. Table 3 compares HyBit against a probabilistic programming system that performs exact inference using algebraic methods i.e. Psi [Gehr et al. 2016]. We put in our best effort to translate the benchmarks for optimal performance in Psi. It would often output a symbolic expression which we would feed to Mathematica for further simplification. Computing and simplifying these algebraic expressions is not a trivial task and hence, Psi timed out on 6 of these benchmarks and Mathematica failed to simplify 4 of these benchmarks. HyBit works for all 19 benchmarks as it reduces the computation to discrete inference on Boolean random variables and approximates the inference query.

5.2 How effective is piece-wise approximation?

We analyze the tradeoff between performance and accuracy when using different numbers of pieces to approximate the continuous distribution. Figure 14 demonstrates the trends of runtime and accuracy with the increase in pieces for different bitwidths for four benchmarks. As the number of linear pieces increases, runtime tends to first decrease and then increase. As the number of pieces increases, the accuracy tends to improve as shown by the lower four plots. This is because as we increase the number of pieces, continuous distributions are replaced with more accurate bit blasted distributions. That accuracy improvement comes at the cost of increased runtime after a certain sweet spot. The appendix provides additional experiments that also justify the usage of piece-wise approximations over an approach based on the central limit theorem.

6 RELATED WORK

Probabilistic programming has been an an active area of research both from the perspective of semantics and inference [Dahlqvist et al. 2023; Milch et al. 2005]. This section positions HyBit with respect related work. At a high level, the key distinction in HyBit is the development of bit blasting for succinct discretization of hybrid probabilistic programs.

Discretization approaches. Prior approaches that discretize continuous or hybrid probabilistic programs estimate the posterior by exhaustively enumerating all of the discretized values [Huang et al. 2021], which does not scale to provide sufficient accuracy in many cases. One prior discretization technique also employs a bit representation [Claret et al. 2013]. However, their approach is not a form of bit blasting, since it is not succinct and still in general produces a representation that is proportional to the number of discretized points. Finally, a recent approach uses discretization to produce upper and lower bounds on the posterior of a probabilistic program [Beutner et al. 2022].

Inference algorithms for hybrid probabilistic programs. Other research specifically targets hybrid probabilistic programs. Leios [Laurel and Misailovic 2020] *continualizes* the hybrid probabilistic program in order to harness the power of existing continuous inference algorithms. `HyBit`, on the other hand discretizes the hybrid programs which helps in scaling inference for hybrid programs specifically with respect to the discrete structure. SPPL supports hybrid programs by translating them to specific representations for inference [Saad et al. 2021]. However, these representations constrain the hybrid programs that can be supported. For instance, SPPL does not support arithmetic on continuous random variables while `HyBit` can. Finally, probabilistic logic programming languages have been extended to support hybrid models using interval traces [Gutmann et al. 2011].

Algebraic approaches. Some PPL inference algorithms produce closed form algebraic expressions to encode probability distributions and then use symbolic techniques to perform exact inference [Gehr et al. 2016; Hur et al. 2014; Narayanan et al. 2016]. However, these systems are necessarily limited in their expressivity and the programs that they can handle, as shown in Table 3.

Path based inference algorithms. A common class of inference algorithms for PPLs are *operational*: they record traces on the program by using concrete values of the random variables. This includes sampling algorithms and variational approximations [Bingham et al. 2019; Carpenter et al. 2017; Chaganty et al. 2013; Dillon et al. 2017; Goodman et al. 2008; Hur et al. 2015; Kucukelbir et al. 2015; Mansinghka et al. 2013, 2018; Minka et al. 2014; Pfeffer 2007; Saad and Mansinghka 2016; Tristan et al. 2014; van de Meent et al. 2015; Wingate and Weber 2013; Wood et al. 2014]. Sampling algorithms like rejection sampling and MCMC methods are universal but have known limitations such as difficulty in handling multi-modality and low-probability evidence, as described in Section 2. More sophisticated techniques like Hamiltonian Monte Carlo and variational approximation address these limitations but impose constraints of continuity and almost-everywhere differentiability, so they must resort to marginalizing out all discrete structure.

Use of a binary representation. Bit blasting has been a widespread technique in software verification, used in constraint solvers to reason about arithmetic using a bit representation [Bruttomesso and Sharygina 2009]. Recent work in scaling inference for probabilistic programs over integers also employs a binary representation for numbers [Cao et al. 2023], in order to exploit conditional independences in that representation. The bit blasting in `HyBit` is inspired by these techniques but has a different purpose and hence a very different technical approach: to develop succinct, and in many cases provably sound, approximations of continuous probability distributions.

7 CONCLUSION AND FUTURE WORK

In this work, we motivated the need for new inference methods for hybrid probabilistic programs. We described *bit blasting*, whereby hybrid probabilistic programs are succinctly discretized and then analyzed using algorithms for discrete inference. We characterized a class of continuous densities — mixed-gamma densities — for which bit blasting is not only succinct but also sound relative to an explicit discretization approach as well as provably efficient to analyze. We then presented a new PPL `HyBit` that employs a novel inference algorithm for hybrid programs based on bit blasting. We demonstrated the performance benefits of `HyBit` over existing approximate inference algorithms.

In future work, we hope to expand the class of distributions that can be bit blasted soundly. We plan to investigate how HyBit can be extended to support hierarchical Bayesian models. We plan to enhance its usability by not requiring user to specify the hyperparameters for every probabilistic program. We are also interested to explore the integration of HyBit with other inference approaches, to leverage their relative strengths for support of a wider range of hybrid probabilistic programs.

ACKNOWLEDGMENTS

The authors would like to thank previous and current members of Star AI lab for helpful discussions and emotional support. This work was funded in part by the DARPA PTG Program under award HR00112220005, the DARPA ANSR program under award FA8750-23-2-0004, NSF grants #IIS-1943641, #IIS-1956441, #CCF-1837129, and a gift from RelationalAI. GVdB discloses a financial interest in RelationalAI.

ARTIFACT

HyBit probabilistic programming system is available as an open source repository on GitHub at https://github.com/Tractables/Dice.jl/tree/hybit with thorough documentation for reusability. It is also available as an archived version on Zenodo [Garg et al. 2024] with comprehensive instructions and scripts for reproducibility of experiments reported in the paper.

REFERENCES

@10kdiver. [n. d.]. @10kdiver. https://twitter.com/10kdiver/status/1503307755976765443?s=20&t=Ux1C55thW5qnCPVOHoxguQ. [Online; accessed 14-March-2022].

Aws Albarghouthi, Loris D'Antoni, Samuel Drews, and Aditya V. Nori. 2017. FairSquare: Probabilistic Verification of Program Fairness. *Proc. ACM Program. Lang.* 1, OOPSLA, Article 80 (Oct. 2017), 30 pages. https://doi.org/10.1145/3133904

S. Arora and B. Barak. 2006. *Computational Complexity: A Modern Approach*. Cambridge University Press. https://theory.cs.princeton.edu/complexity/book.pdf

Raven Beutner, Luke Ong, and Fabian Zaiser. 2022. Guaranteed Bounds for Posterior Inference in Universal Probabilistic Programming. https://doi.org/10.48550/ARXIV.2204.02948

Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. 2017. Julia: A fresh approach to numerical computing. *SIAM Review* 59, 1 (2017), 65–98. https://doi.org/10.1137/141000671

Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. 2018. Pyro documentation. https://pyro.ai/examples/enumeration.html

Eli Bingham, Jonathan P Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D Goodman. 2019. Pyro: Deep universal probabilistic programming. *The Journal of Machine Learning Research* 20, 1 (2019), 973–978.

P. G. Bissiri, C. C. Holmes, and S. G. Walker. 2016. A general framework for updating belief distributions. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 78, 5 (feb 2016), 1103–1130. https://doi.org/10.1111/rssb.12158

Roberto Bruttomesso and Natasha Sharygina. 2009. A scalable decision procedure for fixed-width bit-vectors. In *2009 IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers*. 13–20. https://doi.org/10.1145/1687399.1687403

canyon289. 2022. Spacex. (2022). https://gist.github.com/canyon289/73890bab211c5cbaea41ad6f32df01a5

William X. Cao, Poorva Garg, Ryan Tjoa, Steven Holtzen, Todd Millstein, and Guy Van den Broeck. 2023. Scaling integer arithmetic in probabilistic programs. In *Proceedings of the Thirty-Ninth Conference on Uncertainty in Artificial Intelligence* (Pittsburgh, PA, USA) *(UAI '23)*. JMLR.org, Article 25, 11 pages.

Bob Carpenter, Andrew Gelman, Matthew D. Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. 2017. Stan: A Probabilistic Programming Language. *Journal of Statistical Software* 76, 1 (2017), 1–32. https://doi.org/10.18637/jss.v076.i01

Arun Chaganty, Aditya Nori, and Sriram Rajamani. 2013. Efficiently sampling probabilistic programs via program analysis. In *Artificial Intelligence and Statistics*. 153–160.

Mark Chavira and Adnan Darwiche. 2008. On Probabilistic Inference by Weighted Model Counting. *J. Artificial Intelligence* 172, 6-7 (April 2008), 772–799. https://doi.org/10.1016/j.artint.2007.11.002

Mark Chavira, Adnan Darwiche, and Manfred Jaeger. 2006. Compiling Relational Bayesian Networks for Exact Inference. *IJAR* 42, 1-2 (May 2006), 4–20.

Irene Y. Chen, Shalmali Joshi, Marzyeh Ghassemi, and Rajesh Ranganath. 2020. Probabilistic Machine Learning for Healthcare. arXiv:2009.11087 [stat.ML]

Guillaume Claret, Sriram K. Rajamani, Aditya V. Nori, Andrew D. Gordon, and Johannes Borgström. 2013. Bayesian Inference Using Data Flow Analysis. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering* (Saint Petersburg, Russia) *(ESEC/FSE 2013)*. Association for Computing Machinery, New York, NY, USA, 92–102. https://doi.org/10.1145/2491411.2491423

Fredrik Dahlqvist, Alexandra Silva, and William Smith. 2023. Deterministic stream-sampling for probabilistic programming: semantics and verification. arXiv:2304.13504 [cs.PL]

Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. 2007. ProbLog : A Probabilistic Prolog and Its Applications to Link. *Proc. of IJCAI* (2007), 2468–2473.

Joshua V Dillon, Ian Langmore, Dustin Tran, Eugene Brevdo, Srinivas Vasudevan, Dave Moore, Brian Patton, Alex Alemi, Matt Hoffman, and Rif A Saurous. 2017. TensorFlow Distributions. *arXiv preprint arXiv:1711.10604* (2017).

Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. 2015. Inference and learning in probabilistic logic programs using weighted Boolean formulas. *J. Theory and Practice of Logic Programming* 15(3) (2015), 358 – 401. https://doi.org/10.1017/S1471068414000076

Poorva Garg, Steven Holtzen, Guy Van den Broeck, and Todd Millstein. 2023. Bit Blasting Probabilistic Programs. arXiv:2312.05706 [cs.PL]

Poorva Garg, Steven Holtzen, Guy Van den Broeck, and Todd Millstein. 2024. Bit Blasting Probabilistic Programs. https://doi.org/10.5281/zenodo.10901544

Timon Gehr, Sasa Misailovic, and Martin Vechev. 2016. Psi: Exact symbolic inference for probabilistic programs. In *International Conference on Computer Aided Verification*. Springer, 62–83.

Noah D. Goodman, Vikash K. Mansinghka, Daniel M. Roy, Keith Bonawitz, and Joshua B. Tenenbaum. 2008. Church: a language for generative models. In *Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence (UAI)*.

Noah D Goodman and Andreas Stuhlmüller. 2014. The Design and Implementation of Probabilistic Programming Languages. http://dippl.org. Accessed: 2022-10-26.

Maria I. Gorinova, Andrew D. Gordon, Charles Sutton, and Matthijs Vá kár. 2021. Conditional Independence by Typing. *ACM Transactions on Programming Languages and Systems* 44, 1 (dec 2021), 1–54. https://doi.org/10.1145/3490421

Maria I Gorinova, Dave Moore, and Matthew D Hoffman. 2020. Automatic Reparameterisation of Probabilistic Programs. *International Conference on Machine Learning (ICML)* (2020).

Bernd Gutmann, Manfred Jaeger, and Luc De Raedt. 2011. Extending ProbLog with Continuous Distributions. In *Inductive Logic Programming*, Paolo Frasconi and Francesca A. Lisi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 76–91.

Steven Holtzen, Guy Van den Broeck, and Todd Millstein. 2020. Scaling Exact Inference for Discrete Probabilistic Programs. *Proc. ACM Program. Lang. (OOPSLA)* (2020). https://doi.org/10.1145/3428208

Zixin Huang, Saikat Dutta, and Sasa Misailovic. 2021. AQUA: Automated Quantized Inference for Probabilistic Programs. In *International Symposium on Automated Technology for Verification and Analysis*. Springer, 229–246.

Chung-kil Hur, Aditya V. Nori, Sriram K. Rajamani, and Selva Sammuel. 2015. A Provably Correct Sampler for Probabilistic Programs. *FSTTCS* FSTTCS (2015), 1–14. https://doi.org/10.4230/LIPIcs.FSTTCS.2015.475

Chung-Kil Hur, Aditya V. Nori, Sriram K. Rajamani, and Selva Samuel. 2014. Slicing probabilistic programs. *Proc. of PLDI* (2014), 133–144. https://doi.org/10.1145/2594291.2594303

Michael I. Jordan. 2010. Stat260: Bayesian Modeling and Inference: The Conjugate Prior for the Normal Distribution. http://www.cs.berkeley.edu/~jordan/courses/260-spring10/lectures/lecture5.pdf

D. Koller and N. Friedman. 2009. *Probabilistic graphical models: principles and techniques*. MIT press.

Alp Kucukelbir, Rajesh Ranganath, Andrew Gelman, and David Blei. 2015. Automatic variational inference in Stan. In *Advances in neural information processing systems*. 568–576.

Jacob Laurel and Sasa Misailovic. 2020. Continualization of Probabilistic Programs With Correction. In *Programming Languages and Systems*, Peter Müller (Ed.). Springer International Publishing, Cham, 366–393.

Edward A. Lee and Sanjit A. Seshia. 2017. Introduction to Embedded Systems, A Cyber-Physical Systems Approach.

Vikash Mansinghka, Tejas D Kulkarni, Yura N Perov, and Josh Tenenbaum. 2013. Approximate bayesian image interpretation using generative probabilistic graphics programs. In *Advances in Neural Information Processing Systems*. 1520–1528.

Vikash K. Mansinghka, Ulrich Schaechtle, Shivam Handa, Alexey Radul, Yutian Chen, and Martin Rinard. 2018. Probabilistic Programming with Programmable Inference. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Philadelphia, PA, USA) *(PLDI 2018)*. ACM, New York, NY, USA, 603–616. https://doi.org/10.1145/3192366.3192409

George Marsaglia. 1971. Random Variables with Independent Binary Digits. *The Annals of Mathematical Statistics* 42, 6 (1971), 1922 – 1929. https://doi.org/10.1214/aoms/1177693058

Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L. Ong, and Andrey Kolobov. 2005. BLOG: Probabilistic Models with Unknown Objects. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence* (Edinburgh,

Scotland) *(IJCAI'05)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1352–1359.

T. Minka, J.M. Winn, J.P. Guiver, S. Webster, Y. Zaykov, B. Yangel, A. Spengler, and J. Bronskill. 2014. Infer.NET 2.6. Microsoft Research Cambridge. http://research.microsoft.com/infernet.

Praveen Narayanan, Jacques Carette, Wren Romano, Chung-chieh Shan, and Robert Zinkov. 2016. Probabilistic inference by program transformation in Hakaru (system description). In *International Symposium on Functional and Logic Programming - 13th International Symposium, FLOPS 2016, Kochi, Japan, March 4-6, 2016, Proceedings*. Springer, 62–79. https://doi.org/10.1007/978-3-319-29604-3_5

Robert Nishihara, Thomas Minka, and Daniel Tarlow. 2013. Detecting Parameter Symmetries in Probabilistic Models. https://doi.org/10.48550/ARXIV.1312.5386

Avi Pfeffer. 2007. A general importance sampling algorithm for probabilistic programs. (2007). http://nrs.harvard.edu/urn-3:HUL.InstRepos:25235125

Jeffrey S. Rosenthal. 2006. *A first look at rigorous probability theory* (second ed.). World Scientific Publishing Co. Pte. Ltd., Hackensack, NJ. xvi+219 pages.

Feras Saad and Vikash Mansinghka. 2016. A Probabilistic Programming Approach To Probabilistic Data Analysis. In *Advances in Neural Information Processing Systems (NIPS)*.

Feras A. Saad, Martin C. Rinard, and Vikash K. Mansinghka. 2021. SPPL: Probabilistic Programming with Fast Exact Symbolic Inference. In *PLDI 2021: Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Design and Implementation* (Virtual, Canada). ACM, New York, NY, USA, 804–819. https://doi.org/10.1145/3453483.3454078

J. R. Shaw, M. Bridges, and M. P. Hobson. 2007. Efficient Bayesian inference for multimodal problems in cosmology. *Monthly Notices of the Royal Astronomical Society* 378, 4 (jun 2007), 1365–1370. https://doi.org/10.1111/j.1365-2966.2007.11871.x

Hyungsuk Tak, Xiao-Li Meng, and David A. van Dyk. 2018. A Repelling–Attracting Metropolis Algorithm for Multimodality. *Journal of Computational and Graphical Statistics* 27, 3 (jul 2018), 479–490. https://doi.org/10.1080/10618600.2017.1415911

Jean-Baptiste Tristan, Daniel Huang, Joseph Tassarotti, Adam Pocock, Stephen J. Green, and Guy L. Steele. 2014. Augur: Data-Parallel Probabilistic Modeling. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2* (Montreal, Canada) *(NIPS'14)*. MIT Press, Cambridge, MA, USA, 2600–2608.

Jan-Willem van de Meent, Hongseok Yang, Vikash Mansinghka, and Frank Wood. 2015. Particle Gibbs with Ancestor Sampling for Probabilistic Programs. In *AISTATS*.

David Wingate and Theophane Weber. 2013. Automated variational inference in probabilistic programming. *arXiv preprint arXiv:1301.1299* (2013).

Frank Wood, Jan Willem Meent, and Vikash Mansinghka. 2014. A new approach to probabilistic programming inference. In *Artificial Intelligence and Statistics*. 1024–1032.

Yi Wu, Siddharth Srivastava, Nicholas Hay, Simon Du, and Stuart Russell. 2018. Discrete-Continuous Mixtures in Probabilistic Programming: Generalized Semantics and Inference Algorithms. https://doi.org/10.48550/ARXIV.1806.02027

Yuling Yao, Aki Vehtari, and Andrew Gelman. 2021. Stacking for Non-mixing Bayesian Computations: The Curse and Blessing of Multimodal Posteriors. arXiv:2006.12335 [stat.ME]

Received 2023-11-16; accepted 2024-03-31