

https://dspace.org/

Markov chain monte carlo methods vs. variational inference approaches in posterior density estimation for bayesian neural networks

Chabumba, Daisy Reneilwe

2025-12
University of the Witwatersrand, Johannesburg
https://hdl.handle.net/10539/50155

Chabumba, Daisy Reneilwe. (2025). Markov chain monte carlo methods vs. variational inference approaches in posterior density estimation for bayesian neural networks. [Master's dissertation, University of the Witwatersrand, Johannesburg]. WIReDSpace. https://hdl.handle.net/10539/50155

Downloaded from a DSpace repository. https://dspace.org/

MSc in e-Science Research Report



Markov chain monte carlo methods vs. variational inference approaches in posterior density estimation for bayesian neural networks

By

Daisy Reneilwe Chabumba

Supervisor(s):

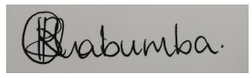
Dr Farai Mlambo
Cyril Chironda

A research report submitted to the Faculty of Science, University of the Witwatersrand, in partial fulfillment of the requirements for the degree of Master of Science

December 4, 2025

Declaration

I, Daisy Reneilwe Chabumba, declare that this research report is my own work. It is being submitted for the Degree of Master of Science in e-Science to the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination at any other university.



December 4, 2025

Dedication

I want to dedicate this research report to the pillars of my life whose unwavering love, guidance, and sacrifices have shaped the person I am today. I dedicate this work to my late grandparents, Alpheus and Chayisa Chabumba; your love and legacy inspire me daily. Though you are no longer here, your teachings and values remain deeply rooted in my heart. This achievement is a tribute to the foundation you built for our family. To my mother, Khomotso Chabumba, your sacrifices, resilience, and unconditional love have been the greatest gifts. You have been my source of strength, pushing me to dream beyond my limits and supporting me through every step of this journey. I cannot thank you enough for everything you have done for me.

To my younger sister, Boitumelo Mashashane, your encouragement and belief in me have meant the world. You have been my rock, my friend, and my greatest cheerleader. I am grateful for your love and support, and I hope this achievement serves as a reminder that perseverance makes anything possible. Most importantly, to my son, Vukosi Mukasi, you are my greatest motivation and the reason I strive to be the best version of myself. Every late night, every sacrifice, and every effort I have put into this journey has been with you in my heart. You are my joy, my purpose, and my inspiration.

I hope this work serves as a testament to the power of dedication and hard work, may it remind you that your dreams are always within reach. To each of you, thank you from the depths of my heart. This achievement is as much yours as it is mine.

Acknowledgement

This research report would not have been possible without the guidance, support, and encouragement of several individuals and institutions I am deeply grateful to. First and foremost, I thank God for granting me the strength, wisdom, and perseverance to complete this work. His grace has been my foundation, guiding me through every challenge and victory.

To my supervisors, Dr. Farai Mlambo and Cyril Chironda, I sincerely thank you for your invaluable guidance and mentorship. Your patience, insight, and dedication have been instrumental in shaping this research, and I deeply appreciate the time and effort you have invested in my academic growth. To Dr. Helen Robertson, thank you for your unwavering support, patience, and guidance throughout my master's journey. You have been a pillar of strength from the beginning, always believing in my potential even when I doubted myself.

To Casey Sparks, I am truly grateful for your assistance and dedication in ensuring I had the necessary support to pursue my studies. Your efforts have played a significant role in making this achievement possible. To the National e-Science Postgraduate Teaching & Training Platform (NEPTTP), I extend my heartfelt appreciation for the financial and academic support that has enabled me to pursue this research. Your commitment to fostering academic excellence has been invaluable.

Each of you has played a vital role in my journey, and I am forever grateful for your contributions. Thank you for your unwavering support, guidance, and encouragement.

List of Figures

3.1 Distribution of Median House Prices Across the Dataset 22
3.2 Feature Distributions for the California Housing Dataset 23
3.3 Correlation Map of Dataset Features 24

4.1 Model Training and Validation Loss Using Full Dataset 37
4.2 Model Training and Validation Loss Using 75% of the Dataset 38
4.3 Model Training and Validation Loss Using 50% of the Dataset 39
4.4 Parameter Trace Plots from MCMC Sampling using Full Dataset 40
4.5 Parameter Trace Plots from MCMC Sampling using 75% of the Dataset . . 41
4.6 Parameter Trace Plots from MCMC Sampling using 50% of the Dataset . . 42

List of Tables

3.1 Overview of Key Variables in the Cleaned California Housing Dataset . . . 22

4.1 Performance Metrics for MCMC and VI Models Using Full Dataset 35
4.2 Performance Metrics for MCMC and VI Models Using 75% of the Dataset 35
4.3 Performance Metrics for MCMC and VI Models Using 50% of the Dataset 35
4.4 Training Time Required for VI and MCMC Across Experiments 36
4.5 Comparison of Peak Memory Usage for VI Training and MCMC Sampling 38
4.6 Statistical Significance of Model Differences Based on T-Test Results . . . 39

Glossary of Key Concepts

Bayesian Inference (BI) An approach to statistical learning where existing beliefs (priors) are updated using observed data to produce a more informed belief (posterior). It's especially useful for modeling uncertainty.

Bayesian Neural Network (BNN) A neural network that models its weights as probability distributions instead of fixed values. This allows the model to express uncertainty in its predictions, making it particularly useful in critical applications.

Burn-in Period (in MCMC) The early phase of Markov Chain Monte Carlo sampling, where initial values are discarded to allow the chain to stabilize and better represent the target distribution.

Conditional Distribution The probability distribution of a variable given the known or fixed values of one or more other variables.

Convergence Behaviour Describes how efficiently an inference algorithm settles into stable, accurate estimates of the target distribution over time.

DenseFlipout A probabilistic layer in TensorFlow Probability used in variational inference, allowing efficient uncertainty estimation in neural networks.

Early Stopping A technique that stops training once model performance stops improving on validation data, helping to avoid overfitting.

Evidence Lower Bound (ELBO) A metric used in variational inference that approximates the likelihood of observed data. Maximizing the ELBO helps approximate the true posterior.

Exploratory Data Analysis (EDA) The initial phase of data analysis where patterns, anomalies, and trends are explored visually and statistically.

Gibbs Sampling (GS) An MCMC method where variables are updated one at a time, using their conditional distributions, to generate samples from a joint posterior.

Joint Probability Distribution The combined probability of two or more random variables occurring together—key to modeling dependencies in Bayesian networks.

Kullback-Leibler (KL) Divergence A measure of the difference between two probability distributions. In VI, it quantifies how close the approximation is to the true posterior.

Markov Chain Monte Carlo (MCMC) A family of algorithms for sampling from complex probability distributions, using Markov chains to approximate posteriors.

Markov Property A principle stating that future states of a system depend only on the current state, not past states—a foundation for Markov-based models.

Metropolis-Hastings (MH) A widely used MCMC algorithm that proposes new samples and accepts or rejects them based on a probability ratio.

Outliers Data points that deviate significantly from the majority of a dataset. They can affect model training and statistical analysis.

p-value A probability value that indicates how likely the observed data would occur under the null hypothesis. Low p-values suggest statistical significance.

Posterior Distribution The updated belief about model parameters after observing new data obtained using Bayes' Theorem.

Random Walk Metropolis (RWM) A Metropolis algorithm where new states are proposed based on small random steps from the current state.

Scalability The ability of an algorithm or model to maintain performance as the size of the data or model complexity increases.

Stationary Distribution The long-run distribution that a Markov chain converges to under certain conditions, regardless of its starting point.

Standardization A data preprocessing step that transforms variables to have a mean of zero and a standard deviation of one.

T-value The test statistic used in T-tests to evaluate the difference between sample means relative to variability within the samples.

TensorFlow (TF) An open-source machine learning library developed by Google for building and deploying deep learning models.

TensorFlow Probability (TFP) An extension of TensorFlow for probabilistic programming, supporting Bayesian modeling and uncertainty estimation.

Uncertainty Quantification The process of evaluating and expressing the confidence level in a model's predictions.

Variational Inference (VI) A method that frames Bayesian inference as an optimization problem, enabling faster and more scalable approximations of posterior distributions.

Contents

1 Introduction **1**
1.1 Background of the Study 1
1.2 Literature Review 2
1.2.1 Markov Chain Monte Carlo for Bayesian Inference 3
1.2.2 Variational Inference for Bayesian Inference 3
1.2.3 Advancements in Bayesian Neural Networks 4
1.3 Statement of the Problem 5
1.4 Research Gap and Motivation 5
1.5 Research Questions 5
1.6 Aim and Objectives 6
1.7 Limitations and Assumptions 6
1.8 Contributions of the Study 6
1.9 Overview of the Report 7

2 Theoretical Review **8**
2.1 Monte Carlo Sampling 8
2.1.1 Mathematical Formulation 8
2.1.2 Applications of Monte Carlo Sampling 9
2.2 Markov Chain Methods 9
2.2.1 Mathematical Formulation 10
2.2.2 Classification of Markov Chains 10
2.2.3 Applications of Markov Chain Methods 10
2.3 Markov Chain Monte Carlo Method 11
2.3.1 Mathematical Formulation 11
2.3.2 Transition Probability and Stationary Distribution 11
2.3.3 Gibbs Sampling 11
2.3.4 Applications of MCMC Methods 12
2.3.5 Metropolis-Hastings Algorithm 12
2.4 Variational Inference Methods 13
2.4.1 Mathematical Formulation 13
2.4.2 Kullback-Leibler (KL) Divergence 13
2.4.3 Evidence Lower Bound (ELBO) 13
2.4.4 Applications of Variational Inference 14
2.4.5 Advantages and Limitations 14
2.5 Bayesian Neural Networks 14
2.5.1 Mathematical Formulation 14
2.5.2 Applications of Bayesian Neural Networks 15

2.5.3 Advantages and Challenges . . . 15
2.6 Evaluation Metrics . . . 15
2.6.1 Predictive Performance Metrics . . . 15
2.6.2 Mean Squared Error (MSE) . . . 15
2.6.3 Root Mean Squared Error (RMSE) . . . 16
2.6.4 Mean Absolute Error (MAE) . . . 16
2.6.5 R-Squared (R^2) . . . 16
2.6.6 Uncertainty Quantification . . . 16
2.6.7 Prediction Interval Coverage Probability (PICP) . . . 16
2.7 Statistical Analysis . . . 17
2.7.1 One-Sample T-Test . . . 17
2.7.2 Independent T-Test . . . 17
2.7.3 Paired T-Test . . . 17
2.7.4 Results of a T-Test . . . 17

3 Methodology **19**
3.1 Data Description . . . 19
3.2 Data Preprocessing . . . 20
3.2.1 Data Overview and Data Transformation . . . 20
3.2.1.1 Data Overview . . . 20
3.2.1.2 Handling Missing Data . . . 20
3.2.1.3 Handling Outliers . . . 21
3.2.1.4 Data Transformation . . . 21
3.2.2 Exploratory Data Analysis . . . 21
3.2.2.1 Descriptive Statistics . . . 21
3.2.2.2 Distribution of Target Variable . . . 22
3.2.2.3 Feature Distributions and Skewness . . . 23
3.2.2.4 Correlation Matrix . . . 24
3.3 Model Setup and Implementation . . . 25
3.3.1 Data Splitting . . . 25
3.3.2 Model Description . . . 25
3.3.3 Model Architecture . . . 26
3.3.3.1 Bayesian Neural Network Implementation Using MCMC Sampling . . . 26
3.3.3.2 Bayesian Neural Network Implementation Using VI . . . 26
3.3.4 Estimation of the Posterior Density . . . 27
3.3.4.1 Posterior Estimation using MCMC . . . 27
3.3.4.2 Posterior Estimation using VI . . . 27
3.4 Model Evaluation . . . 27
3.5 Comparative Analysis . . . 28
3.5.1 Scalability . . . 28
3.5.1.1 Computational Time . . . 28
3.5.1.2 Memory Usage . . . 28
3.5.1.3 Convergence Behavior . . . 28
3.5.1.4 Scalability for Large Models . . . 29
3.5.2 Computational Efficiency . . . 29
3.5.2.1 Computational Time Analysis . . . 29

3.5.2.2 Convergence Behavior for VI . . . 30
3.5.2.3 Convergence Behavior for MCMC . . . 30
3.5.2.4 Memory Usage Analysis . . . 30
3.5.3 Practical Implementation and Ease of Use . . . 31
3.5.3.1 Code Complexity . . . 31
3.5.3.2 Library Support . . . 31
3.5.3.3 Training and Tuning . . . 31
3.5.3.4 User-Friendliness . . . 31
3.5.3.5 Interpretability and Debugging . . . 32
3.6 Declaration of AI Usage . . . 32

4 Results and Discussion **33**
4.1 Results . . . 33
4.1.1 Summary of Findings . . . 33
4.1.1.1 Bayesian Neural Network Using VI . . . 34
4.1.1.2 Bayesian Neural Network Using MCMC . . . 34
4.1.2 Evaluation of MCMC and VI Models . . . 35
4.1.2.1 Experiment 1 . . . 35
4.1.2.2 Experiment 2 . . . 35
4.1.2.3 Experiment 3 . . . 35
4.1.3 Computational Time . . . 35
4.1.3.1 Training Time Comparison . . . 36
4.1.4 Convergence Behavior . . . 36
4.1.4.1 Convergence Behavior for VI . . . 36
4.1.4.2 Convergence Behavior for MCMC . . . 37
4.1.5 Memory Usage Comparison: VI vs. MCMC . . . 38
4.1.6 Statistical Analysis . . . 39
4.2 Discussion of Results . . . 40
4.2.1 Interpretation of Results . . . 40
4.2.2 Discussion of Findings in Relation to Literature . . . 41
4.2.3 Implications . . . 43
4.3 Contributions of the Study . . . 44
4.4 Limitations . . . 45

5 Conclusion and Future Work **47**
5.1 Conclusion . . . 47
5.2 Future Work . . . 48

6 Python Code **50**

Bibliography **62**

Chapter 1

Introduction

This chapter provides the background to the study and a detailed literature review, including an examination of Markov Chain Monte Carlo for Bayesian inference, Variational Inference for Bayesian inference, and recent advancements in Bayesian Neural Networks. It then outlines the statement of the problem, highlights the research gap and motivation, and presents the research questions. The chapter further sets out the aim and objectives of the study, discusses its limitations and assumptions, and describes the expected contributions of the study. Finally, it gives an overview of the report structure.

1.1 Background of the Study

In recent years, scientific machine learning has emerged as a powerful approach for tackling complex problems across various scientific disciplines, including engineering, chemistry, and physics (Brunton and Kutz, 2022). Among the different methodologies, Bayesian Inference (BI) stands out due to its strength in quantifying uncertainty in predictive models. A central component of the Bayesian framework is Markov Chain Monte Carlo (MCMC), which facilitates the estimation of posterior distributions in Bayesian models (Brunton and Kutz, 2022). MCMC is a collection of techniques for drawing samples from probability distributions (Banerjee et al., 2014). Commonly used MCMC algorithms, such as Gibbs Sampling (GS) (Brooks et al., 2011) and Metropolis-Hastings (MH) (Zhang et al., 2022), are frequently employed to evaluate the joint posterior distribution in neural network models.

Although MCMC is the popular method that provides accurate results, it is criticized for its computational inefficiency and slow convergence, making it the worst choice for large-scale applications such as BNNs. To address the challenges with MCMC, researchers have explored the VI, which is treated as an alternative. VI is a technique used in Statistics to approximate complex probability distribution. While MCMC relies sorely on sampling, VI deals with the inference problem as an optimization problem, approximating the posterior with a parametrized distribution (Kingma and Welling, 2014). VI offers faster convergence and scalability, which makes it more suitable for large datasets and complex models. It introduces an approximation bias, meaning that the quality of the posterior density depends on the choice of variational family and optimization strategy (Neal, 2012).

With the rise of BNNs, these inference techniques have obtained significant attention. Unlike the standard neural networks, which use fixed point estimates, BNNs treat their weights

as probability distributions, which allows them to capture uncertainty in predictions (Blundell et al., 2015). This capability is valuable in high-stakes decision-making tasks, such as medical diagnostics, autonomous systems, and financial forecasting (Gelman et al., 2013). However, estimating the posterior distribution of network weights remains a computationally challenging problem. A thorough examination of BNNs has been conducted in (Arbel et al., 2023; Magris and Iosifidis, 2023). MCMC and VI offer different solutions to this challenge, each with distinct advantages and limitations. The selection between the methods is influenced by computational efficiency, scalability, and uncertainty quantification.

BNNs combine the predictive power of the neural networks with BI to offer strong uncertainty estimation in model predictions (Gelman et al., 2013). They are useful where uncertainty quantification is essential, such as in risk assessment and decision-making under uncertainty, active learning, and strong predictions in safety-critical domains (Blundell et al., 2015; Gelman et al., 2013). It is computationally difficult to estimate the posterior density in BNNs. As a result, approximate inference methods are employed, MCMC methods and VI approaches being the most prominent (Gelman et al., 2013).

MCMC produces samples from the posterior distribution by creating a Markov chain that gradually converges to the target distribution (Neal, 2012). These methods are known for their theoretical guarantee of asymptotically exact samples, which ensures high accuracy in posterior estimation. Conversely, VI uses a more straightforward, parameterised distribution to approximate the posterior distribution (Kingma and Welling, 2014). The main aim is to reduce the true posterior's and the approximation distribution's Kullback-Leibler (KL) divergence (Kingma and Welling, 2014). VI methods, including Mean-Field VI and Black-box VI, offer significant computational advantages over MCMC by providing faster convergence and improved scalability to high-dimensional parameter spaces and large data sets.

1.2 Literature Review

By incorporating prior distributions, BNNs reduce overfitting and improve model generalization, making them particularly useful in data-limited applications (Bishop, 1997). They integrate prior distributions by providing a justified approach to uncertainty estimation (Lampinen and Vehtari, 2001). This feature allows BNNs to offer stronger predictions, particularly in domains with limited data or high uncertainty (Fan et al., 2008). The Hebbian learning rule and Bayesian combination function further enhance BNNs' generalization capabilities, making them competitive against other classification methods in various domains (Kononenko, 1989). However, the challenge in implementing BNNs lies in posterior inference, where the distribution of the network's weights is estimated from the observed data. This computationally expensive task requires efficient inference techniques such as MCMC and VI. Recent studies have investigated the convergence of MCMC and VI methods in Bayesian neural networks. Although MCMC has struggled with scalability in larger models and big data, emerging techniques such as Langevin proposal distributions show promise in addressing these challenges (Chandra et al., 2023).

1.2.1 Markov Chain Monte Carlo for Bayesian Inference

MCMC is an effective simulation-based method for estimating complex model statistics, particularly in BI (Gilks, 2005). It plays a critical role in assessing posterior distributions in Bayesian models (Skaanning, 2006). A notable extension of MCMC is the Reversible Jump MCMC, which provides flexibility in model selection and inference (Fan and Sisson, 2010). In genetics, MCMC methods are extensively applied in pedigree and linkage analysis to estimate posterior probabilities of specific genes (Skaanning, 2006).

Additionally, (Fan and Sisson, 2010) discussed the use of likelihood-free MCMC. This method bypasses the need for explicit computation of likelihood functions, making it highly suitable for complex models with stubborn likelihoods. The MH and GS algorithms are the most popular MCMC techniques for generating samples from probability distributions (Maruyama, 2014). These methods are essential in Bayesian computation, offering strong solutions for posterior simulation across various applications (Müller, 2004). In financial econometrics, MCMC methods are employed in asset pricing models to address parameter uncertainty and incorporate varying degrees of belief (Verhofen, 2005).

Furthermore, MCMC techniques have proven to be costly in molecular evolution studies, where they enable the analysis of complex probability distributions and effectively solve high-dimensional integration problems (Larget, 2005). Despite its strengths, MCMC has notable challenges when compared to VI. (Nummenmaa et al., 2007) highlighted the limitations of using a single-peaked variational distribution as a proxy for the complex and often multi-modal genuine posterior distribution in hierarchical Bayesian estimations. MCMC methods provide more flexible approximations but at a higher computational cost. (Saadi et al., 2011) explored the use of MCMC algorithms for parameter estimation in dynamic systems, emphasizing their power in approximating complex integrals and simulating joint distributions.

(Salimans et al., 2015; **?**) highlighted the potential of combining these approaches for posterior density estimation in BNNs, emphasizing the benefits of fast posterior approximation and the ability to trade off computational efficiency for accuracy. The applications of MCMC go beyond statistics, extending to areas like particle technology, where it is used to model processes such as mixing and grinding (Li, 2024). Recent developments in MCMC include using adaptive algorithms to improve efficiency and reliability (Craiu et al., 2022; Vihola, 2020). As research in BI and MCMC continues to evolve, these methods remain vital for addressing complex probabilistic systems across various disciplines (Craiu et al., 2022).

1.2.2 Variational Inference for Bayesian Inference

VI has been explored as a faster and cost-efficient alternative to mitigate MCMC's computational cost. VI approximates the posterior by optimizing a parametric distribution to be close to the true posterior (Nummenmaa et al., 2007). While VI is computationally efficient, it introduces approximation errors because it relies on a single-peaked variational distribution, which may oversimplify the true posterior (Nummenmaa et al., 2007).

VI has been widely adopted for its ability to efficiently approximate posterior distributions, particularly in high-dimensional Bayesian models where MCMC is computationally infeasible. (Mukherjee and Sen, 2021) examined the application of VI in linear regression models with high-dimensional feature spaces, deriving a variational formula for the posterior distribution and establishing conditions under which the mean-field approximation achieves accurate results. (Emerson and Copestake, 2017) applied VI to logical inference in text-based semantic representations, demonstrating improved convergence during training. (Harik and Shazeer, 2010) introduced a framework for variational program inference, which estimated the Kullback-Leibler (KL) divergence between the true posterior and a guided posterior distribution.

Additionally, (Opper, 2019) explored the use of VI for stochastic differential equations, approximating the distribution over unknown paths and the likelihood of drift processes. Several VI techniques have been developed specifically for posterior density estimation in BNNs. (Maceda et al., 2024) suggested a framework that used a neural network trained on simulated data to estimate posterior distributions of model parameters efficiently. (Shridhar et al., 2018) presented a method for uncertainty estimation in Bayesian convolutional neural networks, capturing both model and data uncertainties.

(Jaakkola and Jordan, 2000) illustrated the application of variational methods for obtaining closed-form approximations to posterior distributions in logistic regression and graphical models. (Krishnan et al., 2019) provided a technique for defining informative weight priors in Bayesian deep neural networks that enhanced scalable VI with reliable uncertainty quantification and faster training convergence. A promising area of research is the hybridization of MCMC and VI, with (Salimans et al., 2015) exploring the integration of these approaches to leverage MCMC's sampling accuracy and VI's computational efficiency, thereby creating more scalable BI frameworks.

1.2.3 Advancements in Bayesian Neural Networks

BNNs have become a powerful extension of conventional neural networks, integrating uncertainty estimation and probabilistic modeling (Chang, 2021; Mullachery et al., 2018). BNNs combined the strengths of neural networks' function approximation capabilities with stochastic models' ability to generate posterior distributions and provide probabilistic guarantees on predictions (Mullachery et al., 2018). Recent advancements in BNNs include improved Bayesian classifiers and ML algorithms for model structuring and integration with various tools such as management decision networks and geographic information systems (Marcot and Penman, 2019). BNNs use probabilistic layers that capture uncertainty over-weights and activations, trained using BI (Chang, 2021).

Research has focused on improving the accuracy and efficiency of posterior estimation in BNNs. (Wrede et al., 2021) proposed a strong, cohesive approach that used BNNs to learn summary statistics and directly estimate the posterior density. This approach was more efficient and strong in large prior spaces. (Ghalebikesabi et al., 2022) extended this work by introducing a Bayesian nonparametric model with an auto-regressive likelihood decomposition and Gaussian process prior, which improved the modeling capacity of existing recursive Bayesian density estimators. (Jantre et al., 2020) further enhanced the Bayesian approach by

introducing a method for quantile regression using a BNN, which was consistent under an incorrectly specified model. These studies collectively demonstrate the potential of BNNs in improving the accuracy and efficiency of posterior density estimation.

1.3 Statement of the Problem

Despite the work done by other researchers regarding the theoretical foundations and practical applications of MCMC and VI, there is a lack of detailed comparative studies evaluating their performance in BNNs. Several key aspects remain underexplored, such as posterior estimation accuracy, computational efficiency, scalability to large datasets, quality of uncertainty quantification, and ease of implementation. Addressing these gaps is essential for guiding researchers and practitioners in selecting the most appropriate inference method for their applications.

1.4 Research Gap and Motivation

While MCMC and VI have been widely applied in BNN-based classification tasks in (Blundell et al., 2015; Mullachery et al., 2018; Shridhar et al., 2018), their effectiveness in regression-based posterior density estimation remains underexplored. Existing works have primarily evaluated uncertainty quantification in categorical decision-making but lack systematic comparisons of computational efficiency, scalability, and uncertainty quantification across different inference techniques in practical regression tasks. This study seeks to fill this gap by systematically evaluating MCMC and VI in BNN regression, offering insights into their comparison for real-world applications.

1.5 Research Questions

This study seeks to explore the following research questions:

- How do MCMC and VI compare in estimating the posterior distribution accuracy of BNNs when the factors that influence their performance are taken into account?
- Which of MCMC or VI demonstrates greater computational efficiency in high-dimensional neural network models, considering how efficiency varies with model complexity?
- How does the scalability of MCMC compare to that of VI across 50%, 75%, and 100% of the original dataset while reflecting key performance comparisons?
- What are the fundamental differences in uncertainty quantification between MCMC-based and VI-based BNNs in terms of their impact on model reliability?
- How do practical implementation challenges and ease of use of MCMC and VI, such as parameter tuning, convergence diagnostics, and similar considerations, affect their real-world applicability?

1.6 Aim and Objectives

This study aims to compare MCMC and VI methods for estimating the posterior density of BNN weights by evaluating these methods across multiple dimensions, including accuracy, computational efficiency, scalability, and uncertainty quantification. It also seeks to provide guidelines for selecting the most suitable BI technique in regression-based neural network applications.

The objectives of this study are to:

- Compare the performance of MCMC and VI in estimating the posterior distribution of BNN weights.
- Evaluate the accuracy of posterior estimations generated by MCMC and VI.
- Evaluate the computational efficiency and scalability of both methods.
- Analyse the quality of uncertainty quantification achieved by each method.
- Compare the two methods regarding practical implementation and ease of use.

1.7 Limitations and Assumptions

MCMC is not a foolproof solution for Bayesian inference. It has some challenges and limitations that should be addressed. MCMC can be slow and inefficient, taking a long time and many samples to reach convergence and obtain accurate estimates (Gelman et al., 2013). It can also be tricky and deceptive, producing misleading results if the correct algorithm, parameters, or samples are not chosen. Furthermore, MCMC can be complex and confusing, involving many technical details that are not familiar to practitioners (Brooks et al., 2011). A drawback of VI is the computational challenge in obtaining the optimal approximation to the actual posterior density (Xu, 2020). VI aims to minimize the gap between the actual and approximation posteriors within a given family, which can be non-convex and computationally demanding (Posch et al., 2019). While recent approaches have aimed to improve accuracy, they may require iterative gradient updates, making them computationally intensive (Kim and Pavlovic, 2020).

1.8 Contributions of the Study

This study contributes to the field of BNNs by providing a comparative analysis of MCMC and VI for posterior estimation. It evaluates their accuracy, computational efficiency, scalability, and uncertainty quantification performance, highlighting the trade-offs between these methods. The findings show that while MCMC delivers more precise posterior estimates, it is computationally expensive, whereas VI offers faster and more scalable inference but introduces approximation bias. Additionally, the study provides practical insights into both methods' implementation challenges and real-world applicability, guiding researchers and practitioners in selecting appropriate inference techniques based on model complexity and dataset size. It also identifies opportunities for hybrid approaches that combine MCMC's accuracy with VI's efficiency, paving the way for future advancements in Bayesian deep learning.

1.9 Overview of the Report

This study is organized as follows. Chapter 2 provides key theoretical foundations. It highlights the strengths and limitations of each approach, setting the stage for the comparative analysis conducted in this research. Chapter 3 details the research methodology, including data preprocessing, model implementation, and inference techniques. This chapter systematically explains how MCMC and VI are applied to the problem and the evaluation metrics used to assess their performance.

Chapter 4 presents the results, offering an in-depth analysis and interpretation. The discussion emphasises the comparison between MCMC and VI regarding accuracy, computational efficiency, scalability, and uncertainty quantification, providing insights into their practical applications. It also discusses the findings concerning the literature. Chapter 5 summarizes the key findings of this study and examines their implications. It also outlines potential directions for future research, emphasizing areas where further exploration could enhance the understanding and application of BI in Machine Learning (ML). Finally, Chapter 6 provides the Python code used for the study.

Chapter 2

Theoretical Review

In this chapter, we present the theoretical foundation by exploring the origins of key concepts and models. We discuss the core principles underlying each approach, their integration with other methodologies, and their mathematical formulations.

2.1 Monte Carlo Sampling

Monte Carlo Sampling (MCS) is a computational technique that uses random sampling to approximate numerical results. It is particularly effective for solving deterministic problems but too complex for analytical solutions (Metropolis and Ulam, 1949). The method involves generating random numbers, evaluating the fraction that satisfies a given property, and estimating numerical solutions to complex problems (Robert and Casella, 2004). The origins of Monte Carlo methods trace back to the 1930s when physicist Enrico Fermi used early versions of the technique to study neutron diffusion. However, it was formally developed by Stanislaw Ulam and John von Neumann at Los Alamos National Laboratory during the Manhattan Project in the 1940s (Metropolis and Ulam, 1949). Ulam conceptualized the approach while attempting to predict neutron behavior in nuclear reactions, leading to the structured development of Monte Carlo techniques as a computational tool (Kalos and Whitlock, 2008).

2.1.1 Mathematical Formulation

MCS estimates the expected value of a function $g(x)$ over a domain D, which is represented by the integral:

$$I = \int_D g(x)\, dx, \tag{2.1}$$

where I is the true value of the integral, $g(x)$ is the function being integrated and D is the domain over which the integration occurs.

To approximate this integral using Monte Carlo methods, the following steps are applied:

1. Random Sampling: Generate N random samples $x_1, x_2, ..., x_N$ uniformly distributed over the domain D.

2. Function Evaluation: Compute the function values $g(x_1), g(x_2), ..., g(x_N)$.

3. Averaging: Estimate the integral using the sample mean:

$$\hat{I} = \frac{1}{N} \sum_{i=1}^{N} g(x_i), \tag{2.2}$$

where $\hat{I}$ is the Monte Carlo estimate of the integral, N is the number of random samples and x_i are randomly drawn sample points from a uniform distribution over D.

According to the Law of Large Numbers, as N increases, $\hat{I}$ converges to the true value of the integral I, with an approximation error that scales as $\frac{1}{\sqrt{N}}$ (Robert and Casella, 2004).

2.1.2 Applications of Monte Carlo Sampling

Monte Carlo methods are widely used across various fields because they can model uncertainty and approximate solutions to complex probabilistic problems. They have widespread applications across various domains.

In physics, these methods are employed in statistical mechanics and quantum field theory to simulate particle interactions and diffusion processes (Landau and Binder, 2014). In finance, they are instrumental in option pricing, risk assessment and portfolio management by modeling stochastic processes in financial markets (Glasserman, 2004). The field of engineering benefits from Monte Carlo methods through structural reliability analysis and optimization problems, enabling system performance assessment under uncertainty (Hurtado, 2004). These methods are used in mathematics for numerical integration, especially in high-dimensional problems where traditional quadrature methods prove inefficient (Kalos and Whitlock, 2008). Lastly, in computer graphics, Monte Carlo techniques are essential for rendering realistic images through ray tracing and global illumination by accurately simulating light transport (Pharr et al., 2016).

MCS has become a fundamental computational tool in numerous scientific and engineering disciplines. Leveraging statistical randomness provides practical solutions for otherwise difficult problems using deterministic methods. Since its formalization by Ulam and von Neumann in the 1940s (Metropolis and Ulam, 1949), Monte Carlo methods have played a critical role in advancements across physics, finance, engineering, and artificial intelligence (Robert and Casella, 2004).

2.2 Markov Chain Methods

A Markov chain is a stochastic process that moves between states within a finite or countable set of possibilities. Its key feature is the Markov property, which asserts that the probability of transitioning to a future state depends only on the current state, not on the sequence of past states (Markov, 1906). This lack of memory makes Markov chains particularly effective for modeling various random processes. Andrey Andreyevich Markov, a Russian mathematician, introduced the concept of Markov chains in the early 20th century. In his 1906 paper, Markov extended the law of large numbers to sequences of dependent variables, laying the foundation for what would later be known as Markov chains (Markov, 1906). He applied the Markov chain theory to studying letter sequences in Alexander Pushkin's poem

"Eugene Onegin," demonstrating the practical applicability of his mathematical concepts (Hayes, 2005).

2.2.1 Mathematical Formulation

A Markov chain is defined by a set of states $S = \{s_1, s_2, ..., s_n\}$ and a transition probability matrix P, where each element P_{ij} represents the probability of transitioning from state s_i to state s_j:

$$P_{ij} = P(X_{t+1} = j | X_t = i), \tag{2.3}$$

where P_{ij} is the probability of transitioning from state i to state j and X_t represents the system's state at time t.

The transition probabilities must satisfy the following conditions:

1. $P_{ij} \geq 0$ (probabilities must be positive).
2. $\sum_j P_{ij} = 1$ (each row in P sums to 1).

The state vector $\pi^{(t)}$ at time t represents the probability distribution over states at that time. The equation governs the evolution of the state vector:

$$\pi^{(t+1)} = \pi^{(t)} P. \tag{2.4}$$

A stationary distribution π is a vector that remains unchanged under the transition matrix P:

$$\pi = \pi P. \tag{2.5}$$

2.2.2 Classification of Markov Chains

A Markov chain is characterized by several important properties influencing its dynamics and suitability for various applications. An irreducible Markov chain is one where every state can be reached from any other state, indicating strong connectivity within the system. If the chain is aperiodic, it lacks regular cycles, meaning it does not return to specific states at fixed intervals. When a Markov chain exhibits irreducibility and aperiodicity, it is called ergodic, implying that it will converge to a unique stationary distribution over time.

2.2.3 Applications of Markov Chain Methods

Markov chains have extensive applications across various fields. In queuing theory, they are used to model customer service systems, allowing for the prediction of wait times and optimization of service efficiency. Markov chains help analyze market trends and consumer behavior in economics and finance, contributing to economic models and financial forecasts. In genetics, they are applied to study gene sequences and evolutionary processes. In computer science, Markov chains are implemented in machine learning algorithms, natural language processing, and the Page-Rank algorithm used by search engines. Additionally, in physics and chemistry, Markov chains are employed to model particle systems and chemical reactions, providing insights into dynamic behaviors at the molecular level.

Markov chain methods provide a strong structure for modeling systems where the future state depends only on the current state. Since their inception by Andrey Markov, these methods have found widespread application in diverse fields, offering insights into complex stochastic processes.

2.3 Markov Chain Monte Carlo Method

Markov Chain Monte Carlo (MCMC) methods are a family of algorithms designed to approximate probability distributions by generating a Markov chain with the desired target distribution as its stationary distribution (Metropolis et al., 1953). These techniques are especially valuable for drawing samples from complex, high-dimensional distributions where direct sampling is challenging or impractical. The MCMC method was first introduced by Nicholas Metropolis and his colleagues in 1953 while working on statistical mechanics problems (Metropolis et al., 1953). Later, Hastings extended the technique in 1970 to create the more general MH algorithm (Hastings, 1970). The GS algorithm, another widely used MCMC method, was introduced by Geman and Geman in 1984 in image processing (Geman and Geman, 1984).

2.3.1 Mathematical Formulation

The objective of MCMC is to generate samples $X_1, X_2, \ldots, X_N$ from a probability distribution $P(X)$ where direct sampling is complex. Instead of sampling independently, MCMC constructs a Markov chain that converges to the target distribution.

2.3.2 Transition Probability and Stationary Distribution

A Markov chain is defined by its transition probability $P(X'|X)$, which determines the probability of moving from state X to X'. The stationary distribution $\pi(X)$ satisfies:

$$\pi(X') = \sum_X \pi(X) P(X'|X), \tag{2.6}$$

where $\pi(X)$ is the stationary distribution and $P(X'|X)$ is the transition probability from state X to X'.

For the Markov chain to converge to $\pi(X)$, it must satisfy the detailed balance condition:

$$\pi(X)P(X'|X) = \pi(X')P(X|X'), \tag{2.7}$$

which ensures that the probability of moving between states remains consistent over time.

2.3.3 Gibbs Sampling

GS is a special case of MCMC where each variable is sampled from its conditional distribution while keeping other variables fixed (Geman and Geman, 1984). Given a multivariate distribution $P(X_1, X_2, ..., X_n)$, GS updates each variable iteratively:

$$X_i^{(t+1)} \sim P(X_i|X_1^{(t)}, ..., X_{i-1}^{(t)}, X_{i+1}^{(t)}, ..., X_n^{(t)}), \tag{2.8}$$

where $X_i^{(t+1)}$ is the new value for the i-th variable and $P(X_i|X_{-i})$ is the conditional probability of X_i, given all other variables X_{-i}.

2.3.4 Applications of MCMC Methods

MCMC methods have extensive applications across various fields. In Bayesian inference, MCMC is employed to approximate posterior distributions, facilitating the estimation of complex models where direct sampling is challenging (Chib and Greenberg, 1995). Within computational physics, MCMC is used in simulations for molecular dynamics and statistical mechanics, enabling the exploration of system configurations and thermodynamic properties (Berg, 2004). In financial modeling, MCMC methods estimate risk and optimize portfolio management by modeling the uncertainty and dynamics of economic systems (Chib and Greenberg, 1996). In genetics, MCMC is used in phylogenetics and gene sequence analysis to reconstruct evolutionary histories and assess genetic variation (Yang, 2014). MCMC methods provide a powerful approach to sampling from complex probability distributions. By leveraging the properties of Markov chains, these methods enable efficient approximation of distributions where direct sampling is infeasible.

2.3.5 Metropolis-Hastings Algorithm

In this study, we implement the Random Walk Metropolis (RWM) Algorithm, which is a special case of the Metropolis–Hastings method. The RWM uses a symmetric proposal distribution, so the acceptance probability simplifies to the ratio of the target densities at the proposed and current states. This variant is applied here to generate posterior samples for the Bayesian Neural Network models.

The MH algorithm represents one of the most important and widely adopted approaches in MCMC. Given a target distribution $\pi(X)$, the algorithm generates samples using a proposal distribution $Q(X'|X)$ and an acceptance probability α given by (Metropolis et al., 1953; Metropolis and Ulam, 1949):

Algorithm 1 Metropolis-Hastings Algorithm

Require: Target distribution $\pi(X)$, proposal distribution $Q(X'|X)$

Ensure: Samples $\{X_t\}_{t=1}^{T}$ approximately distributed according to $\pi(X)$

Initialize X_0

for $t = 1$ to T **do**

Sample candidate state $X' \sim Q(X'|X_t)$

Compute acceptance probability:

$$\alpha = \min\left(1, \frac{\pi(X')Q(X|X')}{\pi(X)Q(X'|X)}\right)$$

if Uniform(0,1) $\leq \alpha$ **then**

Accept X' ▷ Move to the candidate state

$X_{t+1} \leftarrow X'$

else

Reject X' ▷ Stay in the current state

$X_{t+1} \leftarrow X_t$

end if

end for

2.4 Variational Inference Methods

VI is a method used in Bayesian statistics and ML to approximate complex probability distributions (Ganguly and Earp, 2021).

2.4.1 Mathematical Formulation

Given a dataset $\mathbf{X}$ and a set of latent variables $\mathbf{Z}$, the goal of VI is to approximate the posterior distribution:

$$p(\mathbf{Z}|\mathbf{X}) = \frac{p(\mathbf{X}, \mathbf{Z})}{p(\mathbf{X})}, \tag{2.9}$$

where $p(\mathbf{Z}|\mathbf{X})$ is the true posterior distribution, $p(\mathbf{X}, \mathbf{Z})$ is the joint distribution of the observed data and latent variables and $p(\mathbf{X}) = \int p(\mathbf{X}, \mathbf{Z}) d\mathbf{Z}$ is the marginal likelihood, which is often intractable.

Since computing $p(\mathbf{Z}|\mathbf{X})$ directly is complex, VI introduces an approximate posterior $q(\mathbf{Z}; \boldsymbol{\lambda})$, parameterized by $\boldsymbol{\lambda}$ and seeks to make it as close as possible to the true posterior.

2.4.2 Kullback-Leibler (KL) Divergence

The closeness between the approximate posterior $q(\mathbf{Z}; \boldsymbol{\lambda})$ and the true posterior $p(\mathbf{Z}|\mathbf{X})$ is measured using the Kullback-Leibler (KL) divergence:

$$D_{KL}(q(\mathbf{Z}; \boldsymbol{\lambda}) \| p(\mathbf{Z}|\mathbf{X})) = \int q(\mathbf{Z}; \boldsymbol{\lambda}) \log \frac{q(\mathbf{Z}; \boldsymbol{\lambda})}{p(\mathbf{Z}|\mathbf{X})} d\mathbf{Z}, \tag{2.10}$$

where $D_{KL}(q\|p)$ measures the divergence between distributions q & p, $q(\mathbf{Z}; \boldsymbol{\lambda})$ is the variational approximation of the true posterior and $p(\mathbf{Z}|\mathbf{X})$ is the true posterior distribution.

Since $p(\mathbf{X})$ is unknown, direct minimization of D_{KL} is not feasible. Instead, we maximize the Evidence Lower Bound (ELBO), which provides a tractable alternative (Ni et al., 2024).

2.4.3 Evidence Lower Bound (ELBO)

The ELBO is derived as follows:

$$\log p(\mathbf{X}) = D_{KL}(q(\mathbf{Z}; \boldsymbol{\lambda}) \| p(\mathbf{Z}|\mathbf{X})) + \mathcal{L}(\boldsymbol{\lambda}), \tag{2.11}$$

where the ELBO is given by:

$$\mathcal{L}(\boldsymbol{\lambda}) = \mathbb{E}_{q(\mathbf{Z};\boldsymbol{\lambda})}[\log p(\mathbf{X}, \mathbf{Z})] - \mathbb{E}_{q(\mathbf{Z};\boldsymbol{\lambda})}[\log q(\mathbf{Z}; \boldsymbol{\lambda})]. \tag{2.12}$$

Since D_{KL} is always positive, maximizing $\mathcal{L}(\boldsymbol{\lambda})$ minimizes the KL divergence, making $q(\mathbf{Z}; \boldsymbol{\lambda})$ a better approximation of the true posterior (Ni et al., 2024).

2.4.4 Applications of Variational Inference

VI is widely applied across various domains because it can efficiently approximate complex posterior distributions. One notable application is latent variable models, which play a significant role in probabilistic modeling. For example, LDA leverages VI to estimate topic distributions within large text corpora, enabling efficient topic modeling in natural language processing. Another critical application is Variational Autoencoders (VAEs), where VI enables efficient training by approximating intractable posterior distributions of latent variables (Kingma and Welling, 2013). Furthermore, BNNs benefit from VI as it provides a scalable method for approximating posterior distributions over neural network weights, thereby enabling strong uncertainty quantification in deep learning models.

2.4.5 Advantages and Limitations

VI offers several advantages over traditional methods like MCMC. One key benefit is its scalability, as VI often converges faster and is more suitable for handling large datasets. Additionally, as an optimization-based approach, VI's deterministic nature provides deterministic approximations, eliminating the randomness inherent in sampling methods. However, VI also comes with certain limitations. The approximation quality heavily depends on the choice of the variational family, which can sometimes restrict the ability to capture complex posterior distributions. Moreover, the optimization process in VI may introduce bias, potentially leading to underestimating uncertainty.

2.5 Bayesian Neural Networks

BNNs are a class of artificial neural networks that incorporate BI principles to model uncertainty in predictions. BNNs incorporate BI into traditional neural networks, allowing for the quantification of uncertainty in forecasts. This approach enhances model robustness and interpretability, addressing challenges such as overfitting and overconfidence in predictions (Arbel et al., 2023).

2.5.1 Mathematical Formulation

In a standard neural network, the goal is to find the optimal weights $\mathbf{w}$ that minimize a loss function $\mathcal{L}(\mathbf{w})$. BNNs, however, treat the weights as random variables and seek to determine their posterior distribution given the observed data $\mathcal{D}$. According to Bayes' theorem:

$$p(\mathbf{w}|\mathcal{D}) = \frac{p(\mathcal{D}|\mathbf{w})\, p(\mathbf{w})}{p(\mathcal{D})}, \tag{2.13}$$

where $p(\mathbf{w}|\mathcal{D})$ is the posterior distribution of the weights given the data, $p(\mathcal{D}|\mathbf{w})$ is the likelihood of the observed data given the weights, $p(\mathbf{w})$ is the prior distribution over the weights and $p(\mathcal{D})$ is the marginal likelihood, serving as a normalization constant.

Direct computation of $p(\mathbf{w}|\mathcal{D})$ is often difficult due to the high dimensionality of $\mathbf{w}$. Approximate inference techniques such as VI or MCMC estimate this posterior distribution (Magris and Iosifidis, 2022).

2.5.2 Applications of Bayesian Neural Networks

BNNs have found successful applications across various fields due to their ability to model uncertainty effectively. In healthcare, BNNs are employed for predictive modeling in medical diagnosis, where they quantify uncertainty in treatment outcomes, thereby enhancing decision-making in clinical settings (Gawlikowski et al., 2021). In autonomous systems, particularly self-driving cars, BNNs contribute to improved decision-making by providing uncertainty estimates in object detection and navigation, which are critical for safety and reliability. Additionally, in financial modeling, BNNs assess risks and enhance decision-making processes by capturing uncertainty in financial predictions, making them valuable tools in dynamic and volatile economic environments.

2.5.3 Advantages and Challenges

BNNs offer notable advantages in ML applications. One key benefit is their ability to quantify uncertainty, enabling probabilistic predictions that allow for better evaluation of model confidence. Additionally, BNNs incorporate prior distributions, which act as a form of regularization and help reduce the risk of overfitting, especially in complex models. However, BNNs also face specific challenges. One major limitation is their computational complexity, as approximate inference methods used in BNNs can be computationally intensive. Furthermore, scalability remains a challenge, as applying BNNs to large-scale problems is often blocked by the complexity of performing accurate inference.

2.6 Evaluation Metrics

Evaluation metrics are important for measuring the effectiveness of predictive models. They offer quantitative insights into how accurately a model captures the true underlying values. In this section, we focus on two main categories of evaluation metrics: Predictive Performance Metrics and Uncertainty Quantification.

2.6.1 Predictive Performance Metrics

Predictive performance metrics measure the accuracy of a model's predictions compared to actual values. Common metrics include the Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and R-Squared (R^2). For all the metrics, M is the total number of observations, i is the index representing each individual data point, where $i = 1, 2, \ldots, M$, $\hat{x_i}$ is the predicted value for the i-th observation and x_i is the actual (true) value for the i-th observation.

2.6.2 Mean Squared Error (MSE)

MSE measures the average squared difference between predicted and actual values, giving higher weight to more significant errors (Pedregosa et al., 2011). A lower MSE indicates that the model's predictions are closer to actual values.

$$MSE = \frac{1}{M}\sum_{i=1}^{M}(\hat{x_i} - x_i)^2. \tag{2.14}$$

2.6.3 Root Mean Squared Error (RMSE)

RMSE is the square root of MSE, making it more interpretable as expressed in the same unit as the target variable (Pedregosa et al., 2011). A lower RMSE indicates better predictions.

$$RMSE = \sqrt{\frac{1}{M}\sum_{i=1}^{M}(\hat{x}_i - x_i)^2}. \tag{2.15}$$

2.6.4 Mean Absolute Error (MAE)

MAE measures the average absolute difference between predicted and actual values (Pedregosa et al., 2011). A smaller MAE indicates better model accuracy.

$$MAE = \frac{1}{M}\sum_{i=1}^{M}|\hat{x}_i - x_i|. \tag{2.16}$$

2.6.5 R-Squared (R^2)

R^2 measures how well the model explains the variance in the actual values (Pedregosa et al., 2011). It ranges from 0 to 1, where a value closer to 1 indicates that the model explains most of the variance, while a value near 0 suggests poor predictive ability. Negative values mean the model performs worse than a simple mean prediction (Pedregosa et al., 2011).

$$R^2 = 1 - \frac{\frac{1}{M}\sum_{i=1}^{M}(\hat{x}_i - x_i)^2}{\frac{1}{M}\sum_{i=1}^{M}(x_i - \bar{x})^2}. \tag{2.17}$$

2.6.6 Uncertainty Quantification

Uncertainty quantification is essential in probabilistic modeling and Bayesian inference, as it provides insights into the reliability of model predictions. One key metric for uncertainty estimation is the Prediction Interval Coverage Probability (PICP).

2.6.7 Prediction Interval Coverage Probability (PICP)

PICP measures the proportion of actual values within the model's predicted confidence intervals. It evaluates how well the model's uncertainty estimation captures the actual observations (Pedregosa et al., 2011).

$$PICP = \frac{1}{M}\sum_{i=1}^{M} 1\{x_i \in [x_i^L - x_i^U]\}. \tag{2.18}$$

For PICP calculation, we used the Z-score for a 95% confidence interval:

$$z = \Phi^{-1}\left(0.5 + \frac{\alpha}{2}\right), \tag{2.19}$$

where Φ^{-1} is the inverse cumulative distribution function of the standard normal distribution. The confidence interval was determined as follows:

$$x_i^L = \hat{x}_i - z \cdot \sigma_i, \quad x_i^U = \hat{x}_i + z \cdot \sigma_i, \tag{2.20}$$

where σ_i represents the standard deviation of the predictions.

2.7 Statistical Analysis

Statistical analysis is important in identifying differences between groups, especially in hypothesis-driven studies. The T-test is one of the most commonly used techniques for comparing the means of two groups. It assesses whether the observed differences are statistically significant or simply the result of random variation (Bevans, 2023). There are three primary types of T-tests: the One-Sample T-Test, the Independent T-Test, and the Paired T-Test, each tailored to specific research contexts (Bevans, 2023).

2.7.1 One-Sample T-Test

The One-Sample T-test is used to check if the mean of a single sample significantly differs from a known population mean. This T-test is calculated as follows:

$$t = \frac{\bar{X} - \mu}{s/\sqrt{n}}, \tag{2.21}$$

where $\bar{X}$ is the sample mean, μ is the population mean, s is the sample standard deviation and n is the sample size.

2.7.2 Independent T-Test

The Independent T-test compares the means of two independent groups to determine whether their differences are statistically significant. This T-test is given by:

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}}, \tag{2.22}$$

where $\bar{X}_1, \bar{X}_2$ is the sample means of both groups, s_1^2, s_2^2 is the variances of both samples and n_1, n_2 is the sample sizes of both groups.

2.7.3 Paired T-Test

The Paired T-test is applied when comparing two related groups, where measurements are found from the same objects under distinct conditions. This T-test is calculated as:

$$t = \frac{\bar{d}}{s_d/\sqrt{n}}, \tag{2.23}$$

where $\bar{d}$ is the mean difference between paired observations, s_d is the standard deviation of differences and n is the number of paired observations.

2.7.4 Results of a T-Test

It is essential to include the T-value, *p*-value, and degrees of freedom when presenting T-test results. These values provide insight into whether the difference between the two groups is statistically significant (Bevans, 2023). The decision to reject or retain the null hypothesis depends on the *p*-value, following these criteria:

- If $p < 0.05$: The group variation is considered statistically significant, and the null hypothesis is rejected.

- If $p > 0.05$: The variation is not statistically significant, indicating that the null hypothesis cannot be rejected.

In summary, a *p*-value below 0.05 provides evidence to reject the null hypothesis and conclude that the group differences are statistically significant, whereas a *p*-value above 0.05 indicates insufficient evidence to reject the null hypothesis.

Chapter 3

Methodology

This chapter presents the research approach used in this study, outlining the steps taken to process the data, implement the models, and evaluate their performance. We provide a structured approach to the experimental design and analysis for the two BI methods.

We describe the dataset in Section 3.1, detailing its attributes, sources, and relevance to the study. Section 3.2 covers data preprocessing techniques, including missing data handling, handling outliers, data transformation, and Exploratory Data Analysis (EDA). Implementing the BNNs using both MCMC and VI is discussed in Section 3.3, where we outline the model architecture, training procedures, model description, and posterior estimation techniques. We use evaluation metrics in Section 3.4 to assess model performance, which includes both predictive accuracy metrics and uncertainty quantification measures. Section 3.5 presents the comparative analysis between the two models, focusing on scalability, computational efficiency, memory usage, convergence behavior, and practical implementation. The scalability experiments were conducted by varying dataset sizes (50%,75% and 100%) to observe how each model adapts to increasing data. Finally, Section 3.6 details the use of Artificial Intelligence (AI) tools in this study, outlining their specific contributions and ensuring transparency in their application.

3.1 Data Description

The California Housing Prices Dataset is popular in ML, and Statistics is mostly used for regression tasks. It contains information about housing prices in California from the 1990 census. The dataset has eight types of metrics and 20,640 districts. The target variable, MedHouseVal (median house value), is predicted based on income, house age, average rooms, population, and location coordinates. The dataset was provided by sklearn.datasets (CaliforniaHousing, 2024).

The dataset was chosen for this study because of its relevance to real-world regression tasks and usefulness for evaluating ML BI methods. This dataset presents several challenges relevant to BI, including continuous-valued features with varying scales and potential non-linear relationships. It is ideal for assessing the accuracy and uncertainty quantification capabilities of both MCMC and VI methods. It provides a rich testing ground for posterior estimation techniques. While more complex datasets such as CIFAR-10 could provide additional insights, the structured numerical nature of the California Housing Prices Dataset allows for a

clear and controlled comparison of MCMC and VI. Moreover, this dataset allows for scalability experiments by varying the dataset size, directly evaluating how dataset complexity affects each method's computational efficiency and convergence.

The California Housing Prices dataset contains these variables:

Metric	**Description**
Median Income	The central income level for the block group.
House Age	The median age of homes in the block group.
Average Rooms	The typical number of rooms per household.
Average Bedrooms	The average number of bedrooms in each household.
Population	The total number of individuals living in the block group.
Average Occupants	The average number of residents per household.
Latitude	The latitude coordinate of the block group.
Longitude	The longitude coordinate of the block group.
Median House Value	The median house value for the households.

3.2 Data Preprocessing

This section outlines the data preprocessing steps to prepare the dataset for modeling. Data preprocessing ensures stable posterior estimation by preventing numerical lack of stability and optimizing model convergence. The process included checking for missing values, removing outliers using the Z-score method (Shukla, 2024) and standardizing features using Z-score transformation to ensure numerical stability (Scikit-Learn, 2024). Standardization aids in maintaining balanced weight distributions, which is essential for both MCMC and VI. Additionally, EDA was performed to examine feature distributions, detect skewness, and analyze correlations between variables. A correlation matrix was also analyzed to identify relationships between features and guide the selection of relevant predictors.

3.2.1 Data Overview and Data Transformation

This section provides an overview of the dataset and describes the preprocessing steps to ensure data quality and consistency. We examine the dataset's structure, check for missing values, and identify potential outliers that may impact model performance.

3.2.1.1 Data Overview

The dataset indicates that all columns contain continuous numerical data of type float64, with feature values varying wildly across observations. Since all columns are of type float64, the dataset comprises continuous numerical features, making it compatible with most statistical and ML models without requiring additional conversion or encoding.

3.2.1.2 Handling Missing Data

We checked the dataset for missing values using isnull function (Pandas, 2024), which revealed that all nine columns are complete, with no missing data. This eliminated the need for imputation or cleaning to handle missing values.

3.2.1.3 Handling Outliers

After removing outliers, the dataset was successfully cleaned using the Z-score method (Shukla, 2024). The final dataset contains 19,794 observations. The extreme outliers in features have been reduced, while the dataset retains meaningful variations necessary for modeling.

3.2.1.4 Data Transformation

We then normalized the numerical features using Z-score transformation from the StandardScaler function to standardize the dataset to zero mean and unit variance (Scikit-Learn, 2024). This step ensured features with larger magnitudes did not dominate the model while improving numerical stability, convergence efficiency, and overall model performance.

Normalization was performed using the Z-score transformation, which is given by this formula (Scikit-Learn, 2024):

$$X_{\text{scaled}} = \frac{X - \mu}{\sigma}, \tag{3.1}$$

where X is the original feature value, μ is the mean of the feature and σ is the standard deviation of the feature.

3.2.2 Exploratory Data Analysis

This section analyzes the dataset to understand its structure, distribution, and relationships between features.

3.2.2.1 Descriptive Statistics

We present the summary statistics for each feature in the dataset in Table 3.1. The summary includes the mean, median, standard deviation, interquartile range, and minimum and maximum values.

The cleaned and standardized dataset now contains 19,794 observations, confirming that outliers have been removed. Each feature has a mean close to 0 and a standard deviation near 1, indicating that the dataset has been properly standardized for modeling. The minimum and maximum values show that extreme values have been effectively handled, particularly in variables such as Average Rooms (-3.58 to 5.87) and Average Bedrooms (-5.44 to 10.71), ensuring that no unrealistic values mislead the analysis.

Despite standardization, the dataset retains its geographical diversity, with latitude ranging from (-1.44 to 2.96) and longitude from (-2.38 to 2.51), preserving the spatial characteristics of housing in California. Additionally, the interquartile range (25% to 75%) shows that most observations are now concentrated within a controlled range, reducing unstableness and improving model stability. The cleaned dataset is now balanced and ready for BNN modeling, allowing for inference using MCMC and VI.

Table 3.1: Overview of Key Variables in the Cleaned California Housing Dataset

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
Count	19794	19794	19794	19794	19794	19794	19794	19794
Mean	-0.0037	0.0032	-0.0020	0.0040	-0.0016	0.0001	0.0060	-0.0051
Std Dev	0.9985	1.0014	0.9957	1.0023	0.9971	1.0357	1.0025	1.0011
Min	-2.0154	-2.2469	-3.5860	-5.4433	-1.6493	-2.5537	-1.4463	-2.3867
25%	-0.7430	-0.7989	-0.6753	-0.4664	-0.6802	-0.5908	-0.7943	-1.1127
50%	-0.1484	0.0055	-0.0492	-0.1499	-0.2257	-0.1342	-0.6395	0.5360
75%	0.5743	0.6490	0.5908	0.2211	0.4302	0.4135	0.9835	0.7758
Max	3.6204	1.8557	5.8695	10.7088	4.2663	36.3208	2.9677	2.5095

3.2.2.2 Distribution of Target Variable

We found the distribution to be right-skewed, indicating that most houses have relatively lower median values, with a steady decline in frequency as the price increases. The peak that occurred around 1.5 to 2.5 (in \$100,000s), suggested that most houses are in that price range. Additionally, a significant spike at the upper limit of 5.0 suggested a possible capping effect, meaning that house values above this threshold may have been limited within the dataset. A density curve confirmed that the distribution does not follow a perfectly normal pattern, emphasizing the importance of using Bayesian methods to account for the uncertainty and non-linearity in housing price predictions.

Figure 3.1 shows the distribution, which displays several key characteristics of the data, as detailed below:

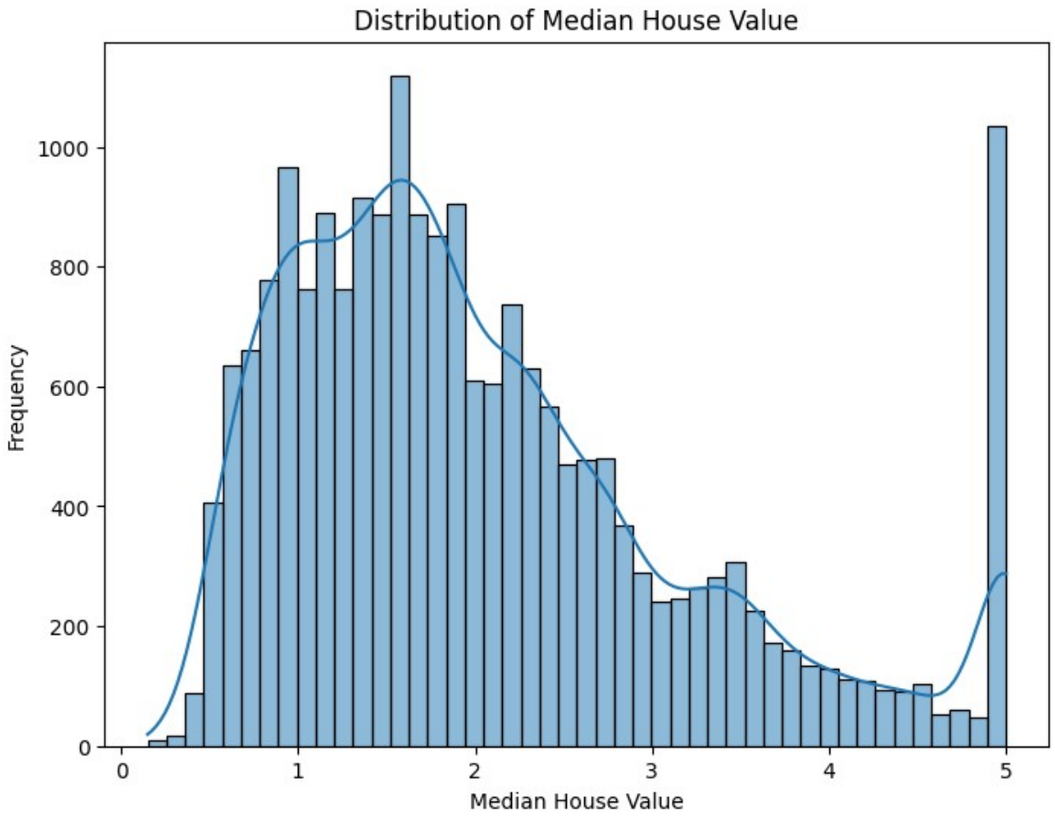


Figure 3.1: Distribution of Median House Prices Across the Dataset

3.2.2.3 Feature Distributions and Skewness

The feature distributions of the dataset revealed important characteristics of the data. Median Income (MedInc) and Average Rooms (AveRooms) showed right-skewed distributions, which suggested that most households had lower income levels and room counts, while fewer observations existed at the higher end of these features. House Age (HouseAge) displayed a relatively uniform spread with multiple peaks, possibly reflecting construction booms during certain time periods.

Average Bedrooms (AveBedrms) showed a strong peak near the lower values, indicating that most houses had a small number of bedrooms. In contrast, Population showed a long right tail, implying that some areas had larger populations. The Average Occupancy (AveOccup) feature was highly skewed, with most values concentrated at lower ranges but with a few extreme cases where the occupancy per household was abnormally high.

The Latitude and Longitude distributions highlighted distinct clusters, indicating geographical variations across different regions of California. These clusters likely represented urban, suburban, and rural areas, influencing housing prices. These features' observed skewness and spread emphasized the need for scaling and potential feature transformations in predictive modeling. Understanding these distributions was essential for handling outliers effectively.

Figure 3.2 illustrates the histograms of all key features, highlighting unique characteristics that may influence the modeling efforts.

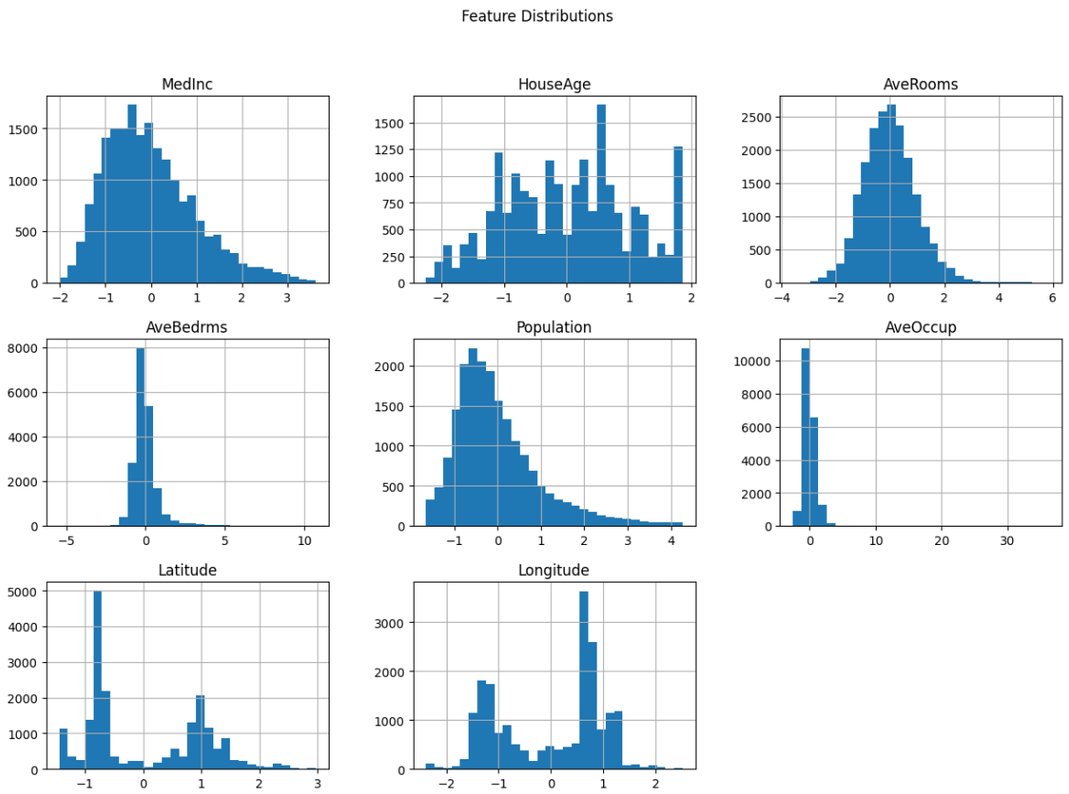


Figure 3.2: Feature Distributions for the California Housing Dataset

3.2.2.4 Correlation Matrix

Figure 3.3 provides insights into the linear relationships between each feature and the target variable. This analysis informs feature selection and potential transformations in the modeling process.

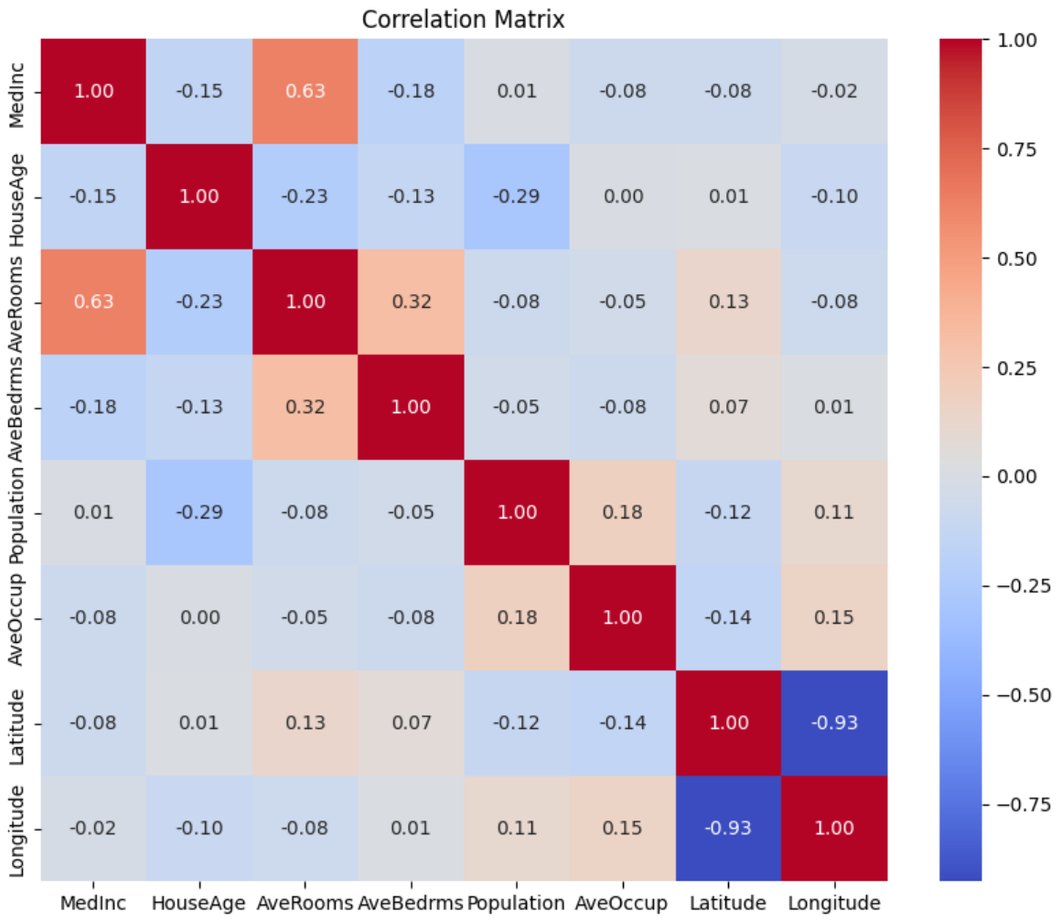


Figure 3.3: Correlation Map of Dataset Features

The correlation matrix provided insights into the relationships between different features in the dataset. Median Income (MedInc) showed a strong positive correlation (0.63) with Average Rooms (AveRooms), indicating that higher-income neighborhoods tended to have larger homes. Average Rooms (AveRooms) and Average Bedrooms (AveBedrms) (0.32) also had a moderate correlation, which suggested that homes with more rooms generally had more bedrooms, though the relationship was not perfectly linear.

House Age (HouseAge) showed a weak negative correlation (-0.29) with Population, implying that older housing areas tended to have smaller populations, possibly due to more established neighborhoods with limited expansion. Latitude and Longitude showed a strong negative correlation (-0.93), reflecting California's geographical orientation, where higher latitudes (northward) corresponded with decreasing longitude values. Other relationships, such as the weak correlation between Population and Median Income (0.01), suggested that population size alone did not strongly influence income levels, emphasizing that housing prices and economic conditions varied across different regions independently of population density.

3.3 Model Setup and Implementation

This section outlines the implementation process for developing BNNs using MCMC and VI. We first describe the data preparation and splitting strategy, ensuring that the dataset is structured for practical model training and evaluation. The architecture of both Bayesian models is then introduced, highlighting how MCMC estimates posterior distributions through sampling (Chib and Greenberg, 1995; Neal, 2012) while VI approximates them using optimization techniques (Blei et al., 2017). Finally, we detail the posterior density estimation process, demonstrating how each method captures uncertainty in predictions. This setup allows for a detailed comparison of the two approaches regarding accuracy, efficiency, and scalability.

3.3.1 Data Splitting

With the data cleaned and preprocessed, the final dataset comprised 19,794 observations and 8 features, with the target variable median house value (MedHouseVal) represented as a one-dimensional array of housing prices. The dataset was then split into training 80% and testing 20% sets, resulting in 15,835 training samples and 3,959 test samples. This ensured effective validation of unseen data to assess generalization and mitigate overfitting. The datasets were then converted into TensorFlow tensors, enabling the use of TensorFlow Probability (TFP) for efficient GPU computations and probabilistic modeling (Abadi et al., 2015).

3.3.2 Model Description

We implemented two BNN models using MCMC and VI to estimate posterior distributions and quantify uncertainty in predictions. The MCMC-based BNN inferred posterior distributions through Random Walk Metropolis (RWM) sampling (Metropolis et al., 1953; Neal, 2012), offering accurate but computationally expensive inference (Gelman et al., 2013). Conversely, the VI-based BNN used DenseFlipout layers to approximate the posterior via KL divergence minimization, providing a faster but less precise alternative (Blei et al., 2017).

The choice of RWM for MCMC sampling was driven by its conceptual simplicity and ease of application, making it a suitable baseline for comparison with VI (Neal, 2012). Although Hamiltonian Monte Carlo (HMC) is often more efficient in high-dimensional spaces due to its gradient-based sampling, it also introduces significant computational complexity and tuning requirements (Gelman et al., 2013). Given the scope of this study, RWM provided a straightforward and strong method for posterior sampling, allowing for a precise analysis of the computational comparison without the additional overhead associated with HMC.

For the VI implementation, DenseFlipout layers were selected due to their compatibility with TFP and efficient parameterization approach (Dillon et al., 2017; Wen et al., 2018). DenseFlipout uses a local reparameterization trick that reduces the variance of gradients during training, which is important for stable and efficient optimization (Wen et al., 2018). Compared to other parameter-efficient alternatives, DenseFlipout layers offer a balance between computational efficiency and uncertainty estimation, making them well-suited for BNNs (Wen et al., 2018). Their ease of integration into existing neural network architectures and the support provided by TF libraries further justified their selection (Dillon et al., 2017).

We chose hyperparameters through empirical tuning, ensuring our BNN trained with VI performed well without unnecessary computational overhead. The learning rate was manually adjusted to keep updates stable and effective throughout training. We set the number of epochs to 100, striking a balance between giving the model enough time to learn and avoiding excessive training. We fixed the batch size at 32, which is a practical choice that ensures a good comparison between model performance and efficiency.

3.3.3 Model Architecture

This section outlines the implementation of BNNs using MCMC sampling and VI. Each approach defines a different strategy for estimating the posterior distribution of network parameters, allowing us to capture model uncertainty and improve predictive reliability.

3.3.3.1 Bayesian Neural Network Implementation Using MCMC Sampling

The architecture of our BNN consists of an input layer, a hidden layer with five neurons, a ReLU activation function, and an output layer for regression. We treated these parameters as random variables with Gaussian priors, starting with a mean of zero and a standard deviation of one, compared to conventional neural networks that learn point values for weights and biases. As a result, we were able to account for model parameter uncertainty.

To approximate the posterior distribution of the weights and biases, we employed MCMC sampling, defining a joint log probability function to model the probabilistic relationships between the network parameters and outputs. We implemented RWM as our transition kernel (Metropolis et al., 1953; Neal, 2012), initializing the Markov chain with zero-valued weights and biases. To ensure convergence, we allowed the chain to burn in for 200 steps, after which we collected 500 posterior samples (Gelman et al., 2013).

Once the sampling process was complete, we computed the posterior mean of the sampled weights and biases, which provided a Bayesian point estimate of the parameters (Gelman et al., 2013). Using these inferred values, we made predictions on the test dataset. Our Bayesian approach included a distribution of predictions, allowing us to quantify the uncertainty associated with each estimate.

3.3.3.2 Bayesian Neural Network Implementation Using VI

We employed VI, which is an alternative to MCMC for posterior estimation. Instead of sampling, VI optimized an approximate posterior distribution by minimizing the Evidence Lower Bound (ELBO) (Blei et al., 2017; Kingma and Welling, 2013). Our network architecture consisted of four layers, each utilizing DenseFlipout layers from TFP, which introduced uncertainty-aware weight distributions (Dillon et al., 2017; Wen et al., 2018). The model included 128, 64, and 32 neurons in successive hidden layers, all with ReLU activations, followed by an output layer with a single neuron.

We compiled the model using the Adam optimizer with a reduced learning rate of 0.001, allowing for finer gradient updates and we used Mean Squared Error (MSE) as the loss function while tracking Mean Absolute Error (MAE) as a key performance metric. We

implemented an early stopping mechanism to prevent overfitting, monitor the validation loss, and restore the best-performing model weights after ten epochs of no improvement.

3.3.4 Estimation of the Posterior Density

This section describes how we applied the two techniques to estimate the posterior distribution of BNNs.

3.3.4.1 Posterior Estimation using MCMC

We defined a joint probability distribution over the network parameters by modeling weights and biases with Gaussian priors and used the RWM algorithm to sample from this posterior (Metropolis et al., 1953; Neal, 2012). We ran the Markov Chain for 500 iterations, discarding the first 200 samples as burn-in and collecting posterior samples for the weights and biases. We then estimated the posterior density by calculating the mean of these samples, providing a Bayesian point estimate of the parameters (Gelman et al., 2013).

3.3.4.2 Posterior Estimation using VI

We approximated the posterior density by using DenseFlipout layers, which introduced uncertainty in the network parameters (Wen et al., 2018). We optimized this variational approximation by minimizing the KL divergence between the variational distribution and the true posterior, effectively learning the posterior through optimization (Blei et al., 2017; Kingma and Welling, 2013). This allowed us to capture the posterior distribution indirectly by fitting a parametric distribution to the observed data.

3.4 Model Evaluation

We applied evaluation metrics to assess the predictive performance and uncertainty quantification of the MCMC-based BNN and the VI-based BNN. These metrics allowed us to quantify how well each model approximated actual housing values and provided a basis for comparing their effectiveness. We evaluated predictive accuracy using MSE as defined in (2.6.2), RMSE in (2.6.3), MAE in (2.6.4) and R-Squared (R^2) in (2.6.5). We calculated MSE and RMSE to measure the average squared error, with RMSE providing an interpretable scale. We used MAE to capture the average absolute differences between predictions and actual values, ensuring robustness against outliers. We also computed R^2 to determine how much variance in the target variable our models explained. Lower MSE, RMSE, and MAE values, along with higher R^2, indicated superior predictive performance.

We applied PICP as defined in (2.6.7) to the VI model to assess uncertainty quantification. Since the VI model produces predictive distributions, we calculated PICP to measure the proportion of actual values that fell within the model's 95% confidence intervals. We computed PICP using the standard deviation of the predictions and a z-score adjustment, ensuring a rigorous assessment of how well the model captured predictive uncertainty. Since the MCMC model did not provide direct interval estimates, we were unable to compute PICP for it.

The evaluation results were compiled into a comparison table to provide insights into the strengths and comparison between both Bayesian models. This analysis allowed us to assess the effectiveness of both methods in capturing predictive uncertainty and overall model performance.

3.5 Comparative Analysis

We systematically compared the efficiency and behavior of the VI and MCMC approaches and conducted a detailed analysis focusing on computational time, memory usage, convergence behavior, and trace diagnostics.

3.5.1 Scalability

To assess the scalability of the VI-based BNN and the MCMC-based BNN, we conducted experiments using 50%, 75% and 100% of the original dataset. This allowed us to systematically evaluate the impact of increasing data volume on computational time, memory usage and convergence behavior.

3.5.1.1 Computational Time

We measured the training duration for the VI model and the sampling time for the MCMC model across all dataset subsets to assess the impact of dataset size on scalability. The execution time for each method was recorded using the time module, providing a quantitative comparison of how computational cost scaled with data size. This analysis highlighted the efficiency of VI's gradient-based optimization compared to the iterative sampling nature of MCMC, particularly in terms of handling larger datasets.

3.5.1.2 Memory Usage

We monitored peak memory consumption throughout the training and sampling processes to analyze the resource demands of each approach as dataset size increased. Using the memory_usage() function, we recorded the memory footprint of both models at different scales, capturing how resource requirements evolved with increasing data complexity. This comparison helped determine whether one method demonstrated superior scalability in terms of memory efficiency, particularly when handling high-dimensional data.

3.5.1.3 Convergence Behavior

We analyzed the convergence behavior of both models by monitoring multiple indicators to ensure the reliability of our parameter estimates.

3.5.1.3.1 Training and validation loss curves

For the VI model, using history.history['loss'] and history.history['val_loss'] to visualize how the loss function evolved over epochs. A smoothly decreasing loss curve with minimal validation fluctuations indicated effective learning and generalization, while large gaps between training and validation loss suggested potential overfitting or poor posterior approximation.

3.5.1.3.2 Trace plots

For key parameters in the MCMC model, using tfp.mcmc.sample_chain() to assess chain mixing and parameter stability over iterations. We evaluated whether the Markov chains properly explored the posterior distribution, whether parameter samples oscillated around a stable mean, and whether different chains explored similar regions of the parameter space. Poor chain mixing or drift signaled the need for longer burn-in periods, better initializations, or improved proposal distributions. These assessments helped ensure that both models achieved stable and reliable convergence.

3.5.1.4 Scalability for Large Models

We evaluated how each method adapted to increasing dataset size and neural network complexity by comparing their efficiency in processing larger datasets. Our analysis included measuring the increase in training time and sampling time as the dataset size grew and analyzing the effect of dataset size on memory consumption. We then assessed the stability of convergence behavior across different data scales to determine the robustness of each method under varying computational demands.

3.5.2 Computational Efficiency

We measured the computational efficiency through execution time, memory usage, and convergence rate. All experiments were conducted on a Lenovo IdeaPad Slim 3 15IAH8 laptop, equipped with a 12th Gen Intel Core i5-12450H processor (2.00 GHz) and 8GB of RAM (7.70 GB usable). The system runs a 64-bit operating system on an x64-based processor. No Graphics Processing Unit (GPU) acceleration was used, ensuring a fair comparison of Central Processing Unit (CPU)-based performance between MCMC and VI methods.

3.5.2.1 Computational Time Analysis

We measured the total computational time for both methods to compare the efficiency of VI and MCMC. Specifically, we recorded the training time for the VI-based BNN and the sampling time for the MCMC-based BNN using the time module. The execution time was logged at the start and end of each process to assess the computational demands of each method.

3.5.2.1.1 VI Training Time

The time taken to optimize the variational posterior was recorded by measuring the duration of the training phase. The computational time for VI was calculated as:

$$\text{Time}_{\text{VI}} = \text{End Time}_{\text{VI}} - \text{Start Time}_{\text{VI}} \tag{3.2}$$

Since VI relies on gradient-based optimization, this measurement reflects the efficiency of parameter updates and convergence speed within the training framework.

3.5.2.1.2 MCMC Sampling Time

The time required to generate posterior samples using the RWM algorithm was recorded. The computational time for MCMC sampling was determined as:

$$\text{Time}_{\text{MCMC}} = \text{End Time}_{\text{MCMC}} - \text{Start Time}_{\text{MCMC}} \tag{3.3}$$

This measurement captured the total time spent in Markov Chain transitions, burn-in steps, and posterior sampling iterations.

3.5.2.2 Convergence Behavior for VI

To analyze the convergence of the VI-based BNN, we plotted the training and validation loss curves over epochs. This visualization helped us determine whether the variational optimization was stable and whether overfitting occurred. The loss function trends provided insight into the quality of the variational approximation.

3.5.2.3 Convergence Behavior for MCMC

To assess the convergence of the MCMC-based BNN, we generated trace plots for selected network parameters. Trace plots provide a visual representation of the sampled values over iterations, allowing us to diagnose the behavior of the Markov Chain. Specifically, we plotted the sampled values for key parameters, including the weights in the first hidden layer (θ_{w_1}), the bias in the first hidden layer (θ_{b_1}), the weights in the output layer (θ_{w_2}) and the bias in the output layer (θ_{b_2}). These plots illustrated how parameter values evolved over iterations, helping to identify potential issues such as poor mixing, convergence problems, or high autocorrelation within the chain.

3.5.2.4 Memory Usage Analysis

We compared the memory consumption of the MCMC-based BNN and the VI-based BNN by tracking peak memory usage during model training and sampling.

3.5.2.4.1 VI Model Training

We recorded the peak memory consumption while optimizing the variational posterior to evaluate the memory footprint during training. This was achieved using the memory_usage() function, which continuously tracked memory utilization at 0.1-second intervals. By monitoring memory allocation in real-time, we assessed VI's efficiency in terms of resource usage and computational demands.

3.5.2.4.2 MCMC Sampling

For the MCMC model, we measured memory usage throughout the posterior sampling process using the RWM algorithm. We tracked the memory usage during the execution of tfp.mcmc.sample_chain(), enabling a detailed analysis of the computational overhead introduced by MCMC sampling. This monitoring helped quantify the resource-intensive nature of MCMC compared to VI, particularly in terms of memory allocation and processing requirements.

3.5.3 Practical Implementation and Ease of Use

We compared the MCMC-based BNN and VI-based BNN in terms of implementation complexity, library support, and ease of use.

3.5.3.1 Code Complexity

The VI model used the DenseFlipout layers, offering a high-level and user-friendly abstraction for constructing BNNs. This built-in functionality simplified the implementation process, reducing the need for extensive manual coding and making it more accessible for users familiar with standard deep learning frameworks. The MCMC model required a more detailed setup, requiring the explicit definition of prior distributions and likelihood functions using JointDistributionCoroutineAutoBatched. This additional complexity increased the effort required for implementation, as it involved careful formulation of the probabilistic model and proper handling of dependencies between variables.

3.5.3.2 Library Support

We used TFP for the implementation of both VI and MCMC. VI benefited significantly from the built-in support provided by the tfp.layers.DenseFlipout module, which simplifies the process of defining Bayesian layers within a neural network. This built-in functionality allowed for automatic uncertainty estimation with minimal manual configuration, making it highly convenient for large-scale applications. MCMC required a more hands-on approach, relying on tfp.mcmc.sample_chain() to perform posterior sampling. This method required careful tuning of hyperparameters such as step size, number of burn-in steps, and transition kernel selection to ensure efficient sampling and convergence.

3.5.3.3 Training and Tuning

The VI model was trained using gradient-based optimization, specifically employing the Adam optimizer. This made the training process more straightforward, as it closely followed standard deep learning optimization techniques. VI required fewer manual adjustments, allowing for more efficient training with minimal intervention. The MCMC model demanded additional hyperparameter tuning to achieve optimal performance. Key parameters, including the number of burn-in steps, step sizes, and transition kernel configurations, needed careful selection to ensure convergence and efficient sampling. This additional complexity made the optimization process more computationally demanding and labor-intensive compared to VI. Moreover, verifying the correctness of the MCMC model required analyzing convergence diagnostics, further increasing the effort needed for fine-tuning.

3.5.3.4 User-Friendliness

VI approach closely resembled standard deep learning pipelines, allowing for seamless integration into existing machine learning workflows. Since it leverages TensorFlow Probability's built-in layers, it requires minimal modifications to standard neural network architectures, making it more accessible for practitioners familiar with deep learning frameworks. The MCMC approach required a deeper understanding of BI concepts and additional expertise in probabilistic modeling. Unlike VI, which provides direct uncertainty estimation,

MCMC requires careful convergence diagnostics to ensure reliable posterior sampling. This involved analyzing trace plots.

3.5.3.5 Interpretability and Debugging

The MCMC model provided direct access into posterior distributions through trace plots, allowing for a clear and explicit representation of uncertainty. This made it possible to analyze the full distribution of model parameters rather than relying on point estimates. However, diagnosing issues such as poor chain mixing was more complex, requiring in-depth analysis of trace behavior and convergence diagnostics. The VI model provided a more easy-to-understand debugging process, as loss curves served as a direct indicator of convergence.

3.6 Declaration of AI Usage

In this study, AI tools were used to enhance efficiency and accuracy in specific aspects of the study. To support the literature review, AI-assisted tools were utilized to identify additional studies. During the methodology and experiment design phase, AI contributed to model tuning and adjustments to the experimental framework. For data analysis, AI-supported coding and statistical methods were employed to interpret results effectively. Additionally, AI-assisted debugging and optimization were applied to Python scripts used in Bayesian inference models to enhance efficiency and accuracy. In the writing and editing process, AI tools were used to improve grammar, structure sections, and enhance overall readability.

All AI-generated contributions were thoroughly reviewed, modified, and validated to maintain academic integrity and accuracy. For full transparency, a signed AI declaration form will be submitted alongside this work.

Chapter 4

Results and Discussion

This chapter presents a comparative analysis of MCMC and VI across key evaluation metrics. The results include posterior estimation accuracy, computational efficiency, scalability, and uncertainty quantification, providing insights into the comparison between these approaches.

Section 4.1 provides a detailed summary of the experimental results for BNNs trained with VI and MCMC across different dataset sizes 50%, 75%, and 100%. It covers the data preparation and outlier handling, the BNN implementations with VI and MCMC, the three experiments comparing their predictive accuracy (MSE, RMSE, MAE), training time, and memory usage, their convergence behaviour for both methods, and the statistical analysis of performance differences. Section 4.2 interprets the experimental findings, comparing the strengths and weaknesses of VI and MCMC. The discussion focuses on model convergence, generalization ability, and predictive performance across different dataset sizes. The section also investigates the experimental findings of this study within the existing literature on BNNs and inference methodologies. Section 4.3 outlines how the study advances understanding of the trade-offs between inference accuracy and computational efficiency, offering valuable insights for real-world applications and future methodological improvements, including the potential for hybrid inference techniques. Section 4.4 provides a critical reflection on the study's constraints, particularly those related to computational costs, dataset generalizability, and evaluation scope, while also suggesting directions for future research to address these challenges and build on the study's findings.

4.1 Results

This section presents the study's key findings based on the experiments conducted to compare MCMC and VI methods for posterior density estimation in BNNs.

4.1.1 Summary of Findings

This study compared MCMC and VI methods for posterior density estimation in BNNs using the California Housing Prices dataset. The dataset initially contained 20,640 samples, which were reduced to 19,794 after outlier removal. For the first experiment using 100% of the dataset, the training set included 15,835 samples, while the test set had 3,959 samples. In the second experiment with 75% of the dataset, the sample size was reduced from 15,480 to 14,835 after removing outliers, resulting in 11,868 training samples and 2,967 test samples.

The third experiment used 50% of the dataset, starting with 10,320 samples and reducing to 9,895 after outlier removal, with 7,916 samples allocated for training and 1,979 for testing.

4.1.1.1 Bayesian Neural Network Using VI

This study investigated the performance of a BNN on three datasets of varying sizes, focusing on its ability to estimate housing prices while addressing outliers and optimizing learning rates. The experiments followed a structured approach, using DenseFlipout layers to capture model uncertainty and ensure strong posterior estimation.

In Experiment 1, the model achieved the lowest final validation MAE of 1.53 after 100 epochs, indicating stable convergence and accurate predictions. Experiment 2, resulted in a slightly higher MAE of 1.93, with greater variance in predictions and some signs of overfitting in later epochs. Experiment 3, showed the highest MAE of 3.02, reflecting greater uncertainty and less stable convergence. The findings highlighted that larger datasets contributed to better stability and accuracy for both MCMC and VI methods, while smaller datasets increased uncertainty and reduced generalization.

Overall, Experiment 1 outperformed the other two, confirming that larger datasets contributed to greater stability and accuracy in BNNs. The performance decline in Experiments 2 and 3 highlighted the role of dataset size in reducing uncertainty and improving generalization.

4.1.1.2 Bayesian Neural Network Using MCMC

This study explored the application of MCMC sampling in a BNN for housing price prediction. The experiments were conducted on datasets of different sizes, with model parameters estimated using probabilistic inference. Each experiment followed the same procedure, which involved removing outliers from the dataset, splitting the data into training and test sets, initializing MCMC sampling with a burn-in phase, and generating predictions.

For all experiments, MCMC sampling was performed using 200 burn-in steps and 500 sampling iterations. The model's chain state was initialized with zero values for weights and biases. After the sampling process, the mean weights and biases were calculated. The first layer contained weight parameters of shape (8, 5) and a bias term of shape (5,). The second layer contained weights of shape (5, 1) and a bias of shape (1,). Predictions were then generated on the test data.

In the first experiment, the predictions generated on the test set had a final shape of (3959,). The second experiment produced predictions with a final shape of (2967,). Lastly, the third experiment resulted in predictions with a final shape of (1979,). The results showed that the MCMC sampling process successfully estimated posterior distributions for the neural network parameters across different dataset sizes. The mean weights and biases remained consistent across experiments, indicating stable sampling results.

4.1.2 Evaluation of MCMC and VI Models

This section presents the evaluation results of the MCMC and VI models based on multiple performance metrics. The models were assessed using MSE, RMSE, MAE, R^2, and the PICP.

4.1.2.1 Experiment 1

Table 4.1 shows that the MCMC model outperformed the VI model in most error-based metrics.

Table 4.1: Performance Metrics for MCMC and VI Models Using Full Dataset

Model	**MSE**	**RMSE**	**MAE**	R^2 **Score**	**PICP**
MCMC	1.2875	1.1347	0.8598	-0.0668	N/A
VI	3.9636	1.9909	1.0173	-2.2843	0.4706

4.1.2.2 Experiment 2

Table 4.2 shows that both models performed relatively similarly, but the MCMC model maintained a slight advantage.

Table 4.2: Performance Metrics for MCMC and VI Models Using 75% of the Dataset

Model	**MSE**	**RMSE**	**MAE**	R^2 **Score**	**PICP**
MCMC	1.2492	1.1177	0.8267	-0.0672	N/A
VI	1.3530	1.1632	0.8814	-0.1559	0.7300

4.1.2.3 Experiment 3

Table 4.3 shows that the MCMC model again showed slightly better performance than the VI model.

Table 4.3: Performance Metrics for MCMC and VI Models Using 50% of the Dataset

Model	**MSE**	**RMSE**	**MAE**	R^2 **Score**	**PICP**
MCMC	1.4155	1.1898	0.8880	-0.1520	N/A
VI	1.6412	1.2811	0.9430	-0.3357	0.9752

The MCMC model consistently achieved lower error values compared to the VI model across all experiments. The differences in MSE, RMSE, and MAE indicate that MCMC provided better predictive accuracy and stability. The R^2 values were negative in all cases, suggesting that neither model fully captured the variance in the dataset. However, the VI model showed better PICP, particularly in Experiment 3, where it reached 0.9752.

4.1.3 Computational Time

This section provides a summary and interpretation of the experimental results comparing VI and MCMC in BNN training for housing price prediction. The experiments were conducted

on datasets of varying sizes to assess the performance differences in terms of training time and predictive accuracy.

4.1.3.1 Training Time Comparison

One of the most notable differences observed between the two methods was training time. Across all experiments, VI required significantly less time to train the BNN compared to MCMC. The training time results are summarized as follows:

Table 4.4: Training Time Required for VI and MCMC Across Experiments

Experiment	**VI Training Time (s)**	**MCMC Sampling Time (s)**
Experiment 1	322.13	689.43
Experiment 2	169.39	419.80
Experiment 3	142.19	232.29

The uneven training time can be attributed to MCMC's iterative nature, which involves generating many samples to approximate the posterior distribution. In contrast, VI approximates the posterior more efficiently using optimization techniques.

4.1.4 Convergence Behavior

This section analyzes the convergence behavior of VI and MCMC methods across different dataset sizes. For VI, training and validation loss trends over 100 epochs were examined. For MCMC, trace plots of BNN parameters were evaluated to assess chain mixing and convergence.

4.1.4.1 Convergence Behavior for VI

The experiments were conducted using 100%, 75%, and 50% of the dataset, respectively. Each plot shows the evolution of training loss and validation loss over 100 epochs.

In the first experiment, as shown in Figure 4.1, the training and validation loss shows significant fluctuations but generally stabilizes as the epochs progress. The decreasing trend suggests that the model is learning effectively, although the variability in loss values indicates sensitivity to the dataset's complexity.

In the second experiment, as shown in Figure 4.2, the loss curves appear smoother than the full dataset experiment. This suggests that reducing the dataset size helps in stabilizing training, possibly by reducing overfitting. However, the final loss values remain comparable to those in the first experiment, indicating that the model still captures essential patterns from the data.

In the third experiment, as shown in Figure 4.3, the loss curves demonstrate more instability, with more significant spikes in validation loss. This behavior suggests that reducing the dataset size further leads to higher variance in training, making the model more susceptible to fluctuations in the data. The increased oscillations indicate that the model struggles with generalization, potentially due to insufficient data for intense learning.

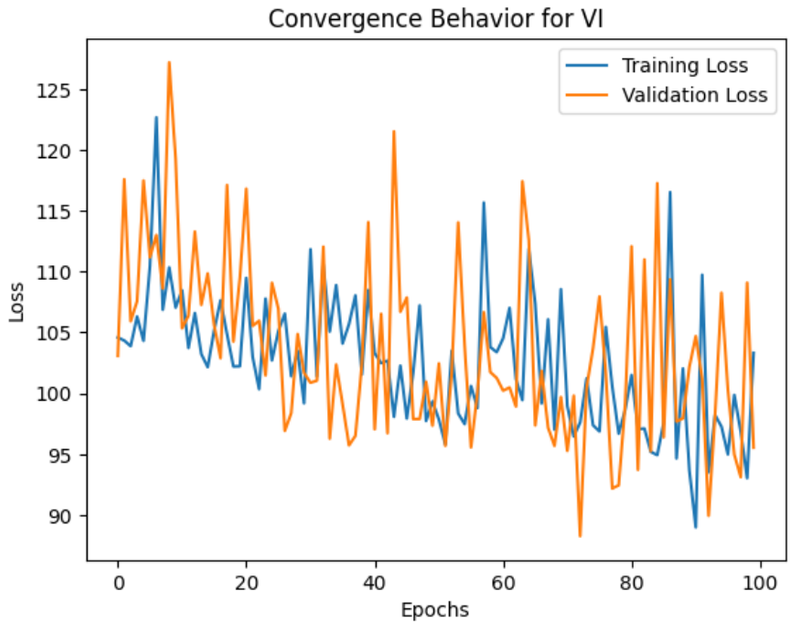


Figure 4.1: Model Training and Validation Loss Using Full Dataset

4.1.4.2 Convergence Behavior for MCMC

The trace plots presented the evolution of BNN parameters across different dataset sizes using MCMC sampling. Each plot represents the variation of parameter values over the course of sampling iterations, offering insights into the stability and convergence of the Markov chains.

The trace plots show relatively smooth transitions and stable fluctuations in Figure 4.4, where the entire dataset 100% was used. The parameter values exhibit a well-mixed Markov chain, suggesting effective convergence. The chain explores the parameter space efficiently without getting stuck in a particular region, indicating reliable parameter estimation.

In Figure 4.5, where only 75% of the dataset was used, the trace plots maintain a similar pattern but with slightly increased fluctuations. Some parameters show more noticeable drift over iterations, which may be attributed to the reduced number of data points. This increased variance suggests that the model is more sensitive to more minor variations in the data, although the chains still appear to be mixing sufficiently.

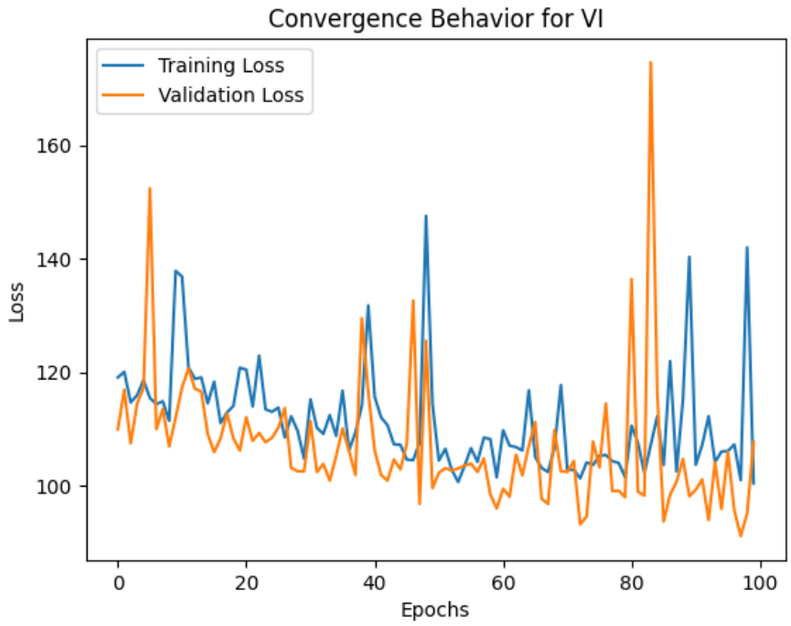


Figure 4.2: Model Training and Validation Loss Using 75% of the Dataset

In Figure 4.6, with only 50% of the dataset used, the trace plots reveal greater volatility in parameter estimates. The fluctuations are more pronounced, indicating increased uncertainty in parameter estimation. Some traces show slower mixing behavior, suggesting that the model struggles to converge as effectively as in the larger dataset scenarios. This behavior is expected, as smaller datasets provide less information for Bayesian updating, leading to wider posterior distributions and less precise estimates.

4.1.5 Memory Usage Comparison: VI vs. MCMC

The memory consumption of VI model training and MCMC sampling was analyzed across three dataset sizes: 100%, 75%, and 50% of the original dataset. The results indicate that MCMC consistently requires more memory than VI training, making it a less efficient choice regarding computational resources. As the dataset size decreases, the memory requirements for both methods also decrease, confirming that memory consumption scales directly proportionally with dataset size, as shown in Table 4.5. The trend suggests that MCMC's high memory consumption can be a limiting factor, especially when dealing with large-scale datasets where computational efficiency is essential.

Table 4.5: Comparison of Peak Memory Usage for VI Training and MCMC Sampling

Dataset Size	Peak Memory Usage (VI) [MB]	Peak Memory Usage (MCMC) [MB]
100%	904.71	2841.76
75%	890.85	1992.49
50%	848.96	1376.95

Given these findings, VI appears to be a more memory-efficient approach, making it more suitable for large datasets and real-world applications where computational resources are

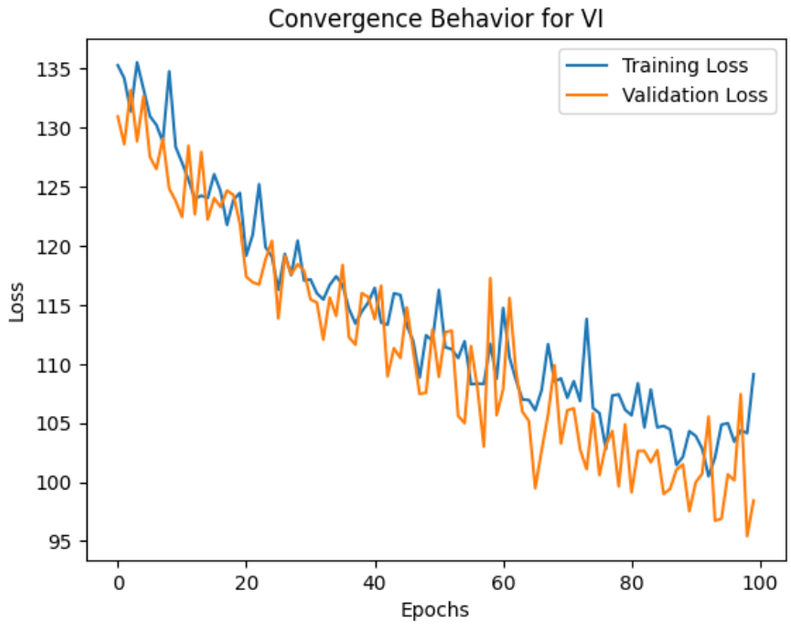


Figure 4.3: Model Training and Validation Loss Using 50% of the Dataset

constrained. However, this efficiency trade-off must be considered alongside inference accuracy, as MCMC provides a more exact posterior estimation at the cost of computational complexity.

4.1.6 Statistical Analysis

We compared MCMC and VI in BNNs to assess differences in predictive accuracy and uncertainty estimation. We used the paired T-test and the one-sample T-test to determine whether the performance variations between the two methods were statistically significant.

Metric	**T-statistic**	***p*-value**
MSE	-1.195750	0.354338
RMSE	-1.258878	0.335106
MAE	-2.603012	0.121309
R2	1.195386	0.354453
PICP (VI Only)	-1.542601	0.262885

Table 4.6: Statistical Significance of Model Differences Based on T-Test Results

The results of the paired T-Tests (2.7.3) show no statistically significant difference between the performance of MCMC and VI across the evaluated metrics. The p-values for MSE (0.3543), RMSE (0.3351), and MAE (0.1213) are all greater than the common significance threshold of 0.05, suggesting that we fail to reject the null hypothesis. Meaning that the differences observed in these error metrics between the two methods could have occurred by chance rather than indicating a true difference in model performance. The R^2 scores for MCMC and VI also show no significant difference, with a T-statistic of 1.1954 and a p-value of 0.3545. Both methods explain variance in the data at comparable levels.

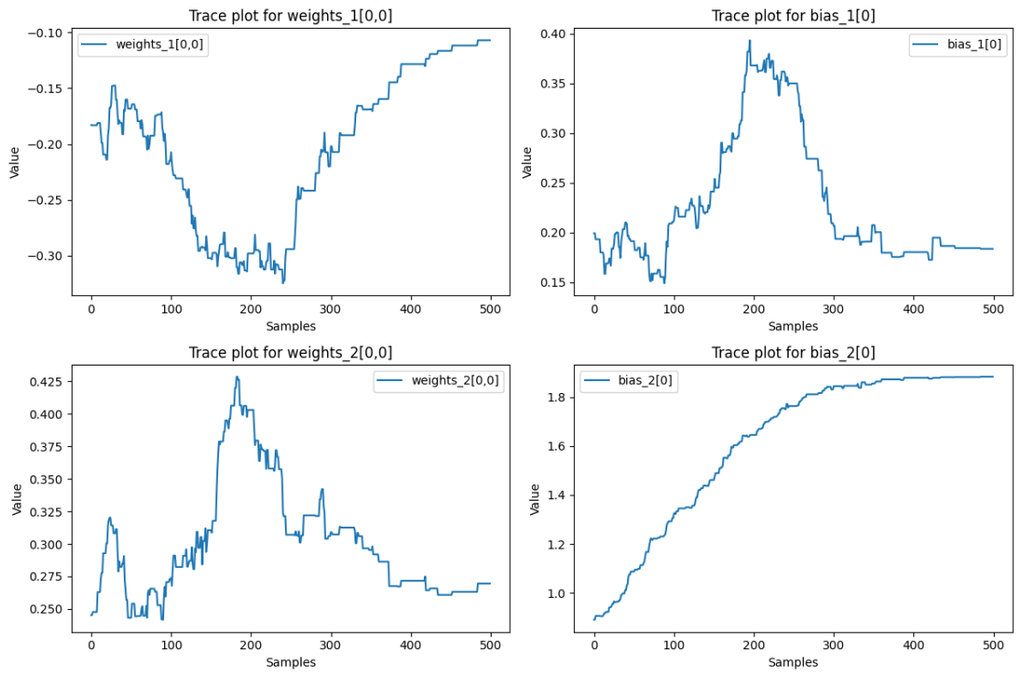


Figure 4.4: Parameter Trace Plots from MCMC Sampling using Full Dataset

Since PICP is only for VI, we conducted the one-sample T-Test (2.7.1) and compared it to an expected theoretical value of 0.95, which resulted in a p-value of 0.2629. The PICP values for VI do not significantly deviate from the expected coverage level. The results indicate that MCMC and VI perform similarly across all tested metrics, with no statistically significant advantage observed for either approach.

4.2 Discussion of Results

This section offers a detailed interpretation of the results, their implications, and limitations, highlighting their significance.

4.2.1 Interpretation of Results

The comparative analysis of BNNs using VI and MCMC revealed unique performance trends across different dataset sizes. The VI-based model generally showed efficient training, achieving lower computational costs and reduced memory usage while maintaining competitive accuracy. However, its predictive performance declined as the dataset size decreased, with increased uncertainty in parameter estimation, as reflected in the widening prediction intervals and the increased MAE.

Conversely, MCMC provided more stable and reliable posterior estimates, particularly in the 100% dataset experiment, where its Markov chains showed smooth convergence and better mixing. MCMC had significantly higher computational demands, requiring nearly three times the memory and extended training time than VI. The evaluation metrics indicated that MCMC consistently outperformed VI in error-based measures, but VI showed higher PICP, especially when trained on reduced dataset sizes.

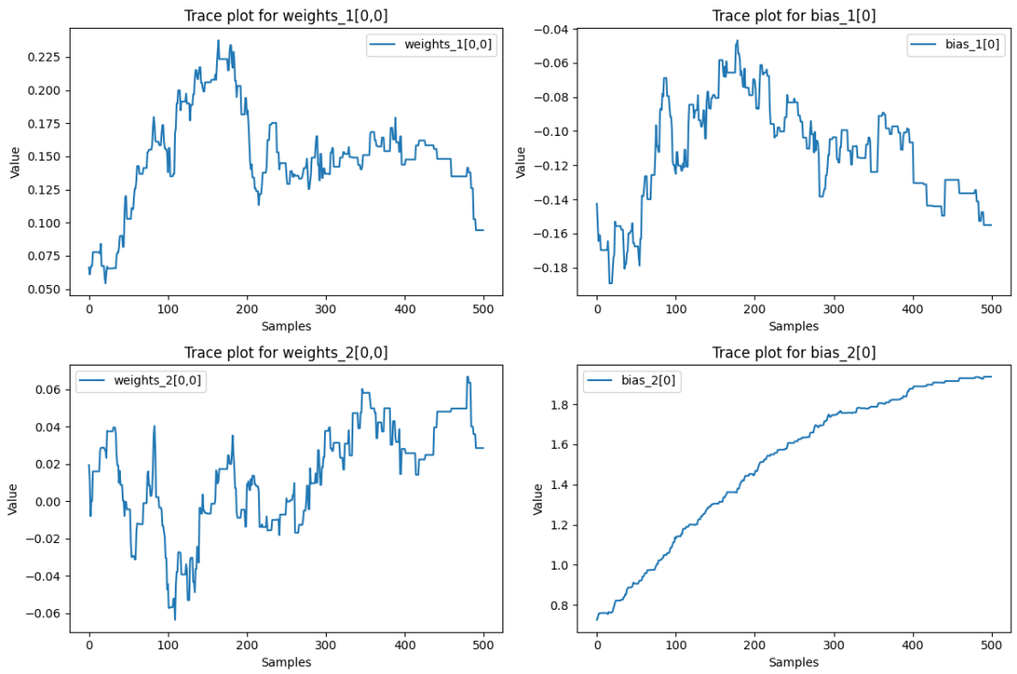


Figure 4.5: Parameter Trace Plots from MCMC Sampling using 75% of the Dataset

This suggests that VI's approximations become less reliable as the dataset decreases, whereas MCMC, though computationally expensive, maintains strong inference capabilities. The convergence behavior further underscored these findings, as MCMC's trace plots showed stable parameter exploration with increasing dataset size. At the same time, VI's loss curves suggested a lack of stability in smaller datasets. Ultimately, the results highlight the comparison between computational efficiency and model accuracy. VI is advantageous for large-scale applications where speed and memory constraints are essential, whereas MCMC is preferable for cases that demand precise posterior estimations despite its demanding nature.

4.2.2 Discussion of Findings in Relation to Literature

This section examines the experimental findings of this study within the existing literature on BNNs and inference methodologies.

The results presented in this study demonstrate that MCMC methods, despite their computational expense, offer highly accurate posterior estimates in high-dimensional Bayesian models. This aligns with (Gilks, 2005), who highlighted MCMC's effectiveness in complex model statistics, and (Neal, 2012), who emphasized its robustness in high-dimensional parameter spaces. The accuracy seen in this study echoes (Fan and Sisson, 2010), who underscored MCMC's suitability for models with intractable likelihoods. The computational burden, including extensive memory usage and longer sampling times, is consistent with challenges noted by (Nummenmaa et al., 2007).

Conversely, VI showcased higher computational efficiency and scalability, particularly when applied to large datasets, a finding supported by (Kingma and Welling, 2014). VI's integration with TFP streamlined its implementation, as also discussed by (Mukherjee and Sen, 2021), who highlighted VI's strength in high-dimensional regression models. This study

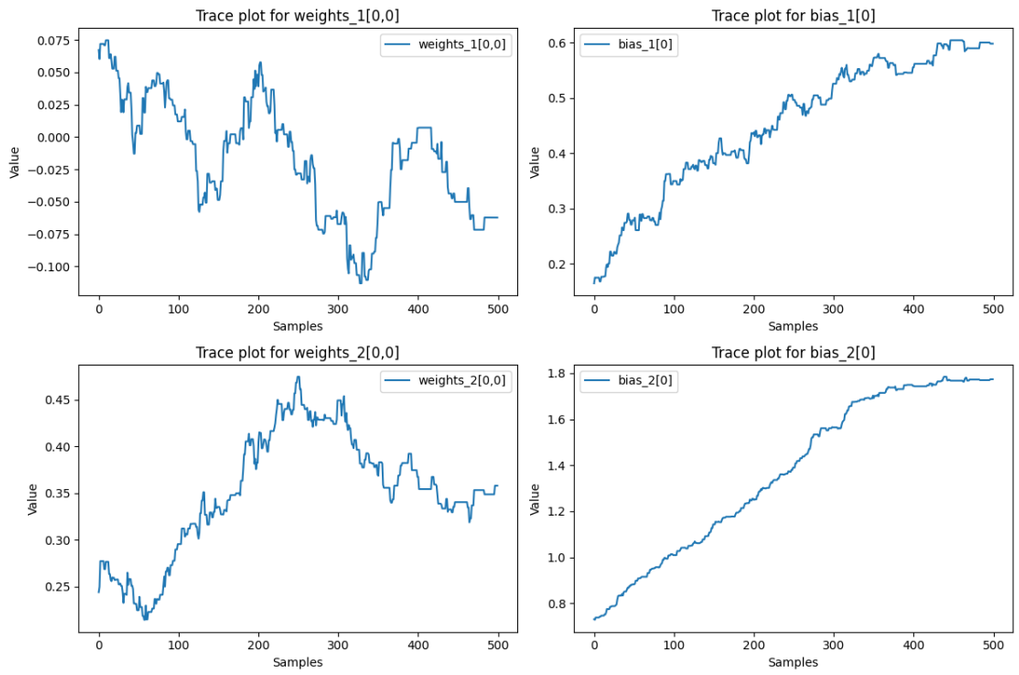


Figure 4.6: Parameter Trace Plots from MCMC Sampling using 50% of the Dataset

found that VI's approximation bias, especially with smaller datasets, often led to unacceptable posterior estimates, a limitation mirrored in prior work by (Opper, 2019) and (Nummenmaa et al., 2007).

The comparative analysis in this study aligns with (Emerson and Copestake, 2017), who showed VI's rapid convergence in text-based inference but highlighted its sensitivity to variational family choice. Similarly, our results showed that while VI achieved faster convergence with lower memory usage, it struggled to maintain accuracy across varying dataset sizes, particularly clear in our 50% dataset experiment, where the final MAE remained high at 3.02. MCMC's high-tuning requirements, including burn-in steps and trace plot analysis, are consistent with challenges highlighted by (Chib and Greenberg, 1995).

Our study's trace plots confirmed the robustness of MCMC's posterior estimates for larger datasets, a finding also noted by (Müller, 2004) in financial econometrics applications. The dataset-specific nature of both methods seen in this study, using the California Housing Prices dataset, is consistent with the literature's caution against over-generalizing findings from specific datasets, as emphasized by (Ghalebikesabi et al., 2022). Future studies could explore hybrid approaches, as suggested by (Salimans et al., 2015), integrating MCMC's accuracy with VI's efficiency.

While many of our findings align with previous research, some are new contributions or challenges to existing assumptions. One of the biggest differences we observed was in the computational performance of MCMC and VI. As expected, MCMC is significantly more computationally expensive than VI, but what's less commonly discussed is how this plays out in real-world BNN applications. Our results show that VI consistently trains faster and uses less memory, contradicting some claims that MCMC can be optimized to be more efficient in large-scale models.

Another surprising difference was in convergence behavior. While it's widely accepted that MCMC provides the most accurate posterior estimates, our findings highlight a significant limitation: the slow convergence for large datasets. VI converges much faster but introduces approximation errors, which become more noticeable as the dataset size decreases. Most studies don't explore how dataset size affects these trade-offs in practice, making this an important insight. Uncertainty quantification is another area where our results offer a different perspective. MCMC is generally considered superior, but when working with smaller datasets, the gap between MCMC and VI is much narrower than expected. The computational cost of MCMC may not always be justified by its advantages.

We provide a more practical, application-driven comparison by evaluating these methods using RMSE, MAE, and training time. Many existing studies focus on theoretical performance, but our findings highlight real-world usability, which will help researchers and practitioners make more informed choices based on dataset size and computational constraints. This study strengthens the comparison highlighted in the existing literature: MCMC provides precision but at a computational cost, while VI offers speed and scalability at the risk of approximation bias. The results contribute to the ongoing conversations by emphasizing the practical considerations of selecting inference methods for BNNs, especially in ML applications involving large datasets and real-time processing.

The research objectives outlined in this study focused on comparing MCMC and VI in terms of posterior estimation accuracy, computational efficiency, scalability, uncertainty quantification, and ease of implementation. The experimental results provide clear insights into these aspects: MSE, RMSE, and MAE were computed for both methods, allowing a direct comparison of their predictive performance. The results showed that MCMC generally provided lower error values, aligning to assess accuracy differences. Training time measurements showed that VI is faster than MCMC, emphasizing the study's objective of evaluating computational cost. The high sampling overhead in MCMC confirmed its inefficiency for large-scale applications.

By conducting experiments with 50%, 75%, and 100% of the dataset, the study assessed how well each method adapts to increasing data sizes. The findings indicate that VI scales more efficiently than MCMC, which suffers from memory and time constraints as dataset size grows. The study computed the PICP for the VI model, while trace plots for MCMC provided insights into uncertainty representation. This comparison supports analyzing the quality of uncertainty estimates in both methods. The evaluation of training complexity, hyperparameter tuning, and debugging difficulty provided insights into the usability of each technique. VI emerged as more user-friendly due to its compatibility with standard deep-learning pipelines.

4.2.3 Implications

The findings of this study offer important implications for the application of BNNs and the selection of appropriate inference methodologies in practice. By examining the trade-offs between MCMC and VI, this research provides valuable guidance for practitioners and researchers seeking to implement uncertainty-aware deep learning models.

Firstly, the superior posterior accuracy achieved by MCMC reaffirms its value in scenarios where model fidelity and uncertainty estimation are paramount. This includes scientific modeling, medical diagnostics, and safety-critical systems, where even minor deviations in posterior estimation could lead to significant consequences. However, the computational burden and prolonged convergence times associated with MCMC make deploying in real-time or large-scale environments impractical. These limitations underscore the need for careful consideration of computational resources and inference latency when opting for MCMC-based BNNs.

On the other hand, VI's efficiency and scalability make it an appealing choice for modern ML applications that involve large datasets, high-dimensional models, or time-sensitive predictions. Its ability to integrate easily into deep learning pipelines—particularly within frameworks such as TFP enhances its appeal for industry applications, including finance, recommender systems, and autonomous technologies. However, the inherent approximation bias in VI highlights the importance of choosing an expressive variational family and investing in techniques that mitigate underfitting in posterior distributions.

From an implementation standpoint, MCMC requires a more diligent setup, including burn-in phase tuning, convergence diagnostics, and sampling strategy refinement. While these requirements enhance the reliability of results, they also demand a higher level of statistical and computational expertise. In contrast, VI offers a more user-friendly implementation, often requiring fewer iterations and less manual intervention. However, it may benefit from future advancements in adaptive variational inference techniques to improve posterior accuracy.

The practical implications of this study emphasize that MCMC remains a powerful tool for applications where accuracy in posterior estimation is non-negotiable and that VI is better suited for large-scale, resource-constrained, or time-critical environments where approximate inference is acceptable, given its operational advantages. These insights contribute to the broader discourse on Bayesian deep learning, supporting more informed decision-making when selecting inference methods based on application-specific constraints. The study thus equips practitioners with a deeper understanding of when and how to apply each inference technique in real-world scenarios.

4.3 Contributions of the Study

This study significantly contributes to BNNs by conducting a comparative analysis of MCMC and VI for posterior density estimation. It empirically evaluates their accuracy, computational efficiency, scalability, and uncertainty quantification performance. The results demonstrate that while MCMC produces more precise posterior estimates, it is computationally expensive and slow, making it less practical for large-scale applications. In contrast, VI offers greater efficiency and scalability but introduces approximation bias, which may affect posterior estimation accuracy, particularly in complex models and smaller datasets. This nuanced evaluation provides essential insights for researchers and practitioners choosing inference techniques based on computational constraints and data size considerations.

Beyond theoretical comparisons, the study examines the practical implications of MCMC and VI in real-world ML applications. It highlights the computational trade-offs between these two methods, showing that MCMC requires extensive memory and tuning while VI integrates more seamlessly into deep learning frameworks. Using the California Housing Prices dataset, the study also reveals that dataset size influences inference performance, with VI excelling in large-scale scenarios and MCMC maintaining stable uncertainty estimates in smaller datasets. These findings are particularly relevant for domains requiring real-time decision-making, such as financial risk modeling, medical diagnostics, and autonomous systems.

Additionally, this research paves the way for hybrid inference techniques that leverage the strengths of both MCMC and VI. The findings suggest that initializing VI with MCMC-generated samples could enhance posterior accuracy while preserving computational efficiency. This opens new research directions, encouraging the development of adaptive inference strategies that balance the precision of MCMC with the speed and scalability of VI. Ultimately, this study provides practical guidelines for selecting Bayesian inference methods, making a valuable contribution to the advancement of probabilistic deep learning and uncertainty quantification in complex machine learning models.

4.4 Limitations

Despite the valuable insights obtained from this study, several limitations must be acknowledged. These limitations primarily stem from computational constraints, methodological approximations, and dataset-specific generalizability, each of which impacts the broader applicability of the findings.

One key limitation is the computational cost associated with MCMC. While MCMC provides highly accurate posterior estimates, its high memory usage and slow convergence present challenges when scaling to large BNNs. The RWM algorithm used in this study required significant computational time and resources, making it impractical for real-time or large-scale applications. This constraint is particularly relevant when working with complex models or large datasets, where alternative inference techniques may be more viable. Conversely, VI provides improved scalability but at the expense of approximation bias. The quality of VI's posterior estimates heavily depends on the chosen variational family, which, if overly restrictive, may fail to capture the true posterior distribution. This trade-off between computational efficiency and posterior accuracy remains a fundamental challenge when selecting an appropriate inference method for Bayesian modeling.

Another limitation relates to the dataset dependency of the performance comparisons. The experiments in this study were conducted using the California Housing Prices dataset, which, while widely used in regression tasks, may not fully represent the complexities found in other domains such as financial forecasting or medical diagnostics. The findings regarding computational efficiency, scalability, and uncertainty quantification may, therefore, not generalize to datasets with different statistical properties. Future research could extend this analysis to datasets with varying noise levels, feature distributions, and problem complexities to validate

the robustness of the results.

Additionally, this study's evaluation of uncertainty quantification was limited to Prediction Interval Coverage Probability (PICP) for VI and trace plots for MCMC convergence diagnostics. While these methods provide valuable insights, other uncertainty estimation metrics, such as Expected Calibration Error (ECE), Bayesian Credible Intervals, or Marginal Likelihood Estimates, could offer a more comprehensive assessment of uncertainty quantification in BNNs. A more diverse evaluation framework would allow for a deeper understanding of the practical implications of these inference techniques in real-world applications.

Finally, the study's comparative analysis of MCMC and VI was conducted within the TFP framework, focusing on direct implementations available in this environment. While TFP provides robust tools for probabilistic modeling, differences in implementation across other libraries (such as PyMC3, Stan, or JAX-based frameworks) may lead to variations in performance and accuracy. Future work could explore cross-framework evaluations to ensure that conclusions are not specific to a particular software ecosystem. Despite these limitations, this study provides valuable empirical insights into the trade-offs between MCMC and VI for posterior estimation in BNNs. These findings contribute to the broader discussion on balancing accuracy, efficiency, and scalability in Bayesian deep learning, serving as a foundation for future advancements in the field.

Chapter 5

Conclusion and Future Work

This chapter provides a concise synthesis of the study's key findings and outlines promising avenues for future research. It first summarises the main outcomes of the comparative analysis of MCMC and VI in Bayesian Neural Networks, highlighting their relative strengths, limitations, and trade-offs. It then proposes directions for further investigation, including methodological improvements and new application areas where uncertainty-aware deep learning can be advanced.

Section 5.1 provides a comparative analysis of MCMC and VI in BNNs, highlighting their strengths, limitations, and comparison in terms of accuracy, computational efficiency, scalability, and uncertainty quantification. Section 5.2 discusses potential research avenues, including hybrid inference techniques combining MCMC and VI's advantages for improved efficiency and accuracy.

5.1 Conclusion

This study compared MCMC and VI for posterior estimation in BNNs, focusing on predictive accuracy, computational efficiency, scalability, and uncertainty quantification. The results highlight the inherent trade-offs between these two inference methods. MCMC delivers superior posterior estimates but at a high computational cost, while VI offers a more scalable and efficient alternative at the expense of approximation bias.

From an accuracy perspective, MCMC consistently outperformed VI, as evidenced by lower MSE, RMSE, and MAE across all dataset sizes. This accuracy advantage is attributed to MCMC's ability to generate asymptotically exact posterior samples, ensuring a more precise representation of uncertainty in BNN predictions. However, this precision comes at the cost of slow convergence, high memory consumption, and extensive computational requirements, making MCMC less feasible for large-scale deep-learning applications.

In contrast, VI demonstrated superior computational efficiency and scalability, completing posterior estimation in significantly less time and with lower memory usage. VI's ability to integrate seamlessly into deep-learning frameworks, particularly through TFP, further enhances its practicality for real-world applications requiring rapid inference. However, the approximation bias introduced by VI, particularly when the chosen variational family is too

restrictive, poses challenges in achieving highly accurate posterior estimates. This limitation is especially relevant in cases where precise uncertainty quantification is critical, such as in high-stakes decision-making tasks.

From an implementation perspective, MCMC required extensive tuning and diagnostics, including burn-in steps, trace plot analysis, and parameter adjustments, to ensure convergence. While these steps enhance the reliability of posterior estimation, they also add complexity to the modeling process. VI offered a more user-friendly alternative. It required fewer hyperparameter adjustments and incorporated smoothly into deep learning pipelines, making it a preferred choice for practitioners seeking a balance between efficiency and ease of use.

Ultimately, the choice between MCMC and VI depends on the application's specific requirements. MCMC remains the preferred approach for high-precision BI, particularly in scenarios where posterior accuracy and uncertainty quantification are paramount. Conversely, VI is better suited for large-scale machine learning applications, where computational efficiency and scalability lead to exact posterior estimation. By systematically evaluating MCMC and VI across multiple dimensions, this study provides valuable empirical insights into their strengths, limitations, and applicability in Bayesian deep learning. These findings serve as a foundation for future research on efficient and scalable BI methods, ensuring that BNNs continue to be a powerful tool for uncertainty-aware machine learning.

5.2 Future Work

While this study has provided a comparative analysis of MCMC and VI for BNNs, several open questions and promising research directions remain. These areas of future work could further enhance the understanding, applicability, and efficiency of BI methods in deep learning.

One promising direction is the development of hybrid inference techniques that integrate the strengths of MCMC and VI. This study demonstrates that MCMC produces more precise posterior estimates, but its computational cost limits its feasibility for large-scale applications. Conversely, VI is more scalable and computationally efficient, yet it introduces approximation bias that affects posterior accuracy. A hybrid approach could leverage MCMC's accuracy and VI's efficiency, potentially using MCMC-generated samples to refine VI approximations. Such adaptive techniques could balance computational feasibility and accuracy, making BI more practical for real-world applications.

Additionally, while the California Housing Prices Dataset effectively benchmarked the performance of MCMC and VI in Bayesian regression tasks, it does not fully capture the complexities of real-world applications. Future studies should explore these inference techniques in diverse and high-stakes domains, such as healthcare, financial modeling, autonomous systems, and climate science. For instance, BI could enhance risk assessment in credit scoring, disease prediction in medical diagnostics, or decision-making in autonomous vehicles, where uncertainty quantification is critical.

Furthermore, uncertainty quantification metrics could be expanded beyond Prediction Interval Coverage Probability (PICP) and trace plots to include Expected Calibration Error (ECE), Bayesian credible intervals, and marginal likelihood estimation. This would provide a more comprehensive understanding of uncertainty estimation and enable a more robust evaluation of model reliability in Bayesian deep learning.

Lastly, cross-framework evaluations of MCMC and VI could be explored. This study implemented both methods using TFP; however, BI frameworks such as PyMC3, Stan, JAX-based methods, or hybrid deep learning architectures could offer different trade-offs in terms of accuracy, efficiency, and ease of use. A comparative study across these frameworks could provide a broader perspective on the implementation challenges and practical choices for Bayesian deep learning. By addressing these directions, future research can contribute to the development of more efficient, scalable, and accurate Bayesian inference methods, ensuring that BNNs continue to be a powerful tool for probabilistic machine learning and uncertainty-aware decision-making.

Chapter 6

Python Code

DATA PREPROCESSING

```python
from sklearn.datasets import fetch_california_housing
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from scipy.stats import zscore

# Load the dataset
data = fetch_california_housing()
df = pd.DataFrame(data.data, columns=data.feature_names)
df["MedHouseVal"] = data.target  # Adding the target variable

# Shuffle the dataset
# Frac = 1 - 100% Dataset, Frac = 0.75 - 75% Dataset,Frac = 0.5 - 50% Dataset
df = df.sample(frac=1, random_state=42).reset_index(drop=True)
df = df.sample(frac=0.75, random_state=42).reset_index(drop=True)
df = df.sample(frac=0.5, random_state=42).reset_index(drop=True)

# Separate features and target variable
X = df.drop(columns=["MedHouseVal"])
y = df["MedHouseVal"]

# **Removing Outliers Using Z-Score Method**
z_scores = X.apply(zscore)  # Compute Z-scores
outlier_mask = (z_scores.abs() < 3).all(axis=1)  # Identify non-outlier rows

# **Apply mask to keep only non-outlier rows**
X_cleaned = X[outlier_mask].reset_index(drop=True)
y_cleaned = y[outlier_mask].reset_index(drop=True)  # Ensure y matches X
```

```
# Print data shape before and after removing outliers
print(f"Original dataset size: {X.shape[0]} samples")
print(f"Dataset size after outlier removal: {X_cleaned.shape[0]} samples")

# Split into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X_cleaned, y_cleaned, test_size=0.2, random_state=42)

# Apply Standard Scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Convert back to DataFrame for better readability
X_train_scaled = pd.DataFrame(X_train_scaled, columns=X_train.columns)
X_test_scaled = pd.DataFrame(X_test_scaled, columns=X_test.columns)

# Show a preview of cleaned data
print("First 5 rows of cleaned training data:")
print(X_train_scaled.head())


# Combine the scaled training and test data for a complete view
df_scaled = pd.concat([X_train_scaled, X_test_scaled], axis=0).reset_index(drop=True)

# Check the dimensions of the scaled dataset
print("Shape of the scaled dataset:", df_scaled.shape)

# Display first few rows of the scaled dataset
print(df_scaled.head())

# Data type and null values information
print(df_scaled.info())
```

```
# Check for missing values
print("Missing values in each column:\n", df_scaled.isnull().sum())

# Summary statistics
print(df_scaled.describe())

import matplotlib.pyplot as plt
import seaborn as sns

# Distribution plot for the target variable
plt.figure(figsize=(8, 6))
sns.histplot(df['MedHouseVal'], kde=True)
plt.title("Distribution of Median House Value")
plt.xlabel("Median House Value")
plt.ylabel("Frequency")
plt.show()

# Plot distributions for all features
df_scaled.hist(bins=30, figsize=(15, 10), layout=(3, 3))
plt.suptitle("Feature Distributions")
plt.show()

# Compute the correlation matrix
correlation_matrix = df_scaled.corr()

# Plot the correlation matrix as a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Correlation Matrix")
plt.show()
```

BNN - VI MODEL

```
BNN - VI MODEL

!pip install tensorflow==2.12.0
!pip install tensorflow-probability==0.17.0

import tensorflow as tf
import tensorflow_probability as tfp


# Convert data to tensors
X_train_scaled = tf.convert_to_tensor(X_train_scaled, dtype=tf.float32)
y_train = tf.convert_to_tensor(y_train, dtype=tf.float32)
X_test_scaled = tf.convert_to_tensor(X_test_scaled, dtype=tf.float32)
y_test = tf.convert_to_tensor(y_test, dtype=tf.float32)
print("Training Data Shape:", X_train_scaled.shape)
print("Test Data Shape:", X_test_scaled.shape)
print("Converted Training Tensors:", X_train_scaled.shape, y_train.shape)

# Define input dimension
input_dim = X_train_scaled.shape[1] if len(X_train_scaled.shape) > 1 else 1
print("Input Dimension:", input_dim)  # Confirm this is an integer

# Build an optimized VI Bayesian Neural Network with more layers and neurons
def build_optimized_vi_bnn(num_features):
    model = tf.keras.Sequential([
        tfp.layers.DenseFlipout(128, activation='relu', input_shape=(num_features,)),
        tfp.layers.DenseFlipout(64, activation='relu'),
        tfp.layers.DenseFlipout(32, activation='relu'),
        tfp.layers.DenseFlipout(1)
    ])
    model.summary()  # Display model summary
    return model
```

```
# Build and compile the optimized VI BNN model
vi_bnn_optimized = build_optimized_vi_bnn(input_dim)

# Reduce learning rate for finer updates
vi_bnn_optimized.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001), loss='mse', metrics=['mae'])
print("Optimized model compiled with a lower learning rate.")

# Set up early stopping callback to prevent overfitting
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# Train the optimized model with early stopping
history_optimized = vi_bnn_optimized.fit(
    X_train_scaled, y_train,
    epochs=100,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stopping],
    verbose=1
)
print("Optimized training complete. Final validation MAE:", history_optimized.history['val_mae'][-1])

# Predictions from the optimized model
y_pred_vi_optimized = vi_bnn_optimized.predict(X_test_scaled)
print(f"Shape of optimized predictions before flattening: {y_pred_vi_optimized.shape}")

# Flatten the predictions if necessary
y_pred_vi_optimized = y_pred_vi_optimized.flatten()
print("Optimized predictions after flattening:", y_pred_vi_optimized[:10])  # Print first 10 predictions for inspection
```

BNN - MCMC MODEL

```
[ ]  import tensorflow as tf
     import tensorflow_probability as tfp
     from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
     from scipy.stats import norm
     import numpy as np

     # Aliases for convenience
     tfd = tfp.distributions

     from scipy.stats import zscore

     # Convert data to tensors
     X_train_scaled = tf.convert_to_tensor(X_train_scaled, dtype=tf.float32)
     y_train = tf.convert_to_tensor(y_train, dtype=tf.float32)
     X_test_scaled = tf.convert_to_tensor(X_test_scaled, dtype=tf.float32)
     y_test = tf.convert_to_tensor(y_test, dtype=tf.float32)
     print("Training Data Shape:", X_train_scaled.shape)
     print("Test Data Shape:", X_test_scaled.shape)
     print("Converted Training Tensors:", X_train_scaled.shape, y_train.shape)

     # Define dimensions for the neural networks
     input_dim = X_train_scaled.shape[1]
     hidden_dim = 5
     output_dim = 1
```

```
# 1. MCMC-based Bayesian Neural Network
def bayesian_neural_network():
    weights_1 = yield tfd.Normal(loc=tf.zeros([input_dim, hidden_dim]), scale=1., name='weights_1')
    bias_1 = yield tfd.Normal(loc=tf.zeros([hidden_dim]), scale=1., name='bias_1')
    weights_2 = yield tfd.Normal(loc=tf.zeros([hidden_dim, output_dim]), scale=1., name='weights_2')
    bias_2 = yield tfd.Normal(loc=tf.zeros([output_dim]), scale=1., name='bias_2')

    x = X_train_scaled
    hidden = tf.nn.relu(tf.matmul(x, weights_1) + bias_1)
    output = tf.matmul(hidden, weights_2) + bias_2
    y = yield tfd.Normal(loc=output, scale=0.5, name='y')

# Define the joint log probability function
def joint_log_prob(y, weights_1, bias_1, weights_2, bias_2):
    model = tfd.JointDistributionCoroutineAutoBatched(bayesian_neural_network)
    return model.log_prob((weights_1, bias_1, weights_2, bias_2, y))

# Define MCMC sampling parameters
num_burnin_steps = 200
num_samples = 500

# Initialize the state
print("Initializing MCMC with Burn-in Steps:", num_burnin_steps)
print("Number of Samples:", num_samples)
initial_chain_state = [
    tf.zeros([input_dim, hidden_dim], dtype=tf.float32),
    tf.zeros([hidden_dim], dtype=tf.float32),
    tf.zeros([hidden_dim, output_dim], dtype=tf.float32),
    tf.zeros([output_dim], dtype=tf.float32)
]
print("Initial Chain State:", initial_chain_state)
```

```python
# Define the target log probability function
def target_log_prob_fn(w1, b1, w2, b2):
    return joint_log_prob(y_train, w1, b1, w2, b2)

# Define the MCMC kernel
print("Setting up MCMC Kernel...")
kernel = tfp.mcmc.RandomWalkMetropolis(
    target_log_prob_fn=target_log_prob_fn,
    new_state_fn=tfp.mcmc.random_walk_normal_fn(scale=0.01)
)
print("Kernel Ready.")

# Sample from the posterior using MCMC
@tf.function
def run_mcmc():
    return tfp.mcmc.sample_chain(
        num_results=num_samples,
        num_burnin_steps=num_burnin_steps,
        current_state=initial_chain_state,
        kernel=kernel,
        trace_fn=lambda current_state, kernel_results: kernel_results
    )

# Run MCMC and get samples
print("Running MCMC Sampling...")
samples, _ = run_mcmc()
print("MCMC Sampling Complete.")
print("Sample Shapes:", [sample.shape for sample in samples])
weights_1_samples, bias_1_samples, weights_2_samples, bias_2_samples = samples
```

```python
# Calculate mean parameters from MCMC samples
mean_weights_1 = np.mean(weights_1_samples.numpy(), axis=0)
mean_bias_1 = np.mean(bias_1_samples.numpy(), axis=0)
mean_weights_2 = np.mean(weights_2_samples.numpy(), axis=0)
mean_bias_2 = np.mean(bias_2_samples.numpy(), axis=0)
print("Mean Weights and Biases Calculated.")
print("Mean Weights 1 Shape:", mean_weights_1.shape)
print("Mean Bias 1 Shape:", mean_bias_1.shape)
print("Mean Weights 2 Shape:", mean_weights_2.shape)
print("Mean Bias 2 Shape:", mean_bias_2.shape)

# Prediction function for MCMC-based model
def predict(X, weights_1, bias_1, weights_2, bias_2):
    hidden = tf.nn.relu(tf.matmul(X, weights_1) + bias_1)
    return tf.matmul(hidden, weights_2) + bias_2

# Predictions from MCMC model
print("Generating Predictions on Test Data...")
y_pred_mcmc = predict(X_test_scaled, mean_weights_1, mean_bias_1, mean_weights_2, mean_bias_2)
y_pred_mcmc = y_pred_mcmc.numpy().flatten()
print("Predictions Complete. Predictions Shape:", y_pred_mcmc.shape)
```

MODEL EVALUATION

```
[ ]  MODEL EVALUATION

     # Calculate mean and std from the VI model's predictions for PICP calculation
     num_samples = 100
     vi_samples = np.array([vi_bnn_optimized.predict(X_test_scaled).flatten() for _ in range(num_samples)])
     mean_predictions_vi = vi_samples.mean(axis=0)
     std_predictions_vi = vi_samples.std(axis=0)

     # Calculate evaluation metrics for both models
     def evaluate_model(y_true, y_pred, std_dev=None, confidence_level=0.95):
         mse = mean_squared_error(y_true, y_pred)
         rmse = np.sqrt(mse)
         mae = mean_absolute_error(y_true, y_pred)
         r2 = r2_score(y_true, y_pred)

         # Calculate PICP if std_dev is provided
         if std_dev is not None:
             z_score = norm.ppf(0.5 + confidence_level / 2)
             lower_bound = y_pred - z_score * std_dev
             upper_bound = y_pred + z_score * std_dev
             picp = np.mean((y_true >= lower_bound) & (y_true <= upper_bound))
         else:
             picp = None

         return mse, rmse, mae, r2, picp

     # Evaluate MCMC model
     mse_mcmc, rmse_mcmc, mae_mcmc, r2_mcmc, _ = evaluate_model(y_test.numpy(), y_pred_mcmc)

     # Evaluate VI model
     mse_vi, rmse_vi, mae_vi, r2_vi, picp_vi = evaluate_model(
         y_test.numpy(), mean_predictions_vi, std_predictions_vi
     )
```

```
# Evaluate MCMC model
mse_mcmc, rmse_mcmc, mae_mcmc, r2_mcmc, _ = evaluate_model(y_test.numpy(), y_pred_mcmc)

# Evaluate VI model
mse_vi, rmse_vi, mae_vi, r2_vi, picp_vi = evaluate_model(
    y_test.numpy(), mean_predictions_vi, std_predictions_vi
)

# Create a DataFrame to compare the results
comparison_results = pd.DataFrame({
    'Metric': ['MSE', 'RMSE', 'MAE', 'R²', 'PICP'],
    'MCMC Model': [mse_mcmc, rmse_mcmc, mae_mcmc, r2_mcmc, 'N/A'],
    'VI Model': [mse_vi, rmse_vi, mae_vi, r2_vi, picp_vi]
})

print(comparison_results)
```

COMPARITIVE ANALYSIS - Computational Time

```
[ ]  !pip install memory-profiler
```

```
[ ]  !pip install memory-profiler
     import time
     import matplotlib.pyplot as plt
     from memory_profiler import profile
     import numpy as np
     from sklearn.metrics import mean_absolute_error

     ### 1. Computational Time Tracking ###

     start_time_vi = time.time()
     history = vi_bnn_optimized.fit(X_train_scaled, y_train, epochs=100, batch_size=32, validation_split=0.2, verbose=1)  # Run the VI training here
     end_time_vi = time.time()
     print(f"VI Training Time: {end_time_vi - start_time_vi:.2f} seconds")


     start_time_mcmc = time.time()
     samples, _ = tfp.mcmc.sample_chain(
         num_results=num_samples,
         num_burnin_steps=num_burnin_steps,
         current_state=initial_chain_state,
         kernel=kernel,
         trace_fn=lambda current_state, kernel_results: kernel_results
     )  # Run MCMC sampling here
     end_time_mcmc = time.time()
     print(f"MCMC Sampling Time: {end_time_mcmc - start_time_mcmc:.2f} seconds")
```

COMPARITIVE ANALYSIS - Convergence Behaviour VI

```
[ ]  import matplotlib.pyplot as plt

     def plot_vi_convergence(history):
         plt.plot(history.history['loss'], label='Training Loss')
         plt.plot(history.history['val_loss'], label='Validation Loss')
         plt.xlabel('Epochs')
         plt.ylabel('Loss')
         plt.legend()
         plt.title("Convergence Behavior for VI")
         plt.show()

     # Call this function after VI training to visualize convergence
     plot_vi_convergence(history)
```

COMPARITIVE ANALYSIS - Convergence Behaviour MCMC

```
[ ]  def plot_mcmc_trace(weights_1_samples, bias_1_samples, weights_2_samples, bias_2_samples):
         plt.figure(figsize=(12, 8))

         plt.subplot(2, 2, 1)
         plt.plot(weights_1_samples[:, 0, 0], label='weights_1[0,0]')
         plt.xlabel("Samples")
         plt.ylabel("Value")
         plt.title("Trace plot for weights_1[0,0]")
         plt.legend()

         plt.subplot(2, 2, 2)
         plt.plot(bias_1_samples[:, 0], label='bias_1[0]')
         plt.xlabel("Samples")
         plt.ylabel("Value")
         plt.title("Trace plot for bias_1[0]")
         plt.legend()

         plt.subplot(2, 2, 3)
         plt.plot(weights_2_samples[:, 0, 0], label='weights_2[0,0]')
         plt.xlabel("Samples")
         plt.ylabel("Value")
         plt.title("Trace plot for weights_2[0,0]")
         plt.legend()
```

```
         plt.subplot(2, 2, 3)
         plt.plot(weights_2_samples[:, 0, 0], label='weights_2[0,0]')
         plt.xlabel("Samples")
         plt.ylabel("Value")
         plt.title("Trace plot for weights_2[0,0]")
         plt.legend()

         plt.subplot(2, 2, 4)
         plt.plot(bias_2_samples[:, 0], label='bias_2[0]')
         plt.xlabel("Samples")
         plt.ylabel("Value")
         plt.title("Trace plot for bias_2[0]")
         plt.legend()

         plt.tight_layout()
         plt.show()

     # Call this after sampling to assess MCMC convergence
     plot_mcmc_trace(weights_1_samples, bias_1_samples, weights_2_samples, bias_2_samples)
```

COMPARITIVE ANALYSIS - Memory Usage

```
[ ]  from memory_profiler import memory_usage

     # Measure memory usage for VI model training
     def train_vi():
         history = vi_bnn_optimized.fit(X_train_scaled, y_train, epochs=100, batch_size=32, validation_split=0.2, verbose=1)
         return history

     print("Memory usage for VI model training:")
     vi_memory = memory_usage((train_vi,), interval=0.1)
     print(f"Peak memory usage for VI training: {max(vi_memory)} MB")

     # Measure memory usage for MCMC sampling
     def run_mcmc():
         samples, _ = tfp.mcmc.sample_chain(
             num_results=num_samples,
             num_burnin_steps=num_burnin_steps,
             current_state=initial_chain_state,
             kernel=kernel,
             trace_fn=lambda current_state, kernel_results: kernel_results
         )
         return samples

     print("Memory usage for MCMC sampling:")
     mcmc_memory = memory_usage((run_mcmc,), interval=0.1)
     print(f"Peak memory usage for MCMC sampling: {max(mcmc_memory)} MB")
```

```
pip install pandas scipy

import pandas as pd
from scipy import stats

# Define the performance metrics for MCMC and VI models
data = {
    "Dataset Size": ["100%", "75%", "50%"],
    "MSE_MCMC": [1.2875, 1.2492, 1.4155],
    "MSE_VI": [3.9636, 1.3530, 1.6412],
    "RMSE_MCMC": [1.1347, 1.1177, 1.1898],
    "RMSE_VI": [1.9909, 1.1632, 1.2811],
    "MAE_MCMC": [0.8598, 0.8267, 0.8880],
    "MAE_VI": [1.0173, 0.8814, 0.9430],
    "R2_MCMC": [-0.0668, -0.0672, -0.1520],
    "R2_VI": [-2.2843, -0.1559, -0.3357],
    "PICP_VI": [0.4706, 0.7300, 0.9752]  # PICP is only available for VI
}

# Convert to a DataFrame
df = pd.DataFrame(data)

# Conduct paired t-tests for all metrics
t_mse, p_mse = stats.ttest_rel(df["MSE_MCMC"], df["MSE_VI"])
t_rmse, p_rmse = stats.ttest_rel(df["RMSE_MCMC"], df["RMSE_VI"])
t_mae, p_mae = stats.ttest_rel(df["MAE_MCMC"], df["MAE_VI"])
t_r2, p_r2 = stats.ttest_rel(df["R2_MCMC"], df["R2_VI"])

# Since PICP is only for VI, we conduct a one-sample t-test to check its deviation from a theoretical mean (e.g., 0.95)
t_picp, p_picp = stats.ttest_1samp(df["PICP_VI"], 0.95)

# Store results in a DataFrame
t_test_results = pd.DataFrame({
    "Metric": ["MSE", "RMSE", "MAE", "R2", "PICP (VI Only)"],
    "t-statistic": [t_mse, t_rmse, t_mae, t_r2, t_picp],
    "p-value": [p_mse, p_rmse, p_mae, p_r2, p_picp]
})

# Display results
print(t_test_results)
```

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org. 25

Arbel, J., Pitas, K., Vladimirova, M., and Fortuin, V. (2023). A primer on bayesian neural networks: Review and debates. *arXiv preprint arXiv:2309.16314*. 2, 14

Banerjee, S., Carlin, B. P., and Gelfand, A. P. (2014). *Hierarchical Modeling and Analysis for Spatial Data (Second ed.)*. CRC Press. p. xix. 1

Berg, B. A. (2004). *Markov Chain Monte Carlo Simulations and Their Statistical Analysis*. World Scientific, Singapore. 12

Bevans, R. (2023). An introduction to t tests: Definitions, formula, and examples. *Scribbr*. Accessed: March 6, 2025. 17

Bishop, C. M. (1997). Bayesian neural networks. *Computer Science*. 2

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112. 25, 26, 27

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight uncertainty in neural networks. *In Proceedings of the 32nd International Conference on Machine Learning (ICML-15) (pp. 1613-1622)*. 2, 5

Brooks, S., Gelman, A., Jones, G., and Meng, X. (2011). *Handbook of markov chain monte carlo*. books.google.com. 1, 6

Brunton, S. and Kutz, J. (2022). Data-driven science and engineering: Machine learning, dynamical systems, and control. *Cambridge University Press*. 1

CaliforniaHousing, S.-L. (2024). California housing dataset — scikit-learn mooc. Accessed: Feb. 28, 2025. 19

Chandra, R., Chen, R., and Simmons, J. (2023). Bayesian neural networks via mcmc: A python-based tutorial. *IEEE Access*. 2

Chang, D. (2021). Bayesian neural networks: Essentials. *arXiv*, abs/2106.11334. 4

Chib, S. and Greenberg, E. (1995). Understanding the metropolis-hastings algorithm. *The American Statistician*, 49(4):327–335. 12, 25, 42

Chib, S. and Greenberg, E. (1996). Markov chain monte carlo simulation methods in econometrics. *Econometric Theory*, 12(3):409–431. 12

Craiu, R. V., Lemieux, C., and Rosenthal, J. S. (2022). Adaptive mcmc algorithms for efficient bayesian computation. *Statistical Science*, 37(3):350–372. 3

Dillon, J. V., Langmore, I., Tran, D., Brevdo, E., Vasudevan, S., Moore, D., Patton, B., Alemi, A., Hoffman, M., and Saurous, R. A. (2017). Tensorflow distributions. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pages 1–6. 25, 26

Emerson, G. E. T. and Copestake, A. A. (2017). Variational inference for logical inference. *arXiv.org*. 4, 42

Fan, C., Gao, F., Sun, S., and Cui, F. (2008). Bayesian neural networks and its application. *Fourth International Conference on Natural Computation*. 2

Fan, Y. and Sisson, S. (2010). Reversible jump markov chain monte carlo. *Computer Science, Mathematics, arXiv: Methodology*. 3, 41

Ganguly, A. and Earp, S. W. F. (2021). An introduction to variational inference. *arXiv preprint arXiv:2108.13083*. 13

Gawlikowski, J., Tassi, C., and Ali, M. (2021). A survey of uncertainty in deep neural networks. *arXiv preprint arXiv:2107.03342*. 15

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., and Rubin, D. B. (2013). Bayesian data analysis. *CRC Press*. 2, 6, 25, 26, 27

Geman, S. and Geman, D. (1984). Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741. 11

Ghalebikesabi, S., Holmes, C., Fong, E., and less, B. L. (2022). Density estimation with autoregressive bayesian predictives. *Mathematics*. 4, 42

Gilks, W. (2005). Markov chain monte carlo. *Computer Science, Mathematics*. 3, 41

Glasserman, P. (2004). *Monte Carlo Methods in Financial Engineering*. Springer. 9

Harik, G. and Shazeer, N. M. (2010). Variational program inference. *arXiv.org*. 4

Hastings, W. K. (1970). Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109. 11

Hayes, B. (2005). First links in the markov chain. *American Scientist*, 93(4):308–313. 10

Hurtado, J. E. (2004). Structural reliability methods. *Computers Structures*, 82(2-3):127–136. 9

Jaakkola, T. and Jordan, M. I. (2000). Bayesian parameter estimation via variational methods. *Computer Science, Mathematics*. 4

Jantre, S. R., Bhattacharya, S., and Maiti, T. (2020). Quantile regression neural networks: A bayesian approach. *Mathematics, Computer Science*. 4

Kalos, M. H. and Whitlock, P. A. (2008). *Monte Carlo Methods: Volume I: Basics*. John Wiley & Sons. 8, 9

Kim, M. and Pavlovic, V. (2020). Recursive inference for variational autoencoders. *arXiv: Learning*. 6

Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*. 14, 26, 27

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. *In Proceedings of the International Conference on Learning Representations (ICLR)*. 1, 2, 41

Kononenko, I. (1989). Bayesian neural networks. *Computer Science, Medicine*. 2

Krishnan, R., Subedar, M., and Tickoo, O. (2019). Efficient priors for scalable variational inference in bayesian deep neural networks. *Computer Science*. 4

Lampinen, J. and Vehtari, A. (2001). Bayesian approach for neural networks–review and case studies. *Computer Science, Mathematics*. 2

Landau, D. P. and Binder, K. (2014). *A Guide to Monte Carlo Simulations in Statistical Physics*. Cambridge University Press. 9

Larget, B. (2005). Introduction to markov chain monte carlo methods in molecular evolution. *Biology*. 3

Li, X. (2024). Advancements in monte carlo methods for high-dimensional bayesian inference. *Journal of Computational Statistics*, 39(2):251–275. 3

Maceda, E., Hector, E. C., Lenzi, A., and Reich, B. J. (2024). A variational neural bayes framework for inference on intractable posterior distributions. *Computer Science, Mathematic*. 4

Magris, M. and Iosifidis, A. (2022). Bayesian learning for neural networks: An algorithmic survey. *arXiv preprint arXiv:2211.11865*. 14

Magris, M. and Iosifidis, A. (2023). Bayesian learning for neural networks: an algorithmic survey. *Artificial Intelligence Review*. 2

Marcot, B. and Penman, T. (2019). Advances in bayesian network modelling: Integration of modelling technologies. *Environmental Modelling Software*, 111:386–399. 4

Markov, A. A. (1906). Rasprostranenie zakona bol'shih chisel na velichiny, zavisyaschie drug ot druga. *Izvestiya Fiziko-matematicheskogo obschestva pri Kazanskom universitete*, 15:135–156. 9

Maruyama, O. (2014). Markov chain monte carlo algorithms. *Biology, Computer Science*. 3

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. (1953). Equations of state calculations by fast computing machines. *Journal of Chemical Physics*, 21 (6), pp. 1087-1092. 11, 12, 25, 26, 27

Metropolis, N. and Ulam, S. (1949). The monte carlo method. *Journal of the American Statistical Association*, 44(247):335–341. 8, 9, 12

Mukherjee, S. and Sen, S. (2021). Variational inference in high-dimensional linear regression. *Journal of machine learning research*. 4, 41

Mullachery, V., Khera, A., and Husain, A. (2018). Bayesian neural networks. *arXiv*, abs/1801.07790. 4, 5

Müller, P. (2004). Monte carlo methods and bayesian computation : Mcmc. *Computer Science, Mathematics*. 3, 42

Neal, R. M. (2012). *Bayesian Learning for Neural Networks*. Springer. 1, 2, 25, 26, 27, 41

Ni, P., Han, Q., Du, X., Fu, J., and Xu, K. (2024). Probabilistic model updating of civil structures with a decentralized variational inference approach. *Mechanical Systems and Signal Processing*, 209. 13

Nummenmaa, A., Auranen, T., Hämäläinen, M., Jääskeläinen, I., Lampinen, J., Sams, M., and less, A. V. (2007). Hierarchical bayesian estimates of distributed meg sources: Theoretical aspects and comparison of variational and mcmc methods. *Medicine*. 3, 41, 42

Opper, M. (2019). Variational inference for stochastic differential equations. *Annals of Physics*. 4, 42

Pandas (2024). *pandas.isnull — pandas 2.2.0 documentation*. Accessed: Feb. 28, 2025. 20

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830. 15, 16

Pharr, M., Jakob, W., and Humphreys, G. (2016). *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann, 3rd edition. 9

Posch, K., Steinbrener, J., and Pilz, J. (2019). Variational inference to measure model uncertainty in deep neural networks. *arXiv: Machine Learning*. 6

Robert, C. P. and Casella, G. (2004). *Monte Carlo Statistical Methods*. Springer, New York, 2nd edition. 8, 9

Saadi, H. A., Ykhlef, F., and Guessoum, A. (2011). Mcmc for parameters estimation by bayesian approach. *Computer Science, Engineering, Mathematics*. 3

Salimans, T., Diederik, A., Kingma, P., Welling, M., Kingma, D. P., Welling, M., and less, U. N. (2015). Markov chain monte carlo and variational inference: Bridging the gap. *Computer Science, Mathematics*. 3, 4, 42

Scikit-Learn (2024). *sklearn.preprocessing.StandardScaler — scikit-learn 1.3.0 documentation*. Accessed: Feb. 28, 2025. 20, 21

Shridhar, K., Laumann, F., and Liwicki, M. (2018). Uncertainty estimations by softplus normalization in bayesian convolutional neural networks with variational inference. *Computer Science, arXiv: Learning*. 4, 5

Shukla, P. (2024). How to detect outliers with z-score. Accessed: Feb. 28, 2025. 20, 21

Skaanning, C. (2006). Markov chain monte carlo methods. *Computer Science, Mathematics*. 3

Verhofen, M. (2005). Markov chain monte carlo methods in financial econometrics. *Economics, Business Financial Markets and Portfolio Management*. 3

Vihola, M. (2020). Ergodicity of adaptive markov chain monte carlo methods. *Journal of Applied Probability*, 57(1):109–131. 3

Wen, Y., Vicol, P., Ba, J., Tran, D., and Grosse, R. (2018). Flipout: Efficient pseudo-independent weight perturbations on mini-batches. In *International Conference on Learning Representations (ICLR)*. 25, 26, 27

Wrede, F., Eriksson, R., Jiang, R., Petzold, L., Engblom, S., Hellander, A., and Singh, P. (2021). Robust and integrative bayesian neural networks for likelihood-free parameter inference. *Computer Science*. 4

Xu, Z. (2020). On the computational asymptotics of gaussian variational inference. *arXiv: Computation*. 6

Yang, Z. (2014). *Molecular Evolution: A Statistical Approach*. Oxford University Press, Oxford. 12

Zhang, K., Su, K., Yao, Y., Li, Q., and Chen, S. (2022). Dynamic evaluation and analysis of the uncertainty of roundness error measurement by markov chain monte carlo method. *Measurement*, 201. 1