Efficient Probabilistic Inference in the Quest for Physics Beyond the Standard Model

Atılım Güneş Baydin,[1] **Lukas Heinrich,**[2] **Wahid Bhimji,**[3]
Bradley Gram-Hansen,[1] **Gilles Louppe,**[4] **Lei Shao,**[5]
Prabhat,[3] **Kyle Cranmer,**[2] **Frank Wood**[6]
[1]University of Oxford; [2]New York University; [3]Lawrence Berkeley National Lab
[4]University of Liège; [5]Intel Corporation; [6]University of British Columbia
{gunes,bradley}@robots.ox.ac.uk; {lukas.heinrich,g.louppe,cranmer}@cern.ch
{wbhimji,prabhat}@lbl.gov; lei.shao@intel.com; fwood@cs.ubc.ca

Abstract

We present a novel framework that enables efficient probabilistic inference in large-scale scientific models by allowing the execution of existing domain-specific simulators as probabilistic programs, resulting in highly interpretable posterior inference. Our framework is general purpose and scalable, and is based on a cross-platform probabilistic execution protocol through which an inference engine can control simulators in a language-agnostic way. We demonstrate the technique in particle physics, on a scientifically accurate simulation of the τ (tau) lepton decay, which is a key ingredient in establishing the properties of the Higgs boson. High-energy physics has a rich set of simulators based on quantum field theory and the interaction of particles in matter. We show how to use probabilistic programming to perform Bayesian inference in these existing simulator codebases directly, in particular conditioning on observable outputs from a simulated particle detector to directly produce an interpretable posterior distribution over decay pathways. Inference efficiency is achieved via inference compilation where a deep recurrent neural network is trained to parameterize proposal distributions and control the stochastic simulator in a sequential importance sampling scheme, at a fraction of the computational cost of Markov chain Monte Carlo sampling.

1 Introduction

Complex simulators are used to express stochastic generative models of data across a wide segment of the scientific community, with applications as diverse as hazard analysis in seismology [1], supernova shock waves in astrophysics [2], market movements in economics [3], and blood flow in biology [4]. In these generative models, complex simulators are composed from low-level mechanistic components. These models are typically non-differentiable and lead to intractable likelihoods, which renders many traditional statistical inference algorithms irrelevant and motivates a new class of so-called likelihood-free inference algorithms [5].

There are two broad strategies for this type of likelihood-free inference problem. In the first, one uses a simulator indirectly to train a surrogate model endowed with a likelihood that can be used in traditional inference algorithms, for example approaches based on conditional density estimation [6–9] and density ratio estimation [10, 11]. Alternatively, approximate Bayesian computation (ABC) [12, 13] refers to a large class of approaches for sampling from the posterior distribution of these likelihood-free models, where the original simulator is used directly as part of the inference engine. While variational inference [14] algorithms are often used when the posterior is intractable, they are not directly applicable when the likelihood of the data generating process is unknown.

Preprint. Work in progress.

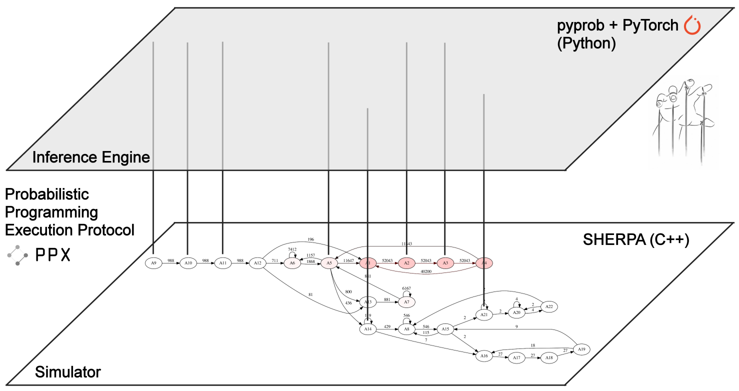


Figure 1: An illustration of the framework, where the *pyprob* package in Python is controlling the SHERPA simulator in C++ through the probabilistic programming execution (PPX) protocol.

The class of inference strategies that directly use a simulator avoids the necessity of approximating the generative model. Moreover, using a domain-specific simulator offers a natural pathway for inference algorithms to provide interpretable posterior samples. In this work, we take this approach, extend previous work in universal probabilistic programming [15] and inference compilation [16] to large-scale complex simulators, and demonstrate the ability to execute existing simulator codes under the control of general-purpose inference engines. This is achieved by creating a cross-platform probabilistic execution protocol (Figure 1) through which an inference engine can control simulators in a language-agnostic way. We implement a range of general-purpose inference engines from the Markov chain Monte Carlo (MCMC) [17] and importance sampling [18] families. The execution framework we develop currently has bindings in C++ and Python, which are languages of choice for many large-scale projects in science and industry, and it can be used by any other language pending the implementation of a lightweight front end.

We demonstrate the technique in a particle physics setting, introducing probabilistic programming as a novel tool to determine the properties of particles at the Large Hadron Collider (LHC) [19, 20] at CERN. This is achieved by coupling our framework with SHERPA[1] [21], a state-of-the-art Monte Carlo event generator of high-energy reactions of particles, which is commonly used with GEANT[2] [22], a toolkit for the simulation of the passage of the resulting particles through detectors. In particular, we perform inference in the case of τ (tau) lepton particle decay in a realistic detector, controlling the simulation within the standard SHERPA software with minimal modification and extracting posterior distributions in agreement with ground truths. To our knowledge this is the first time that universal probabilistic programming has been applied in this domain and in this scale, controlling a codebase of nearly one million lines of code. Our approach is readily scalable to more complex events and full detector simulators, paving the way to its use in the discovery of new fundamental physics.

2 Particle Physics and Probabilistic Inference

Our work is primarily motivated by applications in high-energy physics (HEP), which studies elementary particles and their interactions using energetic events created in particle accelerators such as the LHC at CERN. In this setting, the observed data are the result of interactions of particles generated in a collision event and observed through particle detectors. From these observations, we would like to infer the properties of the particles and interactions that generated them. Collisions happen millions of times per second, creating cascading particle decays in complex detectors instrumented with millions of electronics channels. These experiments then seek to filter the vast volume of (petabyte-scale) resulting data to make discoveries that shape our understanding of fundamental physics.

The complexity of the underlying physics and of the detectors have, until now, prevented the community from employing inference techniques. However, they have developed sophisticated simulator

[1]**S**imulation of **H**igh-**E**nergy **R**eactions of **Pa**rticles. `https://sherpa.hepforge.org/`

[2]**Ge**ometry **an**d **T**racking. `https://geant4.web.cern.ch/`

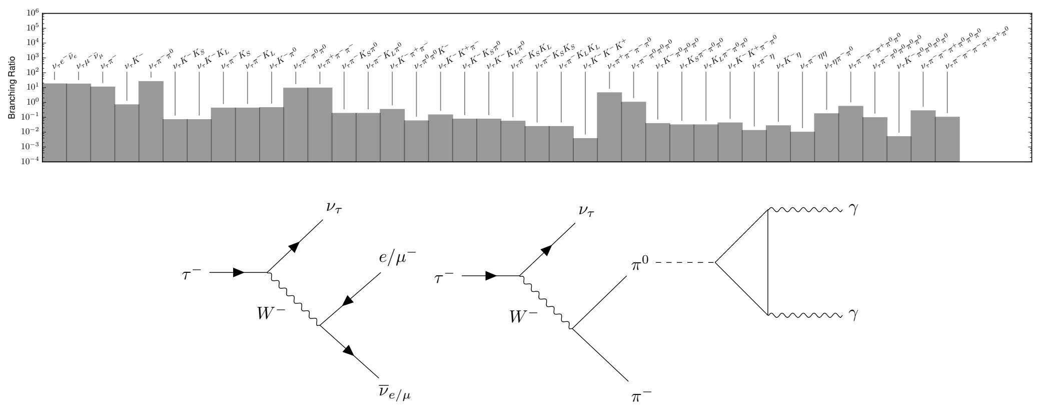


Figure 2: *Top:* branching ratios of the τ lepton, effectively the prior distribution of the decay channels in SHERPA. Note that the scale is logarithmic. *Bottom:* Feynman diagrams for τ decays illustrating that these can produce multiple detected particles.

packages such as SHERPA [21], GEANT [22], PYTHIA [23], Herwig++ [24], and MadGraph [25] to model physical processes and the interactions of particles with detectors. This is interesting from a probabilistic programming point of view, because HEP simulators are essentially very accurate probabilistic algorithms implementing the Standard Model and the passage of particles through matter (i.e., particle detectors). These simulators are coded in languages with unbounded recursion, and performing inference in such a setting *requires* using inference techniques developed for universal probabilistic programming that cannot be handled via more traditional inference approaches that apply to, for example, finite probabilistic graphical models [26]. Thus we focus on creating an infrastructure for the interpretation of existing simulator packages as probabilistic programs, which lays the groundwork for running inference in scientifically-accurate models using general-purpose probabilistic inference algorithms.

The τ Lepton Decay. The specific HEP setting we focus on in depth in this paper is the decay of a τ lepton particle inside an LHC-like detector. This is is a real use case in particle physics currently under active study by LHC physicists [27] and it is also of interest due to its importance to establishing the properties of the recently discovered Higgs boson [19, 20] through its decay to τ particles. Once produced, the τ decays to further particles observed within the detector according to certain decay channels. The probabilities of these decays or "branching ratios" are shown in Figure 2, which have been measured by other experiments and provide prior estimations for inference.

3 Related Work

3.1 Probabilistic Programming

Our work belongs in the family of sampling-based approximate inference techniques, which have been conventionally based on importance sampling [28] and Markov chain Monte Carlo (MCMC) methods [17] such as the Metropolis–Hastings (MH) algorithm [29]. These are computationally inefficient on large-scale models, due to the difficulty in choosing correct proposal distributions and handling increasing model dimensionality. Recent developments in inference algorithms, such as variational methods [30], extensions that combine deep learning [31, 8], MCMC samplers based on physical dynamics such as the No-U-Turn Sampler (NUTS) [32] and Stochastic Gradient Langevin Dynamics [33], and methods that use deep neural networks to amortize the cost of inference [34] such as inference compilation (IC) [16], have been targeting fast and scalable inference.

Probabilistic programming languages (PPLs) attempt to decouple inference algorithms from model building, by creating a simple, yet expressive, syntax that allows one to take advantage of these powerful inference algorithms on any probabilistic generative model expressed as a regular computer program. *Universal* PPLs allow the expression of unrestricted probability models in a Turing-complete fashion [35–37], and there is a recent trend in combining these with variational inference and deep learning, leading to tools such as Pyro [38], ProbTorch [39], and Edward [40]. This is in contrast to languages such as Stan [41] that target the more restricted model class of probabilistic graphical models [26].

3.2 Data Analysis in High-Energy Physics

Inference for an individual collision event in HEP is often referred to as reconstruction [42]. Reconstruction algorithms can be seen as a form of structured prediction: from the raw event data they produce a list of candidate particles together with their types and point-estimates for their momenta. The variance of these estimators is characterized by comparison to the ground truth values of the latent variables from simulated events. Bayesian inference on the latent state of an individual collision is rare in HEP given the complexity of the latent structure of the generative model. Until now, inference for the latent structure of an individual event has only been possible by accepting a drastic simplification of the high-fidelity simulators [43–58]. In contrast, inference for the fundamental parameters is based on hierarchical models and probed at the population level. Recently, machine learning techniques have been employed to learn surrogates for the implicit densities defined by the simulators as a strategy for likelihood-free inference [59].

Currently HEP simulators are run in forward mode to produce substantial datasets that often exceed the size of datasets from actual collisions within the experiments. These are then reduced to considerably lower dimensional datasets of a handful of variables using physics domain knowledge, which can then be directly compared to collision data. Machine learning and statistical approaches for classification of particle types or regression of particle properties can be trained on these large pre-generated datasets produced by the high-fidelity simulators developed over many decades. The field is increasingly employing deep learning techniques allowing these algorithms to process high-dimensional, low-level data [60–62]. However, these approaches do not estimate the posterior of the full latent state nor provide the level of interpretability our probabilistic inference framework enables by directly tying inference results to the latent process encoded by the simulator.

4 Probabilistic Inference in Large-Scale Simulators

In this section we describe the main components of our probabilistic inference framework, which consists of (1) *pyprob*, a PyTorch-based [63] PPL and associated inference engines in Python, (2) PPX, a probabilistic programming execution protocol that defines a cross-platform interface for connecting models and inference engines implemented in different programming languages and executed in separate processes, (3) *pyprob_cpp*, a lighweight C++ front end that allows the execution of models written in C++ under the control of *pyprob*.

4.1 Designing a PPL for Existing Large-Scale Simulators

A shortcoming of the current state-of-the-art in PPLs is that they are not designed to directly support existing codebases, severely limiting their applicability to a very large body of existing probabilistic models implemented as domain-specific simulators in many fields across academia and industry. A PPL, by definition, is a programming language with additional constructs for *sampling* random values from probability distributions and *conditioning* values of random variables via observations [15]. Domain-specific simulators in HEP and other fields are commonly probabilistic in nature, thus satisfying the behavior random *sampling*, albeit generally from simplistic distributions such as the continuous uniform. By automatically "reinterpreting" these existing codebases with a proper rewiring of the (pseudo-)random number generator and introducing a construct for *conditioning*, we can execute existing simulators under the control of general-purpose inference engines designed for probabilistic programming. This enables the application of Bayesian inference techniques in these simulators, essentially treating the existing simulator as a joint prior distribution of latent and observed variables of a model, and obtaining posterior distributions over latent variables conditioned on realizations of observed variables.

To realize our framework, we implement *pyprob*,[3] a universal PPL specifically designed to control models written not only in Python but also in other languages. Because the main inference technique we use in this PPL is based on deep neural networks, we base our PPL on PyTorch [63], whose automatic differentiation (AD) [64] feature with support for dynamic computation graphs has been

[3] `https://github.com/probprog/pyprob`

crucial in our implementation. Our PPL currently has two families of inference engines: (1) MCMC of the lightweight Metropolis–Hastings (LMH) [35] and random-walk Metropolis–Hastings (RMH) [65] varieties, and (2) sequential importance sampling (IS) [66, 18] with its regular (i.e., sampling from the prior) and inference compilation (IC) [16] varieties. The IC technique, where a deep neural network is trained in an amortized inference setting to guide (control) a probabilistic program conditioning on observed inputs, forms our main inference method for performing efficient inference in large-scale simulators. The LMH and RMH engines we implement are specialized for sampling in the space of execution traces of probabilistic programs, and provide way of sampling from the true posterior—at a high computational cost.

A probabilistic program can be expressed as a sequence of random samples $(x_t, a_t, i_t)_{t=1}^T$, where x_t, a_t, and i_t are respectively the value, address, and instance (counter) of a sample, the execution of which describes a joint probability distribution between latent (unobserved) random variables $\mathbf{x} := (x_t)_{t=1}^T$ and observed random variables $\mathbf{y} := (y_n)_{n=1}^N$ given by

$$p(\mathbf{x}, \mathbf{y}) := \prod_{t=1}^{T} f_{a_t}\left(x_t | x_{1:t-1}\right) \prod_{n=1}^{N} g_n(y_n | x_{\prec n}) \ , \tag{1}$$

where $f_{a_t}(\cdot | x_{1:t-1})$ denotes the prior probability distribution of a random variable with address a_t conditional on all preceding values $x_{1:t-1}$, and $g_n(\cdot | x_{\prec n})$ is the likelihood density given the sample values $x_{\prec n}$ preceding observation y_n. A PPL is a regular programming language equipped with `sample` and `observe` statements [15] for sampling random variables with given prior probability distributions and conditioning random variables upon particular observed values.

Once a model $p(\mathbf{x}, \mathbf{y})$ is expressed as a probabilistic program, we are interested in performing inference in order to get posterior distributions $p(\mathbf{x}|\mathbf{y})$ of latent variables $\mathbf{x}$ conditioned on observed variables $\mathbf{y}$. In the sequential IS scheme, a weighted set of samples $\{(w^k, \mathbf{x}^k)_{k=1}^K\}$ is used to construct an empirical approximation of the posterior distribution $\hat{p}(\mathbf{x}|\mathbf{y}) = \sum_{k=1}^K w^k \delta(\mathbf{x}^k - \mathbf{x}) / \sum_{j=1}^K w^j$, where δ is the Dirac delta function. The importance weights for a probabilistic program are expressed as

$$w^k = \prod_{n=1}^{N} g_n(y_n | x^k_{1:\tau_k(n)}) \prod_{t=1}^{T^k} \frac{f_{a_t}(x^k_t | x^k_{1:t-1})}{q_{a_t, i_t}(x^k_t | x^k_{1:t-1})} \ , \tag{2}$$

where $q_{a_t,i_t}(\cdot | x^k_{1:t-1})$ is known as the proposal distribution and may be identical to the prior f_{a_t} (as in regular IS). In the IC technique, we are training a deep neural network to receive the observed values $\mathbf{y}$ and return a set of adapted proposals $q_{a_t,i_t}(x_t | x_{1:t-1}, \mathbf{y})$ such that their joint $q(\mathbf{x}|\mathbf{y})$ is close to the true posterior $p(\mathbf{x}|\mathbf{y})$. This is achieved by using a Kullback–Leibler divergence training objective

$$\begin{aligned} \mathcal{L}(\phi) &:= \mathbb{E}_{p(\mathbf{y})}\left[D_{\mathrm{KL}}\left(p(\mathbf{x}|\mathbf{y}) \mid\mid q(\mathbf{x}|\mathbf{y}; \phi)\right)\right] && (3) \\ &= \int_{\mathbf{y}} p(\mathbf{y}) \int_{\mathbf{x}} p(\mathbf{x}|\mathbf{y}) \log \frac{p(\mathbf{x}|\mathbf{y})}{q(\mathbf{x}|\mathbf{y}; \phi)} \, \mathrm{d}\mathbf{x} \, \mathrm{d}\mathbf{y} \\ &= \mathbb{E}_{p(\mathbf{x},\mathbf{y})}\left[-\log q(\mathbf{x}|\mathbf{y}; \phi)\right] + \text{const.} \ , && (4) \end{aligned}$$

where ϕ represents the neural network weights. The neural network weights ϕ are optimized to minimize this objective by continually drawing training pairs $(\mathbf{x}, \mathbf{y}) \sim p(\mathbf{x}, \mathbf{y})$ from the probabilistic program (i.e., the generative model, or the simulator). To simplify the task of training, only a subset of all addresses (a_t, i_t) are handled by the neural network, and the remaining addresses are left to use the prior f_{a_t} as proposal during inference. The IC controlling of an address is exposed as a boolean flag called `control`, which can be applied to individual `sample` statements or delimited regions of the codebase. Expressed in simple terms, taking a desired outcome $\mathbf{y}$ from the probabilistic program as its input, the neural network learns to control the random number draws of latents $\mathbf{x}$ during the execution in such a way that makes the desired outcome likely.

The neural network architecture in IC is based on a stacked LSTM [67] recurrent core that gets executed for as many time steps as the probabilistic trace length. The input to this LSTM in each time step is a concatenation of embeddings of the observation $f^{\text{obs}}(\mathbf{y})$, the previously sampled value $f^{\text{smp}}_{a_{t-1},i_{t-1}}(x_{t-1})$, the current distribution type $f^{\text{type}}(a_t)$, and the current address $f^{\text{addr}}(a_t, i_t)$. f^{obs} is a neural network specific to the domain (such as a 3D convolutional neural network for volumetric inputs), f^{smp} are feed-forward modules, f^{type} are one-hot vectors denoting a prior distribution type from the set of supported distributions, f^{addr} are learned address embeddings optimized via backpropagation for each (a_t, i_t) pair encountered in the program execution. The addressing scheme a_t [35] is the main link between semantic locations in the probabilistic program and the inputs to the neural network. The addressing scheme in Python is based on an analysis of Python bytecode of the location where the PPL `sample` or `observe` statement is called, and in the PPX protocol (Section 4.2) the addresses a_t are produced and supplied by the side hosting and executing the model.

The joint proposal distribution of the neural network $q(\mathbf{x}|\mathbf{y})$ is factorized into proposals in each time step q_{a_t,i_t}, whose type depends on the type of the prior f_{a_t}. In the experiments presented in this paper (Section 5) the system uses categorical and continuous uniform distributions in the prior, for which we use, respectively, categorical and mixture of Kumaraswamy [68, 69] distributions as proposals parameterized by the neural network.

A common challenge for inference in real-world scientific models, such as those in HEP, is the presence of large dynamic ranges of prior probabilities for various outcomes. For instance, some particle decays are much more probable than others (Figure 2), and the prior distribution for a particle momentum can be steeply falling. Therefore some cases may be much more likely to be seen by the neural network during training relative to others. For this reason, the proposal parameters and the quality of the inference would vary significantly according to the frequency of the observations in the prior. To address this issue, we apply a technique called "prior inflation" for automatically adjusting the measure of the prior distribution during training to generate more instances of these unlikely outcomes. This applies only to the training data generation for the IC neural network, and the unmodified original model is used during inference, ensuring that the importance weights (Eq. 2) and therefore the empirical posterior are correct under the unmodified real model.

4.2 A Cross-Platform Probabilistic Execution Protocol

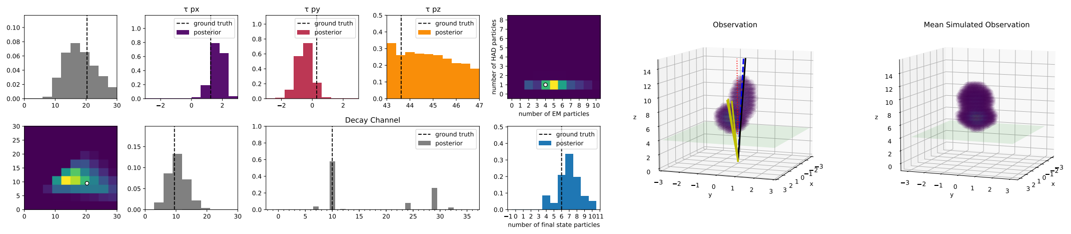
To couple our PPL and inference engines with simulators in a language-agnostic way, we introduce a probabilistic programming execution (PPX)[4] protocol that defines a schema for the execution of probabilistic programs. The protocol covers language-agnostic definitions of common probability distributions and message pairs covering the call and return values of (1) program entry points (2) `sample` statements, and (3) `observe` statements. The implementation is based on flatbuffers,[5] which is an efficient cross-platform serialization library through which we compile PPX into the officially supported languages C++, C#, Go, Java, JavaScript, PHP, Python, and TypeScript, enabling very lightweight PPL front ends in these languages—in the sense of requiring only an implementation to call `sample` and `observe` statements over the protocol. We exchange these flatbuffers-encoded messages over ZeroMQ[6] [70] sockets, which allow seamless communication between separate processes in the same machine (using inter-process sockets) or across a network (using TCP).

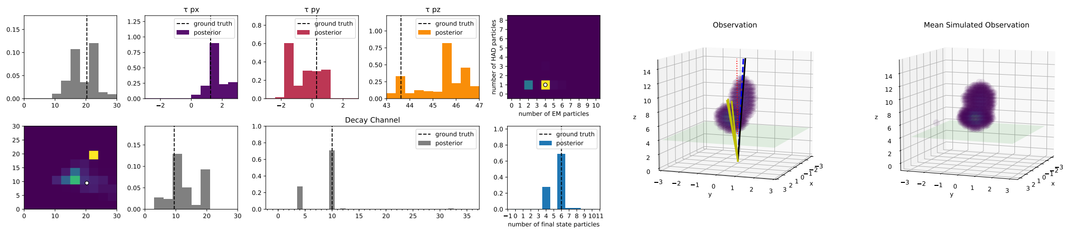
Besides its use with *pyprob*, the PPX protocol defines a very flexible way of coupling any PPL system and model so that they can be (1) implemented in different programming languages and (2) executed in separate processes and on separate machines across networks. Thus PPX is similar in spirit to, and indeed inspired by, the Open Neural Network Exchange (ONNX)[7] project for interoperability between machine learning frameworks. Note that, more than a serialization format, PPX enables runtime execution of probabilistic models under the control of inference engines in separate processes. We are releasing this language-agnostic protocol as a separately maintained project, together with the rest of our work in Python and C++.

4.3 Controlling SHERPA and the Standard Model

In this paper our target simulator is SHERPA [21], a Monte Carlo event generator of high-energy reactions of particles, which is a state-of-the-art simulator of the Standard Model of particle physics developed as an international effort within the HEP community. SHERPA, like many other large-scale

[4] `https://github.com/probprog/ppx`
[5] `http://google.github.io/flatbuffers/`
[6] `http://zeromq.org/`
[7] `https://onnx.ai/`


(a) IC proposal distribution of a selection of latents (9,600 traces).


(b) IC posterior distribution of a selection of latents (9,600 traces).

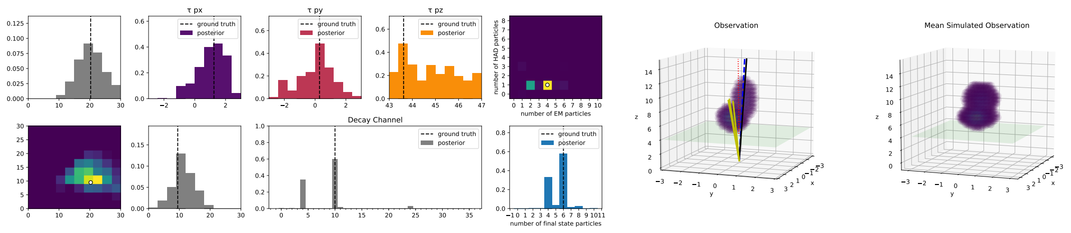

(c) RMH MCMC posterior distribution of a selection of latents (20,000 traces).

Figure 3: Proposal and posterior distributions of a subset of latent variables in the τ lepton decay problem conditioned on the same observation. (See Figure 5 for the full latent structure.) In each subfigure, the lower left and the two adjacent plots show the energies of the two most energetic final state particles and their joint probability. To the right, the distribution of the originating momentum components of the τ lepton and its decay channel are shown. In the middle we show the event composition as characterized by the number of mainly electromagnetically interacting and hadronically interacting final state particles as well as the number of decay products. To the right we show the original observation as well as the mean simulated calorimeters generated during inference. Vertical lines in histograms mark the ground truth values that generated the test observation.

scientific projects, is implemented in C++, and therefore we implement a C++ front end for PPX, called *pyprob_cpp*.[8]

We couple SHERPA to *pyprob_cpp* by a system-wide rerouting of the calls to the random number generator, which is made easy by the existence of a third-party random number generator interface (External_RNG) already present in SHERPA. Through this setup, we can repurpose, with little effort, any stochastic simulation written in SHERPA as a probabilistic generative model in which we can perform inference using probabilistic programming techniques.

Differing from the conventions in the probabilistic programming community, random number draws in C++ simulators are commonly performed at a lower level than the actual prior distribution that is being simulated. This applies to SHERPA where the only samples are from the standard uniform distribution $U(0, 1)$, which subsequently get used for different purposes using transformations or rejection sampling. In our experiments (Section 5) we work with all uniform samples except for a problem-specific single address that we know to be responsible for sampling from a categorical distribution for choosing the τ lepton decay channel. The modification of this address to use the proper categorical prior allows an effortless application of the prior inflation technique (Section 4.1) to generate training data equally representing each channel.

[8]`https://github.com/probprog/pyprob_cpp`

Rejection sampling [71] sections in the simulator pose a problem for our approach, as they define execution traces that are a priori unbounded; and since the inference network has to backpropagate through every sampled value, this makes the training significantly slower. Rejection sampling is key to the application of Monte Carlo methods for evaluating matrix elements [72] and other stages of event generation in particle physics; thus an efficient treatment of this construction is primal. We address this problem by implementing a novel trace evaluation scheme where during training we only consider the last (thus accepted) instance i_{last} of any address (a_t, i_t) that fall within a rejection sampling loop. During inference, we use this same proposal distribution $q_{a_t, i_{\text{last}}}$ in each loop execution. In other words, this corresponds to training the inference network with the state that concludes the loop (i.e., satisfies the acceptance criterion), effectively selecting proposal distributions such that the rejection loop is concluded in as few iterations as possible. This scheme works by annotating the `sample` statements within long-running rejection sampling loops with a boolean flag called `replace`, which, when set true, enables the behavior described for the given sample address.

5 Experiments

An important decay of the Higgs boson is to τ leptons, whose subsequent decay products interact in the detector. This constitutes a rich and realistic case to simulate, and directly connects to an important line of current research in particle physics. During simulation, SHERPA stochastically selects a set of particles to which the initial τ lepton will decay—a "decay channel"—and samples the momenta of these particles according to a joint density obtained from underlying physical theory. These particles then interact in the detector leading to observations in the raw sensor data. While GEANT is typically used to model the interactions in a detector, for our initial studies we implement a fast, approximate detector simulation for a calorimeter with longitudinal and transverse segmentation (with resolution $20 \times 35 \times 35$). The fast detector simulation deposits most of the energy for electrons and π^0 into the first layers and charged hadrons (e.g., $\pi^{\pm}$) deeper into the calorimeter with larger fluctuations. Given raw 3D calorimeter observations, we would like to infer primarily the decay channel that the τ lepton followed and the initial momenta p_x, p_y, and p_z. Using our framework, we compute posterior distributions for the decay channel, initial momenta, and other latent quantities in the model conditioning on various simulated observations with known ground truth. The discrete variable for decay channel has a known prior distribution (Figure 2) given by the branching ratio of the τ into 38 possible decay channels [73].

In Figure 3 we show inference results obtained from the IC and RMH MCMC engines, for a single observation $\mathbf{y}$ sampled from the model joint prior $p(\mathbf{x}, \mathbf{y})$ by running the simulation. The IC proposals are generated by an inference network trained with 1.6 million execution traces, and the IC engine controls (i.e., makes proposals different from the prior for) 47 addresses,[9] 17 of these in replacement (rejection sampling) mode, out of a total of 24,429 addresses. The RMH engine, by its very nature, controls all addresses encountered in the simulation. During IC network training, 440 trace types are encountered (Table 1), which represent the reoccurrence of the same sequence of addresses with different actual sample values. Traces reach lengths up to 7,514 and 1,190 respectively when looking at all and controlled-only samples (Figure 4).

As can be seen in Figure 3 (a), the network proposes values in agreement with the ground truth values. Figure 3 (b) shows the posterior after importance sampling guided by these proposals: this shows the correct posterior over particle decays was identified and also that related τ decays are shown as possible alternatives with correct uncertainty, in agreement with RMH samples from the correct posterior in Figure 3 (c). Furthermore, correlations between the final state particle momenta are well reproduced. In order to make the RMH results for the test observation tractable, we start the chain from the ground truth trace (i.e., eliminating the need for burn in), which would not be possible for inference on real experimental data. This shows good agreement with the decay channel and event composition posteriors obtained from the IC engine that has access to observation $\mathbf{y}$ only (i.e., did not start from ground truth), and that would be used for fast inference with real experimental data.

The ability to connect posterior samples to the simulator code is a key advantage of our method in scientific applications. This connection enables inference results to be interpretable in the context of the physically-motivated latent process encoded by the simulator. Note that the posterior distributions

[9]The controlled addresses are those that fall within sections of the codebase deemed fundamental in the solution of the τ decay problem, based on domain knowledge.

presented in Figure 3 do not show all of the posterior information this technique encodes. Probabilistic programming gives us posteriors over the full space of execution traces covering the entire latent structure of the simulator, which we show at different levels of detail in Figure 5. Table 2 provides several examples of how the actual addresses (a_t, i_t) look like within C++, allowing us to pinpoint all individual nodes in the codebase where the model behaves probabilistically. For instance, this approach gives us the ability to inspect aspects such as the chain of particle decays and interactions within the detector that led to particular posterior predictions, mirroring the standard task of event reconstruction [74]. This capability is not present in inference techniques that do not have access to the simulator, such as those solely based on neural networks.

6 Conclusions

Our work is the first step in subsuming the vast existing body of scientific simulators, which are essentially accurate generative models with decades of development behind them in many instances, into a universal probabilistic programming framework. The ability to scale probabilistic programming to large-scale simulators is of fundamental importance to the field of probabilistic programming and the wider modeling community. It is a hard problem that requires innovations in many areas such as model–inference engine interface, handling of priors with long tails and rejection sampling routines, addressing schemes, and IC network architecture, which make it difficult to cover in depth in a single paper. A main limitation of the introduced technique, currently, is the need for domain expert decisions in marking regions of codebase as controlled (only needed for the IC engine, and not needed for MCMC), which can potentially be automated in future work.

Our advancement allows one to perform model-based machine learning with interpretability, meaning that we understand the exact processes behind how the predictions are produced and the uncertainty in each prediction. With this novel framework providing a clearly defined interface between existing scientific simulators and probabilistic machine learning techniques, we expect to influence both communities to perform research at the intersection of science and machine learning.

Acknowledgments

Baydin and Wood are supported under DARPA PPAML through the U.S. AFRL under Cooperative Agreement FA8750-14-2-0006, Sub Award number 61160290-111668. Gram-Hansen is supported by the UK EPSRC CDT in Autonomous Inteligent Machines and Systems. Cranmer, Louppe, and Heinrich are supported through NSF ACI-1450310, PHY-1505463, PHY- 1205376, and the Moore-Sloan Data Science Environment at NYU. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a DOE Office of Science User Facility supported by the Office of Science of the U.S.Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] Alexander Heinecke, Alexander Breuer, Sebastian Rettenberger, Michael Bader, Alice-Agnes Gabriel, Christian Pelties, Arndt Bode, William Barth, Xiang-Ke Liao, Karthikeyan Vaidyanathan, et al. Petascale high order dynamic rupture earthquake simulations on heterogeneous supercomputers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 3–14. IEEE Press, 2014.

[2] Eirik Endeve, Christian Y Cardall, Reuben D Budiardja, Samuel W Beck, Alborz Bejnood, Ross J Toedte, Anthony Mezzacappa, and John M Blondin. Turbulent magnetic field amplification from spiral SASI modes: implications for core-collapse supernovae and proto-neutron star magnetization. *The Astrophysical Journal*, 751(1):26, 2012.

[3] Marco Raberto, Silvano Cincotti, Sergio M. Focardi, and Michele Marchesi. Agent-based simulation of a financial market. *Physica A: Statistical Mechanics and its Applications*, 299 (1):319 – 327, 2001. ISSN 0378-4371. doi: 10.1016/S0378-4371(01)00312-0. Application of Physics in Economic Modelling.

[4] Paris Perdikaris, Leopold Grinberg, and George Em Karniadakis. Multiscale modeling and simulation of brain blood flow. *Physics of Fluids*, 28(2):021304, 2016.

[5] Florian Hartig, Justin M Calabrese, Björn Reineking, Thorsten Wiegand, and Andreas Huth. Statistical inference for stochastic simulation models–theory and application. *Ecology Letters*, 14(8):816–827, 2011.

[6] Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation. *Journal of Machine Learning Research*, 17(205):1–37, 2016.

[7] George Papamakarios, Theo Pavlakou, and Iain Murray. Masked autoregressive flow for density estimation. *arXiv preprint arXiv:1705.07057*, 2017.

[8] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

[9] Diederik P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4743–4751. Curran Associates, Inc., 2016.

[10] Kyle Cranmer, Juan Pavez, and Gilles Louppe. Approximating likelihood ratios with calibrated discriminative classifiers. *arXiv preprint arXiv:1506.02169*, 2015.

[11] Ritabrata Dutta, Jukka Corander, Samuel Kaski, and Michael U Gutmann. Likelihood-free inference by penalised logistic regression. *arXiv preprint arXiv:1611.10242*, 2016.

[12] Richard David Wilkinson. Approximate Bayesian computation (ABC) gives exact results under the assumption of model error. *Statistical Applications in Genetics and Molecular Biology*, 12 (2):129–141.

[13] Mikael Sunnåker, Alberto Giovanni Busetto, Elina Numminen, Jukka Corander, Matthieu Foll, and Christophe Dessimoz. Approximate Bayesian computation. *PLoS Computational Biology*, 9(1):e1002803, 2013.

[14] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.

[15] Andrew D Gordon, Thomas A Henzinger, Aditya V Nori, and Sriram K Rajamani. Probabilistic programming. In *Proceedings of the Future of Software Engineering*, pages 167–181. ACM, 2014.

[16] Tuan Anh Le, Atılım Güneş Baydin, and Frank Wood. Inference compilation and universal probabilistic programming. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, volume 54 of *Proceedings of Machine Learning Research*, pages 1338–1348, Fort Lauderdale, FL, USA, 2017. PMLR.

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

[18] Arnaud Doucet and Adam M Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. *Handbook of Nonlinear Filtering*, 12(656-704):3, 2009.

[19] G. Aad, T. Abajyan, B. Abbott, J. Abdallah, S. Abdel Khalek, A. A. Abdelalim, O. Abdinov, R. Aben, B. Abi, M. Abolins, and et al. Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. *Physics Letters B*, 716: 1–29, September 2012. doi: 10.1016/j.physletb.2012.08.020.

[20] S. Chatrchyan, V. Khachatryan, A. M. Sirunyan, A. Tumasyan, W. Adam, E. Aguilo, T. Bergauer, M. Dragicevic, J. Erö, C. Fabjan, and et al. Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC. *Physics Letters B*, 716:30–61, September 2012. doi: 10.1016/j.physletb.2012.08.021.

[21] T. Gleisberg, Stefan. Hoeche, F. Krauss, M. Schonherr, S. Schumann, F. Siegert, and J. Winter. Event generation with SHERPA 1.1. *Journal of High Energy Physics*, 02:007, 2009. doi: 10.1088/1126-6708/2009/02/007.

[22] J. Allison, K. Amako, J. Apostolakis, P. Arce, M. Asai, T. Aso, E. Bagli, A. Bagulya, S. Banerjee, G. Barrand, B.R. Beck, A.G. Bogdanov, D. Brandt, J.M.C. Brown, H. Burkhardt, Ph. Canal, D. Cano-Ott, S. Chauvie, K. Cho, G.A.P. Cirrone, G. Cooperman, M.A. Cortés-Giraldo, G. Cosmo, G. Cuttone, G. Depaola, L. Desorgher, X. Dong, A. Dotti, V.D. Elvira, G. Folger, Z. Francis, A. Galoyan, L. Garnier, M. Gayer, K.L. Genser, V.M. Grichine, S. Guatelli, P. Guèye, P. Gumplinger, A.S. Howard, I. Hřivnáčová, S. Hwang, S. Incerti, A. Ivanchenko, V.N. Ivanchenko, F.W. Jones, S.Y. Jun, P. Kaitaniemi, N. Karakatsanis, M. Karamitros, M. Kelsey, A. Kimura, T. Koi, H. Kurashige, A. Lechner, S.B. Lee, F. Longo, M. Maire, D. Mancusi, A. Mantero, E. Mendoza, B. Morgan, K. Murakami, T. Nikitina, L. Pandola, P. Paprocki, J. Perl, I. Petrović, M.G. Pia, W. Pokorski, J.M. Quesada, M. Raine, M.A. Reis, A. Ribon, A. Ristić Fira, F. Romano, G. Russo, G. Santin, T. Sasaki, D. Sawkey, J.I. Shin, I.I. Strakovsky, A. Taborda, S. Tanaka, B. Tomé, T. Toshito, H.N. Tran, P.R. Truscott, L. Urban, V. Uzhinsky, J.M. Verbeke, M. Verderi, B.L. Wendt, H. Wenzel, D.H. Wright, D.M. Wright, T. Yamashita, J. Yarba, and H. Yoshida. Recent developments in GEANT4. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 835 (Supplement C):186 – 225, 2016. ISSN 0168-9002. doi: 10.1016/j.nima.2016.06.125.

[23] Torbjörn Sjöstrand, Stephen Mrenna, and Peter Skands. Pythia 6.4 physics and manual. *Journal of High Energy Physics*, 2006(05):026, 2006.

[24] Manuel Bähr, Stefan Gieseke, Martyn A Gigg, David Grellscheid, Keith Hamilton, Oluseyi Latunde-Dada, Simon Plätzer, Peter Richardson, Michael H Seymour, Alexander Sherstnev, et al. Herwig++ physics and manual. *The European Physical Journal C*, 58(4):639–707, 2008.

[25] Johan Alwall, R Frederix, S Frixione, V Hirschi, Fabio Maltoni, Olivier Mattelaer, H-S Shao, T Stelzer, P Torrielli, and M Zaro. The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations. *Journal of High Energy Physics*, 2014(7):79, 2014.

[26] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[27] Georges Aad et al. Reconstruction of hadronic decay products of tau leptons with the ATLAS experiment. *Eur. Phys. J.*, C76(5):295, 2016. doi: 10.1140/epjc/s10052-016-4110-0.

[28] S Agapiou, O Papaspiliopoulos, D Sanz-Alonso, and AM Stuart. Importance sampling: Intrinsic dimension and computational cost. *arXiv preprint arXiv:1511.06196*, 2015.

[29] Walter R Gilks, Sylvia Richardson, and David Spiegelhalter. *Markov chain Monte Carlo in practice*. CRC press, 1995.

[30] David Wingate and Theophane Weber. Automated variational inference in probabilistic programming. *arXiv preprint arXiv:1301.1299*, 2013.

[31] Diederik P Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[32] Matthew D Hoffman and Andrew Gelman. The No-U-turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15(1):1593–1623, 2014.

[33] Max Welling and Yee W Teh. Bayesian learning via stochastic gradient langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688, 2011.

[34] Samuel J Gershman and Noah D Goodman. Amortized inference in probabilistic reasoning. In *Proceedings of the 36th Annual Conference of the Cognitive Science Society*, 2014.

[35] David Wingate, Andreas Stuhlmueller, and Noah Goodman. Lightweight implementations of probabilistic programming languages via transformational compilation. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 770–778, 2011.

[36] Noah Goodman, Vikash Mansinghka, Daniel M Roy, Keith Bonawitz, and Joshua B Tenenbaum. Church: a language for generative models. *arXiv preprint arXiv:1206.3255*, 2012.

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

[38] Bingham Eli, Jonathan P Chen, Martin Jankowiak, Theofanis Karaletsos, Fritz Obermeyer, Neeraj Pradhan, Rohit Singh, Paul Szerlip, and Noah Goodman. Pyro: Deep probabilistic programming. `https://github.com/uber/pyro`, 2017.

[39] N. Siddharth, Brooks Paige, Jan-Willem van de Meent, Alban Desmaison, Noah D. Goodman, Pushmeet Kohli, Frank Wood, and Philip Torr. Learning disentangled representations with semi-supervised deep generative models. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5927–5937. Curran Associates, Inc., 2017.

[40] Dustin Tran, Matthew D Hoffman, Rif A Saurous, Eugene Brevdo, Kevin Murphy, and David M Blei. Deep probabilistic programming. *arXiv preprint arXiv:1701.03757*, 2017.

[41] Andrew Gelman, Daniel Lee, and Jiqiang Guo. Stan: A Probabilistic Programming Language for Bayesian Inference and Optimization. *Journal of Educational and Behavioral Statistics*, 40 (5):530–543, 2015.

[42] W Lampl, S Laplace, D Lelas, P Loch, H Ma, S Menke, S Rajagopalan, D Rousseau, S Snyder, and G Unal. Calorimeter Clustering Algorithms: Description and Performance. Technical Report ATL-LARG-PUB-2008-002. ATL-COM-LARG-2008-003, CERN, Geneva, Apr 2008. URL `https://cds.cern.ch/record/1099735`.

[43] K. Kondo. Dynamical Likelihood Method for Reconstruction of Events With Missing Momentum. 1: Method and Toy Models. *J. Phys. Soc. Jap.*, 57:4126–4140, 1988. doi: 10.1143/JPSJ.57.4126.

[44] V. M. Abazov et al. A precision measurement of the mass of the top quark. *Nature*, 429: 638–642, 2004. doi: 10.1038/nature02589.

[45] Pierre Artoisenet and Olivier Mattelaer. MadWeight: Automatic event reweighting with matrix elements. *PoS*, CHARGED2008:025, 2008.

[46] Yanyan Gao, Andrei V. Gritsan, Zijin Guo, Kirill Melnikov, Markus Schulze, and Nhan V. Tran. Spin determination of single-produced resonances at hadron colliders. *Phys. Rev.*, D81:075022, 2010. doi: 10.1103/PhysRevD.81.075022.

[47] J. Alwall, A. Freitas, and O. Mattelaer. The Matrix Element Method and QCD Radiation. *Phys. Rev.*, D83:074010, 2011. doi: 10.1103/PhysRevD.83.074010.

[48] Sara Bolognesi, Yanyan Gao, Andrei V. Gritsan, Kirill Melnikov, Markus Schulze, Nhan V. Tran, and Andrew Whitbeck. On the spin and parity of a single-produced resonance at the LHC. *Phys. Rev.*, D86:095031, 2012. doi: 10.1103/PhysRevD.86.095031.

[49] Paul Avery et al. Precision studies of the Higgs boson decay channel $H \to ZZ \to 4l$ with MEKD. *Phys. Rev.*, D87(5):055006, 2013. doi: 10.1103/PhysRevD.87.055006.

[50] Jeppe R. Andersen, Christoph Englert, and Michael Spannowsky. Extracting precise Higgs couplings by using the matrix element method. *Phys. Rev.*, D87(1):015019, 2013. doi: 10.1103/PhysRevD.87.015019.

[51] John M. Campbell, R. Keith Ellis, Walter T. Giele, and Ciaran Williams. Finding the Higgs boson in decays to $Z\gamma$ using the matrix element method at Next-to-Leading Order. *Phys. Rev.*, D87(7):073005, 2013. doi: 10.1103/PhysRevD.87.073005.

[52] Pierre Artoisenet, Priscila de Aquino, Fabio Maltoni, and Olivier Mattelaer. Unravelling $t\bar{t}h$ via the Matrix Element Method. *Phys. Rev. Lett.*, 111(9):091802, 2013. doi: 10.1103/PhysRevLett.111.091802.

[53] James S. Gainer, Joseph Lykken, Konstantin T. Matchev, Stephen Mrenna, and Myeonghun Park. The Matrix Element Method: Past, Present, and Future. In *Proceedings, 2013 Community Summer Study on the Future of U.S. Particle Physics: Snowmass on the Mississippi (CSS2013): Minneapolis, MN, USA, July 29-August 6, 2013*, 2013. URL `http://inspirehep.net/record/1242444/files/arXiv:1307.3546.pdf`.

[54] Doug Schouten, Adam DeAbreu, and Bernd Stelzer. Accelerated Matrix Element Method with Parallel Computing. *Comput. Phys. Commun.*, 192:54–59, 2015. doi: 10.1016/j.cpc.2015.02.020.

[55] Till Martini and Peter Uwer. Extending the Matrix Element Method beyond the Born approximation: Calculating event weights at next-to-leading order accuracy. *JHEP*, 09:083, 2015. doi: 10.1007/JHEP09(2015)083.

[56] Andrei V. Gritsan, Raoul Röntsch, Markus Schulze, and Meng Xiao. Constraining anomalous Higgs boson couplings to the heavy flavor fermions using matrix element techniques. *Phys. Rev.*, D94(5):055023, 2016. doi: 10.1103/PhysRevD.94.055023.

[57] Till Martini and Peter Uwer. The Matrix Element Method at next-to-leading order QCD for hadronic collisions: Single top-quark production at the LHC as an example application. 2017.

[58] Davison E. Soper and Michael Spannowsky. Finding physics signals with shower deconstruction. *Phys. Rev.*, D84:074002, 2011. doi: 10.1103/PhysRevD.84.074002.

[59] Johann Brehmer, Kyle Cranmer, Gilles Louppe, and Juan Pavez. A Guide to Constraining Effective Field Theories with Machine Learning. 2018.

[60] Lily Asquith et al. Jet Substructure at the Large Hadron Collider : Experimental Review. 2018.

[61] Gulrukh Khattak Vitória Pacela Maurizio Pierini Jean-Roch Vlimant Maria Spiropulu Wei Wei Matt Zhang Benjamin Hooberman, Amir Farbin and Sofia Vallecorsa. Calorimetry with Deep Learning: Particle Classification, Energy Regression, and Simulation for High-Energy Physics, 2017. Deep Learning in Physical Sciences (NIPS workshop). `https://dl4physicalsciences.github.io/files/nips_dlps_2017_15.pdf`.

[62] Gregor Kasieczka. Boosted Top Tagging Method Overview. In *10th International Workshop on Top Quark Physics (TOP2017) Braga, Portugal, September 17-22, 2017*, 2018. URL `http://inspirehep.net/record/1647961/files/arXiv:1801.04180.pdf`.

[63] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS 2017 Autodiff Workshop: The Future of Gradient-based Machine Learning Software and Techniques, Long Beach, CA, US, December 9, 2017*, 2017.

[64] Atılım Güneş Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research (JMLR)*, 18(153):1–43, 2018. URL `http://jmlr.org/papers/v18/17-468.html`.

[65] Tuan Anh Le. Inference for higher order probabilistic programs. *Masters thesis, University of Oxford*, 2015.

[66] M Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.

[67] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[68] Ponnambalam Kumaraswamy. A generalized probability density function for double-bounded random processes. *Journal of Hydrology*, 46(1-2):79–88, 1980.

[69] Pablo A Mitnik and Sunyoung Baek. The kumaraswamy distribution: median-dispersion re-parameterizations for regression modeling and simulation-based estimation. *Statistical Papers*, 54(1):177–192, 2013.

[70] Pieter Hintjens. *ZeroMQ: messaging for many applications*. O'Reilly Media, Inc., 2013.

[71] Walter R Gilks and Pascal Wild. Adaptive rejection sampling for gibbs sampling. *Applied Statistics*, pages 337–348, 1992.

[72] Frank Krauss. Matrix elements and parton showers in hadronic interactions. *Journal of High Energy Physics*, 2002(08):015, 2002.

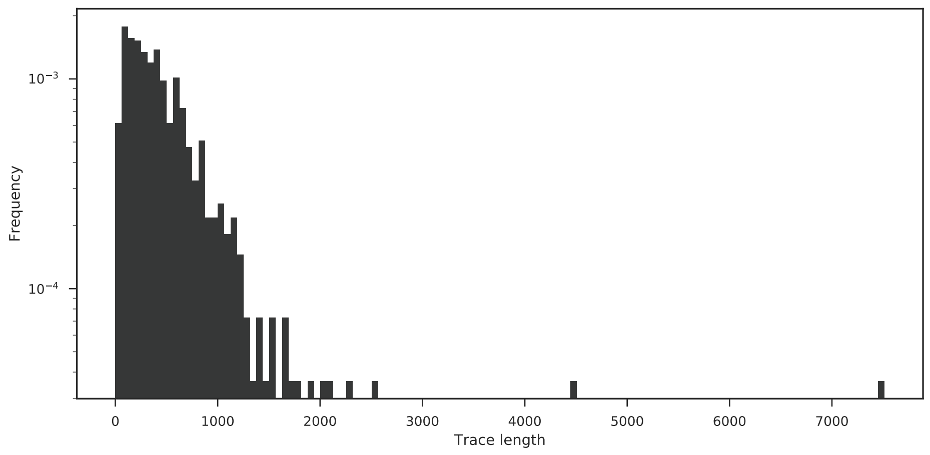
[73] C Patrignani, DH Weinberg, CL Woody, RS Chivukula, O Buchmueller, Yu V Kuyanov, E Blucher, S Willocq, A Höcker, C Lippmann, et al. Review of particle physics. *Chinese Physics C*, 40:100001, 2016.

[74] Rainer Mankel. Pattern recognition and event reconstruction in particle physics experiments. *Reports on Progress in Physics*, 67(4):553, 2004.

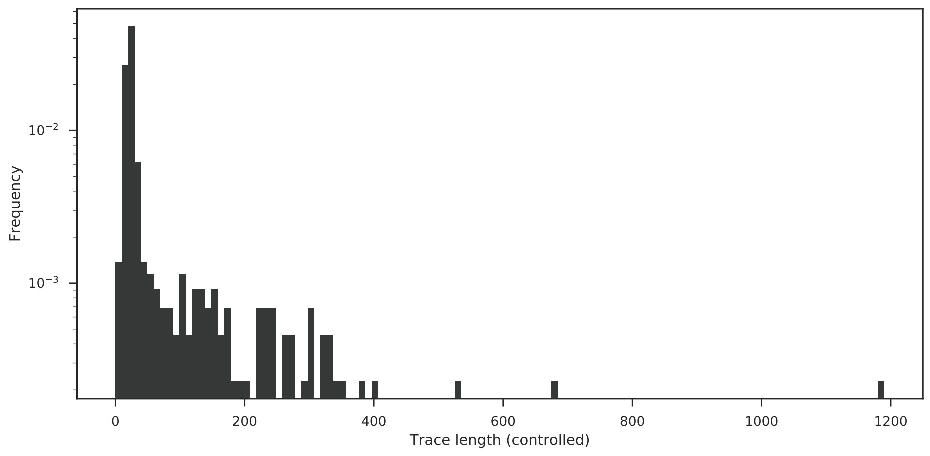
A Appendix

Table 1: Trace types encountered in the τ lepton decay model, identified according to the address sequence contained in each trace. Only the first 36 most frequent trace types are shown out of a total of 440 types encountered over 1,602,880 executions. Note that even when the address sequence is the same, the sampled values in each trace of the same type would be different.

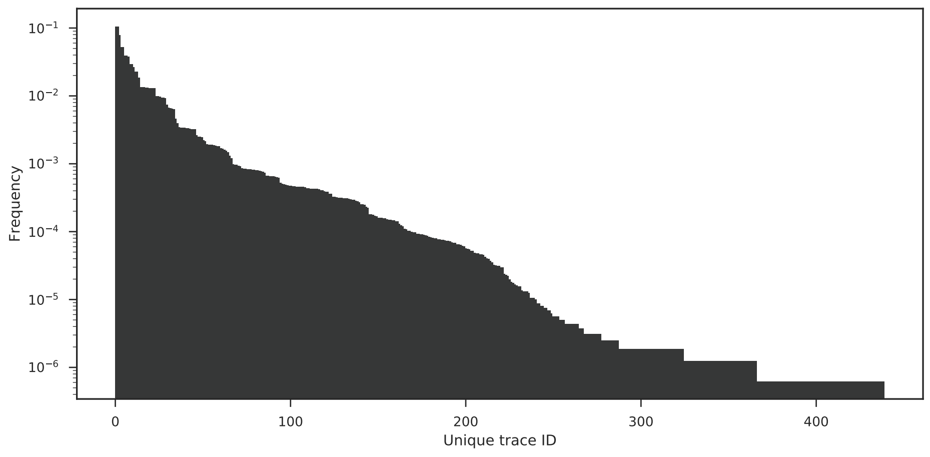
Freq.	Length	Addresses (showing controlled only)
0.106	72	A1, A2, A3, A5, A6, A32, A33, A31
0.105	41	A1, A2, A3, A5, A6, A499, A31
0.078	1,780	A1, A2, A3, A5, A6, A7, A8, A9, A10, A31
0.053	188	A1, A2, A3, A5, A6, A7, A8, A9, A10, A17, A18, A26, A31
0.053	100	A1, A2, A3, A5, A6, A7, A8, A9, A10, A17, A18, A99, A100, A101, A102, A31
0.039	56	A1, A2, A3, A5, A6, A499, A17, A18, A26, A31
0.039	592	A1, A2, A3, A5, A6, A499, A17, A18, A99, A100, A101, A102, A31
0.038	162	A1, A2, A3, A5, A6, A7, A8, A9, A10, A17, A500, A99, A100, A101, A102, A31
0.030	240	A1, A2, A3, A5, A6, A7, A8, A9, A10, A17, A18, A20, A21, A41, A42, A26, A99, A100, A101, A102, A31
0.029	836	A1, A2, A3, A5, A6, A7, A8, A9, A10, A17, A18, A20, A21, A41, A42, A99, A100, A101, A102, A26, A31
0.027	643	A1, A2, A3, A5, A6, A7, A8, A9, A10, A17, A507, A99, A100, A101, A102, A31
0.023	135	A1, A2, A3, A5, A6, A7, A8, A9, A10, A17, A18, A20, A21, A41, A42, A44, A45, A26, A99, A100, A101, A102, A31
0.023	485	A1, A2, A3, A5, A6, A7, A8, A9, A10, A17, A18, A20, A21, A41, A42, A44, A45, A99, A100, A101, A102, A26, A31
0.019	316	A1, A2, A3, A5, A6, A32, A33, A17, A500, A99, A100, A101, A102, A31
0.014	68	A1, A2, A3, A5, A6, A7, A8, A9, A10, A17, A18, A20, A21, A26, A99, A100, A101, A102, A31
0.013	422	A1, A2, A3, A5, A6, A32, A33, A17, A500, A20, A1496, A99, A100, A101, A102, A31
0.013	298	A1, A2, A3, A5, A6, A32, A33, A17, A18, A20, A21, A26, A31
0.013	283	A1, A2, A3, A5, A6, A32, A33, A17, A18, A20, A21, A26, A99, A100, A101, A102, A31
0.013	608	A1, A2, A3, A5, A6, A32, A33, A17, A18, A20, A21, A99, A100, A101, A102, A31
0.013	424	A1, A2, A3, A5, A6, A7, A8, A9, A10, A17, A18, A20, A21, A99, A100, A101, A102, A31
0.013	50	A1, A2, A3, A5, A6, A7, A8, A9, A10, A17, A18, A20, A21, A26, A31
0.013	204	A1, A2, A3, A5, A6, A7, A8, A9, A10, A17, A18, A20, A21, A99, A100, A101, A102, A26, A31
0.013	252	A1, A2, A3, A5, A6, A32, A33, A17, A18, A20, A21, A99, A100, A101, A102, A26, A31
0.010	234	A1, A2, A3, A5, A6, A7, A8, A9, A10, A17, A18, A20, A21, A41, A42, A99, A100, A101, A102, A31
0.010	58	A1, A2, A3, A5, A6, A7, A8, A9, A10, A17, A18, A20, A21, A41, A42, A26, A31
0.010	502	A1, A2, A3, A5, A6, A499, A17, A18, A20, A1496, A99, A100, A101, A102, A31
0.009	216	A1, A2, A3, A5, A6, A499, A17, A500, A20, A21, A99, A100, A101, A102, A31
0.009	1,053	A1, A2, A3, A5, A6, A499, A17, A18, A20, A1496, A26, A99, A100, A101, A102, A31
0.009	800	A1, A2, A3, A5, A6, A499, A17, A500, A20, A21, A99, A100, A101, A102, A26, A31
0.007	92	A1, A2, A3, A5, A6, A7, A8, A9, A10, A17, A500, A26, A31
0.007	32	A1, A2, A3, A5, A6, A7, A8, A9, A10, A17, A507, A26, A31
0.007	78	A1, A2, A3, A5, A6, A7, A8, A9, A10, A17, A507, A99, A100, A101, A102, A510, A511, A898, A31
0.006	120	A1, A2, A3, A5, A6, A7, A8, A9, A10, A17, A507, A20, A508, A99, A100, A101, A102, A31
0.006	118	A1, A2, A3, A5, A6, A7, A8, A9, A10, A17, A507, A99, A100, A101, A102, A510, A511, A882, A883, A884, A885, A31
0.005	553	A1, A2, A3, A5, A6, A7, A8, A9, A10, A17, A500, A20, A21, A41, A42, A99, A100, A101, A102, A26, A31



(a) Distribution of trace lengths (all addresses). Min: 13, max: 7,514, mean: 383.58.



(b) Distribution of trace lengths (controlled addresses only). Min: 6, max: 1,190, mean: 13.61.

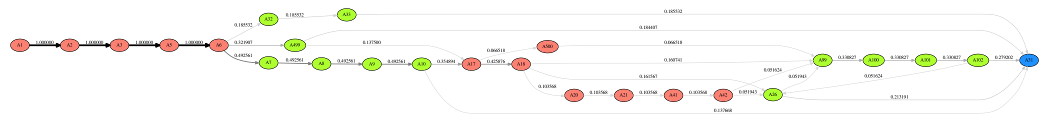


(c) Distribution of trace types, sorted in decreasing frequency.

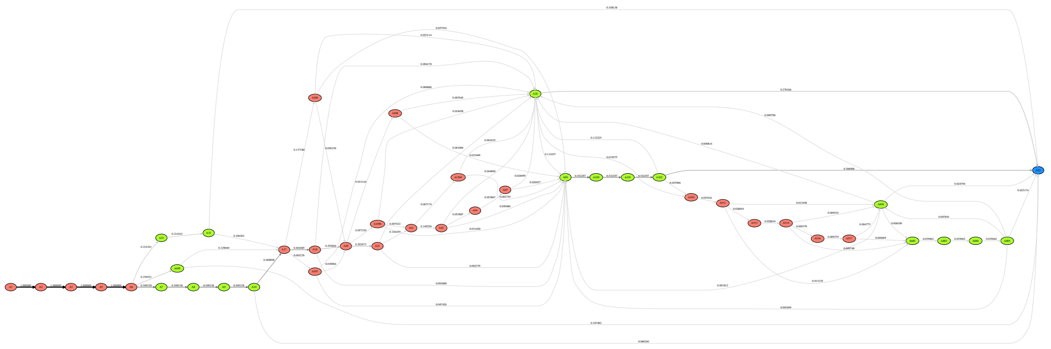
Figure 4: Probabilistic trace statistics in the τ decay model automatically extracted from SHERPA executions via PPX.

Table 2: Addresses in the τ lepton decay problem (C++). Only the first 6 addresses are shown out of a total of 24,382 addresses encountered over 1,602,880 executions.

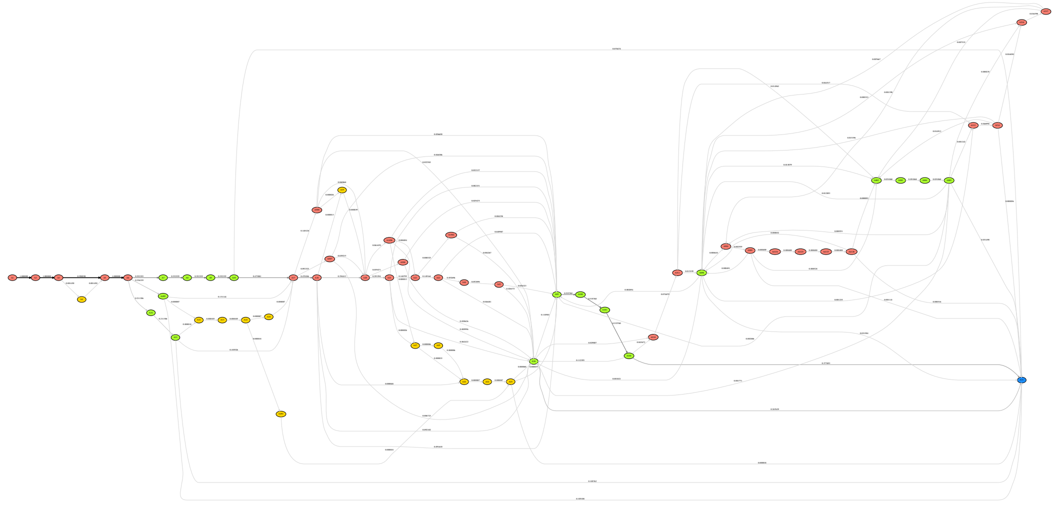
Address ID	Full address
A1	[forward(xt:: xarray_container<xt:: uvector<double, std:: allocator<double> >, (xt:: layout_type)1, xt:: svector<unsigned long, 4ul, std:: allocator<unsigned long>, true>, xt:: xtensor_expression_tag>)+0x5f; SherpaGenerator:: Generate()+0x36; SHERPA:: Sherpa:: GenerateOneEvent(bool)+0x2fa; SHERPA:: Event_Handler:: GenerateEvent(SHERPA:: eventtype:: code)+0x44d; SHERPA:: Event_Handler:: GenerateHadronDecayEvent(SHERPA:: eventtype:: code&)+0x45f; ATOOLS:: Random:: Get(bool, bool)+0x1d5; probprog_RNG:: Get(bool, bool)+0xf9]_Uniform_1
A2	[forward(xt:: xarray_container<xt:: uvector<double, std:: allocator<double> >, (xt:: layout_type)1, xt:: svector<unsigned long, 4ul, std:: allocator<unsigned long>, true>, xt:: xtensor_expression_tag>)+0x5f; SherpaGenerator:: Generate()+0x36; SHERPA:: Sherpa:: GenerateOneEvent(bool)+0x2fa; SHERPA:: Event_Handler:: GenerateEvent(SHERPA:: eventtype:: code)+0x44d; SHERPA:: Event_Handler:: GenerateHadronDecayEvent(SHERPA:: eventtype:: code&)+0x477; ATOOLS:: Random:: Get(bool, bool)+0x1d5; probprog_RNG:: Get(bool, bool)+0xf9]_Uniform_1
A3	[forward(xt:: xarray_container<xt:: uvector<double, std:: allocator<double> >, (xt:: layout_type)1, xt:: svector<unsigned long, 4ul, std:: allocator<unsigned long>, true>, xt:: xtensor_expression_tag>)+0x5f; SherpaGenerator:: Generate()+0x36; SHERPA:: Sherpa:: GenerateOneEvent(bool)+0x2fa; SHERPA:: Event_Handler:: GenerateEvent(SHERPA:: eventtype:: code)+0x44d; SHERPA:: Event_Handler:: GenerateHadronDecayEvent(SHERPA:: eventtype:: code&)+0x48f; ATOOLS:: Random:: Get(bool, bool)+0x1d5; probprog_RNG:: Get(bool, bool)+0xf9]_Uniform_1
A4	[forward(xt:: xarray_container<xt:: uvector<double, std:: allocator<double> >, (xt:: layout_type)1, xt:: svector<unsigned long, 4ul, std:: allocator<unsigned long>, true>, xt:: xtensor_expression_tag>)+0x5f; SherpaGenerator:: Generate()+0x36; SHERPA:: Sherpa:: GenerateOneEvent(bool)+0x2fa; SHERPA:: Event_Handler:: GenerateEvent(SHERPA:: eventtype:: code)+0x44d; SHERPA:: Event_Handler:: GenerateHadronDecayEvent(SHERPA:: eventtype:: code&)+0x8f4; ATOOLS:: Particle:: SetTime()+0xd; ATOOLS:: Flavour:: GenerateLifeTime() const+0x35; ATOOLS:: Random:: Get()+0x18b; probprog_RNG:: Get()+0xde]_Uniform_1
A5	[forward(xt:: xarray_container<xt:: uvector<double, std:: allocator<double> >, (xt:: layout_type)1, xt:: svector<unsigned long, 4ul, std:: allocator<unsigned long>, true>, xt:: xtensor_expression_tag>)+0x5f; SherpaGenerator:: Generate()+0x36; SHERPA:: Sherpa:: GenerateOneEvent(bool)+0x2fa; SHERPA:: Event_Handler:: GenerateEvent(SHERPA:: eventtype:: code)+0x44d; SHERPA:: Event_Handler:: GenerateHadronDecayEvent(SHERPA:: eventtype:: code&)+0x982; SHERPA:: Event_Handler:: IterateEventPhases(SHERPA:: eventtype:: code&, double&)+0x1d2; SHERPA:: Hadron_Decays:: Treat(ATOOLS:: Blob_List*, double&)+0x975; SHERPA:: Decay_Handler_Base:: TreatInitialBlob(ATOOLS:: Blob*, METOOLS:: Amplitude2_Tensor*, std:: vector<ATOOLS:: Particle*, std:: allocator<ATOOLS:: Particle*> > const&)+0x1ab1; SHERPA:: Hadron_Decay_Handler:: CreateDecayBlob(ATOOLS:: Particle*)+0x4cd; PHASIC:: Decay_Table:: Select() const+0x76e; ATOOLS:: Random:: Get(bool, bool)+0x1d5; probprog_RNG:: Get(bool, bool)+0xf9]_Uniform_1
A6	[forward(xt:: xarray_container<xt:: uvector<double, std:: allocator<double> >, (xt:: layout_type)1, xt:: svector<unsigned long, 4ul, std:: allocator<unsigned long>, true>, xt:: xtensor_expression_tag>)+0x5f; SherpaGenerator:: Generate()+0x36; SHERPA:: Sherpa:: GenerateOneEvent(bool)+0x2fa; SHERPA:: Event_Handler:: GenerateEvent(SHERPA:: eventtype:: code)+0x44d; SHERPA:: Event_Handler:: GenerateHadronDecayEvent(SHERPA:: eventtype:: code&)+0x982; SHERPA:: Event_Handler:: IterateEventPhases(SHERPA:: eventtype:: code&, double&)+0x1d2; SHERPA:: Hadron_Decays:: Treat(ATOOLS:: Blob_List*, double&)+0x975; SHERPA:: Decay_Handler_Base:: TreatInitialBlob(ATOOLS:: Blob*, METOOLS:: Amplitude2_Tensor*, std:: vector<ATOOLS:: Particle*, std:: allocator<ATOOLS:: Particle*> > const&)+0x1ab1; SHERPA:: Hadron_Decay_Handler:: CreateDecayBlob(ATOOLS:: Particle*)+0x4cd; PHASIC:: Decay_Table:: Select() const+0x9d7; ATOOLS:: Random:: GetCategorical(std:: vector<double, std:: allocator<double> > const&, bool, bool)+0x1a5; probprog_RNG:: GetCategorical(std:: vector<double, std:: allocator<double> > const&, bool, bool)+0x111]_Categorical(length_categories:38)_1



(a) Latent probabilistic structure of the 10 most frequent trace types.



(b) Latent probabilistic structure of the 100 most frequent traces types.



(c) Latent probabilistic structure of the 250 most frequent traces types.

Figure 5: Interpretability of the latent structure of the τ lepton decay process, automatically extracted from SHERPA executions via PPX. Showing model structure with increasing detail by taking an increasing number of most common trace types into account. Note that the flow is probabilistic at the shown nodes and deterministic along edges. Node labels denote address IDs (A1, A2, etc.) that correspond to uniquely identifiable parts of model execution such as those in Table 2. Addresses A1, A2, A3 correspond to momenta p_x, p_y, p_z, and A6 corresponds to the decay channel in Figure 3. Edge labels denote the overall frequency an edge is taken. *Red:* controlled node; *green*: controlled node with replacement ("rejection sampling mode"); *blue*: observed node; *yellow*: uncontrolled node.