Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Oscar Selnes Bognæs

Radix Spline Parameter Optimization

Achieving fast lookup times with minimal memory usage

Master's thesis in Informatics
Supervisor: Svein Erik Bratsberg
June 2023

NTNU
Norwegian University of
Science and Technology

Oscar Selnes Bognæs

Radix Spline Parameter Optimization

Achieving fast lookup times with minimal memory usage

Master's thesis in Informatics
Supervisor: Svein Erik Bratsberg
June 2023

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

ABSTRACT

In this study, we investigate the possibility of predicting the optimal parameter configuration for the radix spline learned index structure. To tackle this challenge, we propose a novel approach utilizing a multi-output neural network model trained on synthetic datasets. By generating synthetic data based on various distributions and extracting relevant features to capture dataset characteristics, we enable the model to learn and predict the crucial parameters of radix bits and spline error. Our experiments demonstrate the model's capability to predict a satisfactory parameter configuration for a considerable number of datasets. However, we also identify certain limitations and areas for improvement, such as the need for generating more complex and diverse datasets, as well as exploring additional features. The findings of this research contribute to the advancement of learned index structures and pave the way for the development of more efficient and automated parameter tuning techniques.

SAMMENDRAG

I denne oppgaven utforsker vi muligheten for å predikere den optimale konfigurasjonen av Radix Spline ved hjelp av et multi-output nevralt nettverk trent på syntetiske datasett basert på ulike distribusjoner. Det nevrale nettverket viser en evne til å forutsi konfigurasjoner som gir tilfredsstillende resultater, ved å ta hensyn til de viktigste egenskapene til hvert datasett som er reflektert i treningsdataene. Imidlertid har vi også identifisert flere begrensninger og utfordringer i denne studien, inkludert mangel på tilstrekkelig treningsdata og manglende representasjon av datasettet gjennom egenskapene som er brukt. Oppdagelsene i denne studien gir et verdifullt bidrag til lærte indekser som et felt og gir et grunnlag for videre forskning innen dette området.

PREFACE

The motivation behind this research stems from my keen interest in databases and search. Throughout my experience in computer science, I have always found analyzing vast amounts of information, identifying patterns, and manipulating data to be intriguing. When I discovered the concept of combining AI and indexes, specifically learned indexes, I was captivated. Given the limited literature on this topic, I conducted a significant amount of experimentation to angle my work towards contributing something entirely new to the field of learned indexes.

I would like to express my sincere gratitude to my supervisor, Svein Erik Bratsberg, for providing the necessary guidance and freedom to complete my thesis. I would also like to express my gratitude to Kjetil Nørvåg for providing me with a space on the Dif-server.

Oscar Selnes Bognæs
Trondheim, June 6, 2023

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

List of all abbreviations in alphabetic order:

- **ART** Adaptive Radix Tree
- **CDF** Cumulative Distribution Function
- **DBMS** Database Management Systems
- **HT** Hist-Tree
- **LSM-Tree** Log-Structured Merge Tree
- **MKIF** Mean Key Interval Frequency
- **ML** Machine Learning
- **MSE** Mean Squared Error
- **MUKI** Mode of Unique Key Intervals
- **NN** Neural Network
- **NUKI** Number of Unique Key Intervals
- **PGM** Piecewise Geometric Model
- **PLEX** Practical Learned Index
- **ReLU** Rectified Linear Unit
- **RMSE** Root Mean Square Error
- **RS** Radix Spline
- **RMI** Recursive Model Index
- **SOSD** Search on Sorted Data
- **NTNU** Norwegian University of Science and Technology

CHAPTER

ONE

INTRODUCTION

Structures such as B-trees, hash-maps, and Bloom filters are common indexing structures used in DBMS. Although these indexes have improved significantly over the years, they still do not assume anything about the data distribution. To address this limitation, one possible solution is to learn the data distribution, which can enable the creation of highly optimized index structures. However, manually creating specialized solutions for every use case would require a lot of resources and is entirely unrealistic. Therefore, introducing machine learning could provide significant benefits when it comes to learning the data distribution.

1.1 Purpose and Motivation

Kraska et al. 2018 introduced the use of machine learning to learn the data distribution. Their study compared b-tree indexes to a learned index consisting of a hierarchy of learned models. The Recursive Model Index (RMI), as it is called, accurately predicts the area of the desired element faster than the b-tree while using less memory.

Existing structures often require multiple training passes over the data, leading to slow build times and limited effectiveness in real-world scenarios. In addition, they can be inefficient and impractical to use. To solve this problem, Kipf et al. 2020 introduced Radix Spline, which only requires a single pass over the underlying data to build the index. Furthermore, Radix Spline can be tuned to balance lookup time and index size, this can be done by modifying two parameters: the number of radix bits and spline error.

However, different datasets have varying characteristics that affect the building of the index. Therefore, to achieve the best lookup time and index size, different datasets require different parameter configurations. In this thesis, I aim to train a neural network to recognize the unique features of various datasets accurately. Hopefully, this will enable the network to accurately predict the optimal configuration for any given dataset.

1.2 Goals and Research Questions

Goal *Contribute to the field of learned indexes by exploring an alternative approach to auto-tune radix spline.*

Research question 1 (RQ1) *Can neural networks effectively learn the optimal configuration to use with radix spline for any given dataset?*

Research question 2 (RQ2) *How does the neural network's (NN) chosen configuration for a specific dataset compare to the worst-case configuration for the same dataset?*

1.3 Content of thesis

In this thesis, we aim to provide a comprehensive overview of the research conducted. Chapter 2 will provide essential definitions and background knowledge necessary for further reading. Chapter 3 delves into related works, exploring various learned index structures, their limitations, and performance characteristics. In Chapter 4, we discuss the experimental setup and methodology employed in this research. The results of our experiments and their evaluation in relation to the research questions are presented in Chapter 5. Chapter 6 covers the implementation details, limitations encountered, and outlines potential future research directions. Finally, Chapter 7 concludes the thesis, summarizing the key findings and contributions.

CHAPTER

TWO

BACKGROUND AND DEFINITIONS

2.1 Definitions

2.1.1 Index Structures

An index structure is a tool used to enhance retrieval speed in databases and file systems. It can collect a specific key, a range of values, or a combination of keys based on a search criterion. The use of an index structure is to reduce the number of data blocks needed to access to meet the query requirement. In an index structure, data is usually arranged in a tree-like structure, with each node corresponding to a range of values or a specific key. Pointers are used in the leaf node to point to the actual data.

2.1.2 B+-tree

A B+-tree is a tree data structure used for organizing and indexing data to facilitate fast retrieval, as discussed in Section 2.1.1. Unlike leaf nodes, each internal node in a B+-tree only stores the key and a pointer to the child nodes, which in turn point to the actual data. The keys in the internal nodes assist in the search by excluding nodes that do not lead to the desired value.

B+-trees are balanced trees that have the same length for all paths leading from the root node to the leaf node. This property results in a time complexity of *O(log n)*, which corresponds to the height of the tree. In addition, B+-trees are optimized for performing range queries, making them a popular choice for searching in databases and file systems (Ramakrishnan and Gehrke 2002). An illustration of a b-tree can be seen in Figure 2.1.1.

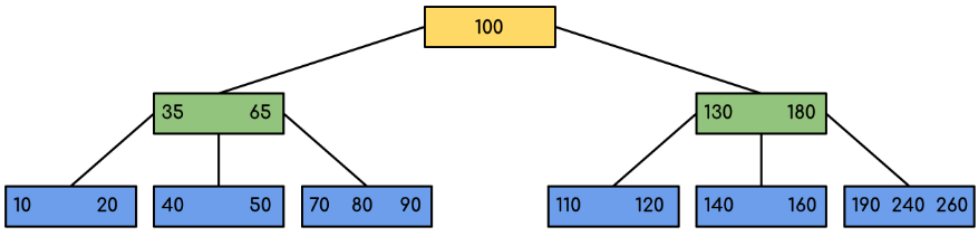


Figure 2.1.1: An illustration of a b-tree (GeekForGeeks 2023a).

2.1.3 Neural Network

A neural network is a powerful machine learning model specifically designed to identify intricate patterns and relationships within vast amounts of data (Alpaydin 2014), encompassing millions of entries. Due to their exceptional pattern detection capabilities, neural networks are instrumental in numerous real-world applications, including statistics and data science.

The structure and function of a neural network is inspired by the human brain. It consists of interconnected layers of nodes, or neurons, which receive input from other neurons, process the information using non-linear activation functions, and produce outputs that are sent forward to other nodes (Alpaydin 2014).

During the training of a neural network model, the weight of each connection is adjusted or tuned to minimize a cost function that measures the error between the predicted outputs and the actual output for a given set of input data. Once the model is trained, it can be used to predict other data it has never seen before by passing it through the network and computing the output of the final layer. Figure 2.1.2 illustrates a feed-forward neural network comprising three input nodes, a single hidden layer, and two output nodes.

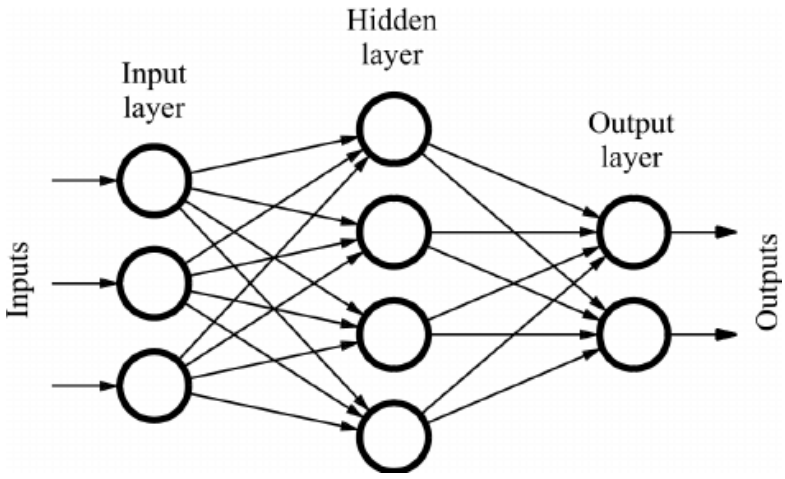


Figure 2.1.2: An illustration of a feed-forward neural network (Quiza and Davim 2011).

2.1.3.1 Feature Extraction

Feature extraction is the process of transforming raw data into a format that highlights the key characteristics of the data, significantly reducing the resources required to describe it. This makes the task more manageable for the machine learning algorithm. It's crucial to select features relevant to the specific domain during this process. Done correctly, this can greatly minimize the amount of noise and irrelevant data (Mathworks n.d.). Feature extraction can be accomplished manually or automatically:

Manual feature extraction
: Manual feature extraction provides greater flexibility in feature selection. Additionally, irrelevant features can be efficiently filtered out, reducing computational complexity and conserving valuable computational resources. However, it necessitates in-depth domain knowledge and a comprehensive understanding of the data.

Automatic feature extraction
: Automatic feature extraction can automatically extract features using specialized algorithms or deep networks, eliminating the need for human intervention. This approach is preferred when there is a requirement for efficient extraction of features from raw data.

2.1.3.2 Model Fitting

In machine learning, the term 'model fit' typically pertains to the extent to which a model can generalize to new data that differs from the training dataset. Essentially, it assesses the model's ability to accurately predict values based on previously unseen or dissimilar data. In machine learning, a model's fit can be characterized as underfitting, overfitting, or achieving a balanced and optimal fit that is appropriate for the task. Figure 2.1.3 illustrates the different types of model fits in a classification problem.

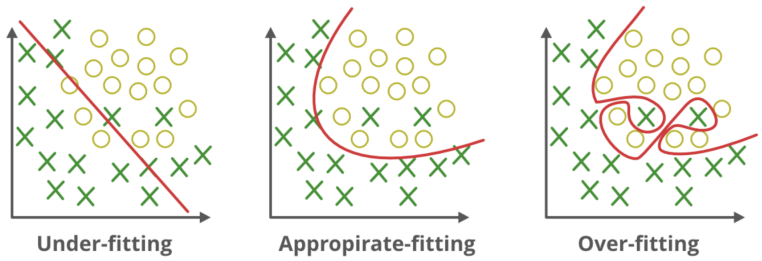


Figure 2.1.3: Model fit in a classification problem (GeekForGeeks 2023b).

2.1.3.3 K-Fold Cross Validation

K-fold cross-validation is a popular technique used in machine learning to evaluate the performance of a predictive model. It involves partitioning the available dataset into K equal-sized subsets, or *folds*. The model is then trained and evaluated K times, each time using a different fold as the validation set and the remaining folds as the training set. This process allows for a comprehensive assessment of the model's performance, as it provides an average performance metric

across multiple iterations. K-fold cross-validation helps in estimating how well the model will generalize to unseen data and provides insights into the robustness of the model. By incorporating validation at each iteration, it helps to identify potential overfitting or underfitting issues. Overall, K-fold cross-validation is a valuable technique for model evaluation and selection, providing a more reliable estimation of the model's predictive capabilities.

2.1.3.4 Root Mean Square Error

The root mean square error (RMSE) is a widely used metric in various fields, including statistics, data analysis, and machine learning, to quantify the accuracy of a predictive model. It measures the average magnitude of the differences between predicted values and the corresponding true values. RMSE provides a single numerical value that summarizes the overall prediction error, with lower values indicating better model performance. The formula for calculating RMSE involves taking the square root of the average of squared differences between predicted values (ŷ) and true values (y) across a dataset of size n:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2.1}$$

In this formula, the difference between each predicted value and the corresponding true value is squared, summed up, and divided by the number of data points. Finally, the square root is taken to obtain the RMSE. This ensures that the RMSE value is in the same unit as the dependent variable, allowing for meaningful interpretation and comparison of prediction errors.

2.1.4 Cumulative Distribution Function

The Cumulative Distribution Function (CDF) quantifies the probability that a random variable X takes on a value less than or equal to a specified value x, providing a cumulative view of the associated probabilities (Haben, Voß, and Holderbaum 2023). The CDF can be defined as:

$$F_X(x) = P(X \leq x) \tag{2.2}$$

2.1.5 Probability Density Function

The Probability Density Function (PDF) is related to the CDF as it helps us understand the likelihood of different outcomes for a random variable. Unlike the CDF, which provides the probability of the random variable being less than or equal to a specified value, the PDF gives us the probability density or likelihood of the random variable taking on a specific value. The PDF finds wide application in statistics for data analysis, parameter estimation, and prediction based on probability distributions.

2.1.6 Z-score Normalization

Z-score normalization is a commonly used technique in data mining that calculates the mean of all data points. Each data point is then assigned a score based on the standard deviation from the mean. These scores provide valuable insights into how each data point compares to the overall distribution, facilitating the identification of outliers or patterns in the data (PATRO and Sahu 2015). The z-score normalization for a list of integers can be defined as:

$$z_i = \frac{x_i - \bar{x}}{s} \tag{2.3}$$

2.1.7 LSM-Trees

The Log-Structured Merge Tree (LSM-Tree) is a powerful data structure utilized in storage and database systems to overcome the limitations of traditional B-trees, particularly in scenarios with high write intensity. The LSM-Tree provides a highly efficient insertion operation with a worst-case time complexity of O(1). In contrast, search and delete operations exhibit a worst-case time complexity of O(n). By optimizing the write path, LSM-Trees offer significant advantages in terms of write throughput, making them well-suited for workloads where frequent updates are required. (O'Neil et al. 1996)

The structure of an LSM-Tree consists of a MemTable for in-memory write buffering and disk-based Sorted String Tables (SSTables). The MemTable serves as a write buffer, storing recently written data before flushing it to disk as immutable SSTables. SSTables are organized in multiple levels, with each level containing sorted files. The compaction process merges and eliminates redundant data to create compacted SSTables in higher levels. This structure enables efficient write operations and optimized read performance by balancing in-memory buffering with disk-based storage and compaction processes. LSM-Trees are well-suited for write-intensive workloads in storage and database systems. Figure 2.1.4 depicts the structure of an LSM-Tree.

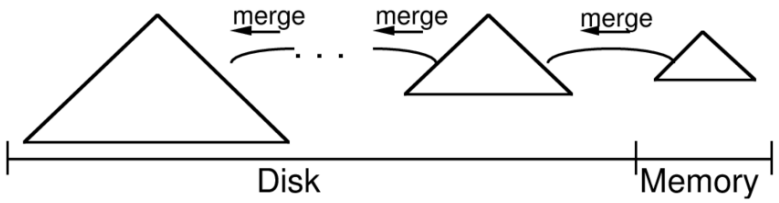


Figure 2.1.4: The merging process in an LSM-Tree (O'Neil et al. 1996).

CHAPTER

THREE

RELATED WORKS

3.1 The First Learned Index Structure

One of the first learned indexes were proposed by Kraska et al. 2018 and laid the foundation for learned indexes and proved how learning the data distribution could have significant benefit when it comes to lookup time and memory footprint. Kraska et al. 2018 argued that any index is essentially already a model and based on this assumption could easily be replaced by a learned model. According to Kraska et al. 2018, a B-Tree is a model because it has an error bound in the form that the B-Tree does not point out the exact position of the key, but rather the page. So in this case the error bound would be *pos - 0* to *pos + pagesize*.

So in theory we can replace this B-Tree with an error bound of *pos - 0 to pos + pagesize* with a learned model (for example a neural net) with an error bound of *pos - min_err* to *pos+max_err*. An illustration of this is shown in Figure 3.1.1.

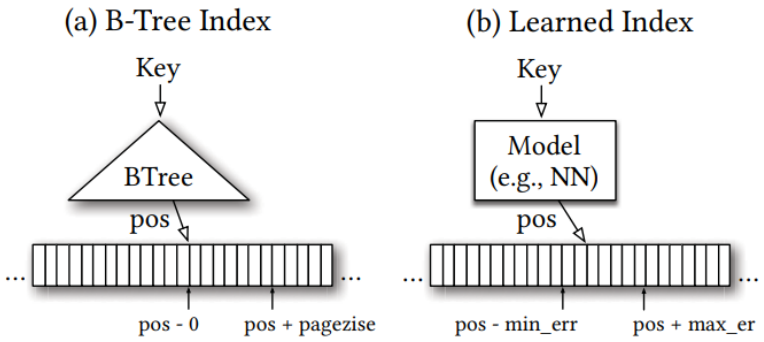


Figure 3.1.1: Replacing B-Trees with learned models models (Kraska et al. 2018).

Learning the entire data distribution can be quite expensive, especially when there is a very complex distribution and a large number of elements. Take for example 100M records, it would be unrealistic to predict the position with high accuracy (search range of 100 for example) using only a single model. Since neural nets are good at learning the general data distribution this model would just not be accurate enough without requiring an unreasonable amount of processing power and memory. Instead, a tree-like tree-like structure called The Recursive Model Index

(RMI) is proposed, which is essentially a hierarchy of models with different stages. This opens up the possibility to use multiple models and exploit the strengths of the different models.

As mentioned above, a single neural net model would not be able to accurately predict the position of a key amongst 100M records. Instead, we could use the RMI and assign the neural net model to the first stage, here the model can reduce the search area from 100M to 10K, a precision gain of 10000. We can then use other models in later stages to further narrow down the search area, like linear regression models. Linear regression models are a good choice because they are inexpensive in lookup time. The general structure of The Recursive Model can be seen in Figure 3.1.2.

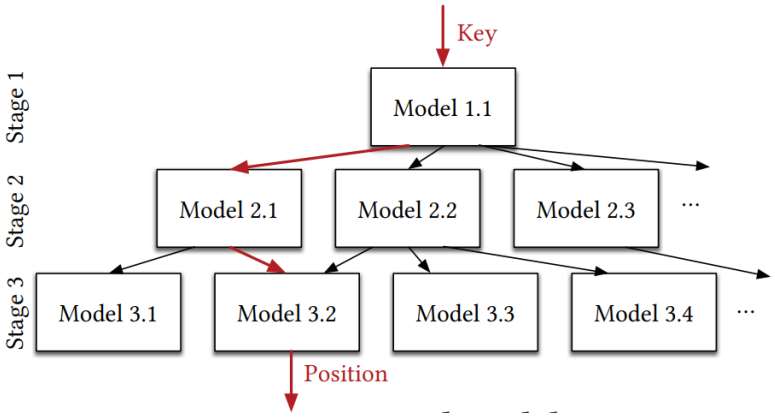


Figure 3.1.2: Recursive Model Index (Kraska et al. 2018).

3.1.1 ALEX

As mentioned in the introduction, learned indexes can provide a great benefit regarding search time and memory footprint. Some of the earlier work done by Kraska et al. 2018 presents a solution that beats B+Tree by a factor of up to three times in search time. When it comes to memory the learned index beats the B+Tree by an order of magnitude. One of the limitations of the original RMI structure is that it only supports static, read-only workloads. Later work done by Ding et al. 2020 proposed a new structure to solve this, called ALEX. ALEX builds on the same RMI structure and even improves the performance by up to 2.2x with an index size that is 15x smaller while simultaneously allowing for point lookups, short-range queries, inserts, updates, deletes, and bulk loading.

ALEX achieves the increase in performance and the possibility for updates by improving the storage layout, and search strategy, and by keeping models accurate with dynamic data distribution and workloads.

For the storage layout, ALEX builds a tree like a B+Tree but allows for different nodes to arbitrarily change the size and leaves extra space between the elements in the array. This gap gradually writes off the cost of shifting keys for every insertion by allocating the closest key to the right of the gap. ALEX also uses a method

for inserting each element as close to the predicted position as possible, called a model-based insert.

ALEX uses exponential search which has superior performance compared to binary search in this case because model-based inserts are used. This utilizes the prediction of the model and starts the search at the predicted position. When the search is performed on an accurate model the exponential search will have better performance than binary search.

Differing from the model proposed by Kraska et al. 2018, ALEX can dynamically change the shape of the model if the data distribution is changed by a large margin after index initialization. Adaptive expansion, node splitting, and, selective model retraining are the implemented methods for handling dynamical changes and are needed to ensure good performance for write workloads.

The goal of this model is to be faster than B+Tree and the previously proposed learned index regarding the lookup time. The model should at least have the same insert time as a B+Tree, and the memory usage should be less than a B+Tree and the RMI. The last goal of this model is to make the space used for data storage less than a dynamic B+Tree.

An illustration of the ALEX Design can be seen in Figure 3.1.3 and shows the different components mentioned in this section. The figure shows how the model is flexible and adaptive in regards to the model height and shape, how gapped arrays are used to allow for variable node sizes, and how exponential search is used from the prediction given by the model.

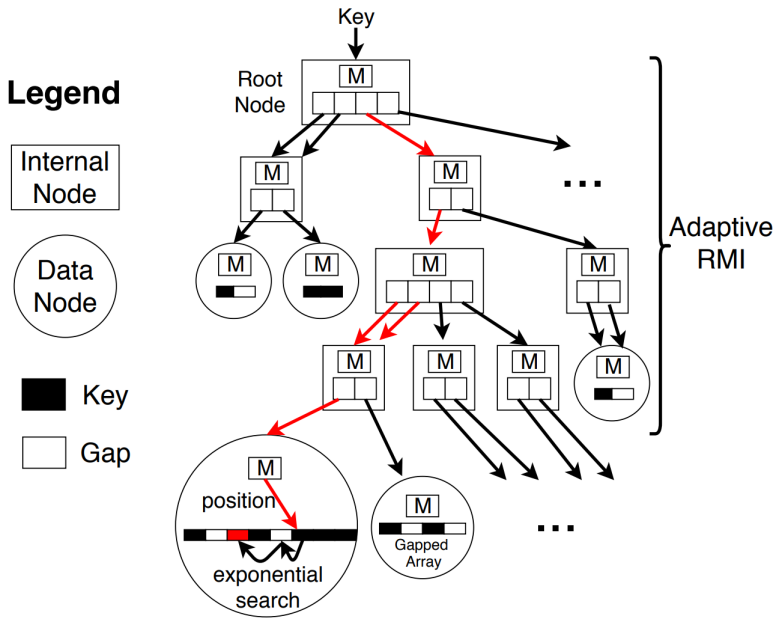


Figure 3.1.3: Alex Design (Leis, Kemper, and Neumann 2013).

3.1.2 Piecewise Geometric Model Index

The Piecewise Geometric Model Index (PGM-Index) is a learned index that supports predecessors, updates, and range searches. The PGM-Index was the work presented in Ferragina and Vinciguerra 2020 and focuses on solving the fully-dynamic indexable dictionary problem.

Like ALEX, the PGM-Index is flexible, and Ferragina and Vinciguerra 2020 propose three types of the PGM-Index. The third one is a *multicriteria* variant that can auto-tune itself to keep up with the space-time constraints, it can do this in a matter of seconds over hundreds of millions of keys.

The PGM-Index creates a linear approximation of the key distribution, as seen in Figure 3.1.4. One line segment in the linear approximation might not be able to represent all the keys within a given error margin, that's why it's called a *piece-wise* linear approximation model (PLA-Model), we can create multiple line segments that cover their portion of the key space.

Performance wise, the PGM-Index with only static setting, meaning no updates, only predecessor and range search, is able to match the query performance of of a cache-optimized static B+Tree. The PGM-Index with static setting beats the same B+Tree when it comes to memory, taking 83x less space.

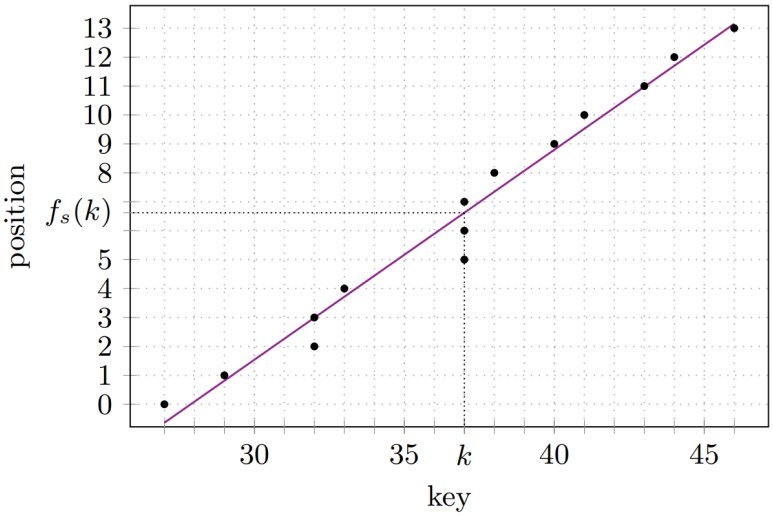


Figure 3.1.4: Linear Approximation of a key distribution (Ferragina and Vinciguerra 2020).

The linear approximation of a set of keys in a given range is shown in Figure 3.1.4. We can see that we need to search at least within 2 neighboring keys to be able to predict the positions of all keys since $key = 37$ is repeated 3 three times. So the function fs has an error of 2 ($\varepsilon = 2$).

The distribution of keys in the above example is exceptionally well distributed and it isn't very demanding to create a single linear approximation with reasonable error bound. But if the keys would have been more clustered and spread out we

would have had to split the linear approximation into multiple line segments that would represent each of their own key space.

The model would then use a hierarchical structure shown in Figure 3.1.5 to be able to predict the position. In the figure, the key $k = 76$ is being searched for. Starting at the top level, the root segment, we compute the position for the next level. At this level, we search for k within our error bounds. This is repeated until the level calculates the final position in our array, also within our error bounds, in this case, it is *+- 1*.

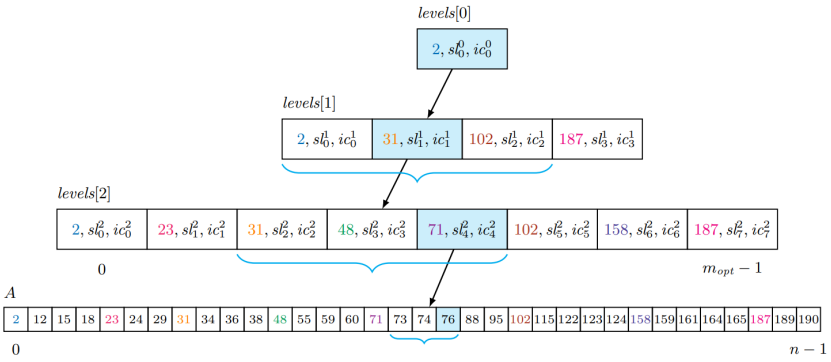


Figure 3.1.5: PGM Index Structure (Ferragina and Vinciguerra 2020).

3.1.3 SOSD Benchmark

The Search On Sorted Data (SOSD) benchmark is an open-source framework created to evaluate learned indexes on both synthetic and real-world datasets. The benchmark includes baseline implementations for comparison, with one of them being the Recursive Model Index (RMI). Notably, the SOSD Benchmark is renowned for having the only publicly available implementation of RMI among the components (Kipf et al. 2019).

Also, the SOSD Benchmark includes methods for generating synthetic datasets and lookup keys, which are instrumental in evaluating the query performance. Each of these datasets can be configured to include a range of 200 million to 800 million 32-bit or 64-bit unsigned integers.

3.1.4 Radix Spline

As mentioned in the introduction, earlier learned index models have shown us that there is a lot of potential when it comes to improving the lookup time of index structures. Even index structures that are highly tuned and read-optimized. The main problem comes when the model needs to be re-trained/re-balanced after an insert or update.

In earlier implementations of learned indexes, the building process had to do multiple passes over the data, this meant that the building process took a significant amount of time. Radix Spline, presented in Kipf et al. 2020 only needs a single

pass over the data and can execute the work per new element in constant time. And yet, it still manages to compete with the leading learned index structures like the Recursive Model Index when it comes to lookup performance and space utilization. But RMI does not support inserts and cannot be constructed in one pass over the data. When it comes to ALEX and the PGM-Index, they do support updates, but they have room for improvement when it comes to building performance.

Radix Spline (RS) assumes that there is no need to support single updates and that the single-pass training algorithm can be done together with the merge process in an LSM-Tree. Since the merge operation is already quite heavy, performance-wise, the time added from training the single-pass learned index would be insignificant. RS comprises of two components: a linear spline model that provides an approximation of the Cumulative Distribution Function (CDF) within a defined error bound, and a radix table that serves as an index for the computed spline points.

Building the Radix Spline takes two steps, which can be done in a single pass over the data. The first step is to fit a linear spline to a CDF based on the underlying data as seen in Figure 3.1.6. This ensures a certain error bound. From this, we get a set of spline points that are spread out on the CDF with an interpolation between them. The second step is to build a flat radix structure, this is called the radix table. The purpose of the radix table is to give an approximate index of the spline points. When both steps are done we can extract a certain radix prefix and use it as an offset in the radix table we just created. To configure RS, the accepted spline error and the radix table size can be adjusted. These parameters directly impact the size of the index and the time required for performing lookups.

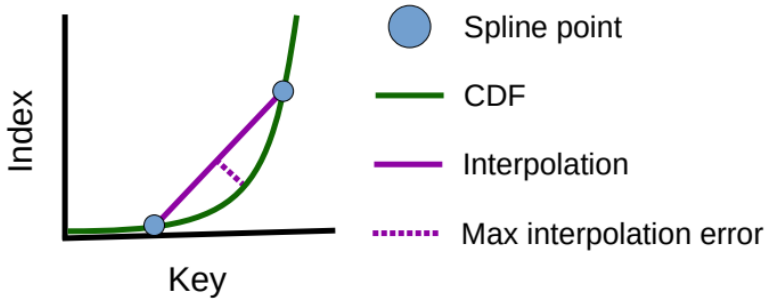


Figure 3.1.6: Spline Segment (Kipf et al. 2020).

When doing a lookup, the lookup key is used to extract an r-bit prefix. In the example shown in Figure 3.1.7, we get 101. This is then used to make an offset access in the radix table from the r-bit prefix. From this, the two pointers are retrieved and used to define a smaller search range. In the figure, the points are 5 and 6. Lastly, binary search is used to find the first occurrence of the key.

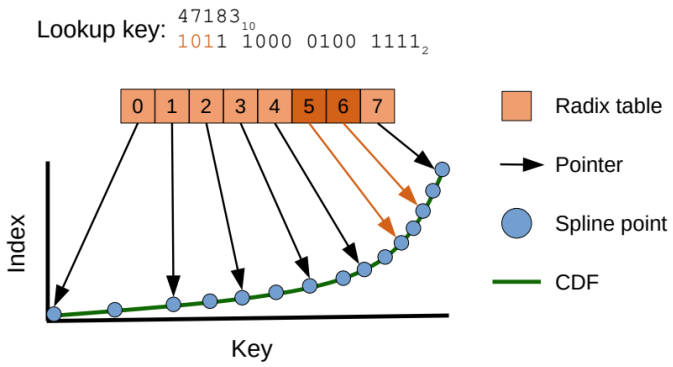


Figure 3.1.7: Spline lookup example (Kipf et al. 2020).

Kipf et al. 2020 proposes the ideal timing to apply the single-pass training algorithm, specifically during the merging process in an LSM-tree. This stage generates data in sorted order, making it optimal for building the index. As mentioned in Section 2.1.7, LSM-trees are well suited for scenarios with frequent updates, and incorporating radix spline with LSM-trees can have a significantly positive impact on the query performance.

3.1.4.1 Build Time, Lookup time, and Index Size

One effective method of illustrating the trade-off between lookup time and index size is by using a heatmap that depicts the various combinations of spline error and number of radix bits, along with their respective impacts on lookup time and index size. This approach was employed in the original radix spline paper, and the corresponding results are presented in Figure 3.1.8.

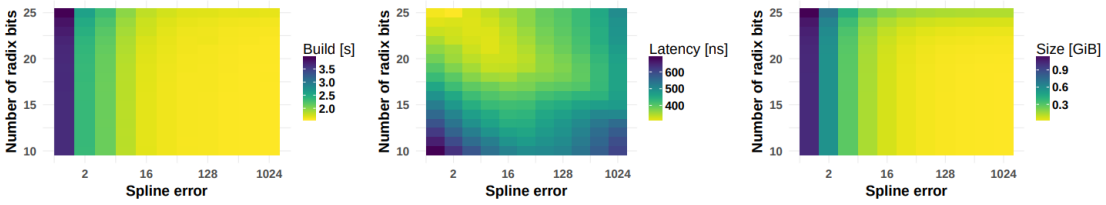


Figure 3.1.8: Build time, lookup latency, and index size for different configurations (Kipf et al. 2020).

When RS was evaluated using real-world datasets, a significant problem became apparent. The index size for the *face_200M_uint64* dataset occupied a considerably larger amount of space compared to the other datasets as seen in Figure 3.1.9. This outlier issue stands out as one of the primary drawbacks of the RS index. Consequently, each dataset requires a different configuration, and it appears that the *face_200M_uint64* dataset was not utilizing the ideal configuration. As stated, the lookup time can be traded for a smaller index size.

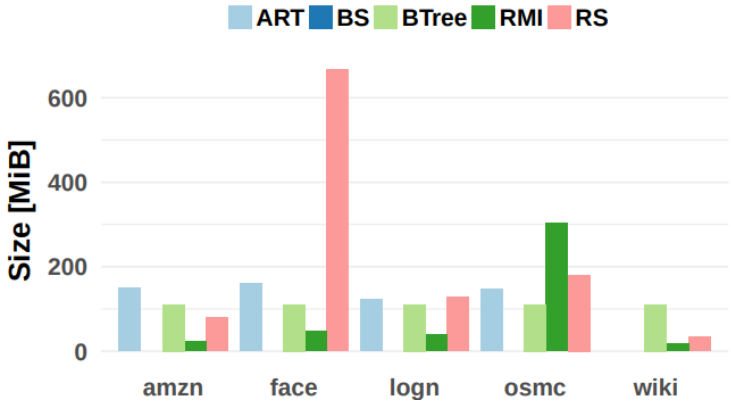


Figure 3.1.9: Index sizes for real world datasets with Radix Spline (Kipf et al. 2020).

3.1.5 Hist-Tree

Hist-Tree (HT) is an indexing method proposed by Crotty 2021 that approximates data distribution using a histogram instead of functions. It utilizes a fast radix tree-based traversal approach to locate the appropriate histogram bucket. Crotty also introduced a compact version of Hist-Tree called Compact Hist-Tree (CHT), which is optimized for read-only workloads. CHT acts as a lookup table, storing nodes and their prefix sums. Importantly, CHT can be built directly from HT or the data itself.

To conduct a lookup in HT, we first check if the key falls within the range covered by HT. If it lies outside this range, the lookup terminates. Next, the algorithm descends through HT by dividing the key with the bin width, calculating the bin index. The sum of all smaller bins is then computed. Subsequently, the new key is derived by subtracting the width multiplied by the bin index from the original key. This process continues until a terminal bin or leaf node is reached. A terminal bin is defined as a bin with a count below a configurable threshold. Once a leaf node or terminal bin is reached, the position is returned.

Let's consider an example from Crotty 2021, as shown in Figure 3.1.10. The HT in this example has a count threshold of 16, designating all nodes with a count lower than 16 as terminal bins, which are marked in grey. Suppose we want to perform a lookup for the key 567. Starting at the root node, we calculate the bin by dividing 567 by 256, yielding a result of 2. We then compute the sum of all smaller bins, which amounts to 114 (20 + 94). Next, we update the key to 55 by subtracting 256 multiplied by 2 from 567. This process continues until we reach a terminal or leaf node. In this specific example, we encounter a bin with a count of 6. Finally, we return the final sum of the counts, which totals 129 (114 + 5 + 3 + 7).

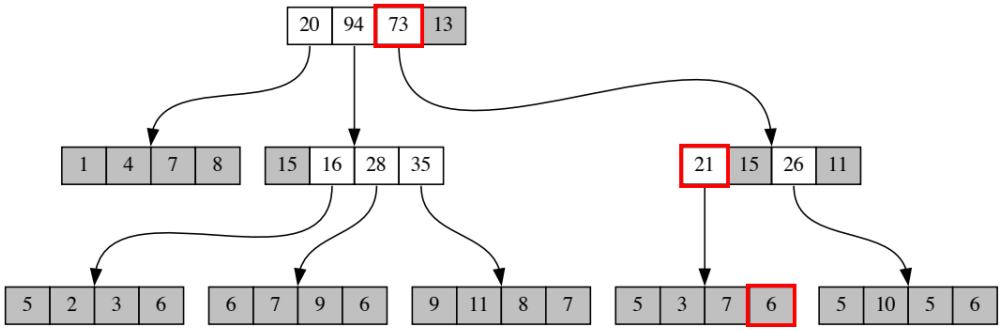


Figure 3.1.10: Hist-tree example (Crotty 2021).

3.1.6 PLEX

The Practical Learned Index (PLEX) is a specific type of learned index that combines elements from CHT and RS. It features a single hyperparameter, ϵ, which controls the trade-off between lookup time and index size. PLEX incorporates a spline layer inspired by radix spline, ensuring an error bound within ϵ, and a radix layer inspired by CHT. This combination results in rapid index construction, error-bounded lookups, and the ability to easily fine-tune the trade-off between lookup time and index size (Stoian et al. 2021).

First, we need to build the linear spline model S, which approximates the position of each key within a specified error bound of ϵ. This spline model ensures that the predicted position of the data remains within ϵ of the actual position in the CDF. The spline model is defined by a set of connected linear spline points, selected from the CDF points. Computing the optimal spline, which minimizes the number of spline points, can be achieved using dynamic programming in $O(N^2)$ time complexity, where N represents the size of the CDF. However, due to scalability concerns, a greedy algorithm is commonly employed in practice. Although this algorithm does not guarantee optimality, it can be implemented in linear time, denoted as $O(N)$.

Secondly, we need to build the CHT. The spline points are selected and indexed in CHT using a new implementation that builds each level level of the tree iteratively by analyzing key chunks. This method is different than the one used in HT, where bulkloading and creating a sparse tree were created first, here the CHT are constucted directly. CHT has two hyperparameters: the number of radix bits (r) for each tree node and the error parameter (δ), which represents the approximation of key positions within the index. If q^* represents the position of the spline point and $\tilde{q}$ denotes the position estimated by the CHT, the position of the spline point can be expressed as follows:

$$q^* \in \{q, \ldots, \tilde{q} + \delta - 1\} \tag{3.1}$$

One limitation of learned indexes has been the manual tuning required for hyperparameters. Consequently, multiple index constructions may be necessary to

identify the optimal configuration that provides the best lookup time while adhering to specific index size constraints. PLEX introduces cost models that enable the estimation of lookup times and accurate computation of space consumption for both the radix table and CHT. These models provide valuable insights without requiring the actual construction of the index. Consequently, the parameters can be selected based on the approximated lookup time and space consumption, allowing for the index to be built just once using the optimal configuration.

3.1.6.1 Radix Table Cost Model

The cost model for the radix table is determined by the parameter r, which divides the input data into 2^r buckets based on the most significant bits of the keys. The keys for the spline points serve as the input data for the radix table. When performing a key lookup, we need to identify the radix bucket b_k in which the key k is stored. Once the bucket is determined, a local search is conducted on the spline nodes within the bucket to locate the precise spline segment. This two-step process ensures efficient retrieval of the desired spline segment corresponding to the key being looked up. Using binary search, the number of steps equals $\lceil \log_2(|bk|) \rceil$, where the $|b_k|$ represents the number of spline points within the bucket b_k. We can then estimate the average lookup time as:

$$\lambda_r = \frac{1}{|D|} \sum_{k \in D} \lceil \log_2(|b_k|) \rceil \tag{3.2}$$

Assuming that only positive lookups are performed, it can be inferred that the key is located within the data set D. Additionally, this cost model has the capability to detect the outlier problem associated with radix spline, as discussed in Section 3.1.4, and can be estimated during the construction of the spline model. The time complexity of this model is represented as $O(r^+|S|)$, where r^+ denotes the maximum allowed r. Moreover, the memory consumption is proportional to $O(2^r)$.

3.1.6.2 CHT Cost Model

To estimate the lookup time and memory consumption for a large set of CHTs with different configurations (r, δ), we employ the same logic as the radix table cost model. The lookup process begins by determining the node v_k in which the key k is located. This corresponds to the node containing the bin that matches k with a size less than or equal to δ. If binary search is employed as the local search method, the number of steps required is given by the sum of the $depth(v_k)$ and $\lceil \log_2(\delta) \rceil$. Notably, $\lceil \log_2(\delta) \rceil$ remains constant for all lookup keys, hence necessitating the calculation of the average depth solely for the leaf nodes.

One limitation of the cost model for CHTs is that calculating the cost using the same method as the radix table cost model would require multiple examinations of the original data. However, this approach is not ideal. Instead, the cost model is simplified by calculating the average tree depth based on lookups from the set of spline keys. Consequently, this simplified cost model does not guarantee the

ability to predict the lookup time from the data itself. The average lookup time can be estimated as follows:

$$\lambda(r, \delta) = \lceil \log_2(\delta) \rceil + \frac{1}{|S|} \sum_{k \in S} \text{depth}(v_k) \tag{3.3}$$

3.1.6.3 Evaluation of PLEX

As stated in the original PLEX paper, PLEX successfully detects the outlier problem mentioned in Section 3.1.4 using the *face_200M_uint64* dataset while maintaining the performance of RS for other datasets. In comparison to ART (Leis, Kemper, and Neumann 2013), BTree, and PGM, PLEX does not suffer performance degradation as the index size increases; this holds true for almost all of the tested datasets. The primary reason for this is that navigating the index no longer saves time due to the significant size, making a binary search more efficient. Figure 3.1.11 illustrates the comparison of lookup time and index size between PLEX and other index structures, both learned and conventional, for four real-world 64-bit datasets.

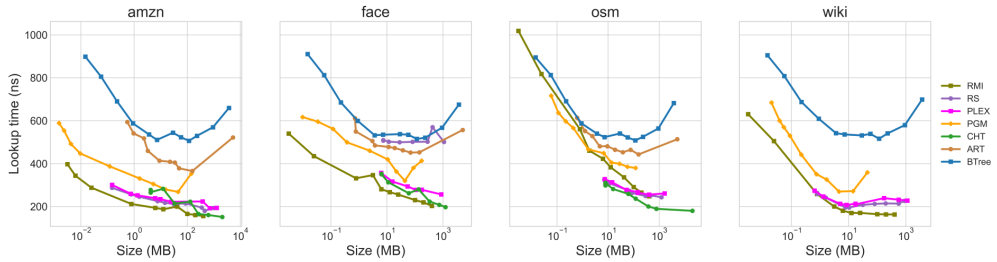


Figure 3.1.11: Size affecting lookup time with PLEX (Stoian et al. 2021).

CHAPTER

FOUR

SETUP AND METHOD

The accuracy of a neural network's predictions heavily relies on the formulation of features within each dataset, as well as the volume, diversity, and variability of the training data. The selection of appropriate features and the availability of sufficient data directly impact the performance of the model.

This chapter focuses on the setup and steps taken to develop a neural network model capable of predicting the hyperparameters to be used with radix spline for any given 32-bit dataset. We will delve into various aspects, including the generation of synthetic data, feature extraction, the identification of the optimal configuration for each dataset, and the subsequent training of the neural network.

4.1 Setup

All the necessary steps to complete the neural network model, including data generation, feature extraction, lookup generation, and index building, were performed on a single system, specifically Linux-5.14.0-1059-oem-x86 kernel with an 12th Gen Intel® Core™ i7-1270p CPU @ 4.80GHz × 16 CPU and 32GB of RAM. This choice of system was critical as variations in build and lookup times of the radix spline index structure could significantly impact the accuracy of the results if different systems were used.

The training datasets for the neural network were generated through random sampling with various distributions using NumPy (Numpy.org 2020). Each dataset comprised either 200 million or 400 million 32-bit unsigned integers. Lookup files containing 1 million keys were generated for each dataset in addition to the datasets themselves. The lookup files were generated using the SOSD benchmark (Kipf et al. 2019), an open-source C++ project designed for benchmarking learned indexes. It is worth mentioning that the core logic of the SOSD benchmark was not altered; instead, it was adopted and customized to meet the specific requirements of this project.

To predict the number of radix bits and the spline error for the Radix spline learned index, a multi-output neural network model was employed. The model,

implemented using TensorFlow (Abadi et al. 2016) in Python, takes four different features as input and generates two outputs. It is comprised of two layers, each containing 128 units, and utilizes the ReLU activation function. Stochastic gradient descent (SGD) is employed as the optimizer, while mean squared error (MSE) serves as the loss function, and root mean squared error (RMSE) is used as the evaluation metric. The model underwent training for 1000 epochs with a batch size of 10, enabling it to learn the relationships between the input features and the desired outputs.

4.1.1 Generating synthetic test data

The SOSD benchmark downloaded data from the Harvard Dataverse database and generated synthetic datasets with uniform, normal, and lognormal distributions. However, these datasets were not sufficient to train the neural network, as they lacked diversity and variability needed for training on a wide range of datasets.

Instead, NumPy was utilized to generate datasets based on 29 unique distributions, each representing a distinct data distribution. The aim was to investigate the capability of neural networks to effectively learn the optimal configuration for radix spline for any given dataset (**RQ1**), and during the data generation process almost all of the available NumPy distributions were utilized. In total, we generated 115 datasets, comprising over 200 GB of synthetic data. Following the methodology of the SOSD Benchmark, the datasets were generated as sorted arrays of uint32 keys and corresponding values, accurately reflecting the underlying data distribution. For a comprehensive list of the distributions and their respective probability density functions (PDFs) or probability mass functions (PMFs), please refer to Table 4.1.1. The specific parameters utilized to create these datasets are provided in Appendix 7.3.

Name	**Size**	**Parameters**	**PDF (PMF)**
Beta	200M	α, β	$\frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)}$
Chi-square	200M	k	$\frac{1}{2^{k/2}\Gamma(k/2)} x^{k/2-1} e^{-x/2}$
Dirichlet	400M	$\alpha_1, \ldots, \alpha_K$	$\frac{1}{B(\boldsymbol{\alpha})} \prod_{i=1}^{K} x_i^{\alpha_i - 1}$
F	200M	d_1, d_2	$\frac{\Gamma(\frac{d_1+d_2}{2})}{\Gamma(\frac{d_1}{2})\Gamma(\frac{d_2}{2})} \left(\frac{d_1}{d_2}\right)^{\frac{d_1}{2}} \frac{x^{\frac{d_1}{2}-1}}{\left(1+\frac{d_1}{d_2}x\right)^{\frac{d_1+d_2}{2}}}$
Gamma	200M	k, θ	$\frac{1}{\Gamma(k)\theta^k} x^{k-1} e^{-x/\theta}$
Gumbel	200M	μ, β	$\frac{1}{\beta} e^{-(z+e^{-z})}$, where $z = \frac{x-\mu}{\beta}$
Laplace	200M	μ, b	$\frac{1}{2b} e^{-\frac{\lvert x-\mu \rvert}{b}}$
Logistic	200M	μ, s	$\frac{e^{-(x-\mu)/s}}{s(1+e^{-(x-\mu)/s})^2}$
Multivariate Normal	400M	μ, Σ	$(2\pi)^{-\frac{k}{2}} \lvert\boldsymbol{\Sigma}\rvert^{-\frac{1}{2}} e^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}$
Negative Binomial	200M	r, p	$\binom{x+r-1}{x}(1-p)^x p^r$

Noncentral Chi-square	200M	k, λ	$\frac{1}{2}e^{-(x+\lambda)/2}(\frac{x}{\lambda})^{\frac{k-2}{4}}I_{k/2-1}(\sqrt{\lambda x})$, where $I_{k/2-1}$ is the modified Bessel function
Noncentral F	200M	d_1, d_2, λ	$\sum_{j=0}^{\infty}(\frac{\lambda}{2})^j \frac{e^{-\frac{\lambda}{2}}}{j!} f_{F_{d_1,d_2}}(x)$, where $f_{F_{d_1,d_2}}(x)$ is the PDF of the central F distribution with degrees of freedom d_1 and d_2
Pareto	200M	x_m, α	$\alpha\frac{x_m^\alpha}{x^{\alpha+1}}$
Rayleigh	200M	σ	$\frac{x}{\sigma^2}e^{-\frac{x^2}{2\sigma^2}}$
Standard Cauchy	200M	-	$\frac{1}{\pi(1+x^2)}$
Standard Exponential	200M	-	e^{-x}
Standard Gamma	200M	-	$\frac{1}{\Gamma(k)}x^{k-1}e^{-x}$
Standard Normal	200M	-	$\frac{1}{\sqrt{2\pi}}e^{-\frac{x^2}{2}}$
Standard t	200M	ν	$\frac{\Gamma(\frac{\nu+1}{2})}{\sqrt{\nu\pi}\Gamma(\frac{\nu}{2})}\left(1+\frac{x^2}{\nu}\right)^{-\frac{\nu+1}{2}}$
Triangular	200M	a, b, c	$\begin{cases}\frac{2(x-a)}{(b-a)(c-a)} & \text{for } a \le x < c,\\ \frac{2(b-x)}{(b-a)(b-c)} & \text{for } c \le x \le b,\\ 0 & \text{otherwise.}\end{cases}$
Von Mises	200M	μ, κ	$\frac{1}{2\pi I_0(\kappa)}e^{\kappa\cos(x-\mu)}$, where $I_0(\kappa)$ is the modified Bessel function of order 0
Wald	200M	μ, λ	$\left(\frac{\lambda}{2\pi x^3}\right)^{1/2} e^{-\frac{\lambda(x-\mu)^2}{2\mu^2 x}}$
Weibull	200M	k, λ	$\frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}$
Zipf	200M	s	$\frac{x^{-s}}{\zeta(s)}$, where $\zeta(s)$ is the Riemann zeta function
Uniform	200M	a, b	$\frac{1}{b-a}$
Exponential	200M	λ	$\lambda e^{-\lambda x}$
Lognormal	200M	μ, σ	$\frac{1}{x\sigma\sqrt{2\pi}}e^{-\frac{(\ln x-\mu)^2}{2\sigma^2}}$
Normal	200M	μ, σ	$\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
Power	200M	a, k	$\frac{k}{a}(x/a)^{k-1}(1-(x/a))^k$

Table 4.1.1: All the distributions used to create the synthetic data. (Numpy.org 2020)

The values of the parameters for each distribution were selected based on the constraints outlined in NumPy's documentation. Additionally, the parameters were chosen with the aim of representing the unique characteristics of each distribution. In other words, the parameters were carefully selected to generate noticeably dif-

ferent datasets for the same distribution. This approach was beneficial in ensuring the generation of a sufficient number of diverse datasets.

For the four datasets that did not require parameterization, the standard datasets. These datasets are the same as the non-standard distribution but with a constant parameter. The standard datasets were also generated five times, with the exception of the *standard exponential* dataset. Although the standard datasets exhibited similarities, they still possessed varying features, which added value to the test set.

Each probability distribution can be visualized with a histogram, which represents the distribution of random samples generated from the respective distribution. The histogram displays the frequencies of values falling into intervals along the x-axis, providing a clear representation of the data distribution. This visualization tool was utilized to select appropriate parameters for generating multiple datasets from a single probability distribution.

Figure 4.1.1 presents an example, showcasing the histograms for three common probability distributions. These histograms offer insights into the shape and characteristics of the distributions. Additionally, Figure 4.1.2 illustrates the CDFs of the same distributions.

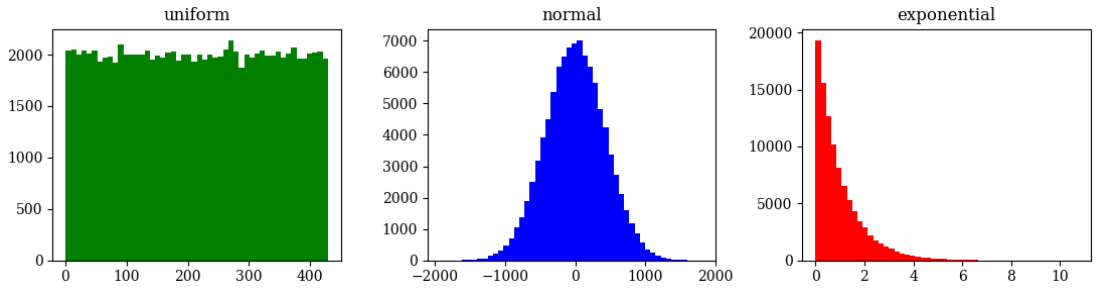
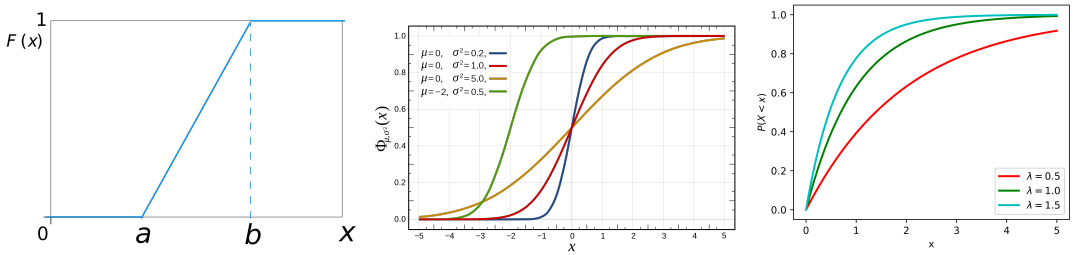


Figure 4.1.1: Graphs depicting three common probability distributions, each consisting of 100,000 elements.



(a) CDF Uniform (Ikamusume-Fan n.d.).

(b) CDF Normal (Inductiveload n.d.).

(c) CDF Exponential (EvgSkv n.d.).

Figure 4.1.2: CDF for three common distributions

4.1.2 Feature extraction

The next step in gathering test data for the neural network is feature extraction, which involves identifying features that have a direct correlation with the index configuration and affect the building of the Radix Spline index. As talked about in Section 2.1.3.1, feature extraction can either be done manually or automatic. In this experiment the features were extracted using manual feature extraction approach.

To ensure the relevance of features and their direct correlation with the optimal configuration for each dataset, a data-driven approach was employed. This approach aimed to identify features that exhibit a direct correlation with the optimal configuration. One of the most common types of measures used to describe a distribution is related to the central tendency, such as mean, median, and mode. These measures provide information about the central or average value of the distribution. Another important measure is kurtosis, which quantifies the heaviness of the distribution's tail. All of these measures were carefully considered when creating the features for each dataset.

To ensure a direct correlation between these measures and the optimal configuration, two 64-bit datasets with distinct distribution complexities were utilized. The first dataset, *fb_200M_uint64*, was specifically chosen due to the outlier problem with RS, providing a unique testing scenario. The second dataset, also a real-world dataset, *wiki_ts_200M_unit64*.

In evaluating the central tendency of each dataset, the number of unique key intervals was employed as a measure. This metric signifies the distinct spaces between keys and provides valuable insights into the spread of the keys within the dataset.

The number of unique key intervals was collected for each dataset, and then the datasets were constructed using 56 different configurations to determine the configuration that yielded the optimal lookup time and index size. Table 4.1.2 presents the variations in features between the two datasets.

Dataset	**Number of unique key intervals**	**Radix bits**	**Spline error**
fb_200M_uint64	76582	10	256
wiki_ts_200M_uint64	4304	16	32

Table 4.1.2: Comparison of number of unique key intervals between two dataset from the Harvard Dataverse database (Harvard n.d.).

Table 4.1.2 could indicates that the spread and variability of the key/value pairs significantly impact the optimal configuration choice. These findings highlight the importance of creating features that reflect these characteristics. Building on this assumption, four features were defined:

1. **Size**
   This metric, also known as the number of key-value pairs, measures the

quantity of data in the dataset. For instance, datasets for this application typically contain either 200M or 400M key-value pairs.

2. **Mean key interval frequency**
   This feature calculates the average frequency of unique intervals between keys and provides a good indication of the Kurtosis of the distribution.

3. **Number of unique key intervals**
   The unique interval count feature calculates the number of distinct spacings between adjacent items in the dataset, which can offer insights into the underlying structure and patterns of the data.

4. **Mode of unique key intervals**
   This feature calculates the highest frequency of repeated unique spacings, which can provide an indication of the kurtosis of the data. A low value indicates that the data is more spread out, while a higher value suggests a greater degree of peakedness.

Modification to the Radix Spline index builder provided in the PLEX project code were implemented to gather features for all 115 datasets. This modification involves an extra pass through the data before building the index, which allows for collecting the necessary information about the distribution of the data. As a result, the implementation of the Radix Spline index becomes a dual-pass learned index, rather than a single-pass learned index as in the original implementation.

4.1.3 Building the indexes

To achieve a high accuracy when testing the performance of each of the datasets using the radix spline learned index, each dataset got tested using 56 different combination of the two hyperparameters used to trade lookup time with index size. These configurations consist of a combination of values for the radix bits parameter (10, 12, 14, 16, 18, 20, 22, and 24) and values for the spline error parameter (4, 8, 16, 32, 64, 128, and 256).

We limited the number of radix bit configs to 8 and spline error configs to 7 in consideration of the long build times and the scope of the project. As demonstrated in the Radix Spline paper, the optimal configurations frequently fall within this range, and values exceeding a spline error of 256 or more than 24 radix bits are typically only required in exceptional circumstances.

4.1.4 Finding the Best and Worst Parameter Configuration

After building 56 indexes with various configurations for each of the 115 datasets, all the necessary features, lookup times, and index sizes were obtained. To create a solution for our training set, we need to determine the optimal configuration. This will enable us to identify the best-performing combination of parameters and build an effective model.

4.1.4.1 Method 1

Two approaches were employed to extract the best configuration for each dataset. One approach involved analyzing all configurations and results using a script. This script identified the configuration with the smallest index size among the top 10% of lookup times. To express this mathematically, the equation below can be used.

$$c_{\min} = \text{argmin}_{c \in C'}\{M(c)\} \tag{4.1}$$

where C' is the set of configurations corresponding to the best 10% of configurations sorted by their lookup time.

4.1.4.2 Method 2

The z-score normalization technique was used as another method to rank the pairs based on their performance. In this approach, each pair was rescaled based on the standard deviation from the mean of all pairs, and scores were computed for each pair. This approach takes into consideration the performance of other pairs and may favor different configurations than the first approach. The equation below was utilized to compute the z-score normalization of the pairs.

$$z_{i,j} = \frac{x_{i,j} - \mu_j}{\sigma_j} \tag{4.2}$$

where $z_{i,j}$ is the z-score normalized value for the ith pair and jth variable, $x_{i,j}$ is the original value for the ith pair and jth variable, μ_j is the mean of the jth variable across all pairs, and σ_j is the standard deviation of the jth variable across all pairs.

4.1.4.3 Finding the Worst Parameter Configuration

To specifically address **RQ2**, the z-score normalization technique was employed to not only identify the optimal parameter configuration for each dataset but also to determine the worst configuration that could be achieved with RS. It is worth noting that the worst configuration is simply the combination of the highest lookup time and the largest index size, without considering any constraints. By doing this, we can directly compare the predicted configuration with the worst possible configuration.

4.1.5 Training the network

As discussed in Section 4.1, our neural network model consists of two layers with 128 units each, and uses the ReLU activation function. The model was trained on our training data for 1000 epochs, with a batch size of 10.

To evaluate the performance of the model, a separate test set was created using datasets downloaded from the Harvard Dataverse database (Harvard n.d.).

Books_200M_uint32 dataset, which is based on real-world data, was included in the test set together with some synthetic generated datasets. Additionally, 20% of the training data was selected and excluded from the training set to increase the size of the test set and provide a more comprehensive evaluation.

Because the data was generated using two different methods to find the optimal configuration, we trained and evaluated the model for each of these methods. In addition, due to the potential for significant variation in results and accuracy when training a neural network with limited data, each method was run up to five times. As a result, we obtained an average accuracy by training and evaluating the network five times for each method.

CHAPTER

FIVE

RESULTS AND EVALUATION

This chapter presents the results obtained through the different steps of the method. Specifically, the outcomes from the feature extraction, the approaches used to find the optimal parameter configuration, and the results from training the neural network are discussed.

5.1 Feature Extraction

As described in Section 4.1.2, each dataset was defined by four distinct features: size (number of key/value pairs), mean key interval frequency (MKIF), number of unique key intervals (NUKI), and mode of unique key intervals (MUKI). Table 5.1.1 shows the results of the feature extraction for one configuration of each dataset, while a comprehensive overview of the features for all 115 datasets is provided in Appendix 7.3.

Dataset	**Size**	**MKIF**	**NUKI**	**MUKI**
Beta	200M	114351	1749	104866000
Chi-square	200M	8251.85	24237	80608500
Dirichlet	400M	923788	433	44888100
F	200M	7939.03	25192	51216700
Gamma	200M	8230.45	24300	28424200
Gumbel	200M	7816.16	25588	38848800
Laplace	200M	7427.76	26926	50963300
Logistic	200M	7612.38	26273	43079000
Multivariate Normal	400M	16780.6	23837	12805200
Negative Binomial	200M	25000000	8	200000000
noncentral Chisquare	200M	8360.85	23921	25823900
noncentral F	200M	9134.51	21895	86893600
Pareto	200M	9932.46	20136	134549000
Rayleigh	200M	9143.69	21873	23272900
Standard Cauchy	200M	114482	1747	199933000
Standard Exponential	200M	7455.73	26825	51046300
Standard Gamma	200M	7693.79	25995	48499000
Standard Normal	200M	8427.79	23731	25700500

Standard T	200M	15584.8	12833	178669000
Triangular	200M	32867.7	6085	11589500
Vonmises	200M	11578.1	17274	38329800
Wald	200M	8322.24	24032	32126900
Weibull	200M	8050.56	24843	28229200
Zipf	200M	216685	923	199980000
Uniform	200M	573066	349	8893560
Exponential	200M	7517.67	26604	51628700
Lognormal	200M	9509.77	21031	142111000
Normal	200M	8193.36	24410	24806900
Power	200M	557103	359	8889000

Table 5.1.1: Features for one dataset per distribution

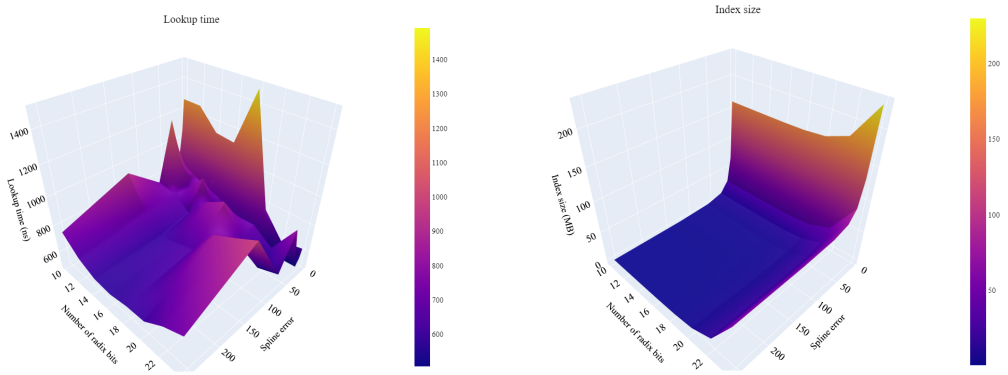
Although Table 5.1.1 only shows one configuration for each dataset, notable variations in the features of each dataset are already evident.

5.2 Finding The Optimal Parameter Configuration

After constructing each of the indexes with 56 different configurations, we obtained pairs of lookup times and index size in bytes. To determine the optimal configuration that provides the best balance between index size and lookup time, we used two approaches discussed in Section 4.1.4.

The utilization of two distinct methods to determine the optimal combination of spline error and number of radix bits stems from the specific requirements of the index. By employing these two different approaches, we are able to assess the model's performance under configurations that address diverse needs, such as index size. While z-score normalization can be considered the more accurate method for scoring each configuration pair, the particular application in which these configurations are employed might necessitate a weighted scoring system that prioritizes lookup time or index size over implementation aspects.

Figure 5.2.1 offer a comprehensive 3D visual representation of the influence of different configurations on both index size and lookup time for the *books_200M_uint32* dataset. Notably, only a limited number of combinations result in the top 10% worst index size and lookup time, this provides a wide range of alternative configurations for applications with specific memory constraints.



(a) Lookup time (b) Index size

Figure 5.2.1: 3D visual representation of lookup time and index size for different configurations

5.2.1 Minimum index for fastest 10% lookups (Method 1)

The results of the first method to determine the optimal parameter configuration are presented in Table 5.2.1. Please note that only one configuration is displayed for each dataset. The complete list of best configurations for all 115 datasets is included in Appendix 7.3.

Dataset	Number of Radix Bits	Spline Error
beta	18	64
chisquare	18	64
dirichlet	16	64
f	16	128
gamma	16	32
gumbel	16	64
laplace	18	64
logistic	16	32
multivariate_normal	18	128
negative_binomial	12	4
noncentral_chisquare	16	32
noncentral_f	18	32
pareto	20	32
rayleigh	16	64
standard_cauchy	24	256
standard_exponential	18	32
standard_gamma	18	32
standard_normal	18	64
standard_t	20	64
triangular	14	32
vonmises	16	32
wald	16	32
weibull	16	32

zipf	20	4
uniform	14	32
exponential	18	256
lognormal	20	32
normal	16	32
power	14	32

Table 5.2.1: Table displaying optimal configuration using the minimum index size for fastest 10% lookup times

5.2.2 Z-Score Normalization (Method 2)

Table 5.2.2 presents the results of the z-score normalization used to determine the optimal parameter configuration. Please note that only one configuration is displayed per dataset, but the complete list of best configurations for all 115 datasets is included in Appendix 1. It is worth noting that the optimal parameter configuration is markedly different when using z-score normalization than when using minimum index size for fastest 10% lookup times.

Dataset	**Number of Radix Bits**	**Spline Error**
beta	10	32
chisquare	10	64
dirichlet	10	8
f	10	16
gamma	12	8
gumbel	10	32
laplace	10	8
logistic	10	8
multivariate_normal	10	128
negative_binomial	10	128
noncentral_chisquare	20	32
noncentral_f	12	256
pareto	12	16
rayleigh	10	8
standard_cauchy	10	8
standard_exponential	10	16
standard_gamma	10	64
standard_normal	16	8
standard_t	10	64
triangular	14	8
vonmises	10	8
wald	10	32
weibull	10	128
zipf	10	8
uniform	10	8
exponential	10	8

lognormal	10	16
normal	12	64
power	10	4

Table 5.2.2: Table displaying optimal configuration using the z-score normalization method

5.3 Neural Network Prediction of Parameters

The neural network was trained using the data resulting from feature extraction and the optimal parameters related to lookup time and index size. Subsequently, the model was evaluated using a set of 25 synthetic datasets generated from various distributions, as well as one real-world dataset.

5.3.1 Method 1

Figure 5.3.1 displays the results obtained from predicting the optimal parameters using the neural network. The model used in this prediction was trained and evaluated with the parameters selected based on the minimum index size for the fastest 10% lookup times. The terms *Num* and *Err* signify the optimal number of radix bits and the corresponding spline error for each dataset. On the other hand, *pNum* and *pErr* represent the predicted values generated by the model, note that these values are rounded to the closes feasible values. The non-rounded-rounded predictions can be found in Appendix 7.3. Additionally, *wNum* and *wErr* denote the worst configurations selected by the z-score normalization method for each dataset. These particular configurations are regarded as the optimal choices for each dataset, according to the criteria established by Method 1.

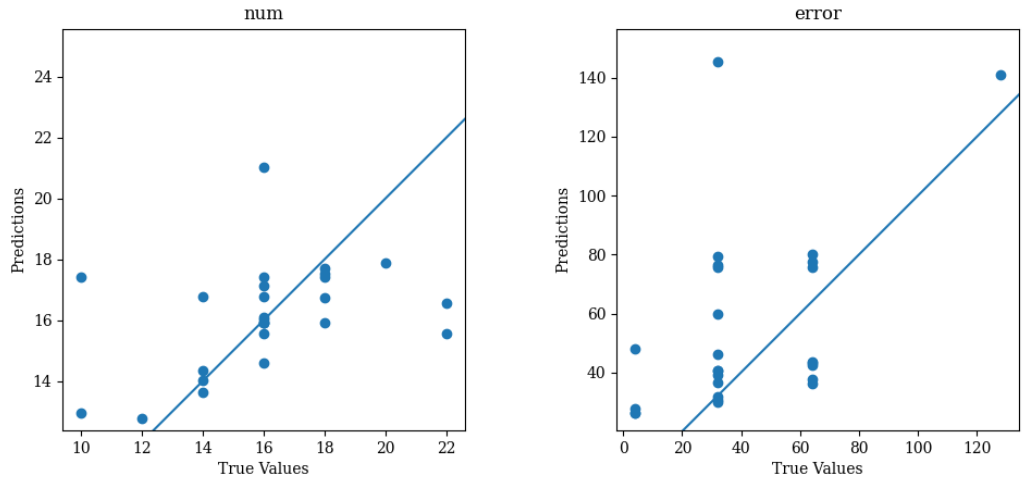
It is important to note that each of the predicted configurations has been rounded to the nearest value for both the number of radix bits and spline error. The table containing the precise predicted values can be found in Appendix 7.3.

Dataset	**Num**	**Err**	**pNum**	**pErr**	**wNum**	**wErr**
books	22	64	16	64	18	4
lognormal	16	4	22	32	10	128
normal	10	4	18	32	24	256
uniform_sparse	14	32	14	32	10	4
beta	16	32	16	64	10	4
chisquare	16	64	16	32	10	4
dirichlet	16	128	16	128	10	4
f	18	64	18	32	24	4
gamma	18	64	16	32	12	4
gumbel	14	64	16	32	12	4
laplace	16	64	18	64	10	4
logistic	16	32	18	32	10	4
negative_binomial	10	4	12	32	24	256

noncentral_f	18	32	16	64	10	4
pareto	18	64	18	64	10	4
rayleigh	16	32	16	32	10	4
standard_cauchy	22	32	16	128	10	8
standard_gamma	16	32	16	32	12	4
standard_normal	16	64	16	32	10	4
standard_t	16	32	16	32	10	4
triangular	16	32	14	32	10	4
vonmises	14	32	14	32	14	4
wald	20	32	18	64	10	4
weibull	18	32	18	64	10	4
zipf	12	4	12	32	10	128
power	14	32	14	32	10	4

Table 5.3.1: Table showing the predicted values, best config, and worst config

Pyplot was used to create a graph (see Figure 5.3.1) that illustrates the model's predictions. The graph depicts the optimal configuration as a line and the predicted configurations as dots, for both the number of radix bits and spline error.



(a) Optimal Number of Radix Bits Predicted by the Model

(b) Optimal Spline Error Predicted by the Model

Figure 5.3.1: Graphs Showing the Models Prediction

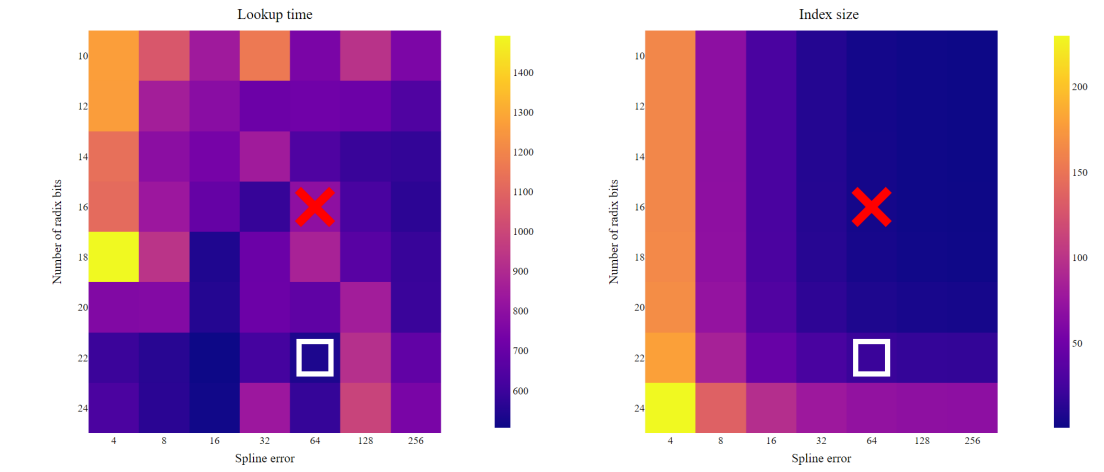
5.3.1.1 Prediction of a real-world dataset

Out of all the datasets employed to assess the model's prediction capabilities, only one was derived from real-world data. The inclusion of this real-world dataset provides insights into the model's performance in a more realistic scenario.

The predicted values for the *books_200M_uint32* dataset using Method 1 were **16** for the number of radix bits and **64** for the spline error. In contrast, the actual optimal configuration for this dataset was **22** for the number of radix bits and

16 for the spline error. Hence, the model successfully predicted the correct spline error but slightly underestimated the optimal number of radix bits.

To offer a comprehensive visual representation of the predicted and actual optimal configurations for the *books_200M_uint32* dataset, two heatmaps are employed. These heatmaps provide a holistic view of the complete range of configurations, encompassing lookup times and index sizes. The best configuration is highlighted with a white square, while the predicted configuration generated by our model is indicated by a red cross for both lookup times and index size. Figure 5.3.2 showcases the heatmap specifically associated with this dataset.



(a) Lookup time (b) Index size

Figure 5.3.2: Heat map showing the optimal configuration (white square) and the predicted configuration (red cross)

5.3.1.2 Evaluation of Method 1

Upon examining Figure 5.3.1, the prediction plot reveals a modest level of agreement between the model's predicted configuration and the actual optimal configuration, indicating the model's ability to capture some essential characteristics of the datasets. However, it is worth noting that only a subset of the predicted values directly align with the actual optimal configuration of the dataset. This observation highlights the model's inadequate accuracy in reliably predicting the optimal configuration for any given dataset (**RQ1**).

As noted in Section 4.1.5, the accuracy of the model varied when it was run multiple times. Because of this, we trained the model five times and calculated the average accuracy. The results displayed in Table 5.3.1 and Figure 5.3.1 only show the results from the first training and evaluation. Figure 5.3.3 presents the RMSE and average RMSE obtained after running the model five times for both *num* and *err*.

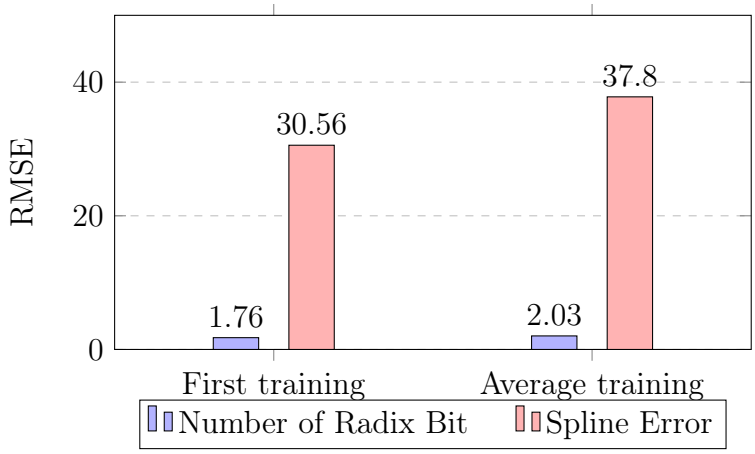


Figure 5.3.3: Graph illustrating the comparison between the RMSE obtained from a single run of the model versus the average RMSE derived from running the model five times. The comparison is performed for both *Num* and *Err*.

5.3.2 Method 2

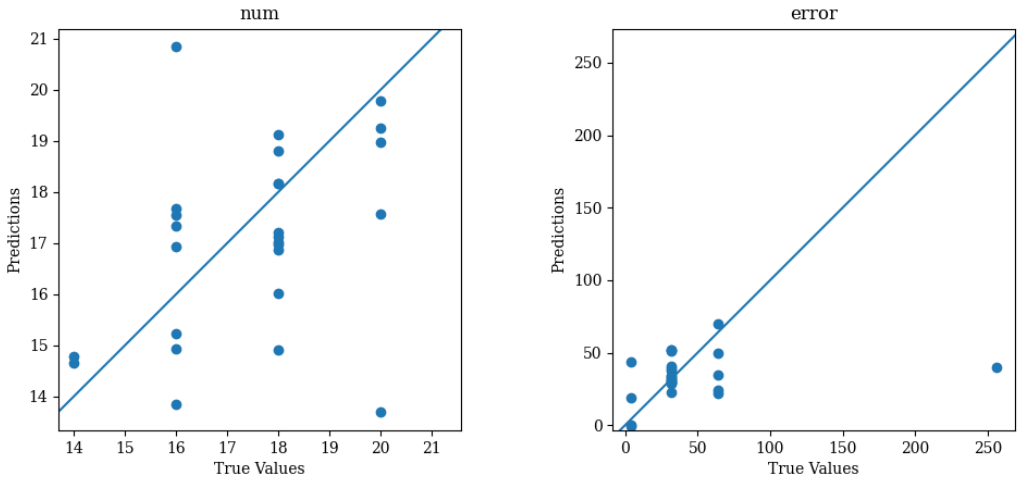
In relation to Method 1, the optimal configuration for the number of radix bits and spline error can be observed in Table 5.3.2, represented as *Num* and *Err*, respectively. The predicted configurations are rounded to the nearest feasible values and indicated as *pNum* and *pErr*. The non-rounded predictions can be found in 7.3. Furthermore, the worst possible configurations are presented as *wNum* and *wErr*. In line with Method 1, z-score normalization was employed to identify the configuration with the highest score among the worst configurations.

Dataset	**Num**	**Err**	**pNum**	**pErr**	**wNum**	**wErr**
books	16	256	16	32	18	4
lognormal	16	4	20	16	10	128
normal	16	4	18	32	24	256
uniform_sparse	14	32	14	32	10	4
beta	18	32	16	16	10	4
chisquare	18	32	16	32	10	4
dirichlet	20	64	18	64	10	4
f	20	64	18	32	24	4
gamma	16	64	16	32	12	4
gumbel	16	32	18	32	12	4
laplace	18	32	18	64	10	4
logistic	18	32	18	32	10	4
negative_binomial	16	4	14	4	24	256
noncentral_f	18	32	18	32	10	4
pareto	18	64	18	64	10	4
rayleigh	18	32	16	32	10	4
standard_cauchy	20	64	20	16	10	8
standard_gamma	18	32	18	32	12	4
standard_normal	16	32	18	32	10	4
standard_t	18	32	18	32	10	4

triangular	16	32	14	32	10	4
vonmises	14	32	14	32	14	4
wald	20	32	20	64	10	4
weibull	18	32	20	64	10	4
zipf	20	4	14	4	10	128
power	18	32	14	32	10	4

Table 5.3.2: Table showing the predicted values, best config, and worst config

Figure 5.3.4 displays prediction plots for number of radix bits and spline error. Here, the optimal configuration is shown as a line and the predicted configuration as dots for both.

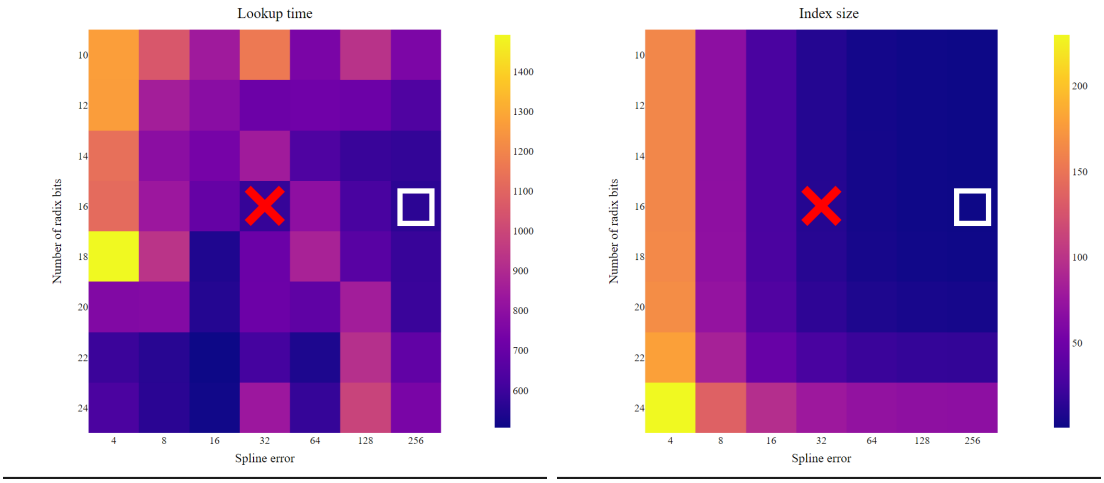


(a) Optimal Number of Radix Bits Predicted by the Model

(b) Optimal Spline Error Predicted by the Model

Figure 5.3.4: Graphs Showing the Models Prediction

To better visualize the results, a heatmap will be used to display all possible configurations in terms of their lookup times and index sizes. The heatmap will highlight the best configuration as a white square and the predicted configuration generated by our model as a red cross. These results are specific to the real-world dataset mentioned earlier, namely *books_200M_uint32*. Figure 5.3.5 shows the heatmap.



(a) Lookup time (b) Index size

Figure 5.3.5: Heat map showing the optimal configuration (white square) and the predicted configuration (red cross)

5.3.2.1 Evaluation of Method 2

The results obtained from Method 2 demonstrated a comparable level of prediction accuracy to Method 1, revealing a degree of agreement between the predicted values and the actual values. However, it should be noted that also for Method 2, only a subset of the predicted values directly correspond to the actual optimal configuration of the dataset. Based on this, it becomes evident that the model's accuracy is inadequate in reliably predicting the optimal configuration for any given dataset (**RQ1**).

In a similar manner to Method 1, the neural network model was trained and evaluated five times to calculate the average RMSE for both the number of radix bits and spline error. However, it is important to note that only the results from the initial training and evaluation are displayed in Table 5.3.2 and Figure 5.3.1. To provide a comprehensive overview, Figure 5.3.6 presents the RMSE and average RMSE obtained after running the model five times for both *num* and *err*.

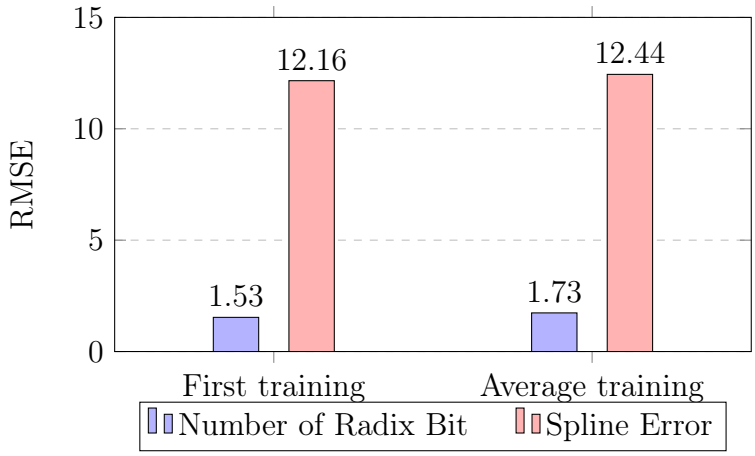


Figure 5.3.6: Graph illustrating the comparison between the RMSE obtained from a single run of the model versus the average RMSE derived from running the model five times. The comparison is performed for both *Num* and *Err*.

In contrast to Method 1, the utilization of z-score normalization enables the assignment of a score to each configuration based on its resulting lookup time and index size. Consequently, all potential configurations can be sorted according to their scores and compared to the predicted configuration, providing a comprehensive visual representation of how closely the prediction aligns with the optimal configuration. This approach allows us to determine how close the predicted configurations are to the optimal configuration for each dataset. Figure 5.3.7 presents the datasets in the same order as presented in Table 5.3.2 on the x-axis. The graph demonstrates that, while the correct optimal configuration was not predicted for several datasets, the predictions still yielded highly acceptable configurations far from the worst-case scenarios. For instance, considering the *books* dataset, represented as the first dataset in the graph, the predicted configuration ranked in the top **87.5%**. Furthermore, none of the predicted configurations fell below the bottom **60%**, with the lowest percentages observed for the *lognormal* and *dirichlet* datasets at **60.7%** and **62.5%**, respectively. This observation suggests that the model effectively captures the inherent pattern between the selected features and the optimal configuration, demonstrating its satisfactory learning capability.

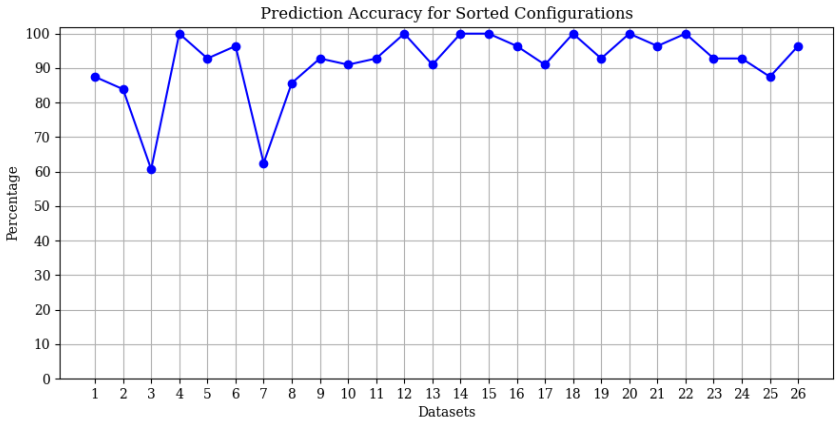


Figure 5.3.7: Prediction accuracy for sorted configurations

5.3.3 Overall Evaluation of the Model

One of the most noteworthy aspects of the predictions is the outliers. Some predictions are way off the true value. For Method 1, the model predicted a *num* of **21.007301**, while the actual optimal *num* was **16** for the *lognormal* dataset. Regarding the *err*, we also observe a more dramatic prediction for the *standard_cauchy* dataset, where the model predicted a spline error of **145.31194**, even though the true value was **32**.

It is possible that the issues with the outliers are attributed to the limited training data available for datasets with unconventional optimal configurations. Upon reviewing the training data using Method 1, it becomes apparent that only two datasets possess an optimal number of radix bits equivalent to **22**. Consequently, the model is less exposed to this specific scenario, which could explain the lower prediction accuracy observed for the *lognormal* dataset. This will be further explored and discussed in Section 6.2.

Table 5.3.1 and 5.3.2 display the worst possible configuration for each of the datasets. This inclusion provides a clearer understanding of how the predicted configuration compares to the worst possible configuration (**RQ2**). It is noteworthy that among all the datasets tested, none of them received a predicted best configuration that matched the worst possible configuration. Furthermore, in the results obtained using Method 1, the predicted configurations exhibited an average discrepancy of **5.8461** and **64.1538** for the predicted *num* and *err*, respectively. For the results stemming from using Method 2 the discrepancy between the predicted and worst configuration were **6.5384** and **55.8461** for the predicted *num* and *err*, respectively.

This can be a good indication that, on average, the predictions deviate considerably from the worst possible configuration. The notable discrepancies observed between the predicted and worst configurations emphasize the model's tendency to produce substantially different results. It is essential to acknowledge that the predicted configurations, both from Method 1 and Method 2, consistently exhibit a significant gap when compared to the worst possible configurations (also seen in Figure 5.3.7). These findings suggest that the model's predictions often stray far from the extreme end of the configuration spectrum, implying a tendency towards more moderate or optimal configurations (**RQ2**).

When comparing the resulting RMSE between the two methods, a clear distinction is evident. Method 2 demonstrates a significantly lower RMSE for the *err* in comparison to Method 1. This disparity can be attributed to the application of z-score normalization in Method 2. By employing z-score normalization, Method 2 identifies distinct pairs of parameter configurations as the optimal ones. Figure 5.3.8 illustrates a comparison of the average optimal configurations for the Number of radix bits and Spline error using both Method 1 and Method 2. Notably, the average optimal *num* remains similar across both methods, while the average *err* for Method 2 is substantially lower. This suggests that Method 2's model was trained within a narrower range for the *err*, resulting in a reduced RMSE for this

particular method.

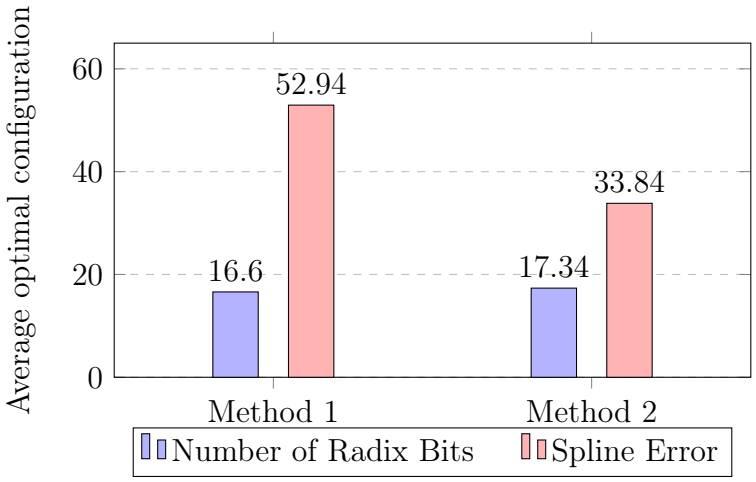


Figure 5.3.8: Graph comparing Method 1 and Method 2 in regards to their average RMSE after running the model five times.

A notable observation in both methods is the significant difference in RMSE between the first training run and the average RMSE from training the model five times. Method 1 exhibited a **12.95%** lower RMSE for *num* and a **19.15%** lower RMSE for *err* compared to the average. This distinction was also apparent in Method 2, albeit to a lesser extent. Method 2 demonstrated an **11.56%** lower RMSE for *num* and a **2.25%** lower RMSE for *err* compared to the average.

The variation observed in the RMSE highlights a compelling indication of the model's inclination towards high variance and inherent instability. Once again, this prevailing trend can be reasonably ascribed to the persistently limited amount of training data.

5.3.4 Z-Score Normalization for Worst Configuration

In both Method 1 and 2, z-score normalization was employed to identify the worst configuration by comparing the predicted values with the worst-case scenario. This approach was chosen because the worst configuration is not subject to constraints, unlike in Method 1. For instance, Method 1 allows for algorithm manipulation to derive the best index sizes from the top 15% of lookup times instead of the standard 10%. However, the worst configuration consistently remains unchanged, determined by the sum of the highest values for index size and lookup time.

CHAPTER

SIX

DISCUSSION

The research presented in this thesis provides valuable insights into a novel approach for predicting the optimal parameter configuration with the radix spline index structure. In Chapter 5, we examine the outcomes obtained from this 'auto-tuner' and briefly discuss certain limitations inherent in the proposed method. This chapter offers a comprehensive analysis of these limitations, shedding light on their impact and implications for future investigations. Additionally, we present avenues for future research and where this approach might be implemented.

6.1 Implementation

As for learned index structures in general they sought to improve query performance and reduce storage requirements, in this section the possibility for this type of auto-tuned radix spline will be discussed in regards to real-world applications.

6.1.1 Dual-Pass Radix Spline

As mentioned in Section 4.1.2, it is necessary to traverse the data once before constructing the index. This step allows for the extraction of features, which are essential for the training of the model to be able to predict the optimal parameter configuration when building the index. This differs from the manually tuned radix spline index structure, since RS is able to build the index after only traversing the data once. Another assumption radix spline makes is that there is no need for single updates and that the single-pass training can be done together with the merge process in an LSM-Tree.

The merge process in an LSM-tree can be resource-intensive, primarily due to the need to merge data with existing on-disk structures, eliminate duplicates, and maintain sorted order. Considering these factors, even building the radix spline index with a dual-pass approach, involving passing over the data twice, would not significantly impact the overall time. In fact, it has the potential to result in time savings as the dual-pass approach better configures the index structure, leading to reduced lookup time. This proposition is also mentioned in Kipf et al. 2020.

Since this research did not specifically investigate the build time, the impact on the overall query time when considering the build process remains uncertain. Further investigation is required to comprehensively assess the effects of the presented approach on both build time and query performance.

6.1.2 Search Engines and Elastic Search

One limitation of building the index during the merge process of an LSM-tree is that RS can only index the data that has been flushed to disk, excluding the data currently residing in the memTables. Since RS is not limited to LSM-trees and can be applied in other scenarios such as search engines or Elasticsearch, it may find alternative applications. However, when dealing with continuously updating data obtained by search engine crawlers, it becomes infeasible to maintain the write-once and read-many principle that RS is well-suited for. Consequently, it is advisable to implement RS specifically for constructing an index on static storage, which can be updated in bulk with a lower rate.

6.2 Limitations

The following section highlights the limitations encountered in the research conducted for this thesis. While some progress has been made in developing an approach to predict the optimal parameter configuration for radix spline, certain constraints and challenges have emerged. By acknowledging these limitations, we gain a comprehensive understanding of the scope and implications of the research findings, thereby paving the way for future investigations and potential improvements in the field.

6.2.1 Comparison with PLEX

As discussed in Section 3.1.6, PLEX is an auto-tuned learned index that simplifies the process of tuning the RS index by requiring only one hyperparameter: the maximum prediction error ϵ. In our experiment, we employed an alternative approach to this problem by utilizing a neural network to train a model capable of predicting the optimal configuration for any given dataset. The results presented in the PLEX paper primarily focused on showcasing the impact of index size on lookup time when compared to other index structures, specifically on 64-bit datasets. Given that our training and prediction were conducted solely on 32-bit datasets, it becomes challenging to directly compare the results obtained from our approach with those of PLEX. But based on the radix table cost model and the CHT cost model, it is reasonable to assume that the accuracy of the predicted configuration is comparable to, or even superior to, the predicted configuration of our model.

6.2.2 Data Generation

The datasets used in the SOSD Benchmark were deemed insufficiently diverse to effectively train the model. Instead, we utilized 29 different distributions to construct a total of 115 datasets. Most of the 29 datasets had parameters

that could be adjusted to manipulate the distribution, except the five standard distributions: *standard_couchy*, *standard_exponential*, *standard_gamma*, *standard_normal*, and *standard_t*.

One inherent limitation of the data is that it consists entirely of synthetic datasets generated using well-known distributions. The parameters used to generate these datasets adhered closely to the specifications outlined in the documentation, with minimal deviation from the default values.

Although the generated datasets incorporated various values for the features, it is important to note that the data remained synthetic, thereby excluding real-world data from the model's training. Consequently, when the model encountered more complex real-world data or synthetic data generated with distributions having unusual parameters, it struggled to capture the underlying patterns effectively. As discussed in Section 2.1.3.2, this suggests that the model may be underfitted. Notably, in certain cases, such as when predicting the optimal configuration for the *lognormal* dataset, the model's performance suffered due to the inability to generalize well and handle unseen data.

6.2.3 Underfitted Model

In the context of neural network training, it is not surprising that the model may exhibit signs of being underfitted, considering the limited variability in the dataset. The generation of only 115 datasets, derived from a mere 29 distributions, constrains the model's exposure to diverse patterns and scenarios. Although the model was able to predict parameter configurations within an acceptable range for the majority of the evaluation datasets, and never selected the worst configuration for any of the test datasets. It is important to note that the test sets were predominantly synthetic generated datasets, with only one dataset representing real-world data. Given that the model was trained primarily on these types of datasets, it is reasonable to attribute the model's prediction performance to the familiarity with such data characteristics.

6.2.4 32-Bit Datasets

The scope of the project led to the generation of 115 datasets, all of which were formatted as 32-bit data. This decision was made due to the substantial amount of time required to generate and process such a large volume of data. Creating lookups, extracting features, and building indexes were already time-consuming tasks, and using 64-bit datasets would have further increased the time required. Additionally, each dataset had a size of 1.6 GiB, resulting in a cumulative dataset collection size exceeding 200 GiB.

During the initial stages of the project, difficulties were encountered when compiling the C++ projects required for generating lookups and building the necessary data on the NTNU *DIF* servers. As a result, an alternative machine with 32 GiB of RAM, as mentioned in Section 4.1, was utilized instead. It should be noted that

the limited memory capacity of this machine posed challenges during the process of building the index and generating lookups. Conducting these tasks with 64-bit datasets would have been unfeasible due to the memory constraints.

6.2.5 K-Fold Cross-Validation

The data used to evaluate the model were selected from the Harvard Dataverse database Harvard n.d., including the *books* dataset, which represents a real-world dataset. Additionally, 20% of the training data was manually selected for evaluation purposes. To ensure diversity in the evaluation set, one dataset was manually selected from each of the five datasets generated per distribution. Since there was limited training data, this had to be done to be able to evaluate the model on the wide range of distributions. Normally, one would use k-fold cross validation to split the data into multiple folds, as mentioned in Section 2.1.3.3.

One limitation of manual dataset selection is that it may not offer a representative sample of the entire dataset. Despite selecting one dataset per distribution, there may still be significant variations within each distribution. This limitation can contribute to a more underfitted model since it does not utilize a systematic and unbiased evaluation technique like k-fold cross-validation. Without such a technique, it becomes challenging to accurately assess the model's generalization ability to unseen data. As a result, the evaluation may lack robustness and may not provide a comprehensive understanding of the model's performance on diverse and unseen data.

6.2.6 Feature Selection

Despite the presence of four different features in the dataset, only three of them significantly impacted the training of the model. This was primarily because the *size* feature remained constant at 200M for the majority of the datasets, with only two exceptions. The remaining features were associated with representing the distribution of keys. This limitation arises from the possibility of other factors influencing the determination of the optimal parameter configuration for RS. Although the selection of features was driven by a data-driven approach, involving testing and identifying correlations between the features and the configuration, there remains the potential for additional influential factors that were not accounted for.

6.2.7 Neural Network

While the approach presented in this thesis demonstrates the potential application of neural networks in predicting the optimal parameter configuration for radix spline, it is not without limitations. Firstly, it is important to note that achieving accurate predictions with a neural network model requires a significant amount of training data, as highlighted in the experiment conducted in this research and discussed as a limitation in Section 6.2.3. Additionally, even with sufficient training data, the model may still fall short in accurately predicting the correct configuration.

Another limitation of using a neural network is the complexity. In this research, a relatively common neural network configuration was utilized. However, to attain an optimally tuned model, further testing and evaluation would be required, which were not conducted in this experiment. Additionally, with the limited training data available, it would not have been useful to perform such extensive testing. This is because the model exhibited significant inaccuracy, making it difficult to discern improvements in prediction performance with other parameters.

Lastly, one of the significant limitations of neural networks lies in their resource requirements for training and execution. Neural networks can be computationally intensive, especially when handling large datasets. The high computational demand associated with neural networks poses challenges in terms of both time and computational resources. This can be impractical when striving to establish efficient index structures that aim to minimize query time. Consequently, the resource-intensive nature of neural networks presents a substantial drawback in the context of databases, where computational efficiency plays a pivotal role.

6.2.8 Write-Once/Read-Many

This approach shares a similar assumption with RS, as both are designed to excel in a write-once/read-many scenario. This means that making frequent changes or updates to the data, such as single updates, would require passing over the entire dataset and rebuilding the index for each update. Hence, it is advantageous to build the index in conjunction with the merging process of an LSM-Tree, as discussed in Section 6.1. However, this approach has some limitations due to the restricted applicability. The index structure's effectiveness is constrained by specific use cases and may not be well-suited for scenarios that involve frequent updates or dynamic data modifications.

6.2.9 Build Time

Throughout this thesis, the experiments have primarily focused on examining the results of index size and lookup time, with less emphasis on the build time. However, it is important to acknowledge the significance of build time in the overall performance of the index structure. Given the considerable time required for feature extraction from each dataset.

Upon examining the results of radix spline presented in Section 3.1.4, it becomes evident that the build time is predominantly influenced by the spline error. Consequently, it is logical to incorporate this understanding into the prediction of the optimal configuration. Notably, one of the predicted spline errors was estimated as 4 for the *negative_binomial* dataset, which would result in a substantial increase in build time. Therefore, leveraging this knowledge to guide the prediction process can lead to more informed decisions and potentially mitigate excessive build times in practical scenarios.

6.3 Future work

The final section of the discussion is dedicated to exploring potential avenues for future research and highlighting recommended next steps based on the obtained results.

6.3.1 Generating More Complex and Diverse Data

The limited scope of data generation poses a significant limitation in this research, impacting the accuracy and reliability of the neural network model in predicting optimal configurations for radix spline. To address this limitation and improve the effectiveness of the proposed approach, future work should focus on expanding the dataset collection by generating a larger and more diverse range of datasets. This would involve incorporating complex real-world data, which better represents the varied scenarios encountered in practical applications of radix spline. By enriching the dataset with a broader spectrum of data, future investigations can enhance the model's performance, robustness, and generalizability, leading to more accurate predictions and better utilization of radix spline in real-world scenarios.

Future work should also focus on enhancing the dataset's variability, especially regarding the size. In the current research, the datasets predominantly consisted of 200M key/value pairs, with only two distributions featuring datasets of 400M pairs. To enhance the robustness and effectiveness of the model, it is crucial to generate datasets with a wider range of sizes. This increased variability will not only bolster the model's performance but also amplify the significance of the *size* feature.

Lastly, future work related to data generation should involve the inclusion of datasets with varying bit sizes, such as 16-bit and 64-bit. By incorporating these variations as features, it would enable training and evaluation on datasets with different bit sizes. This would allow us to assess the model's ability to accurately provide a satisfactory configuration for the *face_200M_uint64* dataset, thereby addressing the outlier problem discussed throughout this thesis.

6.3.2 Exploring Different Features

There are numerous potential features associated with the data distribution. As discussed in Section 6.2.6, the features employed in this research may not be the most optimal ones for capturing the underlying patterns and correlations between the dataset and the optimal parameter configuration.

In order to enhance the predictive capabilities of the model and capture a more comprehensive understanding of the relationship between the dataset and the optimal parameter configuration, future work should consider incorporating additional features or exploring different feature sets. Specifically, focusing on features that directly relate to the distribution of keys. By expanding the feature space, the model can potentially uncover new patterns and correlations that were not cap-

tured by the existing set of features.

6.3.3 Improving the Neural Network Model

In this study, the neural network model was configured using standard settings, without any specific process to fine-tune it for optimal prediction performance. As this was the initial exploration of the approach, the focus was not on fine-tuning the model. However, future work should dedicate resources to tuning the model, considering factors such as the number of epochs, the number of layers, and the number of nodes per layer. These adjustments can have a substantial impact on the model's performance. Additionally, exploring different activation functions is another avenue worth investigating to improve the model's predictive capabilities.

Another crucial aspect to improve in the model is the evaluation process. If the model is trained using a significantly larger dataset, it is recommended to employ k-fold cross-validation. This technique provides a more accurate evaluation by validating the model on multiple subsets of the data, allowing for a comprehensive assessment of the performance with unseen data. By incorporating k-fold cross-validation, valuable insights can be gained into how the model generalizes and performs across different data partitions, enhancing the overall reliability and robustness of the evaluation results.

6.3.4 Comparing the Results Against a Non-Tuned Radix Spline

It would have been compelling to observe the comparative analysis between RS with the predicted optimal configuration and the standard RS. This analysis should include an examination of the lookup time and index size across the two implementations. Furthermore, it is essential to extend this evaluation to incorporate the build time, considering that the data is traversed twice before building the index.

6.3.5 Using A Weighted Z-Score Normalization

In our pursuit of identifying the optimal parameter configuration, we employed two methodologies. Firstly, we implemented Method 1, a script that discerned configurations with the smallest index size among the top 10% lookup times, as mentioned in Section 4.1.4. By adopting this approach, we accorded higher priority to the optimization of lookup time over index size. It is worth noting that within the top 10% lookup times, all configurations may potentially exhibit the poorest 10% index size.

Due to this consideration, it is highly recommended to incorporate weighted z-score normalization in future research research. This approach aims to effectively accommodate the constraints that are specific to each system, facilitating a more refined calibration of lookup time and index size constraints according to the distinct requirements of individual systems.

CHAPTER

SEVEN

CONCLUSIONS

In this thesis, we have embarked on an investigation to predict the optimal parameter configuration for the radix spline learned index structure. Our research journey encompassed several critical steps aimed at unraveling the potential and limitations of this approach. We commenced by generating synthetic data based on different data distributions, capturing a broad range of dataset structures. Subsequently, we diligently extracted relevant features to encapsulate the essential characteristics inherent in the datasets. Employing z-score normalization, we identified the configurations that yielded the most favorable outcomes in terms of lookup time and index size. Through training and evaluation, our objective was to assess the performance and accuracy of our model. In this section, we present the culmination of our efforts, delving into the noteworthy findings and their direct implications in relation to our research questions.

7.1 Research Question 1

The model's predictions were assessed by comparing them to the best configurations obtained for each dataset. While some of the model's predictions aligned with the optimal configurations specific to certain datasets, the majority of the predictions did not match the optimal configurations determined by the established models discussed in Section 4.1.4. Consequently, it can be concluded that the experiment conducted does not consistently achieve the prediction of optimal configurations for any dataset. It is important to acknowledge that the obtained results should not be interpreted as indicating poor performance. It is unrealistic to expect the proposed approach to achieve 100% accuracy.

7.2 Research Question 2

To address Research Question 2, the worst configurations for all the tested datasets were identified and compared to the predictions made by the model. None of the predicted values matched the worst possible configurations for their respective datasets, thereby demonstrating the models' capability to steer away from the worst-case configuration. Upon comparing the predicted configurations to the worst possible configurations, it was observed that there existed a degree of

deviation between them. This highlights a discrepancy between the predicted configurations and the worst-case scenarios.

7.3 Research Goal

The primary objective of this thesis was to make a contribution to the field of learned indexes by exploring an alternative approach to auto-tune radix spline. To this end, we have conducted a thorough investigation comprising a series of experiments and analyses. The results obtained in this study provide compelling evidence that the proposed approach exhibits a certain degree of effectiveness in predicting optimal configurations. Notably, it successfully avoids the worst-case configuration and did not encounter the outlier problem for any of the tested datasets. As a result, this thesis makes a valuable contribution to the field of learned indexes by shedding light on the practical application of neural networks for auto-tuning radix spline, thereby establishing groundwork for future research endeavors in this area.

REFERENCES

Kraska, Tim et al. (2018). "The Case for Learned Index Structures". In: URL: https://arxiv.org/abs/1712.01208.

Kipf, Andreas et al. (2020). "RadixSpline: a single-pass learned index". In: *Proceedings of the Third International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, aiDM@SIGMOD 2020, Portland, Oregon, USA, June 19, 2020*, 5:1–5:5. DOI: 10.1145/3401071.3401659. URL: https://doi.org/10.1145/3401071.3401659.

Ramakrishnan, Raghu and Johannes Gehrke (2002). *Database Management Systems*. McGraw-Hill. ISBN: 0072465638.

GeekForGeeks (2023a). *Introduction of B-Tree*. URL: https://www.geeksforgeeks.org/introduction-of-b-tree-2/.

Alpaydin, Ethem (2014). *Introduction to Machine Learning*. The MIT Press. ISBN: 0262028182.

Quiza, Ramon and J. Davim (Jan. 2011). "Computational Methods and Optimization". In: pp. 177–208. ISBN: 978-1-84996-449-4. DOI: 10.1007/978-1-84996-450-0.

Mathworks (n.d.). *Feature extraction for machine learning and deep learning*. MathWorks. Accessed: 04.05.2023. URL: https://www.mathworks.com/discovery/feature-extraction.html.

GeekForGeeks (2023b). *ML | Underfitting and Overfitting*. URL: https://www.geeksforgeeks.org/underfitting-and-overfitting-in-machine-learning/.

Haben, Stephen, Marcus Voß, and William Holderbaum (May 2023). "Primer on Statistics and Probability". In: pp. 23–39. ISBN: 978-3-031-27851-8. DOI: 10.1007/978-3-031-27852-5_3.

PATRO, S GOPAL and Dr-Kishore Kumar Sahu (Mar. 2015). "Normalization: A Preprocessing Stage". In: *IARJSET*. DOI: 10.17148/IARJSET.2015.2305.

O'Neil, Patrick et al. (June 1996). "The log-structured merge-tree (LSM-tree)". In: *Acta Informatica* 33, pp. 351–385. DOI: 10.1007/s002360050048.

Ding, Jialin et al. (2020). "ALEX: An Updatable Adaptive Learned Index". In: SIGMOD '20. New York, NY, USA: Association for Computing Machinery. URL: https://doi.org/10.1145/3318464.3389711.

Leis, Viktor, Alfons Kemper, and Thomas Neumann (2013). "The adaptive radix tree: ARTful indexing for main-memory databases". In: *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pp. 38–49. DOI: 10.1109/ICDE.2013.6544812.

Ferragina, Paolo and Giorgio Vinciguerra (2020). "The PGM-index: a fully-dynamic compressed learned index with provable worst-case bounds". In: *PVLDB* 13.8. ISSN: 2150-8097. URL: https://pgm.di.unipi.it.

Kipf, Andreas et al. (2019). "SOSD: A Benchmark for Learned Indexes". In: *NeurIPS Workshop on Machine Learning for Systems*.

Crotty, Andrew (2021). "Hist-Tree: Those Who Ignore It Are Doomed to Learn". In: *Conference on Innovative Data Systems Research*. URL: https://cs.brown.edu/people/acrotty/pubs/cidr2021_paper20.pdf.

Stoian, Mihail et al. (Aug. 2021). "PLEX: Towards Practical Learned Indexing". In: URL: https://www.researchgate.net/publication/353838541_PLEX_Towards_Practical_Learned_Indexing.

Numpy.org (2020). *Random sampling*. Wikipedia. Accessed: 02.03.2023. URL: https://numpy.org/doc/1.16/reference/routines.random.html.

Abadi, Martín et al. (May 2016). "TensorFlow: A system for large-scale machine learning". In: URL: https://www.researchgate.net/publication/353838541_PLEX_Towards_Practical_Learned_Indexing.

IkamusumeFan (n.d.). *Uniform CDF*. Wikipedia. Accessed: 19.03.2023. URL: https://commons.wikimedia.org/wiki/File:Uniform_cdf.svg.

Inductiveload (n.d.). *Normal Distribution CDF*. Wikipedia. Accessed: 19.03.2023. URL: https://en.wikipedia.org/wiki/File:Normal_Distribution_CDF.svg.

EvgSkv (n.d.). *Exponential Distribution CDF*. Wikipedia. Accessed: 19.03.2023. URL: https://commons.wikimedia.org/wiki/File:Exponential_distribution_cdf_-_public_domain.svg.

Harvard (n.d.). *Harvard Dataverse*. Accessed: 01.02.2023. URL: https://dataverse.harvard.edu/.

APPENDICES

A - DETAILED TABULAR DATA

A1 - Parameter Values - All distributions

Name	**Parameters**	**Parameter Value**
Beta	α, β	['0.1, 0.9', '0.3, 0.9', '0.5, 0.9', '0.5, 0.5', '0.9, 0.1']
Chi-square	k	['1', '2', '5', '10', '20']
Dirichlet	$\alpha_1, \ldots, \alpha_K$	['[1,5]', '[5, 20]', '[20, 2]', '[1, 20]', '[1, 100]']
F	d_1, d_2	['182.3, 24.9', '200, 200', '2050, 10.5', '78, 4000', '8, 20']
Gamma	k, θ	['7.5, 2', '0.5, 1', '1.0, 2', '2, 2', '5, 1']
Gumbel	μ, β	['0.5, 2', '1, 2', '1.5, 3', '3, 4', '5, 6']
Laplace	μ, b	['0, 1', '0, 2', '0, 4', '5, 4', '16, 22']
Logistic	μ, s	['5, 2', '9, 3', '9, 4', '6, 2', '2, 1']
Multivariate Normal	μ, Σ	[[10, 50], [[10, 0],[0, 100]]', '[0, 0], [[10, 0],[0, 50]]', '[150, 0], [[150, 0],[0, 10]]', '[30, 0], [[50, 0], [0, 200]]', '[0, 0], [[1, 0],[0, 100]]]
Negative Binomial	r, p	['1, 0.1', '2, 0.1', '5, 0.1', '10, 0.4', '15, 0.2']
Noncentral Chi-square	k, λ	['1, 1.0', '5, 0.1', '10, 0.1', '50, 2.4', '100, 50.2']
Noncentral F	d_1, d_2, λ	['50, 12.0, 1', '80, 44.1, 2', '100, 1500, 8', '2, 25421.4, 2', '2, 54207.2, 2']
Pareto	x_m, α	['5', '10', '20', '50', '100']
Rayleigh	σ	['100', '500', '1000', '2000', '5000']
Standard Cauchy	-	[]
Standard Exponential	-	[]
Standard Gamma	-	[]
Standard Normal	-	[]
Standard t	ν	[]
Triangular	a, b, c	['0,(nK/2)','(nK-1),(nK/8)','(nK-1),(nK/1.5)','(nK-1),(nK/15)','(nK-1),(nK/1.1)']
Von Mises	μ, κ	['(nK/2), 16','(nK/4), 8','(nK/8), 4','(nK/16), 2','(nK/32), 1']
Wald	μ, λ	['1, 16', '1, 8', '1, 4', '1, 2', '1, 1']

Weibull	k, λ	['16', '8', '4', '2', '1']
Zipf	s	['2', '3', '4', '5', '6']
Uniform	a, b	['1, 4294967295']
Exponential	λ	['4294967295']
Lognormal	μ, σ	['0,1']
Normal	μ, σ	['1, 4294967295']
Power	a, k	['1', '1.2', '1.4', '1.5', '1.8']

Table A.1: All the parameter values for the distributions used to create the synthetic data. (nK = number of keys)

A2 - Feature extraction - All datasets

Dataset	Size	MKIF	NUKI	MUKI
beta	200M	114351	1749	104866000
beta	200M	269542	742	32123700
beta	200M	396825	504	14582900
beta	200M	412371	485	14153000
beta	200M	113122	1768	104885000
chisquare	200M	8251.85	24237	80608500
chisquare	200M	7612.96	26271	52412800
chisquare	200M	8027.94	24913	36822000
chisquare	200M	8134.05	24588	30186000
chisquare	200M	8637.82	23154	29907300
dirichlet	400M	923788	433	44888100
dirichlet	400M	59232.9	6753	56454500
dirichlet	400M	21510	18596	76949900
dirichlet	400M	19506.5	20506	108089000
dirichlet	400M	35884.1	11147	218959000
f	200M	7939.03	25192	51216700
f	200M	8342.02	23975	28826100
f	200M	9973.07	20054	108943000
gamma	200M	8230.45	24300	28424200
gamma	200M	7252.42	27577	71749900
gamma	200M	7977.66	25070	54596600
gamma	200M	8127.77	24607	40981600
gamma	200M	8659.13	23097	33184800
gumbel	200M	7816.16	25588	38848800
gumbel	200M	7668.42	26081	37639100
gumbel	200M	7702.08	25967	38011500
gumbel	200M	7869.06	25416	39143700
gumbel	200M	7883.02	25371	39289900
laplace	200M	7427.76	26926	50963300
laplace	200M	7248.22	27593	49895600
laplace	200M	7977.98	25069	54839400
laplace	200M	7625.73	26227	52553100
laplace	200M	7483.35	26726	51355500
logistic	200M	7612.38	26273	43079000
logistic	200M	7493.72	26689	41711700
logistic	200M	7405.49	27007	41354400
logistic	200M	7611.22	26277	42746100
logistic	200M	7675.19	26058	43553400
multivariate_normal	400M	16780.6	23837	12805200
negative_binomial	200M	25000000	8	200000000
negative_binomial	200M	28571400	7	200000000

negative_binomial	200M	22222200	9	200000000
negative_binomial	200M	40000000	5	200000000
negative_binomial	200M	22222200	9	200000000
noncentral_chisquare	200M	8360.85	23921	25823900
noncentral_f	200M	9134.51	21895	86893600
noncentral_f	200M	7951.34	25153	38777000
noncentral_f	200M	8496.9	23538	26918100
noncentral_f	200M	7959.25	25128	42669900
noncentral_f	200M	8422.12	23747	46121400
pareto	200M	9932.46	20136	134549000
pareto	200M	9196.67	21747	89921000
pareto	200M	7882.39	25373	62662700
pareto	200M	7606.88	26292	56538500
pareto	200M	7924.24	25239	56397800
rayleigh	200M	9143.69	21873	23272900
rayleigh	200M	8849.56	22600	22002000
rayleigh	200M	8964.99	22309	22497000
rayleigh	200M	8702.08	22983	21524500
rayleigh	200M	8933.36	22388	22446500
standard_cauchy	200M	114482	1747	199933000
standard_cauchy	200M	67476.4	2964	199852000
standard_cauchy	200M	63031.8	3173	199838000
standard_cauchy	200M	74046.6	2701	199870000
standard_cauchy	200M	62754.9	3187	199837000
standard_exponential	200M	7455.73	26825	51046300
standard_gamma	200M	7693.79	25995	48499000
standard_gamma	200M	7083.91	28233	70000100
standard_gamma	200M	7503.28	26655	150223000
standard_gamma	200M	8507.02	23510	27339200
standard_gamma	200M	8494.01	23546	26601400
standard_normal	200M	8427.79	23731	25700500
standard_normal	200M	8403.36	23800	25637800
standard_normal	200M	8276.09	24166	25054500
standard_normal	200M	8323.62	24028	25332200
standard_normal	200M	8243	24263	24790700
standard_t	200M	15584.8	12833	178669000
standard_t	200M	10968.5	18234	111556000
standard_t	200M	8953.35	22338	58465800
standard_t	200M	7748.03	25813	34727300
standard_t	200M	8345.5	23965	30394000
triangular	200M	32867.7	6085	11589500
triangular	200M	32530.9	6148	11584100
triangular	200M	32727.9	6111	11592000
triangular	200M	32867.7	6085	11585000
triangular	200M	32530.9	6148	11582300
vonmises	200M	11578.1	17274	38329800

vonmises	200M	8525.88	23458	32366900
vonmises	200M	7016.81	28503	25971700
vonmises	200M	63211.1	3164	17927300
vonmises	200M	224719	890	12238600
wald	200M	8322.24	24032	32126900
wald	200M	7776.96	25717	34190800
wald	200M	7568.59	26425	41467000
wald	200M	7502.72	26657	53927000
wald	200M	7483.63	26725	64530500
weibull	200M	8050.56	24843	28229200
weibull	200M	8729.05	22912	23214200
weibull	200M	9924.08	20153	19248100
weibull	200M	9152.9	21851	23525400
weibull	200M	7279.08	27476	49773100
zipf	200M	216685	923	199980000
zipf	200M	3333330	60	199999000
zipf	200M	11111100	18	200000000
zipf	200M	20000000	10	200000000
zipf	200M	28571400	7	200000000
uniform	200M	573066	349	8893560
exponential	200M	7517.67	26604	51628700
lognormal	200M	9509.77	21031	142111000
normal	200M	8193.36	24410	24806900
power	200M	557103	359	8889000
power	200M	299850	667	9125670
power	200M	126904	1576	9616390
power	200M	90826.5	2202	9912750
power	200M	44692.7	4475	10886500

Table A.2: The features extracted for all the datasets

A3 - Best Configuration - All datasets

Dataset	Number of Radix Bits	Spline Error
beta	18	64
beta	16	128
beta	16	64
beta	12	64
beta	16	32
chisquare	18	64
chisquare	18	64
chisquare	18	64
chisquare	16	32
chisquare	16	64
dirichlet	16	64
dirichlet	14	64
dirichlet	16	128
dirichlet	14	128
dirichlet	16	128
f	16	128
f	14	64
f	18	64
gamma	16	32
gamma	18	32
gamma	18	32
gamma	18	64
gamma	18	64
gumbel	16	64
gumbel	18	64
gumbel	16	32
gumbel	16	64
gumbel	14	64
laplace	18	64
laplace	18	64
laplace	18	64
laplace	16	64
laplace	16	64
logistic	16	32
logistic	20	8
logistic	16	32
logistic	16	32
logistic	16	32
multivariate_normal	18	128
negative_binomial	12	4
negative_binomial	14	4

negative_binomial	12	4
negative_binomial	12	8
negative_binomial	10	4
noncentral_chisquare	16	32
noncentral_f	18	32
noncentral_f	16	32
noncentral_f	18	64
noncentral_f	16	32
noncentral_f	18	32
pareto	20	32
pareto	20	32
pareto	18	64
pareto	16	64
pareto	18	64
rayleigh	16	64
rayleigh	18	64
rayleigh	16	32
rayleigh	16	32
rayleigh	16	32
standard_cauchy	24	256
standard_cauchy	22	32
standard_cauchy	24	256
standard_cauchy	24	256
standard_cauchy	22	32
standard_exponential	18	32
standard_gamma	18	32
standard_gamma	18	32
standard_gamma	20	16
standard_gamma	16	32
standard_gamma	16	32
standard_normal	18	64
standard_normal	16	32
standard_normal	16	32
standard_normal	16	32
standard_normal	16	64
standard_t	20	64
standard_t	20	32
standard_t	18	32
standard_t	16	32
standard_t	16	32
triangular	14	32
triangular	14	32
triangular	14	32
triangular	14	32
triangular	16	32
vonmises	16	32

vonmises	16	32
vonmises	18	64
vonmises	16	32
vonmises	14	32
wald	16	32
wald	16	32
wald	16	32
wald	18	32
wald	20	32
weibull	16	32
weibull	18	64
weibull	16	32
weibull	16	32
weibull	18	32
zipf	20	4
zipf	12	4
zipf	10	4
zipf	12	4
zipf	12	4
uniform	14	32
exponential	18	256
lognormal	20	32
normal	16	32
power	14	32
power	14	32
power	14	32
power	14	32
power	14	32

Table A.3: Table displaying optimal configuration using Method 1

A4 - Best Configuration - All datasets

Dataset	Number of Radix Bits	Spline Error
beta	10	32
beta	10	8
beta	16	256
beta	12	8
beta	10	256
chisquare	10	64
chisquare	10	32
chisquare	16	8
chisquare	12	16
chisquare	12	128
dirichlet	10	8
dirichlet	10	16
dirichlet	18	32
dirichlet	10	32
dirichlet	24	128
f	10	16
f	14	8
f	12	16
gamma	12	8
gamma	10	8
gamma	10	8
gamma	10	8
gamma	12	8
gumbel	10	32
gumbel	10	8
gumbel	14	8
gumbel	10	8
gumbel	10	16
laplace	10	8
laplace	10	8
laplace	10	8
laplace	12	8
laplace	10	8
logistic	10	8
logistic	10	8
logistic	10	64
logistic	10	8
logistic	10	8
multivariate_normal	10	128
negative_binomial	10	128
negative_binomial	10	32

negative_binomial	12	64
negative_binomial	10	8
negative_binomial	14	64
noncentral_chisquare	20	32
noncentral_f	12	256
noncentral_f	10	8
noncentral_f	10	8
noncentral_f	10	16
noncentral_f	12	16
pareto	12	16
pareto	10	64
pareto	12	16
pareto	12	128
pareto	10	128
rayleigh	10	8
rayleigh	10	8
rayleigh	10	8
rayleigh	10	8
rayleigh	12	16
standard_cauchy	10	8
standard_cauchy	10	8
standard_cauchy	10	32
standard_cauchy	12	16
standard_cauchy	10	64
standard_exponential	10	16
standard_gamma	10	64
standard_gamma	10	8
standard_gamma	10	32
standard_gamma	10	32
standard_gamma	10	8
standard_normal	16	8
standard_normal	10	8
standard_normal	10	8
standard_normal	10	8
standard_normal	10	8
standard_t	10	64
standard_t	10	8
standard_t	10	8
standard_t	10	64
standard_t	12	16
triangular	14	8
triangular	10	8
triangular	12	8
triangular	10	8
triangular	12	16
vonmises	10	8

vonmises	12	8
vonmises	10	8
vonmises	10	8
vonmises	10	64
wald	10	32
wald	10	8
wald	20	256
wald	10	16
wald	10	8
weibull	10	128
weibull	10	64
weibull	10	8
weibull	10	32
weibull	10	8
zipf	10	8
zipf	10	8
zipf	12	16
zipf	10	8
zipf	10	8
uniform	10	8
exponential	10	8
lognormal	10	16
normal	12	64
power	10	4
power	20	64
power	22	32
power	18	64
power	10	128

Table A.4: Table displaying optimal configuration using Method 2

A5 - Predicted configurations - Non rounded

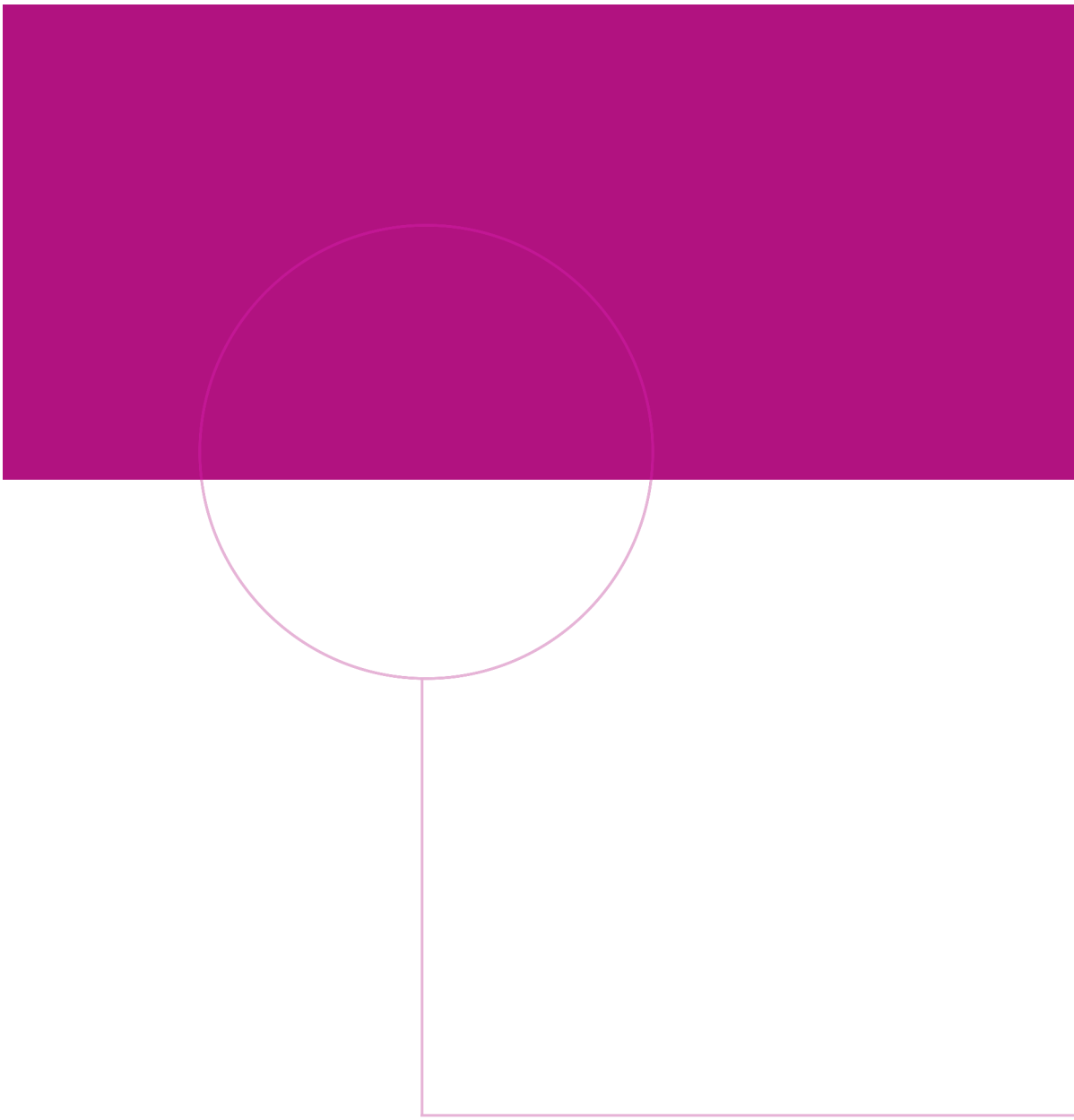
Dataset	Num	Err	Predicted num	predicted err
books	22	64	16.559944	77.46585
lognormal	16	4	21.007301	27.707832
normal	10	4	17.420912	47.90944
uniform_sparse	14	32	13.637132	31.597145
beta	16	32	16.760975	59.900875
chisquare	16	64	15.928265	37.796852
dirichlet	16	128	15.566213	140.82547
f	18	64	17.693745	43.662228
gamma	18	64	15.928947	36.20685
gumbel	14	64	16.759672	42.36778
laplace	16	64	17.418465	79.975266
logistic	16	32	17.140032	46.077526
negative_binomial	10	4	12.952838	26.212002
noncentral_f	18	32	16.719208	76.41333
pareto	18	64	17.528069	75.83009
rayleigh	16	32	16.096148	40.621628
standard_cauchy	22	32	15.556631	145.31194
standard_gamma	16	32	15.915553	40.504124
standard_normal	16	64	15.919629	43.231827
standard_t	16	32	16.029676	39.025566
triangular	16	32	14.595111	30.816849
vonmises	14	32	14.006728	36.674625
wald	20	32	17.878548	75.5414
weibull	18	32	17.402845	79.34965
zipf	12	4	12.783704	26.212002
power	14	32	14.328399	29.890652

Table A.5: Table displaying the accurate predictions from the model (Method 1)

A6 - Predicted configurations - Non rounded

Dataset	Num	Err	Predicted num	predicted err
books	16	256	15.221355	39.66526
lognormal	16	4	20.837484	19.160227
normal	16	4	17.55197	43.504898
uniform_sparse	14	32	14.785072	28.278358
beta	18	32	16.020077	22.995024
chisquare	18	32	16.972857	33.569336
dirichlet	20	64	18.966248	69.83592
f	20	64	17.568811	24.4113
gamma	16	64	16.92615	34.6729
gumbel	16	32	17.668081	37.401703
laplace	18	32	18.797457	52.26239
logistic	18	32	18.168694	40.64132
negative_binomial	16	4	13.84967	-0.07691622
noncentral_f	18	32	17.025091	39.185253
pareto	18	64	18.161207	49.709393
rayleigh	18	32	16.873299	31.104576
standard_cauchy	20	64	19.776594	22.108393
standard_gamma	18	32	17.115662	32.476448
standard_normal	16	32	17.32502	31.832014
standard_t	18	32	17.200897	33.782513
triangular	16	32	14.921661	29.681276
vonmises	14	32	14.643021	29.92548
wald	20	32	19.242348	50.81876
weibull	18	32	19.132484	51.679188
zipf	20	4	13.68548	-0.25703
power	18	32	14.905012	29.351665

Table A.6: Table displaying the accurate predictions from the model (Method 2)



NTNU

Norwegian University of
Science and Technology