Published for SISSA by Springer

Received: January 18, 2024
Revised: February 23, 2024
Accepted: February 29, 2024
Published: March 14, 2024

Accelerating HEP simulations with Neural Importance Sampling

Nicolas Deutschmann[a] **and Niklas Götz**[b,c]

[a] *IBM Research GmbH,*
Säumerstrasse 4, 8803 Rüschlikon, Switzerland
[b] *Department of Physics, Institute for Theoretical Physics, Goethe University Frankfurt,*
Max-von-Laue-Straße 1, 60438 Frankfurt, Germany
[c] *Frankfurt Institute for Advanced Studies,*
Ruth-Moufang-Strasse 1, 60438 Frankfurt am Main, Germany

E-mail: nicolas.deutschmann@gmail.com, goetz@itp.uni-frankfurt.de

Abstract: Many high-energy-physics (HEP) simulations for the LHC rely on Monte Carlo using importance sampling by means of the VEGAS algorithm. However, complex high-precision calculations have become a challenge for the standard toolbox, as this approach suffers from poor performance in complex cases. As a result, there has been keen interest in HEP for modern machine learning to power adaptive sampling. While previous studies have shown the potential of normalizing-flow-powered neural importance sampling (NIS) over VEGAS, there remains a gap in accessible tools tailored for non-experts. In response, we introduce ZüNIS, a fully automated NIS library designed to bridge this divide, while at the same time providing the infrastructure to customise the algorithm for dealing with challenging tasks. After a general introduction on NIS, we first show how to extend the original formulation of NIS to reuse samples over multiple gradient steps while guaranteeing a stable training, yielding a significant improvement for slow functions. Next, we introduce the structure of the library, which can be used by non-experts with minimal effort and is extensivly documented, which is crucial to become a mature tool for the wider HEP public. We present systematic benchmark results on both toy and physics examples, and stress the benefit of providing different survey strategies, which allows higher performance in challenging cases. We show that ZüNIS shows high performance on a range of problems with limited fine-tuning.

Keywords: Automation, Higher-Order Perturbative Calculations, Scattering Amplitudes

ArXiv ePrint: 2401.09069

https://doi.org/10.1007/JHEP03(2024)083

JHEP03(2024)083

Contents

JHEP03(2024)083

1 Introduction

High-Energy-Physics (HEP) simulations are at the heart of the Large Hadron Collider (LHC) program for studying the fundamental laws of nature. Most HEP predictins are expressed as expectation values, evaluated numerically as Monte Carlo (MC) integrals. This permits both the integration of the very complex functions and the reproduction of the data selection process by experiments.

Most HEP simulations tools [1–3] perform MC integrals using importance sampling, which allows to adaptively sample points to speed up convergence while keeping independent and identically distributed samples, crucial to reproduce experimental analyses which can only ingest uniformly-weighted data, typically produced by rejection sampling (see appendix A).

The most popular tool to optimize importance sampling in HEP is by far the VEGAS algorithm [4], which fights the curse of dimensionality by assuming no correlations between the variables. While this is rarely the case in general, a good understanding of the integrand function can help significantly. Indeed optimized parametrizations using multichannelling [5–7] have become bread-and-butter tools for HEP event generation simulators, with good success for leading-order (LO) calculations. However, as simulations get more complex, either by having more complex final states or by including higher orders in perturbation theory, performance degrades fast.

While newer approaches to adaptive importance sampling have been developed, for example Population Monte Carlo [8–11] and its extensions [12–16], and have been successfully applied to other fields, VEGAS and its recent updates and variants [6, 17–20] remain one of the most commonly used algorithms in practically all modern simulation tools [1, 3, 21]. For example, the WHIZARD event generator [2] uses VAMP [6], which is an improvement of the orginal VEGAS implementation.

We suspect that the reasons for this are twofold. First of all the lack of exchange between the two scientific communities is probably at cause and should probably not to be neglected. Second of all however, the impetus in the HEP community is to move toward more "black box" approaches where little is known of the structure of the integrand. This is one reason why approaches like PMC are less common, as it is sensitive to the initial proposal density [12, 22]. As the main practical goal of this paper is to reduce the reliance of HEP simulations on the careful tuning of integrators, we will focus on comparing our work with the *de facto* HEP standard: the VEGAS algorithm. It is worth noting that alternative adaptive approaches, such as Nested Sampling [23, 24], as employed in the SHERPA event generator [25], also exist. However, for the sake of maintaining focus and as it is the most widespread tool, we will concentrate on comparing our work with VEGAS. Our comparison will be based on one of the most recent implementations [20], which makes use of stratified sampling to boost the performance considerably.

There is much room for investing computational time into improving sampling [26]: modern HEP theoretical calculations are taking epic proportions and can require hours for a single function evaluation [27]. Furthermore, unweighting samples can be extremely inefficient, with upwards of 90% sampled points discarded [28]. More powerful importance sampling algorithms would therefore be a welcome improvement [29, 30].

JHEP03(2024)083

Several approaches have been explored to tackle the challenge of efficient sampling in high-energy physics simulations. Initial efforts utilized classical neural networks for sampling, but often encountered prohibitive computational costs [31–33]. Another avenue of research has been the adoption of generative models, such as generative adversarial networks (GANs), which have shown promise in accelerating sampling speed significantly [34–39]. While such approaches do improve sampling speed by a large factor, they have major limitations. In particular, they have no theoretical guarantees of providing a correct answer on average [40, 41].

On the other hand, tools like SHERPA offer an unbiased NN-powered approach for unweighting [42]. It is worth noting that reweighting techniques, such as the recently developed Deeply Conditionalized Transporter Networks (DCTR) [43–46], have become standard in boosting the efficiency generated samples. Despite these advancements, drawbacks persist. For instance, the quality of information in the reweighted samples is restricted to the training sample [40, 41, 47].

To avoid these disadvantages, our work exploits Neural Importance Sampling (NIS) [48, 49], which relies on normalizing flows and has strong theoretical guarantees.

A number of works have been published on using NIS for LHC simulations [50–55], as well as closely related variations [40, 56], but most studies have focused on preliminary investigation of performance and less on the practical usability of the method. Indeed, training requires function evaluations, which we are trying to minimize and data-efficiency training is therefore an important but often under-appreciated concern. Similarly, stability of training without the need of fine-tuning is essential for non-expert users. Furthermore, few authors have provided easily usable open source code, making the adoption of the technique in the HEP community difficult.

Our contribution to improve this situation can be summarized in three items:

- The introduction of a new adaptive training algorithm for NIS. This permits the re-use of sampled points over multiple gradient descent steps, therefore making NIS much more data efficient, while at the same time guaranteeing stable and reliable training.
- A comparative study on the behaviour of different loss functions and their impact on performance for different processes.
- The introduction of ZüNIS, a `PyTorch`-based library providing robust and usable NIS tools, usable by non-experts. It implements previously-developped ideas as well as our new training procedure and is extensively documented.

2 Background

2.1 Importance sampling as an optimization problem

Importance sampling relies on the interpretation of integrals as expectation values. Indeed, let us consider an integral over a finite volume:

$$I = \int_\Omega dx f(x), \quad \text{where } V(\Omega) = \int_\Omega dx \text{ is finite.} \tag{2.1}$$

JHEP03(2024)083

Let p be a strictly positive probability distribution over Ω, we can re-express our integral as the expectation of a random variable

$$I = \int_\Omega p(x)dx \frac{f(x)}{p(x)} = \mathop{\mathbb{E}}_{X_i \sim p} \frac{1}{N} \sum_{i=1}^{N} \frac{f(X_i)}{p(X_i)}, \tag{2.2}$$

whose mean is indeed I and whose standard deviation is $\frac{\sigma(f,p)}{\sqrt{N}}$, where $\sigma(f,p)$ is the standard deviation of $f(X)/p(X)$ for $X \sim p$:

$$\sigma^2(f,p) = \mathop{\mathbb{E}}_{x \sim p} \left(\left(\frac{f(x)}{p(x)} \right)^2 \right) - I^2. \tag{2.3}$$

The problem statement of importance sampling is to find the probability distribution function p that minimizes the variance of our estimator for a given N. In the case of multichanneling [7, 57], finding the probability distribution function p is replaced by finding weights α_i and mappings g_i such that

$$g(x) = \sum_i^m \alpha_i g_i(x), \quad \sum_i^m \alpha_i = 1 \tag{2.4}$$

and

$$I = \sum_i^m \int dx \alpha_i g_i(x) \frac{f(x)}{g(x)} \tag{2.5}$$

The g_i are often chosen according to prior physics knowledge. As we are interested in a blackbox approach, we do not use different channels but instead try to learn an optimal probability distribution function. In Neural Importance Sampling, we rely on Normalizing Flows to approximate the optimal distribution, which we can optimize using stochastic gradient descent. However, it has been shown that both approaches can be successfully combined [54, 55]. Nevertheless, in this work we focus on achieving improvements with a single channel, in order to minimize the required prior knowldge of the integrand.

2.2 Normalizing flows and coupling cells

Normalizing flows [58–61] provide a way to generate complex probability distribution functions from simpler ones using parametric changes of variables that can be learned to approximate a target distribution. As such, normalizing flows are diffeomorphisms: invertible, (nearly-everywhere) differentiable mappings with a differentiable inverse.

Indeed, if $u \sim p(u)$, then $T(u) = x \sim q(x)$ where

$$q\left(x = T(u)\right) = p(u) \left| J_T(u) \right|^{-1}, \tag{2.6}$$

where J_T is the Jacobian determinant of T:

$$J_T(u) = \det \frac{\partial T_i}{\partial u_j}(u). \tag{2.7}$$

In the world of machine learning, T is called a normalizing flow and is typically part of a parametric family of diffeomorphisms $(T(\cdot, \theta))$ such that gradients $\nabla_\theta J_T$ are tractable.

JHEP03(2024)083

Coupling cell mappings perfectly satisfy this requirement [62–64]: they are neural-network-parametrized bijections whose Jacobian factor can be obtained analytically without backpropagation or expensive determinant calculation. As such, they provide a good candidate for importance sampling as long as they can be trained to learn an unnormalized target function, which is exactly what neural importance sampling proposes.

The coupling cells used in the following implement the transformations proposed in [62–64], which are also used in i-flow and MadNIS. Although all the transformations are implemented, we focus our study on the use piecewise-quadratic layers, as these reach a high level of expressiveness without requiring a too large number of parameters. This is important, as a too high number of transformation parameters becomes expensive to learn.

Other than the architecture of NNs used by MadNIS in [54] and also the architecture chosen by i-flow, we use by default considerably larger NN (rectangular DNN with 8 hidden layers, each with 256 nodes), as the we learn the probability distribution function directly instead of splitting it into multiple channels. This guarantees also enough complexity in order to allow stable training on a wide range of integrands. Although we also use LeakyReLU activation functions, we additionally add an input activation layer which scales the input and removes any offset to increase stability of trainging. Additionally, we implement the optimal masking function proposed by i-flow [50].

2.3 Neural importance sampling

Neural importance sampling was introduced in the context of computer graphics [64] and proposes to use normalizing flows as a family of probability distributions over which to solve the minimization problem of importance sampling.

$$\mathcal{L}(\theta) = \int_\Omega dx \frac{f^2(x)}{p\,(x,\theta)}. \tag{2.8}$$

Of course, to actually do so, one needs to find a way to explicitly evaluate $\mathcal{L}(\theta)$ and the original neural importance sampling approach proposes to approximate it using importance sampling. One needs to be careful that the gradient of the estimator of the loss need not be the estimator of the gradient of the loss. The gradient of the loss can be expressed as

$$\nabla_\theta \mathcal{L}(\theta) = -\int_\Omega dx \frac{f^2(x)}{p\,(x,\theta)} \nabla_\theta \log p\,(x,\theta)\,, \tag{2.9}$$

for which an estimator is proposed as

$$\widehat{\nabla}_\theta \mathcal{L}(\theta) = -\sum_{i=0}^{N} \left(\frac{f(X_i)}{p\,(X_i,\theta)}\right)^2 \nabla_\theta \log p\,(X_i,\theta)\,, \quad X_i \sim p. \tag{2.10}$$

The authors also observed that other loss functions are possible which share the same global minimum as the variance based loss: for example, the Kullback-Leibler divergence D_{KL} between two functions is also minimized when they are equal. Such alternative loss functions are not guaranteed to work for importance sampling, but they prove quite successful in practice. Indeed, for certain processes they are shown to be crucial for optimal performance, as can be seen in section 4.3. After training to minimize the loss estimator of eq. (2.10), the normalizing flows provides a tractable probability distribution function from which to sample points and estimate the integral.

JHEP03(2024)083

3 Concepts and algorithms

In this section we describe the original contributions of this paper. The major conceptual innovation we provide in ZüNIS is a more flexible and data-efficient way of training normalizing flows in the context of importance sampling. This relies on a more rigorous formulation of the connection between the theoretical expression of ideal loss functions in terms of integrals and their practical realisations as random estimators than in previous literature, combined with an adaptive strategy to switch between these realistions. We describe this improvement in section 3.1. We also give a high-level overview of the organisation of the ZüNIS library, which implements this new training procedure.

3.1 Efficient training for importance sampling

In this section, we describe how we train probability distributions within ZüNIS using gradient-based optimizers. While the solution proposed in the original formulation of NIS defined eq. (2.10) works and has been successfully used by i-flow, its main drawback is that it samples points from the same distribution that it tries to optimize. As a result, new points X_i must be sampled from p after every gradient step, which is very inefficient for slow integrands.

Our solution to this problem is to remember that the loss function is an integral, which can be evaluated by importance sampling using any PDF, not only p. We will therefore define an auxiliary probability distribution function $q(x)$, independent from θ, from which we sample to estimate our loss function:

$$\int dx \frac{f(x)^2}{p(x,\theta)} = \mathop{\mathbb{E}}_{x\sim q} \frac{f(x)^2}{p(x,\theta)q(x)}. \tag{3.1}$$

This is the basis for the general method we use for training probability distributions within ZüNIS, described in algorithm 1. Because the sampling distribution is separated from the model to train, the same point sample can be reused for multiple training steps, which is not possible when using eq. (2.10). This approach is similiar to the "buffered training" used in MadNIS [54]. This is particularly important for high-precision particle physics predictions that involve high-order perturbative calculations or complex detector simulations because function evaluations can be extremely costly. We show in section 4, in particular in figure 4 that reusing data indeed has a very significant impact on data efficiency.

After training, q is discarded and the integral is estimated from the optimized p.

The only constraint on q is that it yields a good enough estimate so that gradient steps improve the model. Much like in other applications of neural networks, we have found that stochastic gradient descent can yield good results despite noisy loss estimates. We propose three schemes for q:

- A uniform distribution (`survey_strategy="flat"`)
- A frozen copy of the model, which can be updated once in a while[1] (`survey_strategy="forward"`)

[1]This is inspired by deep-Q learning, where two copies of the value model are used: a frozen one used to sample actions, and a trainable one used to estimate values in the loss function. Here the frozen copy is used to sample points, and the trainable model is used to compute PDF values used in the loss function.

JHEP03(2024)083

Algorithm 1: Backward sampling training in ZüNIS.

Data: Parametric PDF $p(x, \theta)$
Result: Trained PDF $p(x, \theta)$
for *M steps* **do**
  Sample a batch $x_1, \ldots, x_{n_{\text{batch}}}$ from q;
  Compute the sampling PDF values $q(x_i)$;
  Compute function values $f(x_i)$;
  Start tracking gradients with respect to θ;
  for *N steps* **do**
    Compute the PDF values from the parametric PDF $p(x_i, \theta)$;
    Estimate the loss $\hat{L} = \frac{1}{N} \sum_{i=1}^{n_{\text{batch}}} \frac{f(x_i)^2}{p(x,\theta)q(x)}$;
    Compute $\nabla_\theta \hat{L}$ using backpropagation;
    Set $\theta \leftarrow \theta - \eta \nabla_\theta \hat{L}$;
    Reset gradients;
  end
end
return $p(x, \theta)$;

- An adaptive scheme starting with a uniform distribution and switching to sampling from a frozen model when it is expected to yield a more accurate loss estimate (`survey_strategy="adaptive_variance"`).

An important point to notice is that the original NIS formulation in eq. (2.10) can be restated as a limiting case of our approach. Indeed, if we take q to be a frozen version of the model $p(x, \theta_0)$, which we update every time we sample points (setting $N = 1$ in algorithm 1), the gradient update in line 9 is

$$\nabla_\theta \left[\mathop{\mathbb{E}}_{x\sim p(x,\theta_0)} \frac{f(x)^2}{p(x,\theta)p(x,\theta_0)} \right] \Bigg|_{\theta_0=\theta} = - \mathop{\mathbb{E}}_{x\sim p(x,\theta)} \frac{f(x)^2}{p(x,\theta)^2} \nabla_\theta \log p(x,\theta). \tag{3.2}$$

The core difference to the "buffered training" approach presented by MadNIS is that we offer an adaptive scheme, which automatically switches between which distribution is chosen to sample from. Using a uniform distribution in the beginning ensures that independent of initialisation every relevant region of the integration space is covered. However, with progressing training the frozen model becomes more desirable. In the adaptive training approaches, we switch once both losses become comparable. This is a crucial improvement, as it allows more stable training even without prior knowledge of the integrand. Additionally, a good coverage at early times of the training can speed up the training process. MadNIS, in comparison, uses saved samples only generated from online training, and achieves good coverage by using prior physics knowledge, which we do not require.

JHEP03(2024)083

3.2 The ZüNIS library

On the practical side, ZüNIS is a `PyTorch`-based library which implements many ideas formulated in previous work but organizes them in the form of a modular software library with an easy-to-use interface and well-defined building blocks. We believe this structure will help non-specialist use it without understanding all the nuts and bolts, while experts can easily add new features to responds to their needs. The ZüNIS library relies on three layers of abstractions which steer the different aspects of using normalizing flows to learn probability distributions from un-normalized functions and compute their integrals:

- `Flows`, which implement a bijective mapping which transforms points and computes the corresponding Jacobian are described in appendix E.1
- `Trainers`, which provide the infrastructure to perform training steps and sample from flow models are described in appendix E.2
- `Integrators`, which use trainers to steer the training of a model and compute integrals are described in appendix E.3

4 Results

In this section, we evaluate ZüNIS on a variety of test functions to assess its performance and compare it to the commonly used VEGAS algorithm [6, 65]. Although there have been recent advances in computing both matrix elements and the integration using VEGAS on GPU [66], we focus on the predominant case of these steps being performed on CPU. We first produce a few low dimensional examples for illustrative purposes, then move on to integrating parametric functions in various dimensions and finally evaluate performance on particle scattering matrix elements.

4.1 Low-dimensional examples

Let us start by illustrating the effectiveness of ZüNIS in a low dimensional setting where we can readily visualize results. We define three functions on the 2D unit hypercube which each illustrate some failure mode of VEGAS (see appendix B).

We ran the ZüNIS `Integrator` with default arguments over ten repetitions for each function and report the performance of the trained integral estimator compared to a flat estimator and to VEGAS in table 1. Overall, ZüNIS `Integrators` learn to sample from their target function extremely well: we outperform VEGAS by a factor 100 for the camel and the slashed circle functions and a factor 30 for the sinusoidal function and VEGAS itself provides no advantage over uniform sampling for the latter two.

We further illustrate the performance of our trained models by comparing the target functions and density histogram for points sampled from the normalizing flows in figure 1, which shows great qualitative agreement.

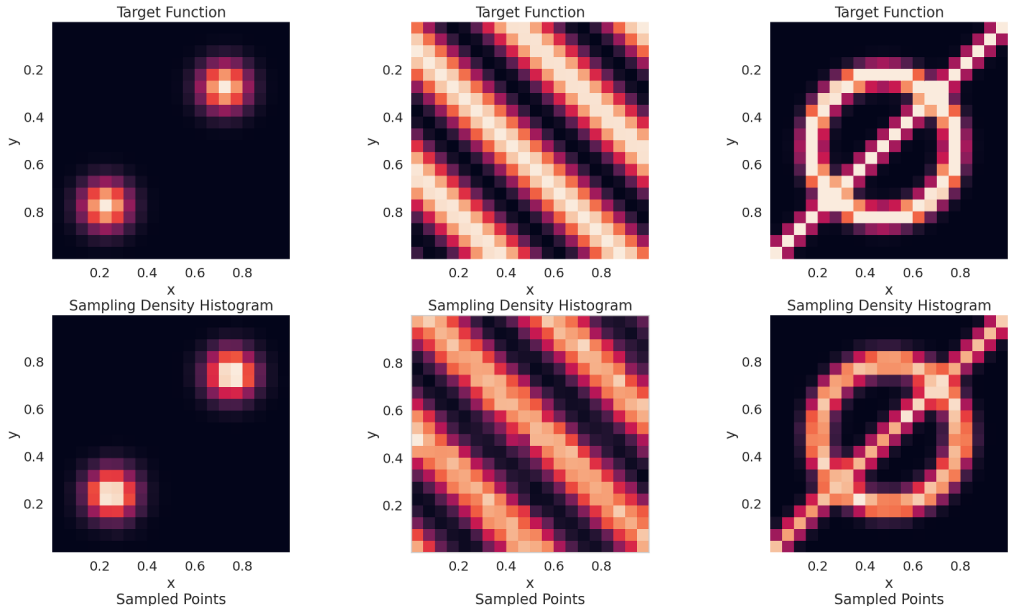
4.2 Systematic benchmarks

Let us now take a more systematic approach to benchmarking ZüNIS. We compare ZüNIS Integrators against uniform integration and VEGAS using the following metrics: integrand

JHEP03(2024)083

Variance Reduction	Camel	Slashed Circle	Sinusoidal
vs. uniform	$1.8 \pm 0.4 \times 10^3$	$8.9 \pm 0.9 \times 10^1$	$2.0 \pm 0.5 \times 10^2$
vs. VEGAS	$7.0 \pm 1.4 \times 10^2$	$8.8 \pm 0.9 \times 10^1$	$1.6 \pm 0.5 \times 10^2$

Table 1. Variance reduction (high is good) for the camel, slashed circle and sinusoidal functions compared to uniform sampling and to VEGAS over 10 repetitions.



(a) Camel function. (b) Sinusoidal function. (c) Slashed circle function.

Figure 1. Comparison between target functions and point sampling densities for 1(a) the camel function, 1(b) the sinusoidal function, 1(c) the slashed circle function. Supplementary figure 7 shows how points are mapped from latent to target space.

variance (a measure of convergence speed, see section 2.1), unweighting efficiency (a measure of the efficiency of exact sampling with rejection, see appendix A) and wall-clock training and sampling.[2]

ZüNIS improves convergence rate compared to VEGAS. For this experiment, we focus on the camel function defined in eq. (B.1) and scan a 35 configurations spanning from 2 to 32 dimensions over function variances between 10^{-2} and 10^2 as shown in table 3.

Except in the low variance limit, ZüNIS can reduce the required number of points sampled to attain a given precision on integral estimates without any parameter tuning, attaining speed-ups of up to $\times 1000$ both compared to uniform sampling and VEGAS-based importance sampling, as shown in figure 2(a)–2(b) and table 4. Compared with the results reported in table II of [50], we see also a significant improvement with respect to i-flow. i-flow reports only moderate reduction of needed function calls up to one third, whereas we find in the same range of dimensions and integrand variances (multiple) order of magnitude reductions in variance, which shows the strength of our approach. Unweighting efficiencies are also boosted

[2]We provide details on hardware in appendix F.

JHEP03(2024)083

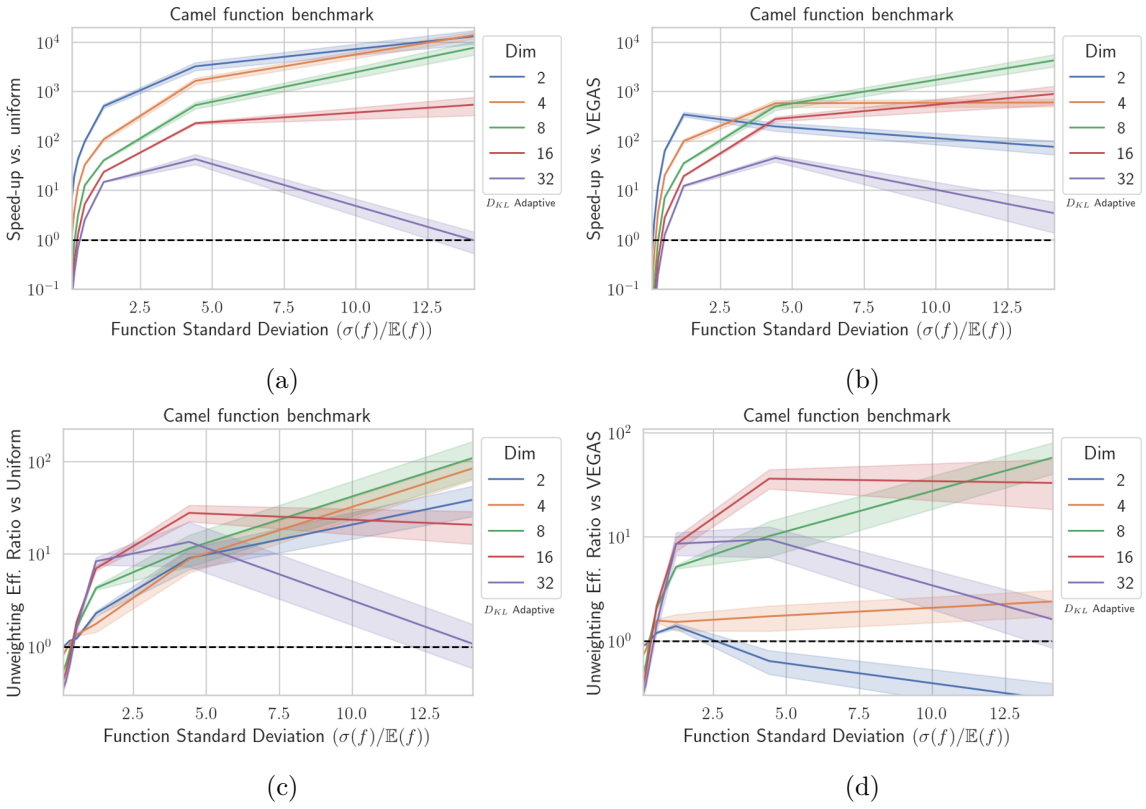


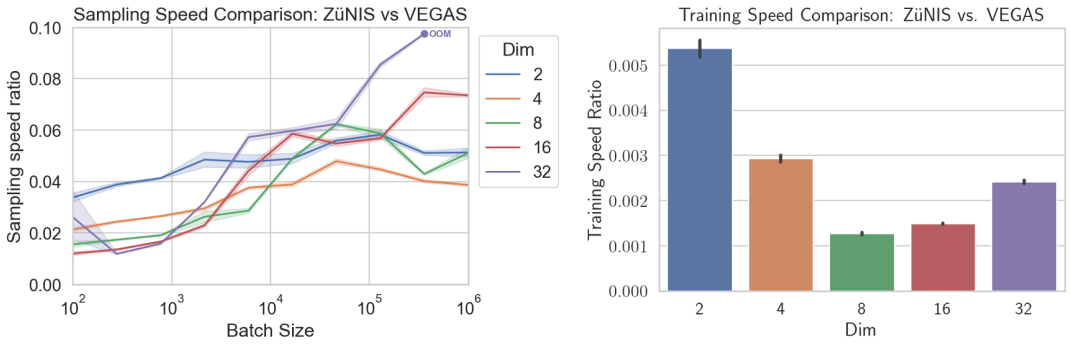
Figure 2. Benchmarking ZüNIS against uniform sampling and VEGAS with default settings. In (2(a)–2(b)), we show the sampling speed-up (ratio of integrand variance) as a function of the relative standard deviation of the integrand, while we show the unweighting speed-up (ratio of unweighting efficiencies) in (2(c)–2(d)).

significantly, although more mildly than variances, as shown in figure 2(c)–2(d), which we could attribute to PDF underestimation in regions with low point density; the nature of the veto algorithm makes it very sensitive to a few bad behaving points in the whole dataset.

ZüNIS is slower than VEGAS. ZüNIS does not, however, outclass VEGAS on all metrics by far: as shown in figure 3, training is a few hundred times slower than VEGAS and sampling is 10-50 times slower, all while ZüNIS runs on GPUs. This is to be expected given the much increased computational complexity of normalizing flows compared to the VEGAS algorithm. As such, ZüNIS is not a general replacement for VEGAS, but provides a clear advantage for integrating time-intensive functions, where sampling is a negligible overhead, such as precision high-energy-physics simulations.

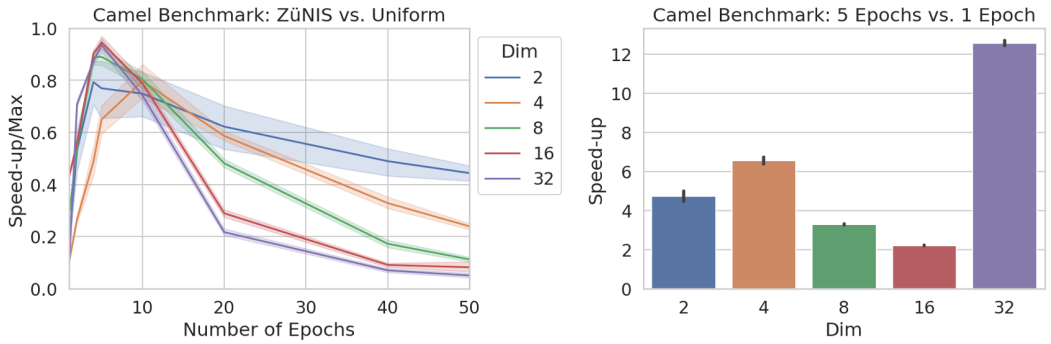
The new loss function introduced in ZüNIS improves data efficiency. We have shown that ZüNIS is a very performant importance sampling and event generation tool and provides significant improvements over existing tools, while requiring little fine tuning from users. Another key result is that the new approach to training we introduced in section 3.1 has a large positive impact on performance. Indeed, as can be seen in figure 4, re-using samples for training over multiple epochs provides a 2- to 10-fold increase in convergence speed, making training much more data-efficient.

JHEP03(2024)083



(a) (b)

Figure 3. Comparison of the training and sampling speed of ZüNIS and VEGAS. As can be expected, ZüNIS is much slower than VEGAS, both for training and sampling, although larger batch sizes can better leverage the power of hardware accelerators.



(a) (b)

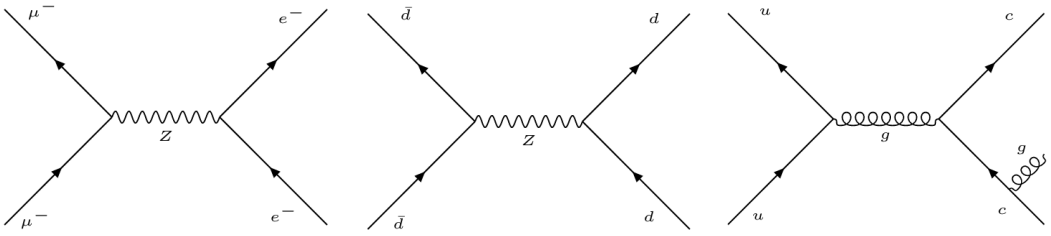
Figure 4. Figure 4(a): effect of repeatedly training on the same sample of points over multiple epochs. For all settings, there is a large improvement when going from one to moderate epoch counts, with a peak around 5-10. Larger number of epochs lead to overfitting, which impacts performance negatively. Figure 4(b): comparison between optimal data reuse (5 epochs) and the original NIS algorithm (1 epoch).

For this experiment, we use forward sampling, where the frozen model is used to sample a batch of points which are then used for training over multiple epochs before resampling from an update of the frozen model. As a result, we reproduce the original formulation of NIS in eq. (2.10) when we use a single epoch as shown in eq. (3.2).

4.3 MadGraph cross section integrals

Cross-sections are integrals of quantum transition matrix elements for a scattering process such as a LHC collision and express the probability that specific particles are produced. Matrix elements themselves are un-normalized probability distributions for the configuration of the outgoing particles: it is therefore both valuable to integrate them to know the overall frequency of a given scattering process, and to sample from them to understand how particles will be spatially distributed as they fly off the collision point.

JHEP03(2024)083



(a) (b) (c)

Figure 5. Sample Feynman Diagrams for $e^-\mu \to e^-\mu$ via Z, $d\bar{d} \to d\bar{d}$ via Z and $uc \to ucg$.

	$e^-\mu \to e^-\mu$ via Z	$d\bar{d} \to d\bar{d}$ via Z	$uc \to ucg$
dimensions	2	4	7
normalized standard deviation	1.45×10^{-2}	6.57×10^{-2}	0.96

Table 2. Comparison of the three test processes.

We study the performance of ZüNIS in comparison to VEGAS by studying three simple processes at leading order in perturbation theory, $e^-\mu \to e^-\mu$ via Z, $d\bar{d} \to d\bar{d}$ via Z and $uc \to ucg$ (with 3-jet cuts based on ΔR), see table 2 and figure 5. We use the first process as a very easy reference while the two other, quark-initiated processes are used to illustrate specific points. Indeed, both feature narrow regions of their integration space with large enhancements, due respectively to Z-boson resonances and infra-red divergences.

We evaluate the matrix elements for these three processes by using the Fortran standalone interface of MadGraph5_aMC@NLO [1]. The two hadronic processes are convolved with parton-distribution functions from `LHAPDF6` [67]. We parametrize phase space (the particle configuration space) using the RAMBO on diet algorithm [68] implemented for `PyTorch` in TorchPS [69]. Using RAMBO on diet has the advantage that no knowledge about the resonance structure of the integrand is required. However, this comes at a cost of reduced performance, as resonances are not reflected appropriately in the phase space sampling. It has been shown this can have a substantial effect on performance [70].

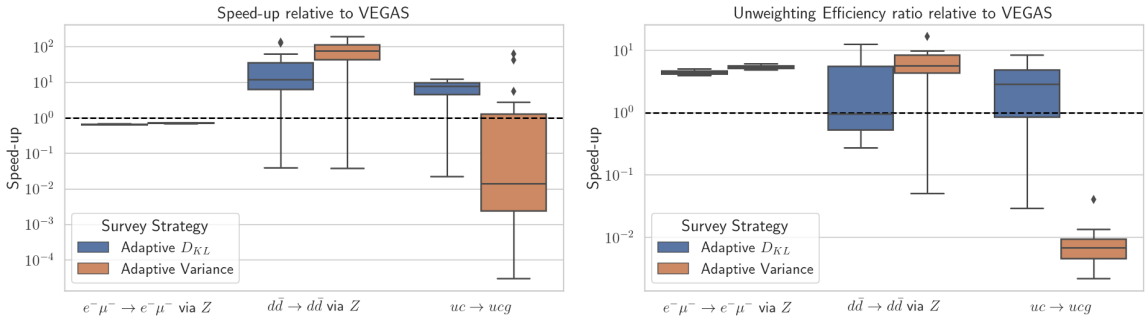
We report benchmark results in figure 6, in which we trained over 500,000 points for each process using near-default configuration, scanning only over variance and Kullback-Leibler losses.

As previously observed, little convergence acceleration is achieved for low variance integrands like $e^-\mu \to e^-\mu$, but unweighting still benefits from NIS. The two hadronic processes illustrate typical features for cross sections: training performance is variable and different processes are optimized by different loss function choices.[3]

The performance of $d\bar{d} \to d\bar{d}$ shows nice improvement with ZüNIS while that of $uc \to ucg$ is more limited. This can be understood by comparing to importance sampling (see appendix C.3): it is in fact VEGAS, which performs significantly better on $uc \to ucg$

[3]Generally, smoother functions are better optimized with the Kullback-Leibler loss while functions with peaks benefit from using the variance loss. As we show in appendix C.4, choosing an adaptive strategy is generally advisable whatever the loss.

JHEP03(2024)083



(a) (b)

Figure 6. Average performance of ZüNIS over 20 runs relative to VEGAS, measured by the relative speed-up in figure 6(a) and by the relative unweighting efficiency in figure 6(b).

compared to $d\bar{d} \to d\bar{d}$ because the parametrization of RAMBO is based on splitting invariant masses, making them aligned with the enhancements in the ucg phase space and allowing great VEGAS performance. This drives a key conclusion for the potential role of ZüNIS in the HEP simulation landscape: not to replace VEGAS, but to fill in the gaps where it fails due to inadequate parametrizations, as we illustrate here by using non-multichanneled $d\bar{d} \to d\bar{d}$ as a proxy for more complex processes.

5 Conclusion

We have showed that ZüNIS can outperform VEGAS both in terms of integral convergence rate and unweighting efficiency on specific cases, at the cost of a significant increase in training and sampling time, which is an acceptable tradeoff for high-precision HEP computations with high costs. In this context, the introduction of efficient training is a key element to making the most of the power of neural importance sampling where function evaluation costs are a major concern. While further testing is required to ascertain how far NIS can fill the gaps left by VEGAS for integrating complex functions, there is already good evidence that ZüNIS can provide needed improvements in specific cases. We hope that the publication of a usable toolbox for NIS such as ZüNIS will stir a wider audience within the HEP community to apply the method so that the exact boundaries its applicability can be more clearly ascertained.

Acknowledgments

We would like to thank Simone Lionetti, Armin Schweizer and Valentin Hirschi for many useful discussions. We are very grateful to Prof. Babis Anastasiou for his continued support providing computational resources. N.D. received funding from ERC grand 69471 at ETHZ and is supported at IBM by SNF grant 200021_192128 / 1. N.G. acknowledges support by the Stiftung Polytechnische Gesellschaft Frankfurt am Main as well as the Studienstiftung des Deutschen Volkes, as well as by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) — Project number 315477589 — TRR 211.

JHEP03(2024)083

6 Reproducibility statement

The library is available on Github or at PyPI `pip install zunis`.

The recommended procedure to reproduce the experiments is to clone the repository and install the `Python` requirements using `pip install -r requirements.txt`.

The data to reproduce the experiments can be generated using scripts provided in the repository at `experiments/benchmarks`, in which `Jupyter` notebooks are also available to reproduce the figures of the paper. The following scripts are available:

- `benchmarks_03/camel/run_benchmark_defaults.sh` to generate camel integration data
- `benchmarks_04/camel/run_benchmark_defaults.sh` to generate camel sampling speed data
- `benchmark_madgraph/ex_benchmark_emu.sh` to generate $e^-\mu \to e^-\mu$ via Z integration data
- `benchmark_madgraph/ex_benchmark_dd.sh` to generate $d\bar{d} \to d\bar{d}$ via Zintegration data
- `benchmark_madgraph/ex_benchmark_ucg.sh` to generate $uc \to ucg$ integration data

These scripts assume that 5 CUDA GPUs are available and run 5 benchmarks in parallel. If fewer GPUs are available, it is recommended to modify the scripts to run the benchmarking scripts sequentially (by removing the ampersand) and to adapt the `--cuda=N` option.

A Event generation with the veto algorithm

Generating i.i.d points following an arbitrary probability distributions in high dimensions is not a priori a trivial task. A straightforward way to obtain such data is to use rejection sampling, which can be based on any distribution q from which we can sample. Given an i.i.d sample $x_1, \ldots, x_N \sim q$, we can define weights $w(x_i) = p(x_i)/q(x_i)$ and keep/reject points with probability $w(x_i)/w_{\max}$.

The main metric for evaluating the performance of this algorithm is the unweighting efficiency: how much data is kept from an original sample of size N on average, which is expressed as

$$\epsilon_{\text{unw}} = \mathop{\mathbb{E}}_{x\sim q} \frac{w(x)}{w_{\max}}. \tag{A.1}$$

B Fundamental limitations of the VEGAS algorithm

We define three functions over the two dimensional hypercube:

$$f_{\text{camel}}(x) = \exp\left(-\left(\frac{x-\mu_1)}{\sigma}\right)^2\right) + \exp\left(-\left(\frac{x-\mu_2)}{\sigma}\right)^2\right), \tag{B.1}$$

$$f_{\varnothing}(x) = \min\left[1,\ \exp\left(-\left(\frac{|x|-r}{\sigma_{\varnothing}}\right)^2\right) + \exp\left(-\left(\frac{a\cdot x}{\sigma_{\varnothing}}\right)^2\right)\right] \tag{B.2}$$

$$f_{\sin}(x) = \cos\left(k\cdot x\right)^2, \tag{B.3}$$

JHEP03(2024)083

to which we refer respectively as the camel, slashed circle and sinusoidal target functions. We set their parameters as follows

$$\mu_1 = \begin{pmatrix} 0.25 \\ 0.25 \end{pmatrix},\ \mu_2 = \begin{pmatrix} 0.75 \\ 0.75 \end{pmatrix},\ a = \begin{pmatrix} 1 \\ -1 \end{pmatrix},\ k = \begin{pmatrix} 6 \\ 6 \end{pmatrix},\ \sigma = 0.1,\ \sigma_{\varnothing} = 0.1,\ r = 0.3 \quad \text{(B.4)}$$

We chose these functions because they illustrate different failure modes of the VEGAS algorithm:

- Because of the factorized PDF of VEGAS, the camel function leads to 'phantom peaks' in the off diagonal. This problem grows exponentially with the number of dimensions but can be addressed by a change of variable to align the peaks with an axis of integration.
- The sinuoidal function makes it nearly impossible for VEGAS to provide any improvement: the marginal PDF over each variable of integration is nearly constant. Again this type of issue can be addressed by a change of variable provided one knows how to perform it.
- The slashed circle function is an example of a hard-for-VEGAS function that cannot be improved significantly by a change of variables. One can instead use multi-channeling, but this requires a lot of prior knowledge and has a computational costs since each channel is its own integral.

JHEP03(2024)083

C Supplementary results

C.1 Qualitative examples

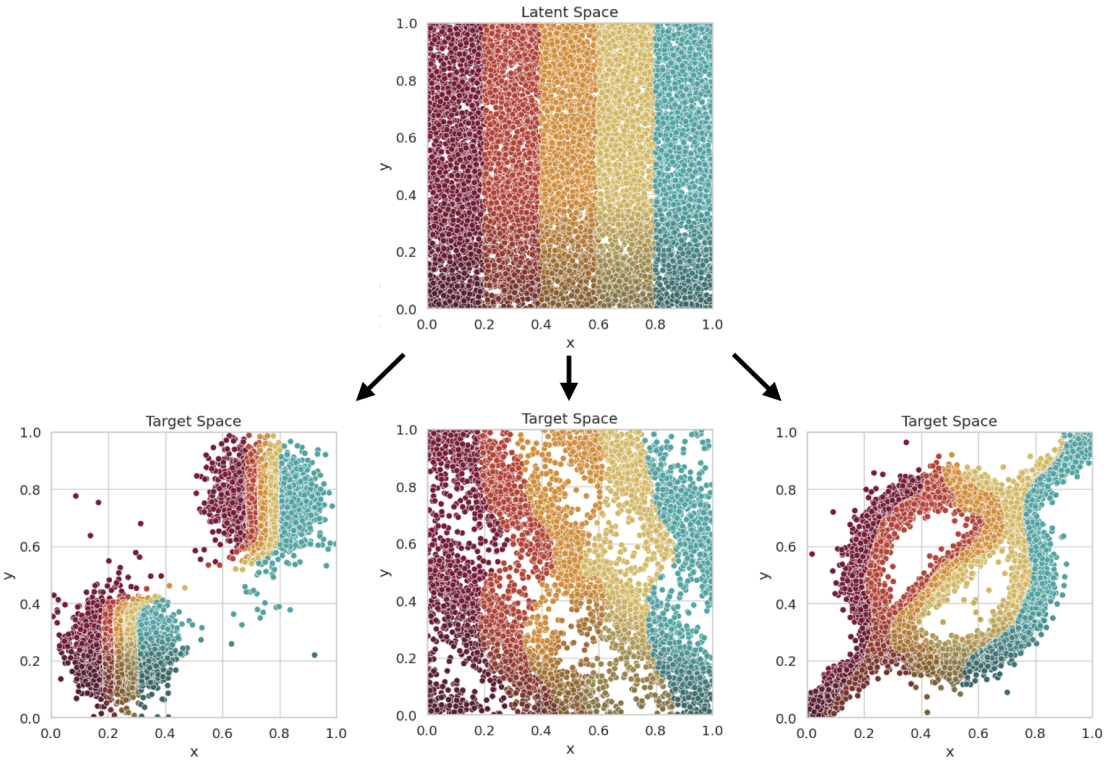


Figure 7. Mapping between the uniform point density in the latent space and the target distribution for the camel function, the sinusoidal function, the slashed circle function. Points are colored based on their position in latent space.

JHEP03(2024)083

C.2 Systematic benchmark details

Camel param. σ	Dimension 2	4	8	16	32	σ_{1d}	rel. st.d
	2.00×10^{-2}	8.93×10^{-2}	2.04×10^{-1}	3.61×10^{-1}	5.06×10^{-1}	0.001	14.1
	6.32×10^{-2}	1.64×10^{-1}	3.14×10^{-1}	4.64×10^{-1}	6.06×10^{-1}	0.01	4.41
	2.21×10^{-1}	3.78×10^{-1}	5.23×10^{-1}	6.67×10^{-1}	8.25×10^{-1}	0.1	1.22
	4.51×10^{-1}	5.93×10^{-1}	7.43×10^{-1}	9.11×10^{-1}	1.10	0.3	0.56
	6.44×10^{-1}	7.99×10^{-1}	9.74×10^{-1}	1.18	1.41	0.5	0.32
	8.62×10^{-1}	1.05	1.26	1.51	1.81	0.7	0.19
	1.21	1.45	1.73	2.07	2.47	1.0	0.10

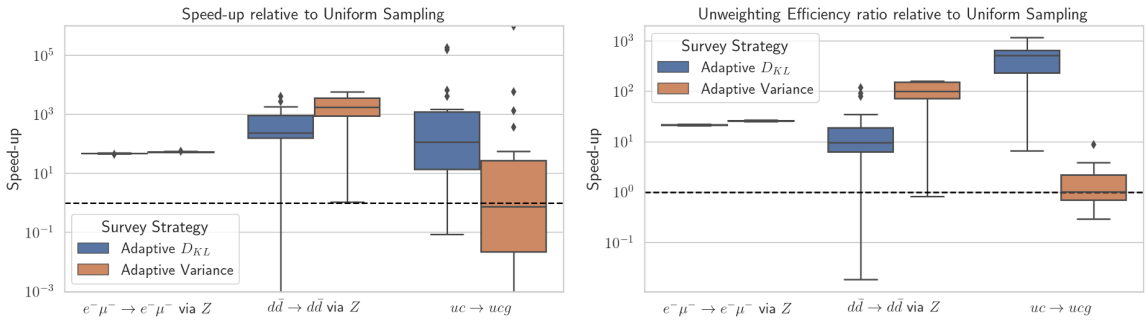
Table 3. Setup of the 35 different camel functions considered to benchmark ZüNIS. We scan over function relative standard deviations, which correspond to different σ parameters for each dimension(eq. (B.1)). We provide the corresponding width of a 1D gaussian (σ_{1d}) with the same variance for reference.

VEGAS speed-up	Dimension 2	4	8	16	32	σ_{1d}	rel. st.d
	$7.6^{+2.4}_{-2.4} \times 10^{1}$	$6.0^{+1.2}_{-1.0} \times 10^{2}$	$4.3^{+1.4}_{-1.0} \times 10^{3}$	$9.0^{+4.7}_{-3.1} \times 10^{2}$	$3.5^{+2.7}_{-1.8} \times 10^{0}$	0.001	14.11
	$2.0^{+0.4}_{-0.4} \times 10^{2}$	$5.8^{+0.8}_{-1.0} \times 10^{2}$	$5.1^{+0.8}_{-0.9} \times 10^{2}$	$2.8^{+0.3}_{-0.3} \times 10^{2}$	$4.5^{+0.6}_{-0.8} \times 10^{1}$	0.01	4.41
	$3.4^{+0.5}_{-0.3} \times 10^{2}$	$1.0^{+0.1}_{-0.1} \times 10^{2}$	$3.5^{+0.1}_{-0.2} \times 10^{1}$	$1.9^{+0.1}_{-0.1} \times 10^{1}$	$1.2^{+0.1}_{-0.1} \times 10^{1}$	0.1	1.22
	$6.3^{+0.4}_{-0.3} \times 10^{1}$	$2.0^{+0.0}_{-0.1} \times 10^{1}$	$7.2^{+0.3}_{-0.3} \times 10^{0}$	$2.8^{+0.1}_{-0.1} \times 10^{0}$	$1.3^{+0.1}_{-0.0} \times 10^{0}$	0.3	0.56
	$1.2^{+0.1}_{-0.1} \times 10^{1}$	$3.5^{+0.1}_{-0.2} \times 10^{0}$	$9.1^{+0.6}_{-0.4} \times 10^{-1}$	$3.7^{+0.2}_{-0.2} \times 10^{-1}$	$2.0^{+0.1}_{-0.1} \times 10^{-1}$	0.5	0.33
	$2.4^{+0.2}_{-0.2} \times 10^{0}$	$5.3^{+0.5}_{-0.4} \times 10^{-1}$	$1.3^{+0.1}_{-0.1} \times 10^{-1}$	$5.5^{+0.4}_{-0.3} \times 10^{-2}$	$3.0^{+0.1}_{-0.2} \times 10^{-2}$	0.7	0.20
	$4.0^{+0.6}_{-0.5} \times 10^{-1}$	$7.8^{+0.6}_{-0.5} \times 10^{-2}$	$1.6^{+0.1}_{-0.1} \times 10^{-2}$	$7.8^{+0.8}_{-0.6} \times 10^{-3}$	$4.8^{+0.3}_{-0.4} \times 10^{-3}$	1.0	0.11

Table 4. Variance reduction factor compared to VEGAS for each of the 35 different camel setups defined in table 3.

JHEP03(2024)083

C.3 Comparing ZüNIS with uniform sampling on matrix elements



(a) (b)

Figure 8. Average performance of ZüNIS over 20 runs relative to flat sampling, measured by the relative speed-up in 8(a) and by the relative unweighting efficiency in 8(b).

C.4 Effect of survey strategies

In the following, we want to investigate how the integration of the three example processes in 4.3 with ZüNIS behaves relative to VEGAS in dependence of the choice of the loss function, the survey strategy and the number of epochs during training. For all other options, the default values are chosen again, except for the number of points during the survey phase, which is set to 500,000.

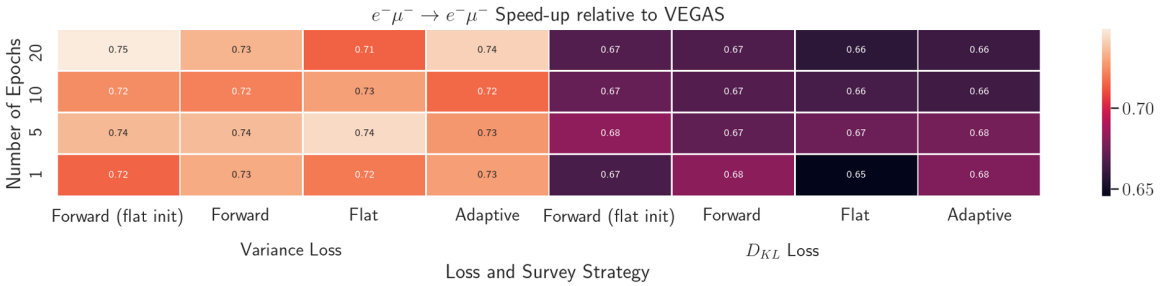
Figure 9(a) shows that for a simple process like $e^-\mu \to e^-\mu$ via Z , where no correlations exist, ZüNIS cannot reach the speed-up achieved by VEGAS. Variance loss seems to lead to higher variance improvements than D_{KL} loss. Contrary, 9(b) shows that ZüNIS can greatly improve the unweighting efficiency for this process. The effect is again consistently stronger when using variance loss. Using a flat survey strategy suffers for both loss functions from overfitting, whereas adaptive sampling in average performs slightly better and does not show overfitting.

$d\bar{d} \to d\bar{d}$ via Z presents a more realistic use case, as the parton distribution functions introduce correlations between the integration variables which present a challenge to the VEGAS algorithm.

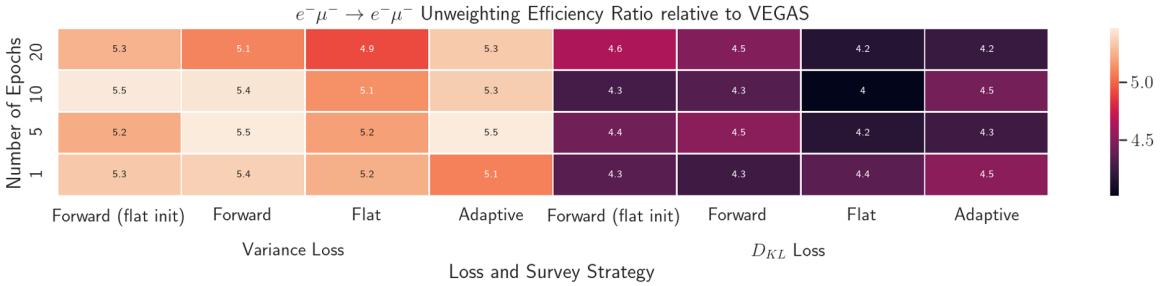
For this process, both the speed-up and the unweighting efficiency ratio clearly favor the variance loss again, which outperforms in both metrics the D_{KL} loss when multiple epochs are used, as can be seen in 10. The richer structure of the integrand reduces the effect of overfitting. Therefore, the performance increases or stays approximately constant except for the combination of D_{KL} loss and the forward survey strategy. For the variance loss, it becomes apparent that the flat survey strategy increases much slower in performance than alternative strategies as a function of the number of epochs. However, for combination of losses and survey strategies, VEGAS could be outperformed substantially in terms of integration speed.

An opposite picture is drawn by the process $uc \to ucg$ in figure 11, for which, apart from the flat survey strategy, D_{KL} loss is in general favored both for integration speed as well as unweighting efficiency ratio. The adaptive survey strategy is here giving the best results, although for a high number of epochs causes overfitting for the unweighting efficiency.

JHEP03(2024)083

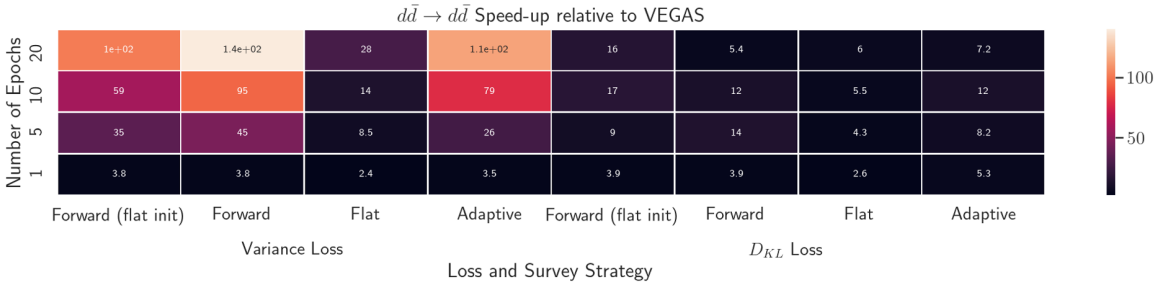


(a)

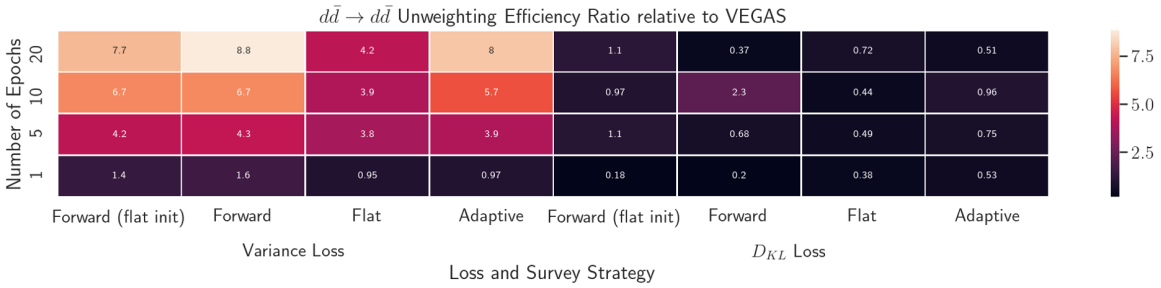


(b)

Figure 9. Median of the performance of ZÜNIS over 20 runs relative to VEGAS for the process $e^-\mu \to e^-\mu$ via Z depending on the loss function, the survey strategy and number of epochs, measured by the relative speed-up in 9(a) and by the relative unweighting efficiency in 9(b).



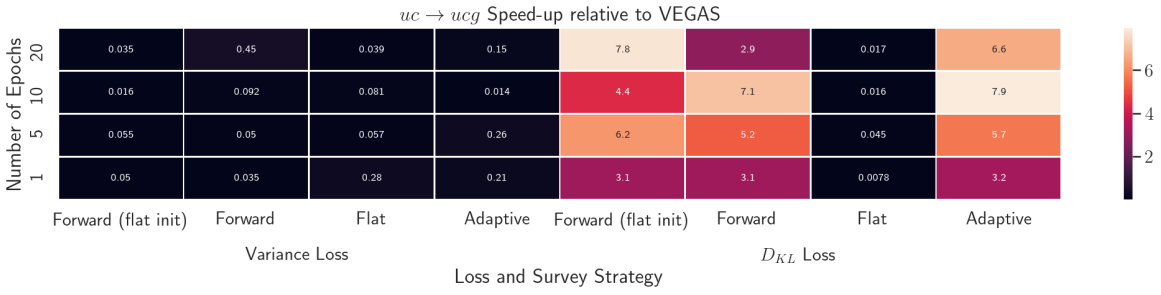
(a)



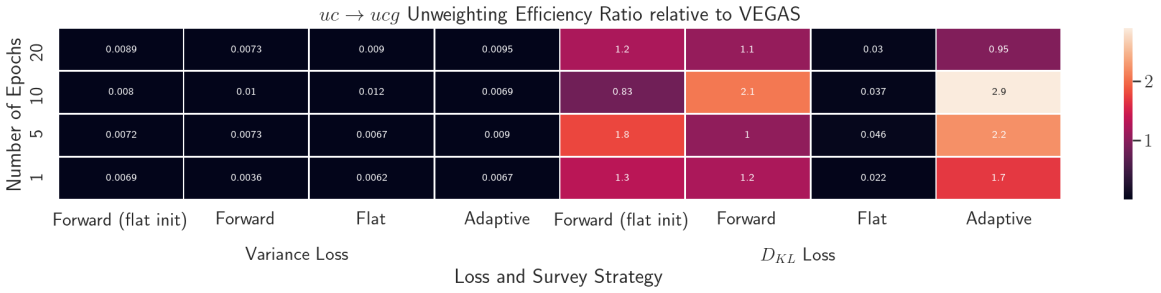
(b)

Figure 10. Median of the performance of ZÜNIS over 20 runs relative to VEGAS for the process $d\bar{d} \to d\bar{d}$ via Z depending on the loss function, the survey strategy and number of epochs, measured by the relative speed-up in 10(a) and by the relative unweighting efficiency in 10(b).

JHEP03(2024)083



(a)



(b)

Figure 11. Median of the performance of ZÜNIS over 20 runs relative to VEGAS for the process $uc \to ucg$ depending on the loss function, the survey strategy and number of epochs, measured by the relative speed-up in 11(a) and by the relative unweighting efficiency in 11(b).

The take-home message of this section is, one the one hand, that the flat survey strategy is in general not recommended. Apart from this, the most important mean to improve the quality of importance sampling are testing whether, independent of the survey strategy, the loss function should be chosen differently.

D Exact minimzation of the neural importance sampling estimator variance

Let us show that the optimal probability distribution for importance sampling is the function itself. Explicitly, as discussed in section 2.1, we want to find the probability distribution p defined over some domain Ω which minimizes the variance of the estimator $f(X)/p(X)$, for $X \sim p(X)$. We showed that this amounts to solving the following optimization problem:

$$\min_p \mathcal{L}(p) = \int_\Omega dx \frac{f(x)^2}{p(x)}, \quad \text{such that } \int dx\, p(x) = 1, \tag{D.1}$$

which we can encode using Lagrange multipliers

$$p = \arg\min \mathcal{L}(p, \lambda) = \int_\Omega dx \frac{f(x)^2}{p(x)} + \lambda \left(p(x) - \frac{1}{V(\Omega)} \right). \tag{D.2}$$

We can now solve this problem by finding extrema with functional derivatives

$$\frac{\delta \mathcal{L}(p, \lambda)}{\delta p(x)} = \lambda - \frac{f(x)^2}{p(x)^2}, \tag{D.3}$$

JHEP03(2024)083

which indeed is zero if $p(x) \propto |f(x)|$. Furthermore, this extremum is certainly a minimum because the loss function is positive and unbounded. Indeed, if we separate Ω into two disjoint measurable subdomains Ω_1 and Ω_2, and define $p_\alpha(x)$ such that points are drawn uniformly over Ω_1 with probability α and uniformly over Ω_2 with probability $1-\alpha$, then the resulting loss function would be

$$\mathcal{L}(p_\alpha) = \frac{V(\Omega_1)}{\alpha} \int_{\Omega_1} dx\, f(x)^2 + \frac{V(\Omega_2)}{1-\alpha} \int_{\Omega_2} dx\, f(x)^2, \tag{D.4}$$

which can be made arbitrarily large by sending α to 0.

E High-level concepts of the ZüNIS API

E.1 Normalizing flows with `Flow` classes

Flows map batches of points and their densities. The ZüNIS library implements normalizing flows by specifying a general interface defined as a Python abstract class: `GeneralFlow`. All flow models in ZüNIS are child classes of `GeneralFlow`, which itself inherits from the Pytorch `nn.Module` interface.

As defined in section 2.2, a normalizing flow in ZüNIS is a bijective mapping between two d dimensional spaces, which in practice are always the unit hypercube $[0,1]^d$ or $\mathbb{R}^d$, with a tractable Jacobian so that it can be used to map probability distributions. To this end, the `GeneralFlow` interface defines normalizing flows as a callable Python object which acts on batches of point drawn from a known PDF p. A batch of points x_i with their PDF values is encoded as a Pytorch `Tensor` object X organized as follows

$$X = (X_1, \ldots, X_{\text{batch}}) \in \mathbb{R}^{\text{batch}} \times \mathbb{R}^{d+1}, \tag{E.1}$$

where each X_i corresponds to a points stacked with its negative log density

$$X_i = \begin{pmatrix} x_{i,1} \\ \vdots \\ x_{i,d} \\ -\log p(x_i) \end{pmatrix}. \tag{E.2}$$

Encoding point densities by their negative logarithm makes their transformation under normalizing flows additive. Indeed if we have a mapping Q with Jacobian determinant j_Q, then $x \sim p(x)$ is mapped to $y = Q(x) \sim \tilde{p}(y)$ such that

$$-\log \tilde{p}(y) = -\log p(x) + \log j_Q(x). \tag{E.3}$$

Coupling Cells are flows defined by an element-wise transform and a mask. All flow models used in ZüNIS in practice are implemented as a sequence of coupling cell transformations acting on a subset of the variables. The abstract class `GeneralCouplingCell` and its child `InvertibleCouplingCell` specifies the general organization of coupling cells as needing to be instantiated with

JHEP03(2024)083

- a dimension d
- a mask defined as a list of boolean specifying which coordinates are transformed or not
- a transform that implements the mapping of the non-masked variables

In release v1.0 of ZüNIS two such classes are provided: `PWLinearCoupling` and `PWQuadraticCoupling`, which respectively implement the piecewise linear and piecewise quadratic coupling cells proposed in [64]. New coupling cells can easily be implemented, as explained in appendix E.4. Both existing classes rely on dense neural networks for the prediction of the shape of their one-dimensional piecewise-polynomial mappings, whose parameters are set at instantiation.

Here is how one can use a piecewise-linear coupling cell for sampling points

```
import torch
from zunis.models.flows.coupling_cells.piecewise_coupling.piecewise_quadratic
    import PWQuadraticCoupling
d=2
N_batch=10
mask = [True,False]
x = torch.zeros((N_batch,d+1))
# Sample the d first entries uniformly, keep 0. for the negative log jacobian
x[...,:-1].uniform_()
print(x[0]) # [0.3377, 0.4362, 0.]
f = PWQuadraticCoupling(d=d,mask=mask)
y = f(x)
print(y[0]) # [0.3377, 0.4411, -0.0314]
```

We provide further details of the use and possible parameters of flows in the documentation of ZüNIS: https://zunis.readthedocs.io/en/stable/.

E.2 Training with the `Trainer` class

The design of the ZüNIS library intentionally restricts `Flow` models to being bijective mappings instead of being ways to evaluate and sample from PDFs so as not to restrict their applicability (see [71] for an example). The specific application in which one uses a normalizing flow, and in our case how precisely one samples from it, is intimately linked to how such a model is trained. As a result, ZüNIS bundles together the low-level training tools for `Flow` models together with sampling tools inside the `Trainer` classes.

The general blueprint for such classes is defined in the `GenericTrainerAPI` abstract class while the main implementation for users is provided as `StatefulTrainer`. At instantiation, all trainers expect a `Flow` model and `flow_prior` which samples point from a fixed PDF in latent space. These two elements together define a probability distribution over the target space from which one can sample.

There are two main ways one interacts with `Trainers`:

- One can sample points from the PDF defined by the model and the prior using the `sample_forward` method.

JHEP03(2024)083

- One can train over a pre-sample batch of points, their sampling PDF and the corresponding function values using `train_on_batch(self, x, px, fx)`

In practice, we expect that the main way users will use `Trainers` is for sampling pretrained models. In the context of particle physics simulations for example, *unweighting* is a common task, which aims at sampling exactly from a known function f. A common approach is the *Hit-or-miss* algorithm [72], whose efficiency is improved by sampling from a PDF approaching f. This is how one would use a trainer trained on f:

```
# [...]
# import or train a trainer ‘pretrained_trainer‘
import torch

# Sampling points
xj = pretrained_trainer.sample_forward(100)
x = x[:, :-1]
px = (-x[:,-1]).exp()
fx = f(x)

# Applying the veto algorithm
fmax = fx.max()
veto = (fx/fmax - torch.zeros_like(fx).uniform_(0.,1.)) > 0.
x_unweighted = x[veto]
# x_unweighted follows the PDF obtained by normalizing f.
```

E.3 Integrating with the `Integrator` class

Integrators are intended as the main way for standard users to interact with ZüNIS. They provide a high-level interface to the functionalities of the library and only optionally require users to know to what lower levels of abstractions really entail and to what their options correspond. From a practical point of view, the main interface of ZüNIS for integration is implemented as the `Integrator`, which is a factory function that instantiates the appropriate integrator class based on a choice of options.

All integrators follow the same pattern, defined in the `SurveyRefineIntegratorAPI` and `BaseIntegrator` abstract classes. Integration start by performing a survey phase, in which it optimizes the way it samples points and then a refine phase, in which it computes the integral by using its learned sampler. Each phase proceeds through a number of steps, which can be set at instantiation or when integrating:

```
# Setting values at instantiation time
integrator = Integrator(d=d, f=f, n_iter_survey=3, n_iter_refine=5)
# Override at integration time
integral_data = integrator.integrate(n_survey=10, n_refine=10)
```

For both the survey and the refine phases, using multiple steps is useful to monitor the stability of the training and of the integration process: if one step is not within a few standard deviations of the next, either the sampling statistics are too low, or something is wrong. For the refine stage, this is the main real advantage of using multiple steps. On

JHEP03(2024)083

the other hand, at each new survey step, a new batch of points is re-sampled, which can be useful to mitigate overfitting.

By default, only the integral estimates obtained during the refine stage are combined to compute the final integral estimate, and their combination is performed by taking their average. Indeed, because the model is trained during the survey step, the points sampled during the refine stage are correlated in an uncontrolled way with the points used during training. Ignoring the survey stage makes all estimates used in the combination independent random variables, which permits us to build a formally correct estimator of the variance of the final result.

E.4 Implementing new coupling cells

To implement a new invertible coupling cell inheriting from `InvertibleCouplingCell`, one must provide an `InvertibleTransform` object and define a callable attribute `T` computing the parameters of the transform. For example, consider a very simple linear coupling cell over $\mathbb{R}$

$$y = Q(x) : \begin{cases} y^A = x^A \\ y^B = \exp\left(T(x^A)\right) \times x^B, \end{cases} \tag{E.4}$$

where $T(x^A)$ is a scalar strictly positive value. This can be defined in the following way in ZüNIS

```
import torch
from zunis.models.flows.coupling_cells.general_coupling import
    InvertibleCouplingCell
from zunis.models.flows.coupling_cells.transforms import InvertibleTransform
from zunis.models.layers.trainable import ArbitraryShapeRectangularDNN

class LinearTransform(InvertibleTransform):
    def forward(self,x,T):
        alpha = torch.exp(T)
        logj = T*x.shape[-1]
        return x*alpha, logj
    def backward(self,x,T):
        alpha = torch.exp(-T)
        logj = -T*x.shape[-1]
        return x*alpha, logj
```

```
class LinearCouplingCell(InvertibleCouplingCell):
    def __init__(self, d, mask, nn_width, nn_depth):
        transform = LinearTransform()
        super(LinearCouplingCell, self).__init__(d=d, mask=mask,transform=transform)
        d_in = sum(mask)
        self.T = ArbitraryShapeRectangularDNN(d_in=d_in,
                                              out_shape=(1,),
                                              d_hidden=nn_width,
                                              n_hidden=nn_depth)
```


JHEP03(2024)083

F Hardware setup details

The computations presented in this work were performed on a computing cluster using a Intel(R) Xeon(R) Gold 5218 CPU @ 2.30GHz with 376 GB RAM. Processes which could be performed on the GPU were done on a GeForce RTX 2080 having 12 GB memory and running on CUDA 11.0.

References

[1] J. Alwall et al., *The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations*, *JHEP* **07** (2014) 079 [arXiv:1405.0301] [INSPIRE].

[2] J. Reuter et al., *New developments on the WHIZARD event generator*, in the proceedings of the *International workshop on future linear colliders*, (2023) [arXiv:2307.14900] [INSPIRE].

[3] SHERPA collaboration, *Event generation with Sherpa 2.2*, *SciPost Phys.* **7** (2019) 034 [arXiv:1905.09127] [INSPIRE].

[4] G.P. Lepage, *Vegas: an adaptive multidimensional integration program*, tech. rep. CLNS-80/447 (1980) [INSPIRE].

[5] R. Kleiss, W.J. Stirling and S.D. Ellis, *A new Monte Carlo treatment of multiparticle phase space at high-energies*, *Comput. Phys. Commun.* **40** (1986) 359 [INSPIRE].

[6] T. Ohl, *Vegas revisited: adaptive Monte Carlo integration beyond factorization*, *Comput. Phys. Commun.* **120** (1999) 13 [hep-ph/9806432] [INSPIRE].

[7] R. Kleiss and R. Pittau, *Weight optimization in multichannel Monte Carlo*, *Comput. Phys. Commun.* **83** (1994) 141 [hep-ph/9405257] [INSPIRE].

[8] M.F. Bugallo et al., *Adaptive importance sampling: the past, the present, and the future*, *IEEE Signal Processing Mag.* **34** (2017) 60.

[9] M.F. Bugallo, L. Martino and J. Corander, *Adaptive importance sampling in signal processing*, *Digital Signal Proc.* **47** (2015) 36.

[10] O. Cappé, A. Guillin, J.-M. Marin and C.P. Robert, *Population Monte Carlo*, *J. Comput. Graph. Statist.* **13** (2004) 907.

[11] Y. Iba, *Population based Monte Carlo algorithms*, *Trans. Jap. Soc. Artif. Intell.* **16** (2001) 279 [cond-mat/0008226] [INSPIRE].

[12] O. Cappé et al., *Adaptive importance sampling in general mixture classes*, *Statist. Comput.* **18** (2008) 447 [arXiv:0710.4242].

[13] E. Koblents and J. Míguez, *A population Monte Carlo scheme with transformed weights and its application to stochastic kinetic models*, *Statist. Comput.* **25** (2013) 407 [arXiv:1208.5600].

[14] V. Elvira, L. Martino, D. Luengo and M.F. Bugallo, *Improving population Monte Carlo: alternative weighting and resampling schemes*, *Signal Proc.* **131** (2017) 77 [arXiv:1607.02758].

JHEP03(2024)083

[15] R. Douc, A. Guillin, J.-M. Marin and C.P. Robert, *Minimum variance importance sampling via population Monte Carlo*, *ESAIM: Probabil. Statist.* **11** (2007) 427.

[16] J.-M. Cornuet, J.-M. Marin, A. Mira and C. Robert, *Adaptive multiple importance sampling*, *Scandinavian J. Statist.* **39** (2012) 798.

[17] S. Jadach, *Foam: multidimensional general purpose Monte Carlo generator with selfadapting symplectic grid*, *Comput. Phys. Commun.* **130** (2000) 244 [physics/9910004] [INSPIRE].

[18] T. Hahn, *CUBA: a library for multidimensional numerical integration*, *Comput. Phys. Commun.* **168** (2005) 78 [hep-ph/0404043] [INSPIRE].

[19] A. van Hameren, *PARNI for importance sampling and density estimation*, *Acta Phys. Polon. B* **40** (2009) 259 [arXiv:0710.2448] [INSPIRE].

[20] G.P. Lepage, *Adaptive multidimensional integration: VEGAS enhanced*, *J. Comput. Phys.* **439** (2021) 110386 [arXiv:2009.05112] [INSPIRE].

[21] J. Bellm et al., *Herwig* 7.0/*Herwig++* 3.0 *release note*, *Eur. Phys. J. C* **76** (2016) 196 [arXiv:1512.01178] [INSPIRE].

[22] F. Beaujean and A. Caldwell, *Initializing adaptive importance sampling with Markov chains*, arXiv:1304.7808.

[23] J. Skilling, *Nested sampling for general Bayesian computation*, *Bayesian Anal.* **1** (2006) 833 [INSPIRE].

[24] W.J. Handley, M.P. Hobson and A.N. Lasenby, *polychord: next-generation nested sampling*, *Mon. Not. Roy. Astron. Soc.* **453** (2015) 4385 [arXiv:1506.00171] [INSPIRE].

[25] D. Yallup, T. Janßen, S. Schumann and W. Handley, *Exploring phase space with Nested Sampling*, *Eur. Phys. J. C* **82** (2022) 8 [arXiv:2205.02030] [INSPIRE].

[26] ATLAS collaboration, *ATLAS HL-LHC computing conceptual design report*, CERN-LHCC-2020-015, CERN, Geneva, Switzerland (2020) [INSPIRE].

[27] S.P. Jones, *Higgs boson pair production: Monte Carlo generator interface and parton shower*, *Acta Phys. Polon. Supp.* **11** (2018) 295 [INSPIRE].

[28] HEP Software Foundation collaboration, *HL-LHC computing review: common tools and community software*, in the proceedings of the *Snowmass* 2021, (2020) [DOI:10.5281/zenodo.4009114] [arXiv:2008.13636] [INSPIRE].

[29] A. Buckley, *Computational challenges for MC event generation*, *J. Phys. Conf. Ser.* **1525** (2020) 012023 [arXiv:1908.00167] [INSPIRE].

[30] HSF Physics Event Generator WG collaboration, *Challenges in Monte Carlo event generator software for High-Luminosity LHC*, *Comput. Softw. Big Sci.* **5** (2021) 12 [arXiv:2004.13687] [INSPIRE].

[31] J. Bendavid, *Efficient Monte Carlo integration using boosted decision trees and generative deep neural networks*, arXiv:1707.00028 [INSPIRE].

[32] M.D. Klimek and M. Perelstein, *Neural network-based approach to phase space integration*, *SciPost Phys.* **9** (2020) 053 [arXiv:1810.11509] [INSPIRE].

[33] I.-K. Chen, M.D. Klimek and M. Perelstein, *Improved neural network Monte Carlo simulation*, *SciPost Phys.* **10** (2021) 023 [arXiv:2009.07819] [INSPIRE].

[34] A. Butter, T. Plehn and R. Winterhalder, *How to GAN LHC events*, *SciPost Phys.* **7** (2019) 075 [arXiv:1907.03764] [INSPIRE].

JHEP03(2024)083

[35] R. Di Sipio, M. Faucci Giannelli, S. Ketabchi Haghighat and S. Palazzo, *DijetGAN: a Generative-Adversarial Network approach for the simulation of QCD dijet events at the LHC*, *JHEP* **08** (2019) 110 [arXiv:1903.02433] [INSPIRE].

[36] A. Butter, T. Plehn and R. Winterhalder, *How to GAN event subtraction*, *SciPost Phys. Core* **3** (2020) 009 [arXiv:1912.08824] [INSPIRE].

[37] SHiP collaboration, *Fast simulation of muons produced at the SHiP experiment using Generative Adversarial Networks*, 2019 *JINST* **14** P11028 [arXiv:1909.04451] [INSPIRE].

[38] B. Hashemi et al., *LHC analysis-specific datasets with Generative Adversarial Networks*, arXiv:1901.05282 [INSPIRE].

[39] S. Carrazza and F.A. Dreyer, *Lund jet images from generative and cycle-consistent adversarial networks*, *Eur. Phys. J. C* **79** (2019) 979 [arXiv:1909.01359] [INSPIRE].

[40] M. Bellagente, M. Haussmann, M. Luchmann and T. Plehn, *Understanding event-generation networks via uncertainties*, *SciPost Phys.* **13** (2022) 003 [arXiv:2104.04543] [INSPIRE].

[41] K.T. Matchev, A. Roman and P. Shyamsundar, *Uncertainties associated with GAN-generated datasets in high energy physics*, *SciPost Phys.* **12** (2022) 104 [arXiv:2002.06307] [INSPIRE].

[42] K. Danziger, T. Janßen, S. Schumann and F. Siegert, *Accelerating Monte Carlo event generation — rejection sampling using neural network event-weight estimates*, *SciPost Phys.* **12** (2022) 164 [arXiv:2109.11964] [INSPIRE].

[43] M. Stoye et al., *Likelihood-free inference with an improved cross-entropy estimator*, arXiv:1808.00973 [INSPIRE].

[44] F.A. Di Bello et al., *Efficiency parameterization with neural networks*, *Comput. Softw. Big Sci.* **5** (2021) 14 [arXiv:2004.02665] [INSPIRE].

[45] S. Diefenbacher et al., *DCTRGAN: improving the precision of generative models with reweighting*, 2020 *JINST* **15** P11004 [arXiv:2009.03796] [INSPIRE].

[46] A. Andreassen and B. Nachman, *Neural networks for full phase-space reweighting and parameter tuning*, *Phys. Rev. D* **101** (2020) 091901 [arXiv:1907.08209] [INSPIRE].

[47] A. Butter et al., *GANplifying event samples*, *SciPost Phys.* **10** (2021) 139 [arXiv:2008.06545] [INSPIRE].

[48] T. Müller, B. Mcwilliams, F. Rousselle, M. Gross and J. Novák, *Neural importance sampling*, *ACM Trans. Graph.* **38** (2019) 1.

[49] Q. Zheng and M. Zwicker, *Learning to importance sample in primary sample space*, *Computer Graphics Forum* **38** (2019) 169 [arXiv:1808.07840].

[50] C. Gao, J. Isaacson and C. Krause, *i-flow: high-dimensional integration and sampling with normalizing flows*, *Mach. Learn. Sci. Tech.* **1** (2020) 045023 [arXiv:2001.05486] [INSPIRE].

[51] E. Bothmann et al., *Exploring phase space with neural importance sampling*, *SciPost Phys.* **8** (2020) 069 [arXiv:2001.05478] [INSPIRE].

[52] C. Gao et al., *Event generation with normalizing flows*, *Phys. Rev. D* **101** (2020) 076002 [arXiv:2001.10028] [INSPIRE].

[53] S. Pina-Otey, V. Gaitan, F. Sánchez and T. Lux, *Exhaustive neural importance sampling applied to Monte Carlo event generation*, *Phys. Rev. D* **102** (2020) 013003 [arXiv:2005.12719] [INSPIRE].

JHEP03(2024)083

[54] T. Heimel et al., *MadNIS — neural multi-channel importance sampling*, *SciPost Phys.* **15** (2023) 141 [arXiv:2212.06172] [INSPIRE].

[55] T. Heimel et al., *The MadNIS reloaded*, arXiv:2311.01548 [INSPIRE].

[56] B. Stienen and R. Verheyen, *Phase space sampling and inference from weighted events with autoregressive flows*, *SciPost Phys.* **10** (2021) 038 [arXiv:2011.13445] [INSPIRE].

[57] S. Weinzierl, *Introduction to Monte Carlo methods*, hep-ph/0006269 [INSPIRE].

[58] E.G. Tabak and E. Vanden-Eijnden, *Density estimation by dual ascent of the log-likelihood*, *Commun. Math. Sci.* **8** (2010) 217.

[59] E.G. Tabak and C.V. Turner, *A family of nonparametric density estimation algorithms*, *Commun. Pure Appl. Math.* **66** (2013) 145 [INSPIRE].

[60] O. Rippel and R.P. Adams, *High-dimensional probability estimation with deep density models*, arXiv:1302.5125.

[61] D.J. Rezende and S. Mohamed, *Variational inference with normalizing flows*, arXiv:1505.05770 [INSPIRE].

[62] L. Dinh, D. Krueger and Y. Bengio, *NICE: Non-linear Independent Components Estimation*, arXiv:1410.8516 [INSPIRE].

[63] L. Dinh, J. Sohl-Dickstein and S. Bengio, *Density estimation using real NVP*, arXiv:1605.08803 [INSPIRE].

[64] T. Müller, B. Mcwilliams, F. Rousselle, M. Gross and J. Novák, *Neural importance sampling*, *ACM Trans. Graph.* **38** (2019) 1.

[65] G.P. Lepage, *A new algorithm for adaptive multidimensional integration*, *J. Comput. Phys.* **27** (1978) 192 [INSPIRE].

[66] E. Bothmann et al., *A portable parton-level event generator for the High-Luminosity LHC*, arXiv:2311.06198 [INSPIRE].

[67] A. Buckley et al., *LHAPDF6: parton density access in the LHC precision era*, *Eur. Phys. J. C* **75** (2015) 132 [arXiv:1412.7420] [INSPIRE].

[68] S. Plätzer, *RAMBO on diet*, arXiv:1308.2922 [INSPIRE].

[69] N. Götz, *NGoetz/TorchPS:-v*1.0.1, https://github.com/NGoetz/TorchPS/tree/v1.0.1, March 2021.

[70] E. Bothmann et al., *Efficient phase-space generation for hadron collider event simulation*, *SciPost Phys.* **15** (2023) 169 [arXiv:2302.10449] [INSPIRE].

[71] J. Brehmer and K. Cranmer, *Flows for simultaneous manifold learning and density estimation*, arXiv:2003.13913 [INSPIRE].

[72] F. James, *Monte Carlo theory and practice*, *Rept. Prog. Phys.* **43** (1980) 1145 [INSPIRE].

JHEP03(2024)083