HAMILTONIAN NEURAL NETWORK AND HAMILTONIAN MONTE CARLO: PART ONE REPORT

SUBMITTED VERSION OF PART 1 FOR JP.MORGAN

Chuanmiao Yan
Academy of Mathematics and Systems Science
Chinese Academy of Sciences
Beijing, 100190
yanchuanmiao22@mails.ucas.ac.cn

December 7, 2024

ABSTRACT

Hamiltonian Monte Carlo (HMC) and No-U-Turn Sampling (NUTS) have become pivotal tools in Bayesian computation, enabling efficient exploration of high-dimensional posterior distributions through gradient-based methods. Recent advancements, including Latent Hamiltonian Neural Networks (L-HNNs), further extend these techniques by integrating machine learning to handle high-dimensional and latent space representations. This paper reviews the foundational literature on HMC and NUTS, identifying their limitations and emphasizing the challenges of high-dimensional Bayesian inference. To address the risk of losing focus on the core problem, the discussion highlights why traditional Monte Carlo methods often fail and how L-HNNs and NUTS provide effective solutions. Critical implementation details, including leapfrog integration, the Metropolis acceptance criterion, and the *BuildTree* function, are also examined to ensure practical applicability. A comprehensive testing strategy, including manual, integration, and unit tests, is outlined, with an emphasis on maintaining consistent configuration updates to acquire the replication results in *TensorFlow*. By bridging theoretical advancements and practical execution, this paper provides a roadmap for leveraging modern sampling methods to address computational challenges in Bayesian inference.

Keywords Hamiltonian Monte Carlo (HMC) · No-U-Turn Sampling (NUTS) · Bayesian inference · *TensorFlow* Replication

Contents

1 Introduction

Statistical modeling and Bayesian inference often require sampling from complex, high-dimensional probability distributions. However, designing efficient sampling algorithms for such problems remains challenging. Classical methods such as random-walk-based Monte Carlo simulations suffer from inefficiency, particularly in high-dimensional spaces where the probability mass is often concentrated in narrow regions. Hamiltonian Monte Carlo (HMC) addresses this issue by introducing auxiliary momentum variables and leveraging Hamiltonian dynamics to explore the probability space more effectively.

Latent Hamiltonian Neural Networks (LHNNs) extend this approach by learning the underlying Hamiltonian dynamics in a latent representation, enabling greater flexibility and scalability. When combined with the No-U-Turn Sampling (NUTS) algorithm, LHNNs can further enhance efficiency by adaptively determining trajectory lengths, removing the need for manual tuning.

This document provides a clear discussion of Hamiltonian Monte Carlo (HMC), No-U-Turn Sampling (NUTS), and extensions such as Latent Hamiltonian Neural Networks (L-HNNs) to address challenges in Bayesian computation and sampling efficiency. The remainder of this paper is organized as follows. Section 2 provides a literature review of foundational methods, including Hamiltonian Monte Carlo (HMC) and No-U-Turn Sampling (NUTS), as well as recent extensions such as Latent Hamiltonian Neural Networks (L-HNNs). This section establishes the theoretical groundwork necessary for understanding modern advancements in Bayesian computation, clarifying key concepts for readers without prior knowledge of Hamiltonian mechanics, Bayesian inference, or Monte Carlo methods. Sections 3, 4, and 5 aim to prevent readers from getting lost in the extensive literature or overlooking the core problem that motivates these methods. Section 3 highlights the challenges posed by high-dimensional probability distributions and the limitations of classical Monte Carlo techniques. Section 4 introduces the structure of L-HNNs, explaining how these networks address these challenges by leveraging latent space representations. Section 5 explores the mechanics of NUTS, showing how specific innovations, including the U-turn condition, overcome critical inefficiencies in traditional sampling approaches. Section 6 complements this discussion by presenting implementation-critical elements that are not fully explored in earlier sections, such as leapfrog integration, the Metropolis acceptance criterion, and the *BuildTree* function, which ensures the practical feasibility of NUTS. Section 7 concludes with a comprehensive description of the testing strategy, covering manual, integration, and unit tests, while underscoring the importance of maintaining updated configuration files to ensure consistency. These sections collectively emphasize the need to focus on the problem itself and highlight how advanced methods bridge theoretical challenges and practical execution.

2 Literature Review

Efficient sampling methods are critical for Bayesian inference, particularly in high-dimensional and complex systems where posterior distributions are difficult to analyze. Traditional Monte Carlo sampling (MCS) methods, including Metropolis-Hastings [Metropolis et al., 1953, Hastings, 1970], have been widely used but suffer from inefficiencies in high-dimensional spaces due to their reliance on random-walk proposals, which result in slow convergence and high autocorrelation between samples. These limitations motivated the development of Hamiltonian Monte Carlo (HMC) and its extensions, which have become powerful tools in modern Bayesian inference.

2.1 Hamiltonian Monte Carlo and No-U-Turn Sampling

HMC, first introduced in the context of molecular dynamics by Duane et al. [1987] and later adapted for Bayesian inference by Neal [2012], combines classical mechanics with Monte Carlo methods to explore probability distributions more efficiently. By introducing auxiliary momentum variables and simulating Hamiltonian dynamics, HMC generates proposals that traverse long distances in the probability space, reducing the random-walk behavior and improving sampling efficiency. These trajectories are guided by the gradient of the log-posterior distribution, enabling HMC to adapt to the geometry of the target distribution and navigate anisotropic curvature effectively.

Despite its advantages, HMC requires careful tuning of the trajectory length and step size, which can be challenging in practice. Hoffman et al. [2014] addressed this issue with the No-U-Turn Sampler (NUTS), an extension of HMC that automatically adapts the trajectory length during sampling. NUTS employs a recursive doubling procedure and stops the trajectory when it detects a "U-turn," ensuring efficient exploration without manual tuning. NUTS has been successfully applied to a wide range of applications and is implemented in popular probabilistic programming frameworks , including Stan Carpenter et al. [2017]. Nishio and Arakawa Nishio and Arakawa [2019] further demonstrated the effectiveness of HMC and NUTS for estimating genetic parameters and breeding values, highlighting their robustness in practical scenarios.

2.2 Extensions and Applications of Monte Carlo Methods

The development of HMC has inspired numerous extensions and applications in diverse fields. Vishnoi [2021] provides an accessible introduction to the theoretical foundations of HMC, offering insights into its utility for sampling in high-dimensional spaces.Che et al. [2021] applied Monte Carlo methods, including Kriging and Variational Bayesian Monte Carlo, to improve predictions of doped UO_2 fission gas release, demonstrating the versatility of these techniques in nuclear energy applications. Similarly, Dhulipala et al. [2022] proposed an active learning framework with multifidelity modeling to improve rare event simulations, showcasing the integration of Monte Carlo methods with machine learning for efficient simulation.

In recent years, the combination of HMC with artificial neural networks has garnered significant attention. Levy et al. [2017] proposed generalizing HMC with neural networks, leveraging their flexibility to model complex systems and improve sampling efficiency. This approach paves the way for methods such as Latent Hamiltonian Neural Networks (LHNNs), which learn Hamiltonian dynamics in a latent space to handle high-dimensional and nonlinearly constrained systems. Applications of neural-network-based HMC for Bayesian inference of reaction kinetics models have further illustrated the potential of these hybrid techniques to address high-dimensional problems. Additionally, hierarchical Bayesian models [Yin et al., 2023] have been explored to design more sophisticated sampling methods that combine domain knowledge with computational efficiency.

2.3 Impact of HMC on Modern Bayesian Inference

Bayesian inference provides a robust framework for estimating parameters and quantifying uncertainty in dynamic and complex systems. Researchers have developed efficient methods to improve Bayesian computation across various domains, including time series analysis, finance, and stochastic modeling. Gerlach et al. [2000] designed Bayesian inference techniques for dynamic mixture models, enabling effective analysis of time-varying phenomena. Kandel et al. [1995] applied Bayesian methods to portfolio efficiency, demonstrating their usefulness in optimizing financial decision-making. Kastner [2017] introduced the stochvol package, delivering computationally efficient tools for analyzing stochastic volatility models in financial data. Whiteley et al. [2010] proposed discrete particle Markov chain Monte Carlo methods, which advanced inference for switching state-space models in discrete-time dynamic systems.

Advancements in Bayesian computation have addressed challenges associated with high-dimensional and complex models. Kastner et al. [2017] developed multivariate factor stochastic volatility models to facilitate inference in large-scale financial data. Han et al. [2022] improved probabilistic model updating by integrating polynomial chaos expansions with Gibbs sampling, which reduced computational costs while maintaining accuracy. These contributions highlight the growing focus on improving computational efficiency in Bayesian inference for modern applications.

Modern methods, including Hamiltonian Monte Carlo (HMC) and No-U-Turn Sampling (NUTS), have revolutionized Bayesian computation by utilizing gradient-based approaches to efficiently explore posterior distributions. Hoffman et al. [2014] proposed NUTS which are regarded as an adaptive extension of HMC, which removes the need for manual tuning of the trajectory length parameter, significantly improving the robustness and accessibility. Dhulipala et al. [2023] expanded this field by introducing Latent Hamiltonian Neural Networks (LHNNs), which integrate HMC with machine learning to sample high-dimensional and latent space representations in a more efficient manner. These advancements have enhanced the potential of Bayesian inference, enabling researchers to address problems that were previously computationally infeasible. By merging traditional methodologies with modern computational frameworks, HMC and NUTS have bridged the gap between classical Bayesian techniques and the computational demands of contemporary scientific and industrial challenges.

3 Motivation: Why Efficient Exploration Is Challenging

Efficient exploration of high-dimensional probability distributions is a core problem in Bayesian inference. These challenges arise due to the inherent nature of high-dimensional spaces, inefficiencies in classical methods, and the need for algorithms that can adapt to the structure of the target distribution. To avoid getting lost in the literature, this section provides an intuitive understanding of these issues and explains why Hamiltonian Monte Carlo (HMC) offers a superior solution.

3.1 Challenges of High-Dimensional Probability Distributions

- **Concentration of Probability Mass**: In high-dimensional spaces, the volume of the space grows exponentially. As a result, the majority of the probability mass concentrates in a narrow "shell" away from the center. For instance, consider a multivariate Gaussian distribution with mean $\mathbf{0}$ and covariance matrix $\mathbf{I}$. The squared

Euclidean distance from the origin follows a χ^2 distribution with d degrees of freedom. As d increases, the density concentrates around $\sqrt{d}$, forming a thin annular region. Random-walk-based methods struggle to locate and stay in this high-probability region because steps taken in random directions often fall outside this shell.

- **Curvature of the Probability Space**: A number of distributions in Bayesian inference exhibit anisotropic curvature, meaning the shape of the distribution stretches more in some directions than others. For example, imagine a stretched-out elliptical valley. If the variance along one direction is much larger than along another, the probability space becomes difficult to traverse. Sampling methods that assume isotropy, such as random-walk proposals, fail to adapt to this curvature, leading to wasted computational effort.
- **Autocorrelation Between Samples**: In classical Monte Carlo methods, new samples are generated based on small perturbations of previous samples, creating a strong dependency between consecutive points. This results in high autocorrelation, reducing the effective sample size. For example, in a random-walk Metropolis algorithm, the next state depends on whether the current proposal lies within the target distribution, leading to slow exploration of the space.

3.2 Limitations of Classical Monte Carlo Methods

Efficient exploration of high-dimensional probability distributions is a core problem in Bayesian inference. These challenges arise due to the inherent nature of high-dimensional spaces, inefficiencies in classical methods, and the need for algorithms that can adapt to the structure of the target distribution. This section provides an intuitive understanding of these issues and explains why Hamiltonian Monte Carlo (HMC) offers a superior solution.

- **Inefficiency in High Dimensions**: The random-walk proposal mechanism generates small, local steps that explore the space incrementally. In high dimensions, the volume of plausible proposals shrinks, and most proposed steps are rejected. A useful metaphor is a person trying to explore a high-dimensional maze by taking random steps—most steps lead to dead ends, and progress is exceedingly slow. Mathematically, the acceptance probability decreases exponentially with the dimensionality d, as small perturbations are less likely to land in high-probability regions.
- **Poor Adaptation to Geometry**: Classical Monte Carlo methods do not account for the local geometry of the target distribution. For example, in a highly anisotropic distribution, random-walk proposals will take many steps to traverse the narrow directions, even if the wide directions are easy to cross. This is analogous to trying to walk along the length of a canyon using random steps—most effort goes into climbing the steep walls instead of advancing along the bottom.
- **High Autocorrelation**: Random-walk methods generate new samples by perturbing the current state, which leads to strong correlations between consecutive samples. This slows down convergence and reduces the effective sample size. For example, if a sample is stuck in a local mode, it can take many steps to escape, resulting in redundant samples that do not represent the full posterior distribution.

These limitations severely affect the performance of classical Monte Carlo methods, particularly when dealing with complex and high-dimensional distributions.

3.3 Why Hamiltonian Monte Carlo (HMC)?

Hamiltonian Monte Carlo overcomes these limitations by leveraging principles from classical mechanics to explore probability distributions more efficiently. The algorithm introduces auxiliary momentum variables $\mathbf{p}$ and simulates the dynamics of a particle moving through the probability space. The system is governed by the Hamiltonian function:

$$H(\mathbf{q}, \mathbf{p}) = U(\mathbf{q}) + K(\mathbf{p}),$$

where $\mathbf{q}$ represents the position (parameters of interest), $\mathbf{p}$ represents the momentum, $U(\mathbf{q}) = -\log P(\mathbf{q})$ is the potential energy derived from the target distribution, and $K(\mathbf{p}) = \frac{1}{2}\mathbf{p}^\top \mathbf{M}^{-1}\mathbf{p}$ is the kinetic energy with mass matrix $\mathbf{M}$. The following features explain why HMC performs better:

- **Efficient Exploration of Probability Space**: Hamiltonian Monte Carlo (HMC) simulates the trajectory of a particle under Hamiltonian dynamics, enabling it to travel long distances in the probability space without getting stuck in local modes. This advantage arises from the use of momentum variables $\mathbf{p}$ and the gradients of the log-probability density, which guide the particle to efficiently explore the target distribution.

In classical Monte Carlo methods, such as the Metropolis-Hastings algorithm, the random-walk proposal mechanism generates small, local steps:

$$\mathbf{q}_{t+1} = \mathbf{q}_t + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2 \mathbf{I}).$$

This approach struggles in high-dimensional scenarios because the volume of plausible proposals shrinks exponentially with the dimensionality d. To see why, consider the acceptance probability in Metropolis-Hastings, which is defined as:

$$\alpha(\mathbf{q}_t, \mathbf{q}_{t+1}) = \min\left(1, \frac{\pi(\mathbf{q}_{t+1})}{\pi(\mathbf{q}_t)}\right),$$

where $\pi(\mathbf{q})$ is the target probability density function (PDF). If the target distribution is a multivariate Gaussian:

$$\pi(\mathbf{q}) = \frac{1}{(2\pi)^{d/2}} \exp\left(-\frac{\|\mathbf{q}\|^2}{2}\right),$$

the acceptance probability depends on the relative distance of the proposed sample $\mathbf{q}_{t+1}$ from the origin compared to the current sample $\mathbf{q}_t$. For a random-walk proposal, the squared distance of the proposed sample from the origin is:

$$\|\mathbf{q}_{t+1}\|^2 = \|\mathbf{q}_t + \epsilon\|^2 = \|\mathbf{q}_t\|^2 + 2\mathbf{q}_t^\top \epsilon + \|\epsilon\|^2.$$

In high dimensions, the term $\|\epsilon\|^2 \sim \chi^2(d)$ grows roughly proportionally to d, while the cross-term $2\mathbf{q}_t^\top \epsilon$ averages to zero because ϵ is independent of $\mathbf{q}_t$. This means that $\|\mathbf{q}_{t+1}\|$ is likely to deviate significantly from the thin shell of high probability (concentrated around $\|\mathbf{q}\| \sim \sqrt{d}$). As a result, the ratio $\pi(\mathbf{q}_{t+1})/\pi(\mathbf{q}_t)$ becomes vanishingly small, causing the acceptance probability to decay exponentially with d. This leads to frequent rejections, slow exploration, and inefficient sampling in high dimensions.

HMC avoids this inefficiency by leveraging Hamiltonian dynamics. Instead of relying on small perturbations, HMC simulates a deterministic trajectory through the probability space, governed by the Hamiltonian:

$$\frac{d\mathbf{q}}{dt} = \nabla_{\mathbf{p}} H, \quad \frac{d\mathbf{p}}{dt} = -\nabla_{\mathbf{q}} H.$$

Here, $\mathbf{q}$ represents the position (parameters of interest), $\mathbf{p}$ is the momentum, and H is the Hamiltonian function that incorporates the target distribution through the potential energy $U(\mathbf{q}) = -\log P(\mathbf{q})$. These dynamics enable the particle to "build up speed" in regions of low curvature (wide valleys) and traverse long distances without being constrained by the dimensionality of the space.

A key reason HMC maintains high acceptance rates, even in high dimensions, is that proposals are informed by the gradients of the log-probability density, $-\nabla U(\mathbf{q})$. This gradient information ensures that the trajectory aligns with the geometry of the target distribution, allowing the sampler to move efficiently into regions of high probability. Unlike random-walk methods, HMC does not propose steps blindly; instead, it uses the structure of the probability space to make long, informed jumps. As a result, the acceptance probability remains stable as the dimensionality d increases, avoiding the exponential decay seen in classical methods.

To provide a concrete comparison, consider a high-dimensional Gaussian distribution with mean $\mathbf{0}$ and covariance matrix $\mathbf{I}$. Classical random-walk proposals must take very small step sizes σ to maintain an acceptable acceptance rate, leading to slow exploration. In contrast, HMC exploits the smoothness of the Gaussian to propose large, efficient steps along trajectories that preserve the probability density, resulting in significantly faster convergence and exploration.

This efficient traversal of the probability space eliminates the slow, incremental steps of random-walk methods and reduces the chance of revisiting the same regions repeatedly. The combination of momentum and gradient-driven proposals allows HMC to explore the space globally, achieving much higher efficiency in high-dimensional settings.

- **Geometry Awareness**: The gradients of the potential energy, $-\nabla U(\mathbf{q})$, guide the trajectory, allowing HMC to adapt to the local geometry of the distribution. For instance, in an anisotropic distribution, the particle moves faster along directions with low curvature and slower along directions with high curvature, akin to navigating a winding river by following the current.
- **Reduced Autocorrelation**: Hamiltonian Monte Carlo reduces autocorrelation between consecutive samples by leveraging deterministic trajectories based on Hamiltonian dynamics. To understand this, consider how classical Monte Carlo methods, such as the Metropolis-Hastings algorithm, generate samples. In these methods, the next state $\mathbf{q}_{t+1}$ is proposed by adding a random perturbation ϵ to the current state $\mathbf{q}_t$:

$$\mathbf{q}_{t+1} = \mathbf{q}_t + \epsilon,$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2\mathbf{I})$. This random-walk behavior leads to small, incremental steps that are heavily dependent on the previous state. As a result, the autocorrelation between consecutive samples is high, especially in high-dimensional spaces where most proposals are rejected, causing the sampler to remain stuck in a local region for many iterations.

In contrast, HMC simulates a trajectory in the probability space using the equations of motion derived from the Hamiltonian:

$$\frac{d\mathbf{q}}{dt} = \nabla_{\mathbf{p}} H, \quad \frac{d\mathbf{p}}{dt} = -\nabla_{\mathbf{q}} H.$$

These dynamics allow the sampler to "leap" across the probability space over many dimensions in a single proposal. Instead of relying on small random steps, HMC uses the gradient of the log-probability to guide the trajectory, enabling large, informed jumps. The resulting state $\mathbf{q}_{t+1}$ depends not only on the current position $\mathbf{q}_t$ but also on the auxiliary momentum $\mathbf{p}$, which introduces a second degree of freedom. This reduces the dependency between consecutive samples and decreases autocorrelation.

A simple comparison of autocorrelation can be made using the effective sample size (ESS), which measures how many independent samples are equivalent to the correlated samples produced by a sampler. Classical Monte Carlo methods often yield small ESS because of high autocorrelation, while HMC achieves a much higher ESS because of its ability to efficiently explore the space without revisiting the same regions repeatedly.

Despite these advantages, HMC requires careful tuning of the step size and trajectory length. Small step sizes improve accuracy but reduce efficiency, while large step sizes risk numerical instability. The No-U-Turn Sampler (NUTS) automates this tuning process by dynamically adjusting the trajectory length to ensure efficient exploration, as discussed in Section 5.

4 Latent Hamiltonian Neural Networks (L-HNN)

Hamiltonian Neural Networks (HNNs) aim to learn the Hamiltonian function H_θ that governs a system's dynamics through Hamilton's equations:

$$\frac{d\mathbf{q}}{dt} = \frac{\partial H_\theta}{\partial \mathbf{p}}, \quad \frac{d\mathbf{p}}{dt} = -\frac{\partial H_\theta}{\partial \mathbf{q}}. \tag{1}$$

The key idea is to model the Hamiltonian H_θ using a neural network parameterized by θ and train it to predict the system's dynamics. Two approaches exist for implementing HNNs: *Direct HNN* and *Latent HNN*. Below, we explain their differences and why Latent HNN provides additional flexibility for complex systems.

4.1 Shared Loss Function

Both Direct HNN and Latent HNN share the same loss function, which measures the discrepancy between the predicted dynamics, derived from the Hamiltonian, and the observed dynamics. Specifically:

$$L = \left\| \frac{\partial H_\theta}{\partial \mathbf{p}} - \frac{\Delta \mathbf{q}}{\Delta t} \right\| + \left\| -\frac{\partial H_\theta}{\partial \mathbf{q}} - \frac{\Delta \mathbf{p}}{\Delta t} \right\|, \tag{2}$$

where $\Delta\mathbf{q}/\Delta t$ and $\Delta\mathbf{p}/\Delta t$ are the observed time derivatives of position $\mathbf{q}$ and momentum $\mathbf{p}$, respectively. This loss function ensures that the learned Hamiltonian H_θ generates dynamics that align with the observed data.

4.2 Direct HNN

In Direct HNN, the Hamiltonian is modeled directly in terms of the observed state variables $\mathbf{q}$ and $\mathbf{p}$:

$$H_\theta = H_\theta(\mathbf{q}, \mathbf{p}), \tag{3}$$

where $\mathbf{q}$ and $\mathbf{p}$ are the generalized position and momentum, respectively. This approach is straightforward and effective when $\mathbf{q}$ and $\mathbf{p}$ are fully observable and accurately represent the state of the system.

However, this direct parameterization imposes limitations on the degrees of freedom and expressivity, reducing adaptability to complex or high-dimensional systems. Consequently, integration errors are more likely to accumulate when predicting Hamiltonian dynamics, particularly during long-term simulations.

4.3 Latent HNN

To address the limitations of Direct HNN, Latent HNN introduces a set of latent variables $\boldsymbol{\lambda} = \{\lambda_1, \lambda_2, \ldots, \lambda_d\}$, where d is the dimension of the latent space. Instead of directly parameterizing the Hamiltonian in terms of $\mathbf{q}$ and $\mathbf{p}$, Latent HNN models the Hamiltonian as:

$$H_\theta = \sum_{i=1}^{d} \lambda_i, \tag{4}$$

where $\boldsymbol{\lambda}$ is the output of a neural network.

Network Architecture

The latent variables $\boldsymbol{\lambda}$ are computed through a multi-layer perceptron (MLP) as follows:

$$\mathbf{u}_p = \phi(\mathbf{W}_{p-1}\mathbf{u}_{p-1} + \mathbf{b}_{p-1}), \quad p \in \{1, \ldots, P\}, \tag{5}$$

$$\boldsymbol{\lambda} = \mathbf{W}_P\mathbf{u}_P + \mathbf{b}_P, \tag{6}$$

where $\mathbf{W}_p$ and $\mathbf{b}_p$ are the weights and biases of the p-th layer, $\phi(\cdot)$ is the activation function, and $\mathbf{u}_0$ represents the input state variables $\mathbf{z} = \{\mathbf{q}, \mathbf{p}\}$.

Advantages of Latent HNN

Through the use of latent variables, Latent HNN provides the following key advantages:

- **Increased Degrees of Freedom:** The latent variables introduce additional degrees of freedom, allowing the model to better capture complex dependencies and dynamics that may not be directly observable in the input space.
- **Enhanced Expressivity:** By leveraging a latent representation, L-HNNs can model a wider range of Hamiltonian functions, making them more flexible compared to traditional HNNs.
- **Reduced Integration Errors:** The increased expressivity and flexibility help reduce integration errors when predicting Hamiltonian dynamics, particularly for long-term simulations or high-dimensional systems.

5 What Is NUTS and Why Use It?

No-U-Turn Sampling (NUTS) is an adaptive version of Hamiltonian Monte Carlo (HMC) designed to eliminate the need for manually tuning the trajectory length, a critical parameter in standard HMC. In traditional HMC, the trajectory length (or equivalently, the number of leapfrog steps) must strike a delicate balance: if it is too short, the sampler behaves like a random walk and explores the space inefficiently; if it is too long, computational resources are wasted, and the sampler risks retracing its path unnecessarily, reducing efficiency.

NUTS addresses this issue by dynamically and adaptively determining the trajectory length during sampling. It uses a recursive doubling procedure to simulate the trajectory and monitors a "U-turn" condition to decide when to stop extending the trajectory. The U-turn condition is given by:

$$(\mathbf{q}_{n+1} - \mathbf{q}_n)^\top \mathbf{p}_{n+1} < 0, \tag{7}$$

where:

- $\mathbf{q}_n$ and $\mathbf{q}_{n+1}$ are the current and next positions in the trajectory,
- $\mathbf{p}_{n+1}$ is the momentum at the next position, and
- the dot product $(\mathbf{q}_{n+1} - \mathbf{q}_n)^\top \mathbf{p}_{n+1}$ measures whether the trajectory is starting to reverse direction.

5.1 Intuition Behind the U-Turn Condition

To understand the U-turn condition intuitively, recall that in HMC, the particle's motion through the probability space is guided by Hamiltonian dynamics. The momentum $\mathbf{p}$ acts like velocity, and the position $\mathbf{q}$ represents the particle's current location. As the particle moves, it explores the probability space, ideally traveling in a direction that efficiently samples the target distribution.

The U-turn condition captures the scenario where the trajectory starts to reverse back toward previously visited regions. Specifically:

- The vector $\mathbf{q}_{n+1} - \mathbf{q}_n$ represents the direction of motion between two successive positions in the trajectory.
- The momentum $\mathbf{p}_{n+1}$ indicates the current "velocity" of the particle.
- The displacement direction $(\mathbf{q}_{n+1} - \mathbf{q}_n)$ and the momentum vectors $\mathbf{p}^{n+1}$) should generally align. That is, the particle is still exploring new areas of the probability space.
- If the dot product $(\mathbf{q}_{n+1} - \mathbf{q}_n)^\top \mathbf{p}_{n+1}$ is negative, it means the particle's momentum is now oriented in the opposite direction of its recent movement. In other words, the trajectory is "turning back" on itself, like making a U-turn. At this point, further extending the trajectory would waste computational resources because the particle would start retracing its path, revisiting regions that have already been explored.

To visualize this, imagine walking through a valley:

- Initially, you walk downhill, guided by the gradients of the terrain (analogous to the gradients of the log-posterior in HMC).
- As you approach a flat or uphill region, your momentum gradually decreases due to resistance.
- If you continue walking too far, you might end up turning back and revisiting the same path you just traveled. The U-turn condition detects this reversal and stops the trajectory before you waste further effort retracing your steps.

By dynamically stopping the trajectory when the U-turn condition is met, NUTS avoids unnecessary computation while ensuring that the sampler remains efficient.

5.2 Why Bidirectional Evolution

The modified bidirectional evolution condition implemented by [Dhulipala et al., 2023] presents a subtle yet important variation from the traditional NUTS algorithm:

- $(\mathbf{q}^+ - \mathbf{q}^-)$ is the displacement vector between the leftmost and rightmost positions of the trajectory,
- $\mathbf{p}^-$ and $\mathbf{p}^+$ are the momenta at $\mathbf{q}^-$ and $\mathbf{q}^+$, respectively, and
- The dot products $(\mathbf{q}^+ - \mathbf{q}^-) \cdot \mathbf{p}^-$ and $(\mathbf{q}^+ - \mathbf{q}^-) \cdot \mathbf{p}^+$ check whether the displacement direction aligns with the momenta at the trajectory's extremes.

The necessity of bidirectional evolution in NUTS algorithm stems from several key considerations:

- **Enhanced Space Exploration**
  - Unidirectional evolution might miss crucial regions of the parameter space
  - Bidirectional sampling provides more comprehensive exploration
  - The algorithm can adapt to local geometric structures in both time directions
- **Time Reversibility**
  - Hamiltonian systems are inherently time-reversible
  - Bidirectional evolution preserves this fundamental physical property
  - Ensures detailed balance in the sampling process
- **Robust U-turn Detection**
  - Considers momentum states at both endpoints ($\mathbf{p}^-$ and $\mathbf{p}^+$)
  - Avoids bias from local trajectory behavior
  - Provides more reliable termination criteria through the condition:

$$s \leftarrow s \cdot \mathbf{1}[(\mathbf{q}^+ - \mathbf{q}^-) \cdot \mathbf{p}^- \geq 0] \cdot \mathbf{1}[(\mathbf{q}^+ - \mathbf{q}^-) \cdot \mathbf{p}^+ \geq 0] \tag{8}$$

- **Symmetric Tree Building**
  - Enables balanced exploration in phase space
  - Reduces correlation between successive samples
  - Improves mixing properties of the Markov chain

6 Additional Implementation Details

This section describes additional key components that are essential for understanding and implementing the discussed ideas. These points include the Metropolis acceptance criterion, leapfrog integration, and specific data generation techniques. While these concepts were not discussed in earlier sections, they are crucial for practical implementation.

6.1 Leapfrog Integration

Leapfrog integration is a symplectic numerical method commonly used to simulate Hamiltonian dynamics. It ensures accurate solutions to the equations of motion while preserving the energy and phase-space volume of the system over long trajectories. The kinetic energy $K(\mathbf{p})$ is assumed to be diagonal and is expressed as:

$$K(\mathbf{p}) = \sum_{i=1}^{d} \frac{p_i^2}{2m_i},$$

where:

- m_i is the mass associated with the i-th degree of freedom (or dimension). It determines the "inertia" of the system in that dimension, i.e., how much a change in momentum p_i affects the velocity $\frac{p_i}{m_i}$.
- d is the total number of dimensions in the system.

In this study, we assume $m_i = 1$ for all dimensions i. This simplifies the equations, as the velocity becomes directly proportional to the momentum ($\frac{p_i}{m_i} = p_i$) and eliminates the need for explicit mass terms in the updates.

The synchronized leapfrog integration scheme updates the position $q_i(t)$ and momentum $p_i(t)$ in each dimension i as follows:

1. **Position Update**:
$$q_i(t + \Delta t) = q_i(t) + \Delta t\, p_i(t) - \frac{\Delta t^2}{2} \frac{\partial H}{\partial q_i(t)}. \tag{9}$$
Here:
   - The term $\Delta t\, p_i(t)$ accounts for the velocity contribution to the position update (since $m_i = 1$).
   - The term $-\frac{\Delta t^2}{2} \frac{\partial H}{\partial q_i(t)}$ accounts for the acceleration due to the potential energy gradient.
   - Δt is the discretization time step that controls the resolution of the simulation.
2. **Momentum Update**: The momentum is updated in two synchronized half-steps:
$$p_i\left(t + \frac{\Delta t}{2}\right) = p_i(t) - \frac{\Delta t}{2} \frac{\partial H}{\partial q_i(t)}, \tag{10}$$
$$p_i(t + \Delta t) = p_i\left(t + \frac{\Delta t}{2}\right) - \frac{\Delta t}{2} \frac{\partial H}{\partial q_i(t + \Delta t)}. \tag{11}$$
The momentum update accounts for the force (negative gradient of the potential energy) acting on the system.
3. **Synchronized Updates**: The leapfrog integration alternates between updating positions and momenta in a synchronized manner:
   - First, $p_i(t)$ is updated by a half-step.
   - Next, $q_i(t)$ is updated by a full step using the updated momentum.
   - Finally, $p_i(t)$ is updated by another half-step, completing the synchronization.

The synchronized leapfrog scheme, as shown in Algorithm 2, ensures that the symplectic property (volume preservation in phase space) and energy conservation are approximately maintained, even over long trajectories. These features make it particularly suitable for Hamiltonian-based methods like Hamiltonian Monte Carlo (HMC).

6.2 Metropolis Acceptance Criterion

In this study, we use the Metropolis Acceptance Criterion in both HNN and L-HNN Monte Carlo. The Metropolis acceptance criterion is used to decide whether to accept or reject a candidate state generated by leapfrog integration. This step ensures that theMonte Carlo sampling process produces samples that follow the target distribution. The update rule is given by:

$$\{\mathbf{q}^i, \mathbf{p}^i\} \leftarrow \{\mathbf{q}^*, \mathbf{p}^*\} \text{ with probability } \alpha,$$

where the acceptance probability α is defined as:

$$\alpha = \min\left(1, \exp\left(H(\mathbf{q}^{i-1}, \mathbf{p}^{i-1}) - H(\mathbf{q}^*, \mathbf{p}^*)\right)\right),$$

and:

- $(\mathbf{q}^*, \mathbf{p}^*)$ is the candidate state after leapfrog integration.
- $(\mathbf{q}^{i-1}, \mathbf{p}^{i-1})$ is the current state.
- $H(\mathbf{q}, \mathbf{p}) = U(\mathbf{q}) + K(\mathbf{p})$ is the Hamiltonian.

If the candidate state is rejected, the current state $(\mathbf{q}^{i-1}, \mathbf{p}^{i-1})$ is retained. This criterion ensures detailed balance and preserves the target distribution.

6.3 Data Generation

Generating data for training Hamiltonian Neural Networks (HNNs) and their extensions involves simulating trajectories of a Hamiltonian system. The following specific steps are used:

1. **Initialize position:** The initial position $\mathbf{q}(0)$ is set to the final position from the previous sample (zero at the beginning):
$$\mathbf{q}(0) = \mathbf{q}^{\text{previous}}.$$
2. **Sample momentum:** The initial momentum $\mathbf{p}(0)$ is drawn randomly from a standard Gaussian distribution:
$$\mathbf{p}(0) \sim \mathcal{N}(0, \mathbf{I}_d),$$
where $\mathbf{I}_d$ is the d-dimensional identity matrix.
3. **Simulate dynamics:** Using leapfrog integration, simulate trajectories by alternating updates to $\mathbf{q}(t)$ and $\mathbf{p}(t)$ over a time interval $[0, T]$ with step size Δt.
4. **Record samples:** Save the position and momentum samples $\{\mathbf{q}(t), \mathbf{p}(t)\}$ at each time step.

These steps ensure that the generated dataset accurately reflects the Hamiltonian system dynamics, yielding suitable training data for HNNs.

Algorithm 1 Hamiltonian neural network training

1: **HNN parameters:** $\boldsymbol{\theta}$; **Training data:** $\{\mathbf{q}, \mathbf{p}\}$
2: Evaluate gradients of the training data $\frac{\Delta \mathbf{q}}{\Delta t}$ and $\frac{\Delta \mathbf{p}}{\Delta t}$
3: Initialize HNN parameters $\boldsymbol{\theta}$
4: Compute HNN Hamiltonian $H_{\boldsymbol{\theta}}$
5: Compute $H_{\boldsymbol{\theta}}$ gradients $\frac{\partial H_{\boldsymbol{\theta}}}{\partial \mathbf{p}}$ and $-\frac{\partial H_{\boldsymbol{\theta}}}{\partial \mathbf{q}}$
6: Compute loss
$$\mathcal{L} = \left\| \frac{\partial H_{\boldsymbol{\theta}}}{\partial \mathbf{p}} - \frac{\Delta \mathbf{q}}{\Delta t} \right\| + \left\| -\frac{\partial H_{\boldsymbol{\theta}}}{\partial \mathbf{q}} - \frac{\Delta \mathbf{p}}{\Delta t} \right\|$$
7: Minimize $\mathcal{L}$ with respect to $\boldsymbol{\theta}$

6.4 Online Monitoring

Online monitoring is a critical component of the proposed sampling workflow, ensuring accuracy and stability when integrating Hamiltonian dynamics. This mechanism evaluates the integration process dynamically and adapts the workflow based on a predefined error metric, reducing the impact of integration errors.

Algorithm 2 Hamiltonian neural networks evaluation in leapfrog integration

Hamiltonian: H; **Initial conditions:** $\mathbf{z}(0) = \{\mathbf{q}(0), \mathbf{p}(0)\}$; **Dimensions:** d; **Steps:** N; **End time:** T
$\Delta t = \frac{T}{N}$
for $j = 0 : N - 1$ **do**
  $t = j\Delta t$
  Compute HNN output gradient $\frac{\partial H_{\boldsymbol{\theta}}}{\partial \mathbf{q}(t)}$
  for $i = 1 : d$ **do**
    $q_i(t + \Delta t) = q_i(t) + \frac{\Delta t}{m_i} p_i(t) - \frac{\Delta t^2}{2m_i} \frac{\partial H_{\boldsymbol{\theta}}}{\partial q_i(t)}$
  end for
  Compute HNN output gradient $\frac{\partial H_{\boldsymbol{\theta}}}{\partial \mathbf{q}(t+\Delta t)}$
  for $i = 1 : d$ **do**
    $p_i(t + \Delta t) = p_i(t) - \frac{\Delta t}{2} \left(\frac{\partial H_{\boldsymbol{\theta}}}{\partial q_i(t)} + \frac{\partial H_{\boldsymbol{\theta}}}{\partial q_i(t+\Delta t)} \right)$
  end for
end for

Algorithm 3 L-HNNs in Hamiltonian Monte Carlo

Hamiltonian: $H = U(\mathbf{q}) + K(\mathbf{p})$; **Samples:** M; **Starting sample:** $\{\mathbf{q}^0, \mathbf{p}^0\}$; **End time for trajectory:** T; **Steps:** N; **Dimensions:** d
for $i = 1 : M$ **do**
  $\mathbf{p}(0) \sim \mathcal{N}(0, \mathbf{I}_d)$
  $\mathbf{q}(0) = \mathbf{q}^{i-1}$
  Compute $\{\mathbf{q}^*, \mathbf{p}^*\} = \{\mathbf{q}(T), \mathbf{p}(T)\}$ with Algorithm 2
  $\alpha = \min\{1, \exp(H(\{\mathbf{q}^{i-1}, \mathbf{p}^{i-1}\}) - H(\{\mathbf{q}^*, \mathbf{p}^*\}))\}$
  With probability α, set $\{\mathbf{q}^i, \mathbf{p}^i\} \leftarrow \{\mathbf{q}^*, \mathbf{p}^*\}$
end for

Algorithm 4 L-HNNs in efficient NUTS with online error monitoring (main loop)

Hamiltonian: $H = U(\mathbf{q}) + K(\mathbf{p})$; **Samples:** M; **Starting sample:** $\{\mathbf{q}^0, \mathbf{p}^0\}$; **Step size:** Δt; **Threshold for leapfrog:** $\Delta_{\text{lf,max}}$; **Threshold for L-HNNs:** $\Delta_{\text{hnn,max}}$; **Number of leapfrog samples:** N_{lf}
Initialize $l_{\text{lf}} = 0$, $n_{\text{lf}} = 0$
for $i = 1 : M$ **do**
  $\mathbf{p}(0) \sim \mathcal{N}(0, \mathbf{I}_d)$
  $\mathbf{q}(0) = \mathbf{q}^{i-1}$
  $u \sim \text{Uniform}([0, \exp\{-H(\mathbf{q}(0), \mathbf{p}(0))\}])$
  Initialize $\mathbf{q}^- = \mathbf{q}(0), \mathbf{q}^+ = \mathbf{q}(0), \mathbf{p}^- = \mathbf{p}(0), \mathbf{p}^+ = \mathbf{p}(0), j = 0, \mathbf{q}^* = \mathbf{q}^{i-1}, n = 1, s = 1$
  if $l_{\text{lf}} = 1$ **then**
    $n_{\text{lf}} \leftarrow n_{\text{lf}} + 1$
  end if
  if $n_{\text{lf}} = N_{\text{lf}}$ **then**
    $l_{\text{lf}} = 0, n_{\text{lf}} = 0$
  end if
  while $s = 1$ **do**
    Choose direction $\nu_j \sim \text{Uniform}(\{-1, 1\})$
    if $j = -1$ **then**
      $\mathbf{q}^-, \mathbf{p}^-, \ldots, \mathbf{q}', \mathbf{p}', n', s', l_{\text{lf}} \leftarrow \text{BuildTree}(\mathbf{q}^-, \mathbf{p}^-, u, \nu_j, j, \Delta t, l_{\text{lf}})$
    else
      $\ldots, \mathbf{q}^+, \mathbf{p}^+, \mathbf{q}', \mathbf{p}', n', s', l_{\text{lf}} \leftarrow \text{BuildTree}(\mathbf{q}^+, \mathbf{p}^+, u, \nu_j, j, \Delta t, l_{\text{lf}})$
    end if
    if $s' = 1$ **then**
      With probability $\min\{1, n'/n\}$, set $\{\mathbf{q}^i, \mathbf{p}^i\} \leftarrow \{\mathbf{q}', \mathbf{p}'\}$
    end if
    $n \leftarrow n + n'$
    $s \leftarrow s' \mathbf{1}[(\mathbf{q}^+ - \mathbf{q}^-) \cdot \mathbf{p}^- \geq 0] \mathbf{1}[(\mathbf{q}^+ - \mathbf{q}^-) \cdot \mathbf{p}^+ \geq 0]$
    $j \leftarrow j + 1$
  end while
end for

Algorithm 5 L-HNNs in efficient NUTS with online error monitoring (build tree function)

function BUILDTREE($\mathbf{q}, \mathbf{p}, u, \nu, j, \Delta t, l_{\text{lf}}$)
  if $j = 0$ **then**
    Base case taking one leapfrog step
    $\mathbf{q}', \mathbf{p}' \leftarrow$ Algorithm 2 with initial conditions: $\mathbf{z}(0) = \{\mathbf{q}, \mathbf{p}\},$ Steps: 1; End Time: Δt
    $l_{\text{lf}} \leftarrow l_{\text{lf}} \vee \mathbf{1}[H(\mathbf{q}', \mathbf{p}') + \ln u > \Delta_{\text{hnn,max}}]$
    $s' \leftarrow \mathbf{1}[H(\mathbf{q}', \mathbf{p}') + \ln u \leq \Delta_{\text{hnn,max}}]$
    if $l_{\text{lf}} = 1$ **then**
      $\mathbf{q}', \mathbf{p}' \leftarrow$ Leapfrog integration with initial conditions: $\mathbf{z}(0) = \{\mathbf{q}, \mathbf{p}\},$ Steps: 1; End Time: Δt
      $s' \leftarrow \mathbf{1}[H(\mathbf{q}', \mathbf{p}') + \ln u \leq \Delta_{\text{lf,max}}]$
    end if
    $n' \leftarrow \mathbf{1}[u \leq \exp\{-H(\mathbf{q}', \mathbf{p}')\}]$
    return $\mathbf{q}', \mathbf{p}', \mathbf{q}', \mathbf{p}', n', s', l_{\text{lf}}$
  else
    Recursion to build left and right sub-trees (follows from Algorithm 3 in Hoffman et al. [2014], with l_{lf} additionally passed to and retrieved from every BuildTree evaluation)
    return $\mathbf{q}^-, \mathbf{p}^-, \mathbf{q}^+, \mathbf{p}^+, \mathbf{q}', \mathbf{p}', n', s', l_{\text{lf}}$
  end if
end function

6.4.1 Error Metric and Threshold

To monitor the accuracy of the integration process, we define an error metric ϵ as:

$$\epsilon \equiv H(q, p) + \ln u,$$

where $H(q, p)$ is the Hamiltonian (the sum of kinetic and potential energy), and u is a uniformly distributed random variable used in the Metropolis acceptance criterion. Ideally, under perfect integration, $\epsilon \leq 0$. However, in practical applications, numerical integration errors can cause $\epsilon > 0$ during iterations.

To handle these cases, we introduce a threshold $\Delta_{\text{max}}^{\text{hnn}}$ for the error metric when using L-HNNs (Hamiltonian Neural Networks). When ϵ exceeds $\Delta_{\text{max}}^{\text{hnn}}$, the L-HNN is considered unreliable, and the sampler automatically switches to standard Leapfrog integration, which uses numerical gradients of the target density.

6.4.2 Cooldown Period with N_{lf}

After switching to standard Leapfrog integration, the sampler undergoes a cooldown period before reintroducing the L-HNN. This cooldown period is controlled by a parameter N_{lf}, which specifies the number of Leapfrog samples to take before returning to L-HNN-based sampling. The cooldown period ensures that the sampler has moved to regions of phase space where the L-HNN is sufficiently accurate, preventing repeated switches caused by transient errors.

6.4.3 Practical Recommendations

In the traditional No-U-Turn Sampler (NUTS), Hoffman and Gelman recommend setting the error threshold for Leapfrog integration, $\Delta_{\text{lf,max}}$, to a large value (e.g., $\Delta_{\text{lf,max}} = 1,000$). However, for L-HNNs, a more conservative threshold is recommended. For instance, setting $\Delta_{\text{hnn,max}} \approx 10$ helps prevent sampling errors caused by the higher integration inaccuracies of L-HNNs.

6.5 *BuildTree* Function

The *BuildTree* function checks this criterion at each recursive step by evaluating whether the trajectory satisfies the stopping condition. As shown in Algorithm 5 if the criterion is violated, the tree-building process halts.

- The *BuildTree* function checks this criterion at each recursive step by evaluating whether the trajectory satisfies the stopping condition. If the criterion is violated, the tree-building process halts.
- *BuildTree* constructs the trajectory by recursively building left and right subtrees. This allows the sampler to balance exploration (new states) and exploitation (accepting samples) while maintaining detailed balance. The recursive nature makes the algorithm computationally efficient, as it avoids the need to store or compute the entire trajectory explicitly.

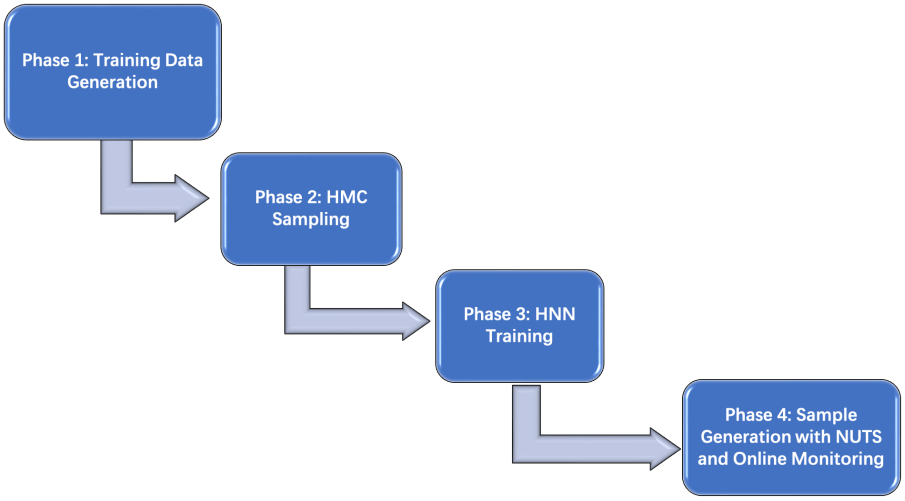

Figure 1: The meta workflow for L-HNN in efficient NUTS

- As the tree expands, *BuildTree* stores candidate samples along the trajectory. It probabilistically selects a sample from the trajectory based on the Hamiltonian and Metropolis acceptance criteria, ensuring that the stationary distribution is preserved.
- In the provided algorithm (Algorithm 5), *BuildTree* incorporates online error monitoring by using thresholds ($\Delta_{\text{hnn,max}}$ and $\Delta_{\text{lf,max}}$. This ensures that the leapfrog integration remains accurate and efficient while building the trajectory.

6.5.1 Summary for Online Monitoring

Online monitoring ensures robust and efficient sampling through the following mechanisms:

- Dynamically evaluating the error metric ϵ during sampling.
- Automatically switching to standard Leapfrog integration when the error exceeds $\Delta_{\text{hnn,max}}$.
- Implementing a cooldown period (N_{lf}) to stabilize the sampler before reintroducing L-HNN-based sampling.
- Preventing numerical instability and improving sample quality by adapting the workflow in real time.
- The *BuildTree* function recursively constructs a balanced binary tree of Hamiltonian trajectory states, ensuring efficient exploration of the parameter space while satisfying the No-U-Turn criterion and maintaining detailed balance.

These features are critical for combining the efficiency of L-HNNs with the reliability of standard Leapfrog integration, ensuring the overall robustness of the sampling process.

6.6 Workflow

Figure 1 shows the overall meta workflow for L-HNNs in efficient NUTS with online error monitoring. The overall workflow consists of four main stages: data generation, HMC sampling, HNN training, and sample generation with online monitoring. Each stage is described below:

1. **Training Data Generation:** The data is generated based on the target Hamiltonian system as described in 6.3. Specifically:
   - Initialize the system with random initial positions and momenta, sampled uniformly or from a Gaussian distribution.
   - Simulate the dynamics of the system over a time span using leapfrog integration to compute trajectories of positions and momenta.
   - Save the trajectories as the dataset for subsequent stages.
2. **HMC Sampling:** Using the leapfrog integration function, Hamiltonian Monte Carlo (HMC) is performed to sample from the target distribution. As demonstrated in Algorithm 2 and 3, the steps are as follows:

- Use the initial dataset to define the target potential energy function $U(q)$ and kinetic energy function $K(p)$.
- Perform leapfrog integration to simulate the Hamiltonian dynamics and generate candidate samples.
- Use the Metropolis acceptance criterion to decide whether to accept or reject the proposed samples.
- Save the accepted samples as training data for the Hamiltonian Neural Network (HNN).

3. **HNN Training:** The HNN is trained on the previously generated samples as shown in Algorithm 1 . The input data consists of $2d$-dimensional vectors, where:
   - The first d dimensions correspond to the positions q (associated with potential energy).
   - The last d dimensions correspond to the momenta p (associated with kinetic energy).

   The HNN learns to predict the derivatives of the Hamiltonian, enabling it to simulate dynamics and generate new samples efficiently.
4. **Sample Generation with NUTS and Online Monitoring:** After training the HNN, the model is used to generate new samples. This stage incorporates the No-U-Turn Sampler (NUTS) with online monitoring to improve sampling efficiency:
   - Use the trained HNN to define the Hamiltonian for the system.
   - Apply NUTS to adaptively determine trajectory lengths and step sizes during sampling through *BuildTree* iteratively function as shown in Algorithm 4 and 5
   - Generate high-quality samples that can be used for downstream tasks or further analysis.

This workflow ensures that the data generation, sampling, training, and sample generation processes are seamlessly integrated, leveraging online monitoring to maximize efficiency and robustness.

7 Unit test and Integration Test Implementation

This section outlines the unit tests and integration tests conducted for the replication code implemented using *TensorFlow*. It is crucial to highlight that, regardless of the type of testing—whether unit testing, integration testing, or manual testing—the configuration files must be updated beforehand to incorporate any required parameter changes. Ensuring the configuration files are up-to-date guarantees consistency across all testing scenarios and prevents discrepancies arising from outdated or incorrect settings.

7.1 Manual Test

To manually run the replicated functions, please perform:

- For the standard Deep Neural Network (DNN) models, execute the script `train_dnn.py` first, followed by `dnn_hmc.py`, `dnn_lmm.py`, and `dnn_nuts_online.py`, all located in the ‘dnns‘ directory.
- For the standard Hamiltonian Neural Network (HNN) models, execute the script `train_hnn.py` first, followed by `hnn_hmc.py`, `hnn_lmm.py`, and `hnn_nuts_online.py`, all located in the `hnns` directory.

7.2 Integration Test

Alternatively, integration tests can be directly conducted in their respective directories using the following command:

```
python -m unittest
```

7.3 Unit Test

The unit test can also be easily applied through command lines. To execute the corresponding tests for DNN, simply replace `hnn` with `dnn` in the command under the directory `dnns`. For example:

- For `test_00_train_dnn` which is the function of `train_dnn.py`:

  ```
  python -m unittest test_integration.TestIntegration.test_00_train_dnn
  ```

- For `test_01_dnn_hmc` which is the function of `dnn_hmc.py` :

```
python -m unittest test_integration.TestIntegration.test_01_dnn_hmc
```

- For `test_02_dnn_lmc` which is the function of `dnn_lmc.py`:

```
python -m unittest test_integration.TestIntegration.test_02_dnn_lmc
```

- For `test_03_dnn_nuts_online` which is the function of `dnn_nuts_online.py`:

```
python -m unittest test_integration.TestIntegration.test_03_dnn_nuts_online
```

7.4 Updating Configuration Files

To modify parameters, follow these steps:

1. Edit the necessary parameters in the `generate_configs.py` file. This file serves as the central configuration generator for all testing scenarios, including unit tests, integration tests, and manual tests.
2. Execute the script to generate the updated configuration files:

```
python generate_configs.py
```

3. Verify that the updated configuration files have been correctly generated in the designated directory (e.g., `configs/`).

References

Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

W Keith Hastings. Monte carlo sampling methods using markov chains and their applications. 1970.

Simon Duane, Anthony D Kennedy, Brian J Pendleton, and Duncan Roweth. Hybrid monte carlo. *Physics letters B*, 195(2):216–222, 1987.

Radford M Neal. Mcmc using hamiltonian dynamics. *arXiv preprint arXiv:1206.1901*, 2012.

Matthew D Hoffman, Andrew Gelman, et al. The no-u-turn sampler: adaptively setting path lengths in hamiltonian monte carlo. *J. Mach. Learn. Res.*, 15(1):1593–1623, 2014.

Bob Carpenter, Andrew Gelman, Matthew D Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus A Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A probabilistic programming language. *Journal of statistical software*, 76, 2017.

Motohide Nishio and Aisaku Arakawa. Performance of hamiltonian monte carlo and no-u-turn sampler for estimating genetic parameters and breeding values. *Genetics Selection Evolution*, 51:1–12, 2019.

Nisheeth K Vishnoi. An introduction to hamiltonian monte carlo method for sampling. *arXiv preprint arXiv:2108.12107*, 2021.

Yifeng Che, Xu Wu, Giovanni Pastore, Wei Li, and Koroush Shirvan. Application of kriging and variational bayesian monte carlo method for improved prediction of doped uo2 fission gas release. *Annals of Nuclear Energy*, 153:108046, 2021.

Somayajulu LN Dhulipala, Michael D Shields, Benjamin W Spencer, Chandrakanth Bolisetti, Andrew E Slaughter, Vincent M Labouré, and Promit Chakroborty. Active learning with multifidelity modeling for efficient rare event simulation. *Journal of Computational Physics*, 468:111506, 2022.

Daniel Levy, Matthew D Hoffman, and Jascha Sohl-Dickstein. Generalizing hamiltonian monte carlo with neural networks. *arXiv preprint arXiv:1711.09268*, 2017.

Yiting Yin, Wenxia Xu, Liping Zhu, Wen Cheng, Yani Mo, and Ren Lin. Bayesian sampling method design based on hierarchical model. 2023.

Richard Gerlach, Chris Carter, and Robert Kohn. Efficient bayesian inference for dynamic mixture models. *Journal of the American Statistical Association*, 95(451):819–828, 2000.

Shmuel Kandel, Robert McCulloch, and Robert F Stambaugh. Bayesian inference and portfolio efficiency. *The Review of Financial Studies*, 8(1):1–53, 1995.

Gregor Kastner. stochvol: Efficient bayesian inference for stochastic volatility (sv) models. 2017.

Nick Whiteley, Christophe Andrieu, and Arnaud Doucet. Efficient bayesian inference for switching state-space models using discrete particle markov chain monte carlo methods. *arXiv preprint arXiv:1011.2437*, 2010.

Gregor Kastner, Sylvia Fr"uhwirth-Schnatter, and Hedibert Freitas Lopes. Efficient bayesian inference for multivariate factor stochastic volatility models. *Journal of Computational and Graphical Statistics*, 26(4):905–917, 2017.

Qiang Han, Pinghe Ni, Xiuli Du, Hongyuan Zhou, and Xiaowei Cheng. Computationally efficient bayesian inference for probabilistic model updating with polynomial chaos and gibbs sampling. *Structural Control and Health Monitoring*, 29(6):e2936, 2022.

Somayajulu LN Dhulipala, Yifeng Che, and Michael D Shields. Efficient bayesian inference with latent hamiltonian neural networks in no-u-turn sampling. *Journal of Computational Physics*, 492:112425, 2023.