Adaptive Selection of Lossy Compression Algorithms Using Machine Learning

Bachelor Thesis

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

Vorgelegt von:	Armin Schaare
E-Mail-Adresse:	3schaare@informatik.uni-hamburg.de
Matrikelnummer:	6423624
Studiengang:	B.Sc. Informatik
Erstgutachter:	Julian Kunkel
Zweitgutachter:	Anastasiia Novikova
Betreuer:	Julian Kunkel

Hamburg, den 29.11.2016

Abstract

This goal of this thesis was to evaluate machine learning model's ability for their use as an automatic decision feature for compression algorithms. Their task would be to predict which compression algorithms perform best on what kind of data. For this, artificially generated data, itself, and its compression was analyzed, producing a benchmark of different features, upon which machine learning models could be trained. The models' goal was to predict the compression and decompression throughput of algorithms Additionally, models had to correctly attribute data to the algorithm producing the best compression ratios. Machine learning approaches under consideration were Linear Models, Decision Trees and the trivial Mean Value Model as a comparison baseline. It was found, that Decision Trees performed significantly better than Linear Models which in turn were slightly better than the Mean Value approach. Nevertheless, even Decision Trees did not produce a satisfying result which could be reliably used for practical applications.

Contents

Disclaimer

The contents of Chapter 1 and 2 are based on the author's same-titled paper. This paper was written within the scope of the "Abschlussarbeiten-Seminar" of the University of Hamburg, which aims to prepare students for their thesis. The seminar was lectured by Dr. Daniel Moldt and the here referenced paper is made available on-line [Sch16].

Acknowledgement

I would like to express my sincere gratitude towards my supervisors who have supported me throughout my work on this thesis. Their constant and reliable commentary were an invaluable resource in this endeavor. Furthermore, while attending the "Abschlussarbeiten-Seminar", Dr. Daniel Moldt and every other attendee provided constructive feedback, structurally and content-wise, for which I am very thankful. Finally, the emotional support and motivation I received from my friends and family helped me to concentrate and stay on course. Thank you all, for your guidance and positivity.

1. Introduction

1.1. Motivation

Encoding digital information, using fewer bits than the original data, is known as data compression and can be used to minimize storage and bandwidth use of computer and network architectures. Compression methods can be divided into lossless and lossy ones, where members of the former are reconstructing the original data perfectly and members of the latter only approximate it. M. Hilbert and P. Lopez (2011) have shown that data compression still has unused potential. They estimate, that all the data in the world could be further compressed losslessly by a factor of 4.5 [HL11]. Therefore, it is obvious that reasonable lossy compression would result in even larger factors of compression. Nevertheless, legitimate concerns about non-acceptable data degradation and compression speeds, lead to a suboptimal use of available compression schemes, especially in scientific contexts, [HWK+13]. For this reason, SCIL — the Scientific Compression Interface Library — for the programming language C is being developed at the Universität Hamburg and the German Climate Computing Center. As part of the AIMES-project, SCIL aims to increase data storage efficiency by providing an interface to many of the most common and modern compression algorithms focusing on lossy ones [Kun16]. Users of SCIL will be able to specify absolute and relative error tolerances as well as compression- and decompression throughputs, which are guaranteed to be enforced. SCIL then automatically decides which algorithm is best suited for the given arguments and data to compress [SK16]. The decision process is the main subject of this thesis.

As there is a whole spectrum of different compression algorithms, choosing the optimal one for the given data is quite involved. The user has to consider the overall structure of the data such as its minimum and maximum value, standard deviation, mean value, etc. Furthermore, an in depth knowledge of each compression method with its — data depended — strengths and weaknesses, is required. Therefore, users tend to have only a few lossless and lossy compression methods in their repertoire, sticking to one of them for every compression. This leads to cases where the compression with a completely different method — unbeknown to the user — would be beneficial regarding compression throughput, ratio or both. SCIL aims to completely abstract the decision process from the user, by applying machine learning to map relevant criteria to the best suitable method [SK16]. With this feature, users are allowed to benefit from novel compression methods without modifying code.

1.2. Goals of the Thesis

The main goal of this thesis is the evaluation of different machine learning models for the task of SCIL's automatic compression decider. Machine learning approaches under evaluation are Linear Models as well as Decision Trees and should be carefully assessed by comparing them to useful baselines. Further goals of this thesis are the following ones:

- Further development of SCIL's framework containing the algorithm decider
- Implementation of a training data generator for machine learning purposes
- Improvement of overall compression rates

The purpose of the training data generator is to create diverse, realistic compressible data (patterns), extract relevant features such as standard deviation, mean value, etc., as well as conducting compression benchmarks for each available algorithm. The resulting data is then summarized into sets, upon which machine learning models will be trained.

After the conclusion of this paper, it should be clear, which machine learning approach under evaluation will be suitable to model SCIL's automatic decision process for compression purposes. The last point is directly dependent on the implementation of the best machine learning approach. Since this paper only covers the evaluation of different machine learning models, the accomplishment of this goal cannot be tested and should be the topic for another thesis in the future.

1.3. Structure of the Thesis

Chapter 2 provides a detailed background of compression, machine learning and SCIL, while focusing, how the automatic compression decider depends on these topics. An explanation of SCIL's inner functioning follows, explaining its structure and possible difficulties which could arise with the inclusion of the compression decider.

Chapter 3 portrays the design of the experiment and the training process for machine learning. It is explained, how compressible as well as training data for the machine learning task is generated Furthermore, the visualization of training data and the method of extracting useful feature information is described.

The evaluation of the considered machine learning models are presented in Chapter 4.

Chapter 5 contains the summary and conclusion, presenting the analysis of the evaluation's outcome. Here, the leading question, which machine learning approach is best suited for the task at hand, is answered. Finally, the thesis finishes with future outlooks of machine learning in conjunction to data compression.

2. Background

In this chapter, the background and theory of each relevant subject for SCIL's automatic compression decider is addressed.

Section 2.1 shortly summarizes the historical development of compression, focusing on the scientific community's stance towards it regarding the application upon measurement data. Furthermore, the community's concerns are illustrated as well as potential ways to mitigate them and using compression reasonably.

An introduction to machine learning is provided in Section 2.2. It presents the general theory of Machine Learning with a strong focus on pitfalls and their mitigation regarding its application. Additionally, each considered machine learning model is explained in detail.

Section 2.3 introduces SCIL's general work-flow. Here, SCIL's context creation and each component of its compression chain is explained thoroughly. Furthermore, all algorithms, currently available under SCIL, are explained.

2.1. Compression

Compression is the procedure of encoding data, in our case represented as an array of bytes, using less space than its uncoded equivalent. It can be classified into two categories: lossy and lossless compression. While lossless compression retains every bit of information, lossy compression represents a trade-off between data degradation and size by approximating the original data. For this reason, lossy algorithms generally produce higher compression ratios[1] than lossless ones. The main applications of data compression are minimizing storage and bandwidth use. Therefore it is widely used in digital audio, video and text as well as commercial and scientific data processing. Nevertheless estimates exist that worldwide data could be further compressed by a factor of 4.5 [HL11].

[1] The Compression ratio is defined by the quotient of uncompressed buffer size and compressed buffer size.

2.1.1. History

This section is prominently made up of excerpts from Stephen Wolframs (2002) book "A New Kind of Science". In its compression history section, Wolfram manages to purposefully convey key facts regarding lossless compression [Wol02].

First applications of data compression were present as early as 1838 with the invention of Morse code. For this, common letters of the English alphabet were encoded by shorter signals in comparison to uncommon letters. Without further use cases for compression at that time, notable advances where delayed to the 1940s and the rise of information theory. Claude Shannon and Robert M. Fano (1949) devised first methods of compression by using statistical approaches. David A. Huffman — at that time a Ph.D. student under Fano — perfected the scheme two years later, which is now known as the famous and widely used *Huffman coding* [Huf52]. In the following years, Huffman coding was implemented directly on hardware before being enhanced by dynamically coding actual encountered data instead of maintaining a static map. The dynamic mapping, developed in the 1970s, was widely used due to the rise of the on-line storing of text files. Shortly after, a fundamentally different algorithm emerged through the work of Abraham Lempel and Jacob Ziv, firstly realized by Terry A. Welch in 1984 [Wel84]. The *Lempel-Ziv-Welch* algorithm (LZW) quickly became the de facto standard due to unprecedented compression ratios and is still widely used today [KA10]. With personal computers capability of storing ever increasing amounts of data, digital images and audio became more popular [Wol02]. First lossy compression methods were developed for commercial and private use, most notably JPEG [Wal91] for digital images and MP3 [JJS93] for audio files. Since a reduction of picture or sound quality does not devalue the data in a way, errors would impact text files or scientific data, it can be reasonably traded-off for smaller file sizes. Both file formats were released in the early 1990s and are still widely in use today, even if there are many alternatives performing objectively better, i.e, JPEG's successor JPEG-2000 [Agu12].

There are also formats, specifically designed for scientific data, which provide lossy compression, such as GRIB, standardized by the World Meteorological Organization. GRIB is able to lossy encode data by quantization and bit-packing or by applying the JPEG-2000 algorithm, for its first and second edition, respectively [Wor03]. Yet, lossy compression is often disregarded in scientific contexts, since it degrades information. Even though institutions like NASA encouraged use of lossy compression for research data, a wide adoption was nowhere to be seen [Nat88]. Science, in most cases, kept relying on lossless methods as of now.

2.1.2. Concerns for Scientific Data

The degradation of scientific data is not the only concern of researches when applying compression. Another major factor is the resulting usage of computing power and memory. For example the Large Hadron Collider at CERN generates filtered data with a rate of 1 Gigabyte per second [fNR12]. There are many lossless algorithms which produce acceptable compression ratios with a reasonable throughput [Nem11]. Such algorithms are widely used with standardized formats such as GRIB. Nevertheless, lossy compression can achieve much higher compression ratios with still acceptable throughputs. Degradation of information can be adjusted to fit the purposes of the data at hand, by constraining the error of lossy compression to reasonable values. For example, by keeping the information loss to values within one standard deviation of measurement errors, justifiable data preservation can be attained. Thus, training personnel in using lossy compression sensibly could be well worth the effort.

2.2. Machine Learning

This section is based on the book "From Curve Fitting to Machine Learning" by Achim Zielesny (2013). Zielensy's book focuses on the responsible training of different machine learning models, taking into account common mistakes, users are tempted to make. Furthermore, he proposes ways to mitigate those mistakes, which this section presents in detail [Zie13].

Machine Learning covers the procedures for model approximation with the help of computers. These approximations are based on empirical data in contrast to analytical models. In general, machine learning addresses the problem of finding functions which sufficiently approximate dependencies of the real world. Data driven finding of a adequate model is done with an aspect of machine learning which is called *supervised learning*. Models under evaluation in this thesis are all trained by supervised learning.

Every machine learning model is represented by a mathematical function. This function consists of input arguments and parameters, whereas parameters describe the shape of the function. Changing the functions parameters leads to different outputs for the same input arguments. Supervised learning makes use of this fact to change a functions parameters such that the model begins to approximates a mapping between predefined input and output data (see A.1). This iterative process comes to a halt when a desirable model is found.

While supervised learning is used to change parameters of the model function, they are not the only variable arguments which determine the suitability of the model for the task at hand. There are also so-called *structural hyperparameters* or *metaparameters*, which are not present in the model function itself as values, but which, i.e., determine the amount of actual model parameters. In the case of Linear Models, they describe for example the grade of polynomials used for curve fitting (see 2.2.1). Hyperparameters are not optimized in supervised learning algorithms themselves, but are up for the user to decide, before the actual training commences. It is crucial to use reasonable values for hyperparameters, since the model's fitness after training directly depends on them. The problems which can arise in the case of non-optimal values for the hyperparameters are known as *underfitting* and *overfitting*.

An underfitted model is not able to comprehend the complexity of a given task. Smaller than optimal values for hyperparameters lead to models being underfitted. In such a case, models can be trained infinitely with arbitrary training algorithms and never produce satisfying results.

Overfitting can happen to models which have an exceedingly complex structure, in terms of its hyperparameters, given their context of application. As a result, the model will be able to almost exactly map the inputs of the training set to their corresponding outputs, while not being able to generalize reasonably. Inputs which were not in the training data lead to imprecise or, in extreme cases, utterly wrong outputs.

There are many more problems than under- and overfitting as well as methods of optimizing the training process. Additional topics regarding optimization of machine learning can be found in Appendix A.

The following subsections present the theory behind every considered model for the task of SCILs automated compression decider, providing a foundation for their analysis afterwards. The exact task for the compressor decider is to map compression relevant parameters to the best suitable method available. Due to their inherently different structure and mechanics, the models under evaluation are bound to have different results.

2.2.1. Linear Regression Models

The most simple case for a Linear Regression Model is a function of the form

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i \tag{2.1}$$

where y_i is the scalar output of the i-th data tuple, x_i is its scalar input, β_0 and β_1 are model parameters and ϵ_i is the error of the model function regarding the i-th data point. Such a model function is often called *Simple Linear Regression*, since it is only two dimensional, representing the most basic Linear Regression Model. SCIL's automatic compressor decider has to be able to map multiple input values to the output. For that reason, a more general approach has to be considered, which is called *Multiple Linear Regression* (*MLR*).

$$Y = X\beta + \epsilon \tag{2.2}$$

With

$$Y^T = \begin{pmatrix} y_0 & \dots & y_n \end{pmatrix} \tag{2.3}$$

$$\beta^T = \begin{pmatrix} \beta_0 & \dots & \beta_n \end{pmatrix} \tag{2.4}$$

$$\epsilon^T = \begin{pmatrix} e_1 & \dots & e_m \end{pmatrix} \tag{2.5}$$

$$X = \begin{pmatrix} 1 & x_{11} & \dots & x_{1n} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{m1} & \dots & x_{mn} \end{pmatrix} \tag{2.6}$$

Here, Y denotes a vector containing all output values of the data set, X is the input matrix, consisting of one multi-dimensional input per row. β^T is the transposed vector, containing all model parameters and ϵ^T is a vector containing the error of every data point. Note that the first column of X is solely made up of ones to produce the β_0 summand in Equation (2.1). Optimizing such a model can barely be considered training, since the process is not iterative and can be completed by analytical means (*Ordinary Least Squares*) via matrix calculus. Figure 2.1 shows a linear curve fit, predicting the *height* of a human depending on his or her *age* and *weight*.
A special case of MLR is *Multiple Polynomial Regression* or *MPR*. Such a model is able to learn factorial compositions of input values based on the provided maximum polynomial degree. For this, the input matrix is adapted to not only contain all single input values in each of its rows, but also all multiplicative combinations of those depending on the

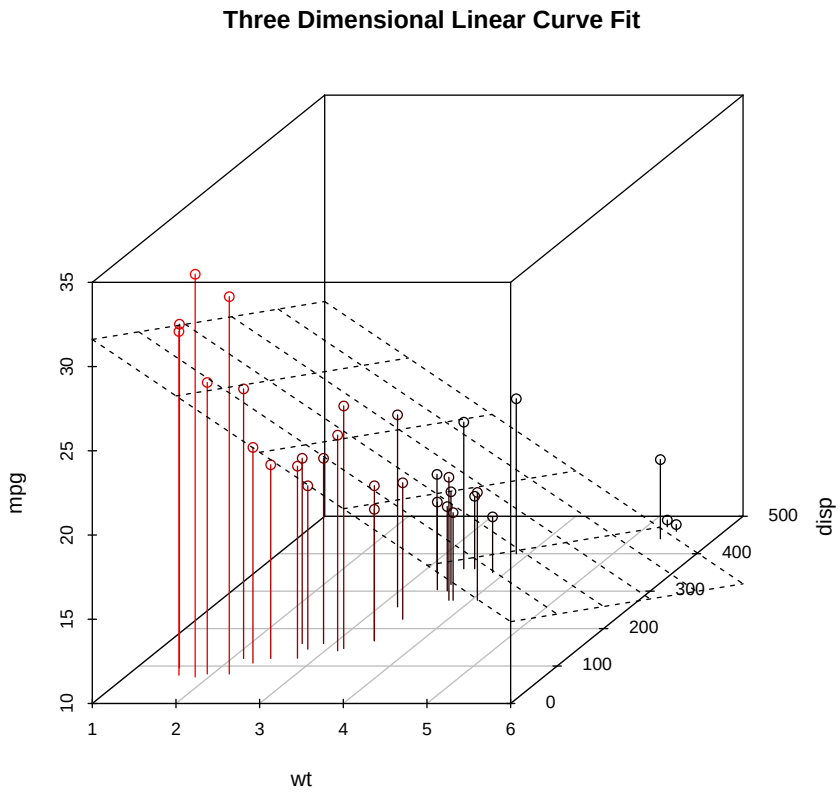


Figure 2.1.: Three dimensional linear curve fit.

polynomial degree of the model. For example a MPR-model of polynomial degree 2 with 3 input variables would have an input matrix consisting of rows of the form:

$$x_i = \begin{pmatrix} 1 & x_{i1} & x_{i2} & x_{i3} & x_{i1}^2 & x_{i1}x_{i2} & x_{i1}x_{i3} & x_{i2}^2 & x_{i2}x_{i3} & x_{i3}^2 \end{pmatrix} \tag{2.7}$$

Since each product of the basic inputs x_{i1}, x_{i2}, x_{i3} can be seen as a separate input variable, the optimization procedure stays the same. Nevertheless, with a polynomial degree of 2 the model would now be able to learn parabolic dependencies between input and output.
While raising a models polynomial degree gives it the ability to model arbitrarily complex dependencies, the models proneness to overfitting increases drastically. Models of a high enough polynomial degree can produce bumps at every data point in the training set. An example for a polynomial curve fit can be seen in Figure 2.2.

2.2.2. Decision Trees

Decision trees are a commonly used model to perform classification tasks in data analysis and machine learning. Their structure is intuitive, often plotted as a simple to understand

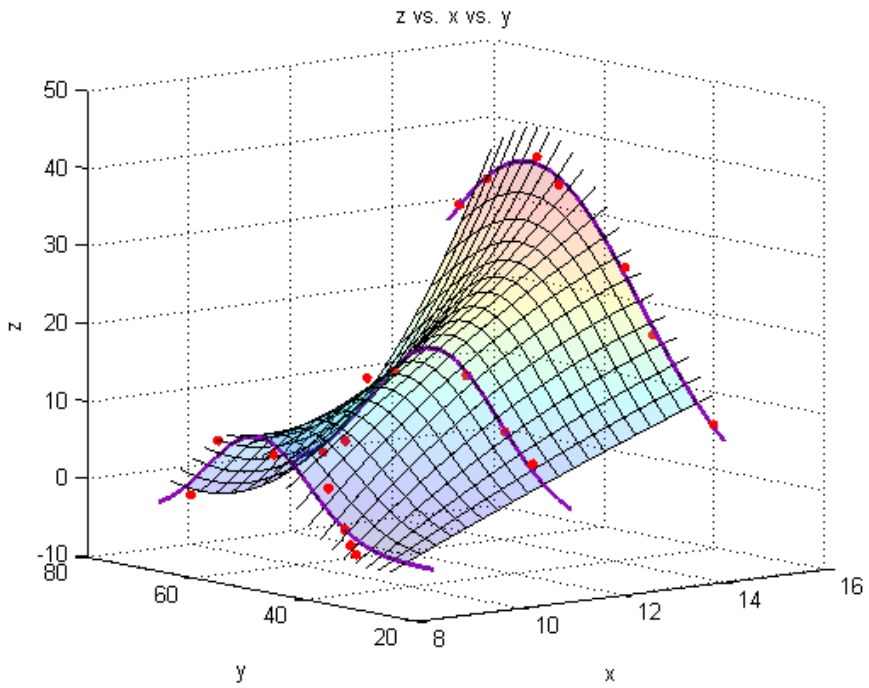


Figure 2.2.: Three dimensional polynomial curve fit [Kit12].

flowchart. Even after just being introduced to decision trees, people are able to grasp the core concepts and can quickly deduce important dependencies of the data at hand.

Decision Trees can be represented as tree graphs growing from top to bottom. They start at the root node, were the first split is conducted based on a single input variable. Each edge after the split of a node contains a mutually exclusive boolean query. A split can either be binary or non-binary, whereas binary trees are preferred due to their more simplistic nature. In fact, every tree containing non-binary splits can be translated into an equivalent binary one, simply by introducing more layers and nodes which depend on another boolean clause of the same input. With this, the decision tree propagates though every node and every corresponding boolean clause until it reaches its leaf nodes which determine the class, the input maps to. Figure 2.3 shows a plot of a decision tree constructed by observing casualties of the famous titanic incident, where *sibsp* denotes the number of siblings.

In contrast to other machine learning approaches, decision trees do not depend on conventional numeric cost functions which are to be minimized. Instead, the training is represented by a greedy search of all possible trees, generating a node and its split at each iteration. The optimal node at each iteration is determined by using the most influential input and a boolean query, separating the further path by two possibilities of value occupancy. For numerical inputs this generally translates to a simple less-than or greater-than relationship. Even though there are many approaches to conduct the training of decision trees, they all share the similarity of using a greedy heuristic search

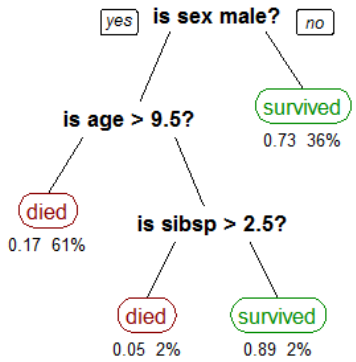


Figure 2.3.: Typical decision tree [Mil11]

trying find the best possible tree. The reason for that is the infinite search space of possible trees, with a complexity of $O(2^N)$ for binary ones of N depth. The difference of the training methods are simply in the heuristic used. There are evolutionary, information entropy maximization and simple comparison approaches to name a few.

Decision trees major advantages are their intuitive nature, their expressiveness, the ability to model non-linear data and their coping with less prepared data [Des11]. Expressiveness denotes the before-mentioned nature of trained trees to include a hierarchy from most to less dependent inputs. This can be especially useful to optimize other machine learning approaches under consideration, by incrementally leaving out the least significant inputs. Nevertheless, decision trees also have their drawbacks. One potentially significant drawback could be their inability to interpolate between data points used for training. This results from rectangular shapes of classification regions in the input data space [Kan11]. Interestingly even with their fundamental difference to other machine learning models, decision trees are prone to overfitting. In most extreme cases, the tree generates one leaf for each training input. With this, the decision tree will be able to perfectly classify each training sample used to generate the tree, while non-sample inputs will produce unpredictable behavior. To overcome this issue, so called *pruning* of the tree can is being carried out, essentially reducing it complexity an thus decreasing the overfitting.

2.3. The Scientific Compression Interface Library (SCIL)

SCIL is a library for the programming language C, which provides an interface for compression purposes focused on scientific data. It encapsulates many of the most modern and popular compression algorithms and offers them in a single includable module. The main goal of SCIL is to enable scientists to further harness the potential of compression regarding effective data size. To pursue this goal, SCIL is able to chain different algorithms and data conditioners together to provide optimal compression. In addition to the automatic decision process for the best available algorithm, it is hoped that SCIL will be able to improve overall compression rates significantly. SCIL's workflow consists of the creation of a context and the following execution of the compression. Providing a compression context, the buffer to compress, its dimensionality as well as the data type of the buffer's values, SCIL's compression procedure can be invoked. SCIL will then generate an algorithm chain to compress the buffer optimally. An automatic selection feature does not exist as of now and will be implemented after considering the results of this thesis. After a compression chain has been created, the compression commences, returning the compressed data to the user [SK16].

2.3.1. Context Creation

For the context creation, the user provides parameters and the data type of the buffer to compress. The parameters are internally called user-hints as they describe the intentions of the user. They consist of the following quantities:

- Absolute error tolerance
- Relative error tolerance
- Significant digits
- Significant bits
- Compression speed
- Decompression speed

The absolute error tolerance describes a parameter for lossy compression methods. With this, SCIL will only consider available algorithms which guarantee that values will differ after a compression decompression cycle by no more than the provided value. The relative error tolerance has a similar function to the absolute error tolerance. Algorithms considered will make sure, that values will not exceed or fall below the given percentage after a compression. Significant digits and bits are different flavors of providing a relative error tolerance, whereas these three metrics are interchangeable. Providing every relative parameter, the most constraining will be applied. Compression and decompression speed will limit the compression to algorithms which are known to yield the desired throughput. If none of the above parameters are used, SCIL will default to ignore them which results in maximum error tolerances and disregarding throughputs. Therefore a compression with default parameters would lead to a discarding of the data.

2.3.2. Algorithm Chain

After obtaining a context, the compression procedure generates the chain of data conditioners and algorithms. A compression chain consists of a pipeline of five distinct steps in the following order:

- General preconditioner (any type to any type)
- Data converter (any type to integer)
- Integer preconditioner (integer to integer)
- Data type specific compressor (any type into byte)
- Byte-wise compressor (byte to byte)

Such a pipelining of tasks allows for a dynamic and customizable approach for compression. Standard algorithms, included in SCIL, solely reside in either the Value- or Byte-wise compressor. With the help of preconditioners, data buffers can be adapted to further increase compression performances of these methods. To illustrate the role of the listed steps, an explanation and example of each one follows.

First Preconditioner. This step modifies the uncompressed data provides a buffer which can be translated back to the original data. Its goal is to increase compression rates of standard algorithms, making use of the specific data properties. First preconditioners of the same or different kind can be used in row at this step to obtain highly adapted data buffers. For example, the data could be translated to each values difference to its previous one. Thus, for smooth data, the obtained buffer will consist of values with a smaller standard deviation. The Abstol algorithm for example will perform better on such an optimized buffer.

Data Converter. The data converter translates data of arbitrary type to an integer representation. An example for this would be scalar quantization, which maps intervals of values into distinct ones [You10]. The Abstol algorithm for example makes use of the quantization data converter (see 2.3.3).

Integer Preconditioner. After the data converter, the option of applying one or more integer preconditioners exists. Integer preconditioners have the same purpose as the first ones, but are applied after the data converter. It is thinkable, that conditioners, applied after converting the data to an integer representation, could increase performances of compression methods. Though, as of now, SCIL does not provide specific methods which fall into the role of integer preconditioners.

Data Type Specific Compressor. Value-wise compressors are the first one to actually reduce the buffer size. They take in a buffer of distinct values and produce a byte buffer. Many lossy and lossless standard algorithms, included by SCIL, fall into this group. A custom value-wise compressor is the Swage algorithm as part of the Abstol method. It truncates a specified number of bits in each value and packs them tightly together in the

target buffer. As a result, the bit representation of values in the target buffer generally do not start and finish at whole bytes.

Byte-wise Compressor. Similar to the value-wise compressors, byte-wise ones effectively reduce the sizes of buffers. In contrast, they can accept input buffers of every data type, since they operate per byte and are generally lossless. The standard lossless algorithms GZIP and lz4 fall into this category.

2.3.3. Compression Algorithms

SCIL's goal of providing optimal compression requires a broad spectrum of algorithms with distinct properties. For this reason, SCIL aims to include many of modern and popular methods. Compression algorithms, currently available in SCIL are shortly explained in the following paragraphs.

Memcpy. The standard library's memcpy function is implemented as the trivial lossless compression procedure. Even though memcpy does not produce smaller buffers, it has a much higher throughput of data. For this reason it is used as a fallback method in case of strict performance parameters, provided by the user, which cannot be attained by other compression algorithms. Furthermore, it serves as a baseline for benchmarking SCIL's non-trivial compression methods.

Abstol. The abstol algorithm was developed alongside SCIL at the University of Hamburg. In the respective paper it is stated, that Abstol has a uniform scalar quantization method at its core [You10]. It is further described, that by separating the interval between the minimum and maximum value of the data into uniform regions, specified by the provided absolute error tolerance, a number of distinct bins is obtained in which each value resides. Each bin is then encoded by its index and used to recreate the original value in a lossy manner [SK16].

Sigbits. Like Abstol, the Sigbits method was also devised with the implementation of SCIL. In the original paper, it is explained, that Sigbits makes use of properties of the data values float representation and the provided relative error tolerance to generalize over all values. For example, if all values are positive, the sign bit of each value can be dropped and instead only be stored once, in the header of the compressed buffer. Furthermore, Sigbits quantizes the values' exponents and drops non-significant bits in the mantissa as specified by the provided relative error tolerance [SK16].

FPZIP. FPZIP is a lossless and lossy hybrid compression algorithm, devised and developed by Peter Lindstrom (2006) at the Lawrence Livermore National Laboratory. Lindstrom says its designed for high-throughput floating point compression. According to their paper, it functions by structurally iterating over all values, predicting newly encountered ones with the help of the Lorenz predictor [ILRS03], based on subset of already encoded data points. The actual value and the prediction is represented as

integers after dropping the least significant bits in case of lossy compression, storing their residuals separately. Afterwards, the residuals are entropy encoded by a specifically devised method [LI06].

ZFP. The ZFP compression method is the successor of FPZIP, also developed by Peter Lindstrom (2014). They state that ZFP is inspired by 2D image compression, separating data into blocks and applying different steps to process each block. According to the paper, these steps are the following ones, in order:

1. Alignment of values with common exponents
2. Conversation to fixed-point value representation
3. Application of an orthogonal block transformation
4. Ordering transform coefficients by magnitude
5. Encoding the coefficients

Lindstrom asserts, that the application of an orthogonal block transform makes use of a specifically devised method, outperforming the resembling DCT-II [YL95] approach regarding 3D data. The ZFP algorithm can be used for both, lossy and lossless compression, while its focus lays on the former of the two [Lin14].

GZIP. The famous an widely used GZIP program is a lossless method, which implements the DEFLATE algorithm [Deu96a] described by P. Deutsch (1996). Deutsch states, that the DEFLATE algorithm is a concatenation of the Lempel-Ziv [LZ77] and Huffman coding [Huf52] approaches [Deu96b].

LZ4fast. As an adaption to the Lempel-Ziv algorithm, LZ4, developed by Yann Collet (2011), focuses on compression and decompression speed while forfeiting compression ratios [Col11].

3. Design

This chapter illustrates the design concept of the whole experiment.

Section 3.1 gives and overview of how the experiment was conducted. For certain steps, references to following sections with more detailed explanations are provided.

All involved features for the machine learning task are presented in Section 3.2. The origin and meaning of each feature under consideration are explained.

Section 3.3 covers the generation of data buffers and subsequent gathering of training data for machine learning. It provides a subsection, explaining the concept and functioning of patterns.

A short description on feature selection is provided in Section 3.4. Here, the recognition of useful input features for the model training is depicted.

In Section 3.5, the strategy of training regression and classification models is presented.

A detailed explanation on the conducted model evaluation can be found in Section 3.6. Thereby, different kinds of errors for the models' fitness evaluation are introduced.

3.1. Methodology

The following list describes the rough procedure of the experiment.

1. Generating diverse set of multidimensional data patterns. A pattern defines how values are distributed in the n-dimensional data space; data can be created using a (potentially smooth) function. For more information on data patterns, see Section 3.3.1.
2. Analyzing the generated data for certain features using descriptive statistics. For this, data relevant characteristics are extracted and explored, i.e, the mean value and standard deviation. See Section 3.2 for a detailed view of all extracted features.
3. Creating training data by compressing the data and benchmarking this process. For this, a specifically developed tool is used, which creates the features and labels for the training data, allowing for an examination of possible dependencies of input and output features. Section 3.3 provides a more in-depth explanation.
4. Selection of relevant features. Here, the actual examination of dependencies is conducted by visually inspecting scatterplots of input versus output features. This suggests, which input parameters should be used to successfully apply machine learning approaches. A more detailed explanation can be found in Section 3.4 .
5. Training machine learning models with this data. After gathering the relevant information, the machine learning can commence. Common procedures like the

splitting of data into a training and validation set are used. Section 3.5 provides a more detailed view of the model training.

6. Evaluating the performances of the trained models. After the models are trained, their performances must be carefully evaluated and compared to each other. For this, well-defined metrics on the validation set are used to quantify the error (see 3.6).

3.2. Definition of Features

A *feature*, in the context of machine learning, is a variable, either provided to, or predicted by the model. Features, provided to a model are called *input features* and are used to predict values which represent an assignment for *output features*. Input features, in SCIL's context, can be further divided into *data characteristics*, which are inherent, quantitative descriptions of data buffers, and *user provided parameters*. User provided parameters on the other hand are arguments, users of SCIL can assign values to, influencing the compression in certain ways. The following paragraphs explain all features, measured and stored in the process of data generation. The suffixes **IF** and **OF** distinguishes input and output features.

Algorithm OF
The name or identifier of the used compression algorithm on the buffer, described by its data characteristics in this row. It is used in the classification task, answering which algorithm produces the best compression ratio for the current input features.

Byte Size IF
The number of bytes, the data buffer occupies, representing a possible input feature.

Total Number of Values IF
The total number of values in the data buffer, described by the current row of data characteristics. It is always one eighth of the byte size, since every buffer consists of double-precision-floating-point numbers in this experiment.

Dimensionality IF
The dimensional layout of the data, either 1-, 2-, 3-, or 4-D and could be used as an input feature. For multidimensional data, each dimension has the same amount of entries, such that the data represents a square, cube or hypercube.

Minimum Value IF
The minimum value, found in the compressible data buffer.

Maximum Value IF
The maximum value, found in the compressible data buffer.

Mean Value IF
The arithmetic mean of all values in the compressible data buffer.

Standard Deviation IF
The standard deviation of all values in the compressible data buffer.

Maximum Step Size IF
The maximum step between all values and their neighbors in each dimension. Calculated by iterating over all values, taking the absolute difference to their neighbors in each direction and storing the maximum encountered.

Absolute Error Tolerance IF
The maximum allowed absolute error tolerance for the compression task. It is determined for each buffer by a logarithmic random distribution in the interval $[2^{-13}, 2^2]$. Does not apply to lossless algorithms or algorithms based on relative error tolerance.

Relative Error Tolerance IF
The maximum allowed relative error tolerance for the compression task, in percent. It is determined for each buffer by a logarithmic random distribution in the interval $[2^{-10}, 2^4]$. Does not apply to lossless algorithms or algorithms based on absolute error tolerance.

Compression Throughput OF
The speed of the compression in megabytes per second. Measured by dividing the size of the uncompressed data by the time, the compression took.

Decompression Throughput OF
The speed of the compression in megabytes per second. Measured by dividing the size of the uncompressed data by the time, the decompression took.

Compression Ratio OF
The ratio of the uncompressed buffer's byte size versus the compressed buffer's byte size. Larger values represent a higher compression.

At this point, it is notable that some features which are used as parameters for the pattern generation (see 3.3) are also contained in the data for model training. Nevertheless, features that share the name of pattern parameters, are not necessarily occupied by the same value, due to the behavior of the compressible data generation. For example, the total number of values is directly dependent on the dimensionality of the data. The number of values on each axis is determined by the formula $N = \lfloor \sqrt[d]{n} \rfloor$. Here, n is the total number of values in the buffer as determined before the pattern generation and d is the dimensionality of the data. Since the result is rounded down, the actual number of values in the buffer is generally less than the predetermined amount.

The same holds true for the minimum and maximum value in the buffer. If the compressible data was generated with the random pattern, it is very likely that no value

will be that of the provided maximum. This is due to the reason, that the random pattern uses the rand()-function of C's standard library, scaling its output into the interval of the provided minimum and maximum arguments. The rand-function produces integer values in the interval between 0 and RAND_MAX, where the latter is simply a value dependent on the systems architecture. On the hardware, the data generation commenced on, RAND_MAX equals to $2^{31} - 1 = 2147483647$. Therefore, buffers are unlikely to contain the provided minimum or maximum argument as a concrete value.

3.3. Generation of Data

Samples of compressible data need to be present before the generation of training data can commence. For this reason, the specifically developed tool generates 10000 data buffers with varying size, containing double-precision-floating-point-numbers, by applying a random pattern of the ones presented in Section 3.3.1. The random number, defining the pattern to use is uniformly distributed, so that afterwards, every pattern was created approximately the same amount of times. Since patterns receive arguments for their creation, the concrete values have to be determined beforehand. These arguments include the minimum and maximum value of the data as well as up to two pattern specific parameters and are randomly determined for each data buffer. Table 3.1 shows the intervals and random distribution of all parameters for pattern generation.

Metric	Minimum	Maximum	Distribution
Pattern	—	—	uniform
No. of Data Points	2^8	2^{22}	logarithmic
Dimensionality	1	4	uniform
Minimum Arg.	-2^{14}	2^{14}	logarithmic
Maximum Arg.	-2^{14}	2^{14}	logarithmic
Argument 1	1	16	uniform
Argument 2	1	16	uniform

Table 3.1.: Arguments for Compressible Data Generation

The entry for the minimum and maximum argument represent their absolute value. It is equally possible for them to be negative or positive, while still being logarithmically distributed in the whole interval of $[-16384, 16384)$. From two determined values in this interval, the smaller one will be treated as the current minimum argument and the larger one as the maximum. For the constant pattern, where the maximum argument does not apply, only one value is determined and used as the minimum argument. Therefore, even for the constant pattern, a symmetric distribution is attained. Both other parameters are chosen from a uniformly distributed, random floating point variable in the interval $[1, 16]$. Their meaning is patten-specific, as described in Section 3.3.1. Values lower or higher than specified by the interval of possible assignments, in general, do not result in data which exhibits further, distinct characteristics for the machine learning task.

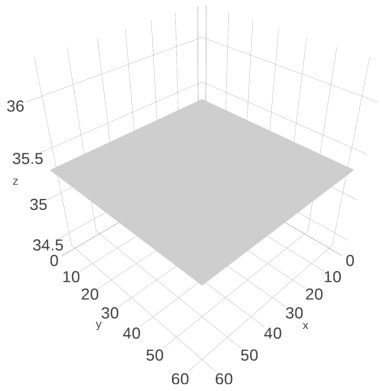
While generating the data buffers, the tool simultaneously gathers their characteristics and benchmarks the compression. In cases of lossy compression, the user provided inputs of absolute and relative error tolerances are required, which also represent input features. These are also randomly generated with a logarithmic distribution in the interval $[2^{-13}, 2^{1}]$ and $[2^{-10}, 2^{2}]$, respectively. Having provided these parameters, the compression commences producing benchmarks in terms of compression and decompression throughput as well as compression ratio for all algorithms. Therefore, each data buffer produces 8 lines, containing the specific input and output features. Finally, the resulting data for every buffer and every algorithm is stored as a comma-separated-values-file, ready for analysis.

3.3.1. Patterns

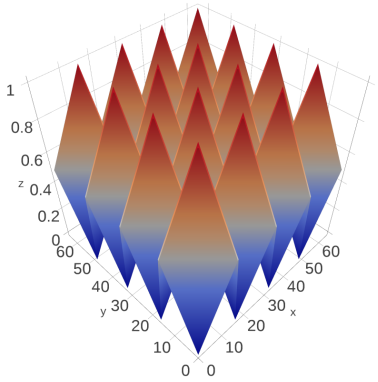
Generating data to compress, for the purpose of benchmarking each algorithm, has the goal of providing diverse data. Diversity of the data — in this context — is defined by the uniform filling of the training sets input space, regarding data characteristics. With more diverse data, machine learning has better spaced points of reference for training, thus producing superior models. For this reason, the compressible data generation makes use of diverse patterns to create data. The patterns used for data generation include:

- Constant
- Steps
- Random
- Sinusoidal
- Perlin-Noise

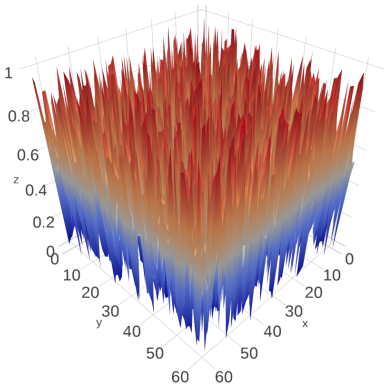
Each pattern constructs structurally different data as can be seen in Figure 3.1.



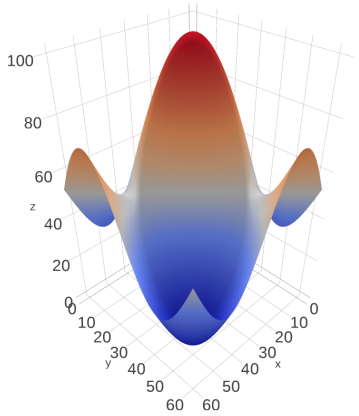
(a) Constant, value 35.3



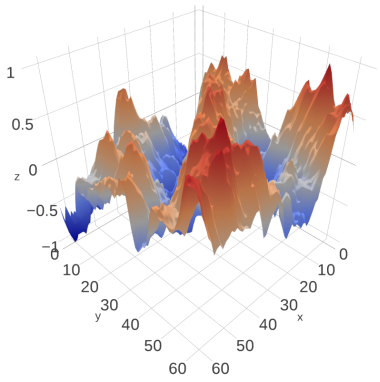
(b) Steps, Argument 100



(c) Random, 0 to 1



(d) Sinusoidal, Scale 1, values 0 to 100



(e) Perlin Noise, Scale 3 and 6 passes

Figure 3.1.: 3-Dimensional data generated with different patterns

Pattern generators can be further controlled by providing different parameters. In the following paragraphs, each pattern's properties and parameters are explained.

Constant. The Constant pattern is the most simple one. Providing a data buffer of arbitrary length and dimension, it sets all values to the provided minimum parameter.

Steps. The Steps pattern makes use of a provided "stepsize"-parameter to construct linear, repeated data. The "stepsize"-parameter denotes the size of repeated sections. For two dimensional data, this results in a straight line which raises from the provided minimum to the maximum in the given step size. Afterwards, it repeats by resetting the line back to the minimum. For multidimensional data, and analog plane is generated, which repeats after step size on each axis.

Random. This pattern creates uniformly distributed random data. Every generated value will be placed in the interval of the minimum and maximum parameter.

Sinusoidal. Sinusoidal patterns generates by applying one or more sinus curves on each input dimension. Available parameters are the base frequency and the count of additive smaller sinus curves. The base frequency is measured per buffer size in each dimension. A base frequency of one would result in one full period of the first or most significant sinus curve over each axis. Each further sinus curve will have a two fold frequency and half the amplitude of the previous one. Finally, the data buffer is normalized to fit between the provided minimum and maximum.

Perlin Noise. The Perlin Noise pattern generates smooth but random data from a seed. Parameters for perlin noise are similar to the ones of the sinusoidal pattern in their functioning. The provided base scale determines the size of the patterns first pass. The other parameter is the number of passes, where the intensity is halved and the scale doubled in comparison to the previous pass.

3.4. Selection of Features

By providing plots, showing the dependencies of input and output features, an assessment of the input features is obtained. With this, an estimate is made for the inputs relevance, considering the machine learning task at hand. For this reason, every combination of output and input features is illustrated with the help of scatterplots and boxplots (for the dimensionality input feature). These plots can be inspected in Appendix B. A measure of relevance for the input data are patterns of dots in the graphics. Dots uniformly distributed along the x-axis suggest no correlation of the input and output feature and are therefor of no interest to model training. If dots are, i.e., culminating at a linear or exponential curve, a strong correlation is observed. In these cases, the output at the y-axis is highly correlated with the input feature, enabling machine learning to interpret the data correctly.

3.5. Model Training

For the evaluation, a multitude of models are trained per algorithm and output feature. Since users of SCIL can provide a minimum compression and decompression throughput, which are dependent on input features and generally unknown beforehand, a regression of these values is needed. Thus, two models will be trained for each algorithm to predict its throughputs based on the provided input features.

For the classification task, each machine learning approach under consideration is trained upon all algorithms, since the output should be a preference of each algorithm. Theses preference values are in the interval of $(0, 1)$, where the highest value denotes the best corresponding algorithm, as estimated by the model.

All in all, 27 models were fitted, evaluated and compared to one another, producing a detailed portrait of their performances. Models under consideration are the Linear, Decision Tree, and the Mean Value approach. The former two where introduced in detail in Section 2.2.1 and 2.2.2 respectively. They represent serious contenders for the role of SCIL's automatic compression decider. In contrast, the Mean Value Model functions as a baseline for comparison purposes of the other models. As the name suggests, the Mean Value approach will average the compression and decompression throughputs for the regression task. For the classification, its prediction simply defaults to the algorithm which produced the best compression ratio most of the time. Therefore, when comparing other, serious models with this one, an assessment is obtained whether the model is slightly or significantly better than the trivial approach.

The Linear models depend on a structural hyperparameter, denoting the polynomial degree used for the fitted curve. To provide a more in-depth analysis of model suitability, possible assignments of the hyperparameter were iterated over, producing a different model in each step. Comparing the different fitness measurements of training and evaluation set predictions (see Section 3.6), the best model of its kind is determined and used for the cross-model comparison. Linear models of degree 1 to 3 were evaluated.

Having obtained the best linear model, the the different errors of all predictions on the evaluation set will be used as a metric to compare their regression results. For classification, the percentage of correct predictions of the best suitable algorithms represents a reasonable measurement.

3.6. Model Evaluation

Trained regression models are measured by using the following error metrics:

- *Root-Mean-Squared-Error* (RMSE)
- *Mean-Absolute-Error* (MAE)
- *Relative-Squared-Error* (RSE)
- *Relative-Absolute-Error* (RAE)

The definition of all error metrics can be found in Appendix A.2. All presented error metrics follow the rule: the smaller the value, the better the fit. While RMSE and MAE represent absolute errors in terms of the output feature analyzed, the RSE and RAE can be used to assess a models fitness relative to the mean value. This results in normalized values, where zero means a perfect fit and one a fit similar in usefulness as the Mean Value Model. Values larger than one suggest a fitness worse than that of the Mean Value Model, suggesting an utterly worthless fit. Even though the RSE and RAE suffice for the task of examining a models fitness, the RMSE and MAE values can be used for context driven comparison of models. The difference between RSME and MAE (or RSE and RAE) is, that the former — by squaring the residuals — gives more weight for rough errors in the calculation, while the latter weights all errors the same. Therefore, large values for RMSE and smaller ones for MAE indicate few strong errors, while the opposite suggest many small ones.

For classification, simply the percentage of correct predictions of the best suitable algorithms represents a reasonable measurement.

4. Evaluation

This chapter provides results of the experiment which are interpreted in Chapter 5.

Section 4.1 presents and interprets the results of the data visualization. Each metric is analyzed, extracting important information for the machine learning process.

The result of the machine learning process are illustrated in Section 4.2. Different kind of error metrics of the Mean Value, Linear Model and Decision Tree approaches are presented and analyzed.

4.1. Data Inspection

In Appendix B, graphs are shown, comparing each training data input to every output for all algorithms. The graphs are all logarithmically on the x-axis to cope with the exponential nature of the data generation used. An exception for this is the discrete dimensionality input. The y-axis for all graphs showing the Compression Ratio is also logarithmically scaled to better visualize strong outliers.

- Output feature in comparison to the **total number of values** in the buffer can be inspected in Figures B.1, B.2 and B.3. These plots are especially remarkable because of the complex patterns they show. Such complex distributions in the graphs are a strong indicator for dependency and therefore, the total number of values should be included in the machine learning process as an input feature.
- Figures B.4, B.5 and B.6 show the outputs in dependency on the **dimensional layout** of the data. Here, most Algorithms do not display significant patterns with the exception of ZFP when using relative error tolerance parameters. The throughput of ZFP-reltol for compression as well as decompression interestingly peeks at 2-dimensional data even though the algorithm is designed for 3D-data. Regarding the compression ratio of 3D-data, ZFP-reltol reliably reaches factors of 200, outperforming other algorithms immensely, except for a few outliers. In addition, GZIP and LZ4fast show minor dependencies. The throughput of GZIP in both — compression and decompression — rises with higher dimensional data. Its compression ratio is significantly higher for multidimensional (2D,3D and 4D) data, with no obvious differences between those layouts. LZ4fast dos not display an obvious change in throughput with differing dimensional layout, but shows a very similar structure to GZIP, regarding its compression ratio. Since there are a few algorithms, which performance in respect to speed and quality of compression is dependent on the dimensionality of the data, this metric should also be considered for the machine learning task.

- Plots of the **mean value** metric are shown in Figure B.7, B.8 and B.9. It should be noted that the x-axis of the plot is logarithmically scaled. There are about the same number of points in the negative spectrum with an almost symmetric distribution. Therefore, the plots should be considered as the absolute value of averages. Most algorithms show no dependency between their performance and the average of the data with the exception of algorithms based on absolute error tolerance. The throughput of Abstol and ZFP-Abstol decline for large averages of the data. Considering the compression ratio, a very similar behavior is observed.
- The algorithm performances regarding **standard deviation** of the data buffers can be inspected in Figure B.10, B.11 and B.12. All plots of the standard deviation are almost identically to their counterparts of the average metric regarding their structure.
- Figure B.13, B.14 and B.15 illustrate the compression performances, subject to the **maximum step** metric. This metric is also structurally identical to the average and standard deviation metrics. A further illustration of those metrics' inter-dependencies can be seen in Figure B.22. As expected, the plot shows a strong linear relationship of the metrics. Nevertheless, many outliers of this linearity can be observed, which suggests that a simultaneous inclusion of these metrics could benefit the training process.
- The performances of algorithms considering the **absolute error tolerance** metric can be seen in Figure B.16, B.17 and B.18. Unsurprisingly, algorithms which do not depend on a clients assignment of an absolute error tolerance do not experience changes in their performances. Abstol and ZFP-Abstol on the other hand are slightly faster in compression as well as decompression for higher provided tolerances. Though, ZFP-Abstol increase in throughput seemingly only applies for some outliers and is rather constant in the normal case. Regarding compression ratio, Abstol and ZFP-Abstol behave as expected, showing a rise for higher tolerances.
- The last potentially influential metric to analyze is the **relative error tolerance** as show in Figure B.19, B.20 and B.21. Surprisingly, no algorithm shows correlated behavior for different metric values. Not even the compression ratios of the Sigbits or ZFP-reltol algorithm seem to depend on the value of the relative error tolerance parameter. In this case, it is very unlikely, that there are in fact no observable dependencies. The reason for this behavior is unknown but could be due to a bug in either the data generation, the SCIL compression API or in the data evaluation software. Using this metric for machine learning will most likely not contribute to better models.

4.2. Model Fitness

The following subsections include tables, illustrating the results of the models' fitness assessments. For this, the RMSE, MAE, RSE and RME values for each model and algorithm are presented. Section 4.2.1 contains the measurements for the regression task on the compression throughput of algorithms. Equivalently, Section 4.2.2 presents the fitnesses of models predicting all algorithms' decompression throughputs. Finally, the percentages of correct classifications of algorithms, producing the best compression ratios, are shown in Section 4.2.3.

4.2.1. Fitness on Compression Throughput

Tables 4.1, 4.2, 4.3 and 4.4 show a clear favorite for estimating the compression throughput. In almost all cases, Decision Trees perform significantly better than the Mean Value and even the Linear Model. Exceptions include GZIP and FPZIP in which cases Decision Trees are about equally fit in comparison to Linear Models. Though in these case both non-trivial models do not show significantly better errors, than the Mean Value approach. The performance of Decision Trees is especially superior to Linear Models regarding the Compression Throughputs of both ZFP-algorithms. Nevertheless, the clearly better fitness of decision trees in comparison to the other models is still unsuitable for SCIL's compression decider feature. A RAE value of 0.5 indicates a 50% better prediction, than the Mean Value approach. To be truly reasonable as a candidate for SCIL's feature, values as low as 0.1 as desired, in which case Models start to be considered "reliable". Since the Decision Trees best RAM value is merely 0.5 and a lot more for other algorithms than ZFP, their performance is still insufficient.

	Memcpy	Abstol	GZIP	Sigbits
Mean Value	3381.75	43.99	50.65	27.87
Linear Model	3069.71	38.01	49.16	27.32
Decision Tree	2322.48	29.68	49.86	24.78

	FPZIP	ZFP-Abstol	ZFP-reltol	LZ4fast
Mean Value	28.88	404.16	211.61	2326.64
Linear Model	27.92	380.19	147.26	2072.87
Decision Tree	27.05	202.50	120.67	1891.52

Table 4.1.: RMSEs for Compression Throughput

	Memcpy	Abstol	GZIP	Sigbits
Mean Value	2694.77	33.94	42.16	20.36
Linear Model	2286.29	27.69	39.74	19.91
Decision Tree	1586.80	21.89	41.18	17.55

	FPZIP	ZFP-Abstol	ZFP-reltol	LZ4fast
Mean Value	20.44	170.89	168.37	1919.32
Linear Model	19.13	158.33	111.76	1700.62
Decision Tree	18.66	87.87	85.56	1420.34

Table 4.2.: MAEs for Compression Throughput

	Memcpy	Abstol	GZIP	Sigbits
Mean Value	1.0000	1.0000	1.0000	1.0000
Linear Model	0.8305	0.7472	0.9417	0.9642
Decision Tree	0.4754	0.4555	0.9686	0.7929

	FPZIP	ZFP-Abstol	ZFP-reltol	LZ4fast
Mean Value	1.0000	1.0000	1.0000	1.0000
Linear Model	0.9343	0.8850	0.4845	0.7946
Decision Tree	0.8769	0.2511	0.3253	0.6616

Table 4.3.: RSEs for Compression Throughput

	Memcpy	Abstol	GZIP	Sigbits
Mean Value	1.0000	1.0000	1.0000	1.0000
Linear Model	0.8561	0.8152	0.9447	0.9994
Decision Tree	0.5942	0.6446	0.9788	0.8809

	FPZIP	ZFP-Abstol	ZFP-reltol	LZ4fast
Mean Value	1.0000	1.0000	1.0000	1.0000
Linear Model	0.9346	0.9434	0.6601	0.8864
Decision Tree	0.9115	0.5236	0.5054	0.7403

Table 4.4.: RAEs for Compression Throughput

4.2.2. Fitness on Decompression Throughput

Tables 4.5, 4.6, 4.7 and 4.8 show results, very similar to the ones in Section 4.2.1. Again, Decision Trees vastly outperform the other models regarding their fitness for predicting decompression throughputs. A minor difference is that Linear Models' performances are not even for GZIP and FPZIP similar to those of Decision Trees. Interestingly, the superior model is slightly better performing while predicting decompression in comparison to compression throughputs, as measured by the mean of the RAE values. Though this slightly better fitness does not suffice by far to convince of the practical use of this model.

	Memcpy	Abstol	GZIP	Sigbits
Mean Value	2175.44	78.01	296.76	23.06
Linear Model	1930.33	67.14	283.21	21.97
Decision Tree	1355.09	47.92	276.33	17.79

	FPZIP	ZFP-Abstol	ZFP-reltol	LZ4fast
Mean Value	24.18	466.56	325.53	2923.56
Linear Model	22.87	431.50	235.16	2687.63
Decision Tree	21.13	269.10	170.49	2058.85

Table 4.5.: RMSEs for Decompression Throughput

	Memcpy	Abstol	GZIP	Sigbits
Mean Value	1794.73	62.84	233.78	17.29
Linear Model	1535.28	50.47	219.22	16.19
Decision Tree	1011.40	36.87	213.55	12.60

	FPZIP	ZFP-Abstol	ZFP-reltol	LZ4fast
Mean Value	18.04	237.46	257.73	2544.42
Linear Model	16.65	207.41	186.95	2215.19
Decision Tree	15.60	142.24	126.49	1621.52

Table 4.6.: MAEs for Decompression Throughput

	Memcpy	Abstol	GZIP	Sigbits
Mean Value	1.0000	1.0000	1.0000	1.0000
Linear Model	0.7916	0.7420	0.9108	0.9105
Decision Tree	0.3901	0.3779	0.8671	0.5972

	FPZIP	ZFP-Abstol	ZFP-reltol	LZ4fast
Mean Value	1.0000	1.0000	1.0000	1.0000
Linear Model	0.8947	0.8554	0.5224	0.8472
Decision Tree	0.7633	0.3327	0.2746	0.4972

Table 4.7.: RSEs for Decompression Throughput

	Memcpy	Abstol	GZIP	Sigbits
Mean Value	1.0000	1.0000	1.0000	1.0000
Linear Model	0.8680	0.8023	0.9393	0.9572
Decision Tree	0.5718	0.5862	0.9150	0.7445

	FPZIP	ZFP-Abstol	ZFP-reltol	LZ4fast
Mean Value	1.0000	1.0000	1.0000	1.0000
Linear Model	0.9241	0.8816	0.7247	0.8763
Decision Tree	0.8657	0.6046	0.4903	0.6415

Table 4.8.: RAEs for Decompression Throughput

4.2.3. Classification Fitness

The evaluation of classification problems also show that in this case, Decision Trees are superior to the Mean Value and Linear Models. Nevertheless, the rise of correct predictions from the least to most performing model is only 3.9%. Such a small increase is not enough to promote the use of Decision Trees in SCIL's compression decider feature. The decision tree producing these results can be seen in Figure 4.1.

Model	Correct Classifications (%)
Mean Value	65.23
Linear Model	67.50
Decision Tree	69.13

Table 4.9.: Correct Classifications Percentage

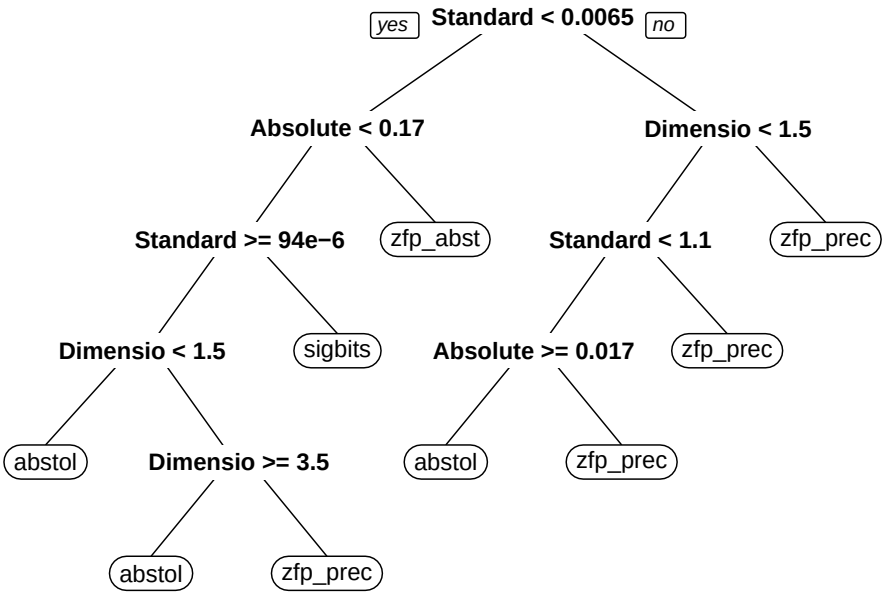


Figure 4.1.: Decision Tree for Classification

5. Summary and Conclusion

The purpose of this thesis was to evaluate machine learning approaches for their suitability in predicting throughputs of compression algorithms and resulting compression ratios. The reason for this research was to find a suitable machine learning model adequately executing the task of choosing the best possible algorithm for data compression. This model — if it performs satisfyingly — would be integrated in the Scientific Compression Interface Library (SCIL) to abstract the users choice of the best compression algorithm. It was shown, that if a successful model was found, users of SCIL would benefit from the trained model by further exploiting the unused potential of data compression.

Chapter 1 provided and introduction to this theses, explaining its motivation and goals. It provided the reader with an understanding of the fundamental problem of the manual decision for compression algorithms. Furthermore, the reader was oriented through this thesis by a short presentation of upcoming chapters. Afterwards, a background of compression and machine learning in general was provided in Chapter 2, preparing the concepts used in the design and evaluation. The fundamental work on compression algorithms and compression in general were shown as well as a short summary of the history of compression. Additionally, machine learning was introduced in general and more specifically in scope of Linear Models and Decision Trees. Following, Chapter 3 illustrated the design and methodology of this work, explaining key concepts and the experiments realization in detail. Finally the evaluation of the results was provided in Chapter 4, visualizing and analyzing potentially relevant features for machine learning and presenting the experiment's outcome.

The results of this work, as depicted in Section 4.2, strongly suggest that Decision Trees are better suited than Linear Models for the task at hand. Decision Trees performed better in almost all instances of throughput prediction regarding compression and decompression. The only exceptions, was the prediction of the GZIP algorithm's compression throughput in which case the Linear Model's fitness was marginally better than the Decision Tree's. For the classification task — which algorithm will result in the best compression ratio — Decision Trees also had a higher percentage of correct predictions. However, the performances of Decision Tree may be better than those of Linear Models in this context but their absolute fitness is not nearly sufficient. To reasonably consider a machine learning model for this task, it should have no higher RAE values than 0.1. The correct classification percentage of Decision Trees was shown to be at 69.13% — only 3.9% higher than that of the Mean Value Model. Therefore the answer of this work is, that none of the trained methods should be used for SCIL's automatic compression decider, as they are.

5.1. Future Work

To further pursue the goal of finding an adequate model for the task of SCIL's decider feature, additional research with prospect of success can be done. The evaluations results suggest two problems with the training of Decision Trees or Linear Models:

1. The feature space is too complex, regarding the dependencies of inputs and outputs.
2. There are unknown input features which would result in a large performance boost, when included in the training process.

For point one, the solution is to evaluate more powerful models such as neural networks, support vector machines or random forests. The drawback of this is the computational intensiveness of these models training and should be met with more powerful hardware.

If the second point turns out to be true, the solution is to measure additional input features, until they are able to predict the output satisfyingly. Though, finding out, what those unknown input features are and how they are defined/measured could be a hard problem.

All in all, further research on this topic is encouraged and could provide a model with sufficient prediction accuracy for the task of automating the decision for the best possible compression algorithm.

Bibliography

[Agu12] Paula Aguilera. Comparison of different image compression formats. https://homepages.cae.wisc.edu/~ece533/project/f06/aguilera_rpt.pdf, 2012. [Online; accessed 05-October-2016].

[Col11] Yann Collet. LZ4 explained. http://fastcompression.blogspot.de/2011/05/lz4-explained.html, May 2011. [Online; accessed 04-November-2016].

[Des11] Bala Deshpande. 4 key advantages of using decision trees for predictive analytics. http://www.simafore.com/blog/bid/62333/4-key-advantages-of-using-decision-trees-for-predictive-analytics, 2011. [Online; accessed 20-September-2016].

[Deu96a] P. Deutsch. DEFLATE Compressed Data Format Specification version 1.3. https://tools.ietf.org/pdf/rfc1951.pdf, May 1996. [Online; accessed 03-November-2016].

[Deu96b] P. Deutsch. GZIP file format specification version 4.3. https://tools.ietf.org/pdf/rfc1952.pdf, May 1996. [Online; accessed 04-November-2016].

[fNR12] CERN European Organization for Nuclear Research. Processing: What to record? http://cds.cern.ch/record/1997399, 2012. [Online; accessed 05-October-2016].

[HL11] M. Hilbert and P. Lopez. The World's Technological Capacity to Store, Communicate, and Compute Information. *Science*, 332(6025):60–65, feb 2011.

[Huf52] David Huffman. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE*, 40(9):1098–1101, sep 1952.

[HWK+13] Nathanael Hübbe, Al Wegener, Julian Kunkel, Yi Ling, and Thomas Ludwig. Evaluating Lossy Compression on Climate Data. In Julian Martin Kunkel, Thomas Ludwig, and Hans Werner Meuer, editors, *Supercomputing*, number 7905 in Lecture Notes in Computer Science, pages 343–356, Berlin, Heidelberg, 06 2013. Springer.

[ILRS03] Lawrence Ibarria, Peter Lindstrom, Jarek Rossignac, and Andrzej Szymczak. Out-of-core compression and decompression of large n-dimensional scalar fields. *Computer Graphics Forum*, 22(3):343–348, sep 2003.

[JJS93] N. Jayant, J. Johnston, and R. Safranek. Signal compression based on models of human perception. *Proceedings of the IEEE*, 81(10):1385–1422, 1993.

[KA10] S. R. Kodituwakku and U. S. Amarasinghe. Compression of Lossless Data Compression Algorithms for Text Data. In *Indian Journal of Computer Science and Engineering*, number 4, pages 416–425, Tamil Nadu, India, 12 2010. Engg Journals Publications. [Online; accessed 04-October-2016].

[Kan11] Mehmed Kantardzic. *Data Mining : Concepts, Models, Methods, and Algorithms*. Wiley-IEEE Press, Somerset, US, 2 edition, 2011. [Online; accessed 20-September-2016].

[Kit12] John Kitchin. Colors, 3D Plotting, and Data Manipulation. `http://matlab.cheme.cmu.edu/2012/06/19/colors-3d-plotting-and-data-manipulation/`, 2012. [Online; accessed 20-September-2016].

[Kun16] Julian Kunkel. AIMES — Advanced Computation and I/O Methods for Earth-System Simulations. `https://wr.informatik.uni-hamburg.de/research/projects/aimes/start#increase_storage_efficiency`, 2016. [Online; accessed 30-September-2016].

[LI06] P. Lindstrom and M. Isenburg. Fast and Efficient Compression of Floating-Point Data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1245–1250, sep 2006.

[Lin14] Peter Lindstrom. Fixed-Rate Compressed Floating-Point Arrays. *IEEE Transactions on Visualization and Computer Graphics*, 20(12):2674–2683, dec 2014.

[LZ77] A. Lempel and J. Ziv. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, may 1977.

[Mil11] Stephen Milborrow. An example of a CART classification tree. `https://upload.wikimedia.org/wikipedia/commons/f/f3/CART_tree_titanic_survivors.png`, 2011. [Online; accessed 20-September-2016].

[Nat88] National Aeronautics and Space Administration. Proceedings of the Scientific Data Compression Workshop. `http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19890012961.pdf`, 1988. [Online; accessed 05-October-2016].

[Nem11] Evan Nemerson. Squash Compression Benchmark. `https://quixdb.github.io/squash-benchmark/#results`, 2011. [Online; accessed 05-October-2016].

[Sch16] Armin Schaare. Adaptive Selection of Lossy Compression Algorithms Using Machine Learning. 09 2016.

[SK16] Armin Schaare and Julian Kunkel. SCIL - Scientific Compression Interface Library. 04 2016.

[Wal91] Gregory K. Wallace. The JPEG still picture compression standard. *Communications of the ACM*, 34(4):30–44, apr 1991.

[Wel84] Terry A. Welch. A Technique for High-Performance Data Compression. `https://www.cs.duke.edu/courses/spring03/cps296.5/papers/welch_1984_technique_for.pdf`, 1984. [Online; accessed 04-October-2016].

[Wol02] Stephen Wolfram. *A New Kind of Science*. Wolfram Media, 2002.

[Wor03] World Meteorological Organization. Introduction to GRIB Edition1 and GRIB Edition 2. `http://www.wmo.int/pages/prog/www/WMOCodes/Guides/GRIB/Introduction_GRIB1-GRIB2.pdf`, june 2003. [Online; accessed 15-November-2016].

[YL95] Boon-Lock Yeo and Bede Liu. Volume rendering of DCT-based compressed 3D scalar data. *IEEE Transactions on Visualization and Computer Graphics*, 1(1):29–43, mar 1995.

[You10] Yuli You. *Audio Coding: Theory and Applications*. Springer, 2010.

[Zie13] Achim Zielesny. *From Curve Fitting to Machine Learning: An Illustrative Guide to Scientific Data Analysis and Computational Intelligence (Intelligent Systems Reference Library)*. Springer, 2013.

Appendices

A. Additional Machine Learning Information

A.1. Training Data

Each data point d_i, is defined in the way

$$d_i = (\boldsymbol{x}_i, \boldsymbol{y}_i, \boldsymbol{\sigma}_i)$$

where $i \in \mathbb{N}$ is the index of the data point, $\boldsymbol{x}_i \in \mathbb{R}^n$ is the argument value, $\boldsymbol{y}_i \in \mathbb{R}^m$ is the dependent value and $\boldsymbol{\sigma}_i \in \mathbb{R}^m$ is the statical error of $\boldsymbol{y}_i$ in the form of its standard deviation. Often, statistical errors are not available in the data or disregarded in used machine learning approaches, in which case they are simply omitted. Such a reduced data point is called an input/output- or simply *I/O pair*, where many of them form a *data set* upon which machine learning models can be trained. For a model function f being able to be trained on a data set, it needs the same dimensionality for its input- as well as output values.

$$f : \mathbb{R}^n \to \mathbb{R}^m$$

Feeding all inputs $\boldsymbol{x}_i$ from the data set into the model, produces outputs which can be compared to the actual outputs of the data set. This allows for a fitness evaluation of the trained machine learning model.

A.2. Qualitative Machine Learning Evaluation

Qualitative measurement, regrading residuals, to illustrate the fitness of models are the metrics in Section 3.6. They are defined in the following way:

$$RMSE = \sqrt{\frac{1}{k}\sum_{i=1}^{k}\left(\boldsymbol{y}_i - f(\boldsymbol{x}_i)\right)^2}$$

$$MAE = \frac{1}{k}\sum_{i=1}^{k}\left|\boldsymbol{y}_i - f(\boldsymbol{x}_i)\right|$$

$$RSE = \frac{\sum_{i=1}^{k}\left(\boldsymbol{y}_i - f(\boldsymbol{x}_i)\right)^2}{\sum_{i=1}^{k}\left(\boldsymbol{y}_i - \bar{\boldsymbol{y}}\right)^2}$$

$$RAE = \frac{\sum_{i=1}^{k}\left|\boldsymbol{y}_i - f(\boldsymbol{x}_i)\right|}{\sum_{i=1}^{k}\left|\boldsymbol{y}_i - \bar{\boldsymbol{y}}\right|}$$

Where f is the model function, $\boldsymbol{y}_i$ is the output, $\boldsymbol{x}_i$ the input of the i-th data point, k is the number of data points and $\bar{\boldsymbol{y}}$ is the mean value of the all outputs.

A.3. Over- and Underfitting

To determine, whether a model is over- or underfitted, two different approaches can be pursued, depending whether statistical errors are present in the data set for training (see A.1).

If statistical errors are present in the training sample, they can be compared to the models residuals when processing the samples inputs. *Residuals* are simply the differences between each given output in the training set and the models prediction, based on the corresponding input. If the residuals are higher than the statistical errors of the training set, underfitting most likely occurred. In addition, residuals in the case of an underfitted model are often forming distinct shapes along one or multiple dimensions of the input. Such shapes of the residuals can be indicators for features, the models function misses. For example, if residuals form a parabolic shape along one of its input-axis, a ax^2 term in the models function could be missing.

In the case of an overfitted model, residuals would simply decline to levels way below the data's statistical errors, through the course of training.

If statistical errors are not present in the training data, underfitting as well as overfitting can still be diagnosed. For this, the available data is split into a training and an evaluation set. Afterwards, the model will only be trained by feeding the data of the training set into the learning algorithm. Then, residuals are computed for both, the training and

evaluation set. Underfitted models in this case will result in unacceptably high residuals of the training set, since the models structure itself forbids a good fit. For overfitted models, the residuals for the training set would be almost zero, while the ones of the evaluation set will be unreasonably high. This is because overfitted models do not generalize data well. The specific inputs of the training data are learned almost perfectly, but input values differing slightly, as contained in the data of the evaluation set, produce strongly deviating results.

A.4. Complexity and Resources

In addition to the problem of overfitting, complex models consume much more resources while training, in terms of time and memory space. Generally the trainings time and space complexity rises polynomial, i.e., $O(n^2)$ or $O(n^3)$) with its number of hyperparameters. Thus, too complex models can be very costly for scientific or commercial endeavors. For this reason, it is encouraged to use the most simplistic model, regarding number of hyperparameters, which provides a satisfying result. As a rule of thumb, the simplest model, which performs adequately, is considered the best one.

B. Data Visualization

B.1. Total Number of Values

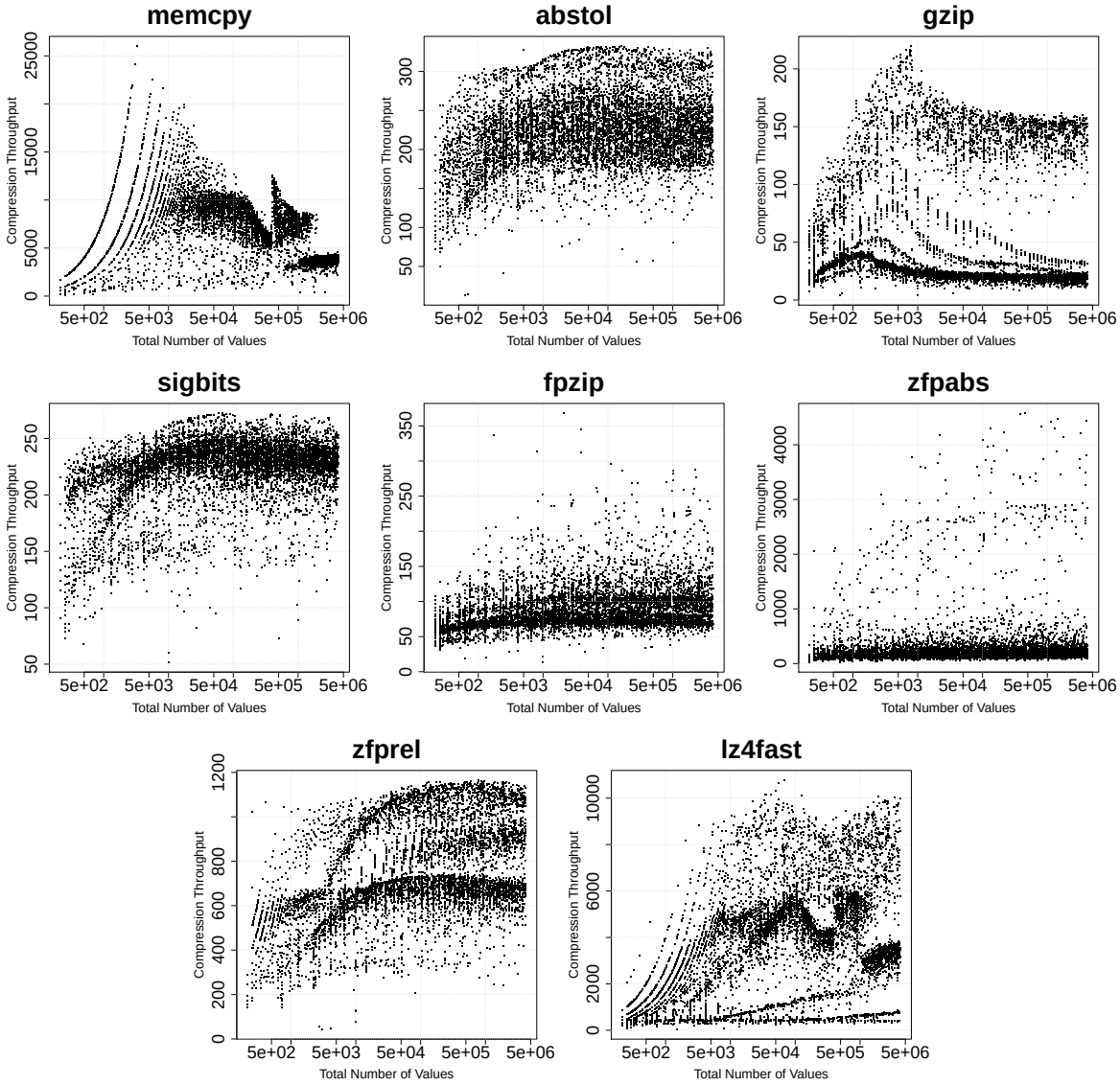


Figure B.1.: Dependencies of Compression Throughput and Number of Values

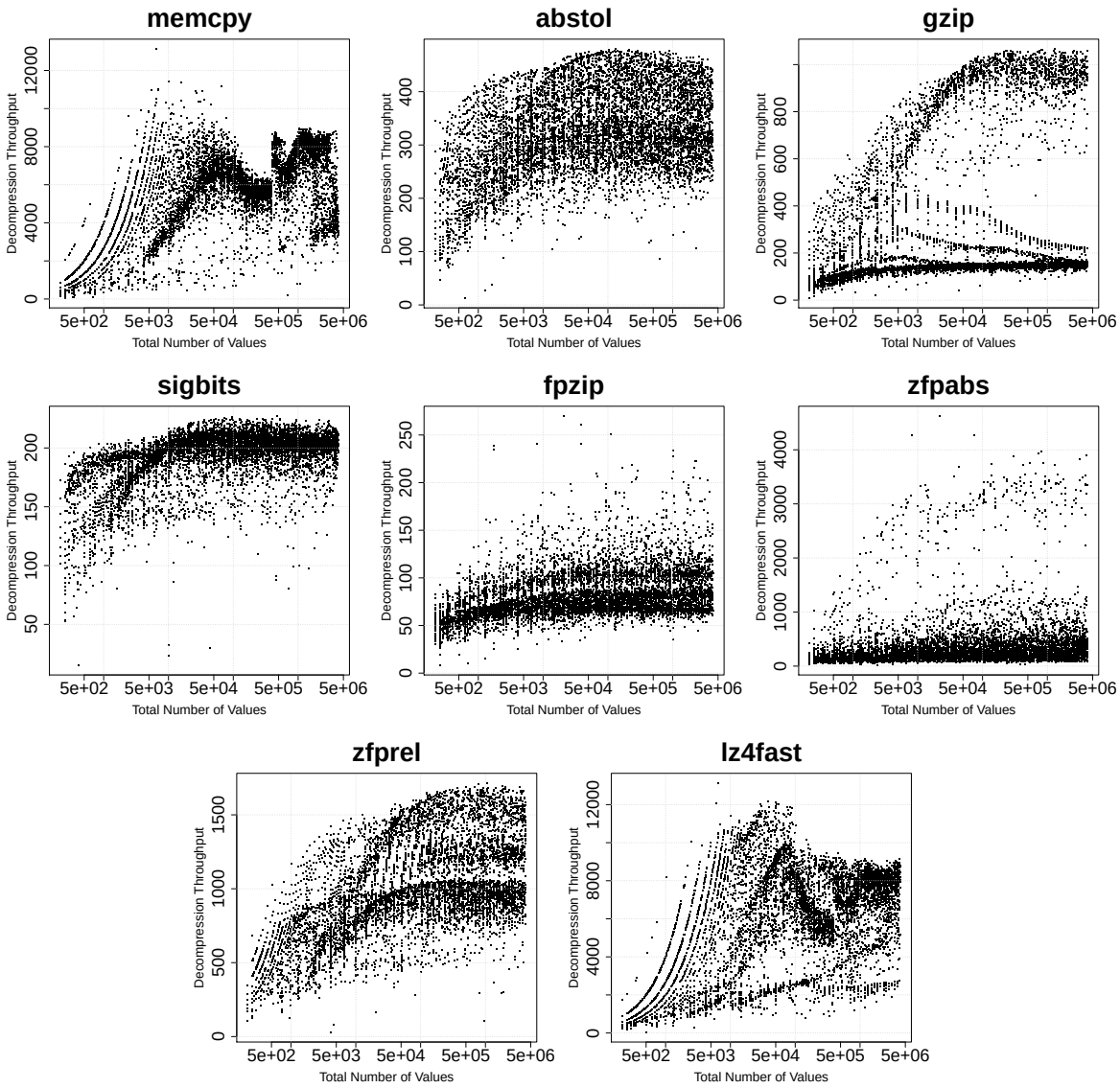


Figure B.2.: Dependencies of Decompression Throughput and Number of Values

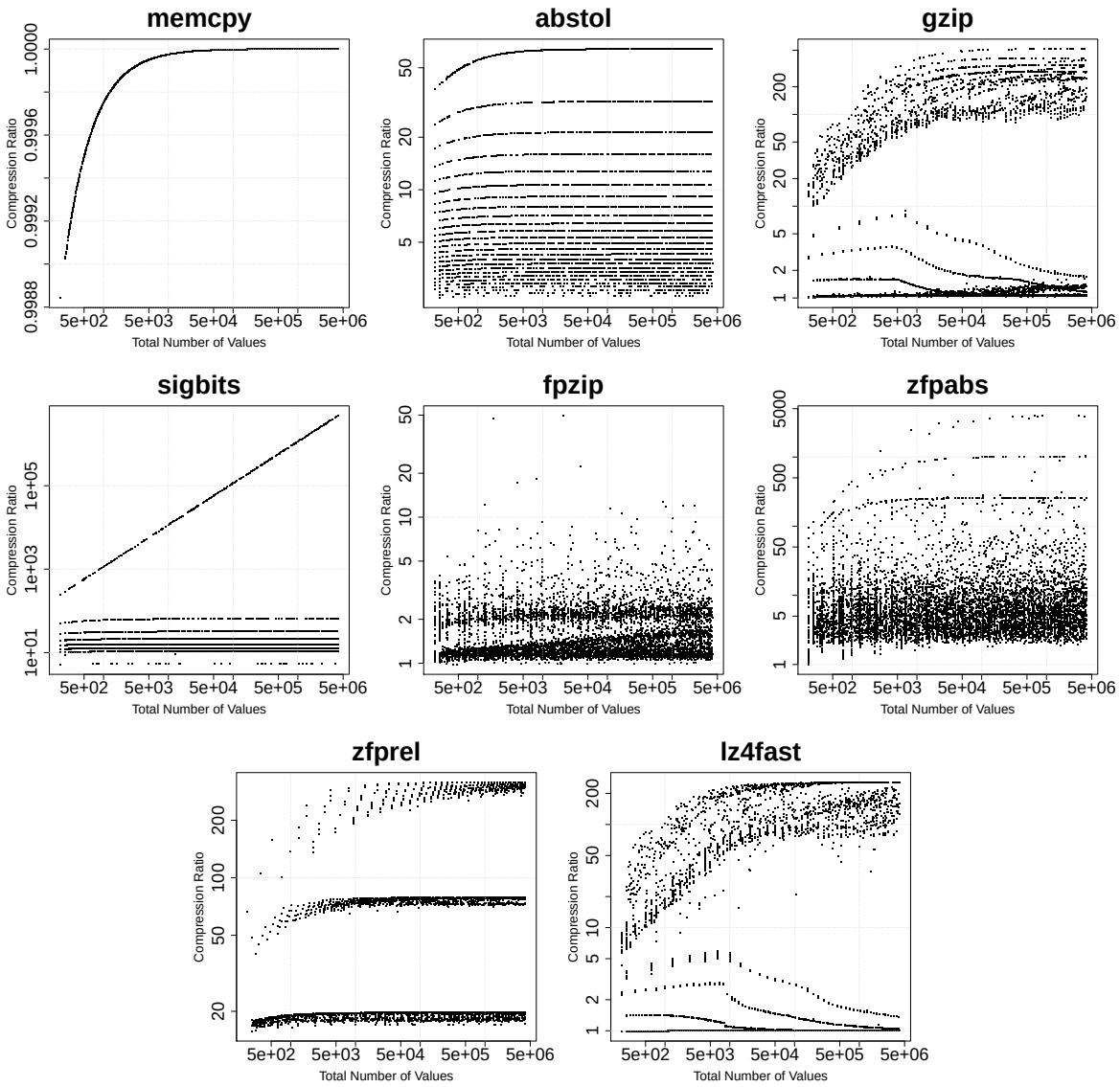


Figure B.3.: Dependencies of Compression Ratio and Number of Values

B.2. Data Dimensionality

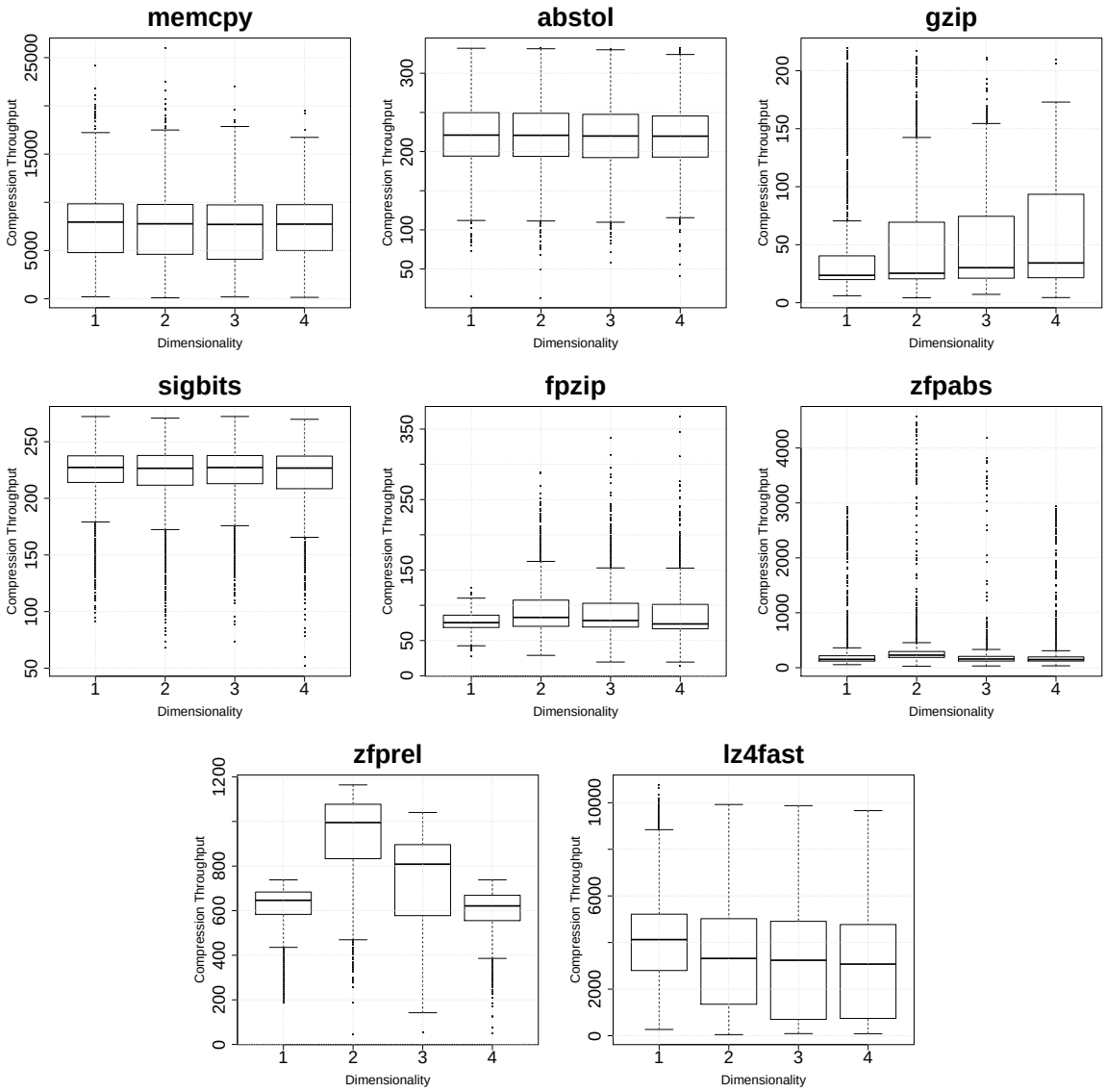


Figure B.4.: Dependencies of Compression Throughput and Data Dimensionality

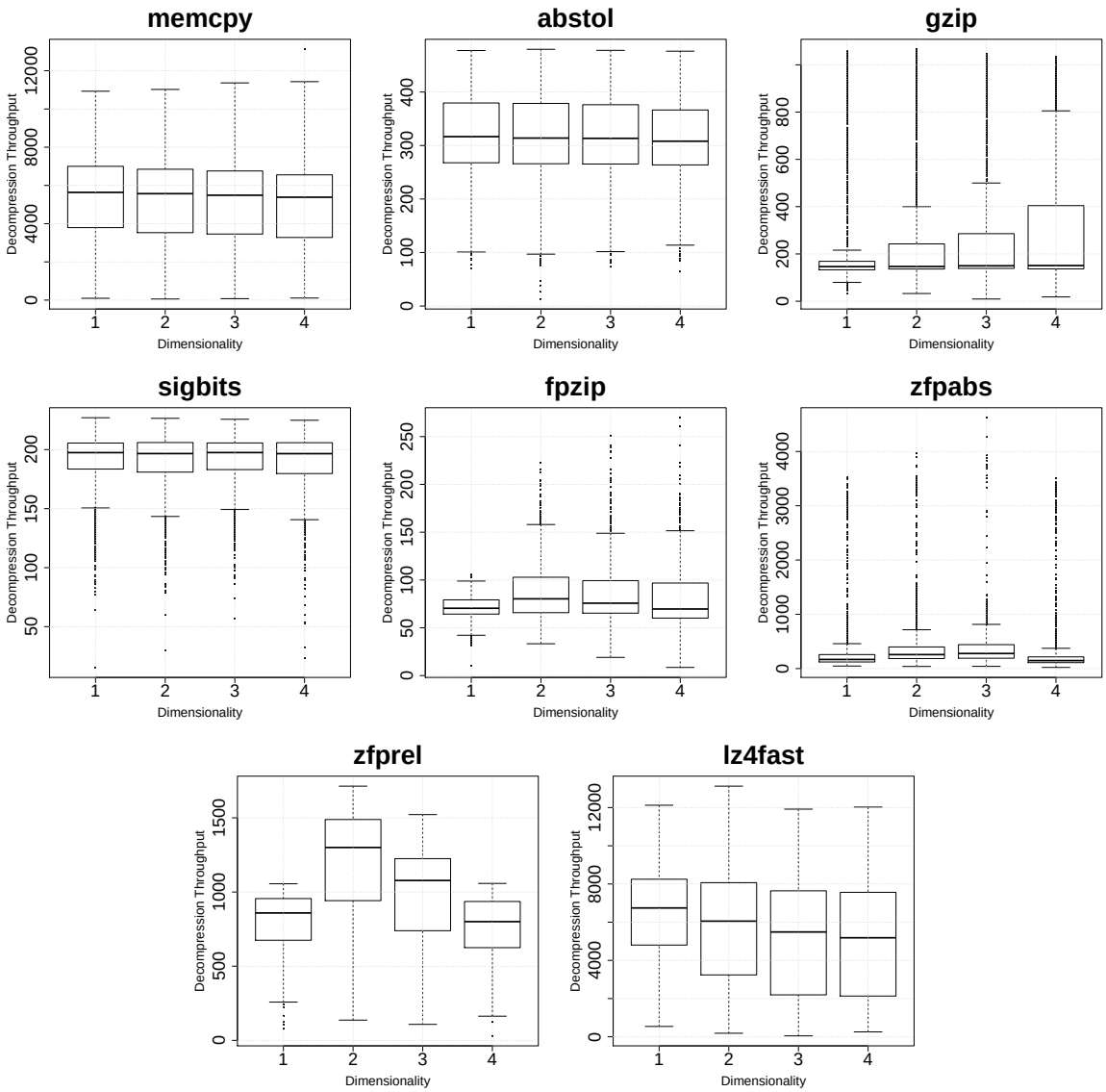


Figure B.5.: Dependencies of Decompression Throughput and Data Dimensionality

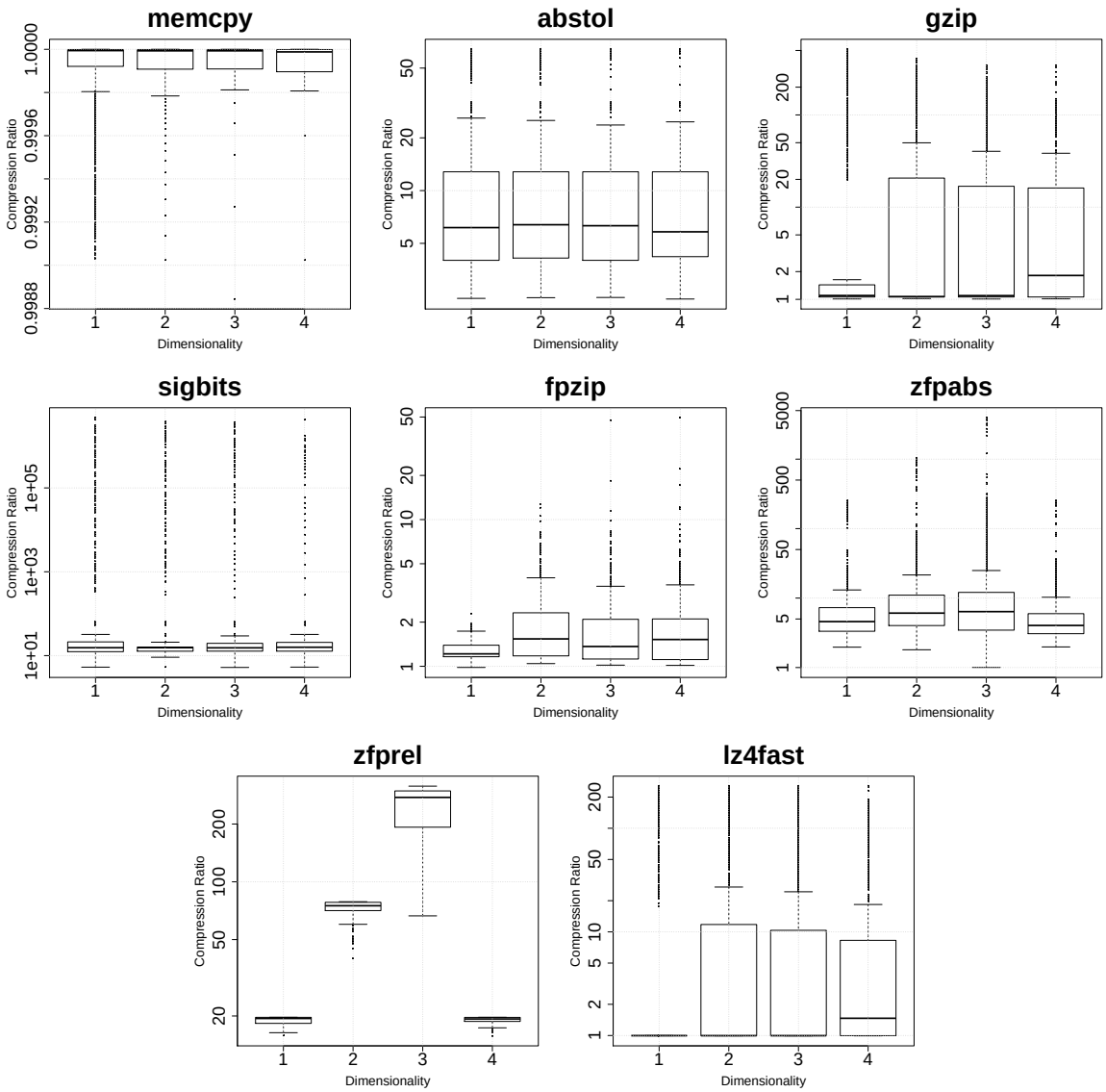


Figure B.6.: Dependencies of Compression Ratio and Data Dimensionality

B.3. Mean Value

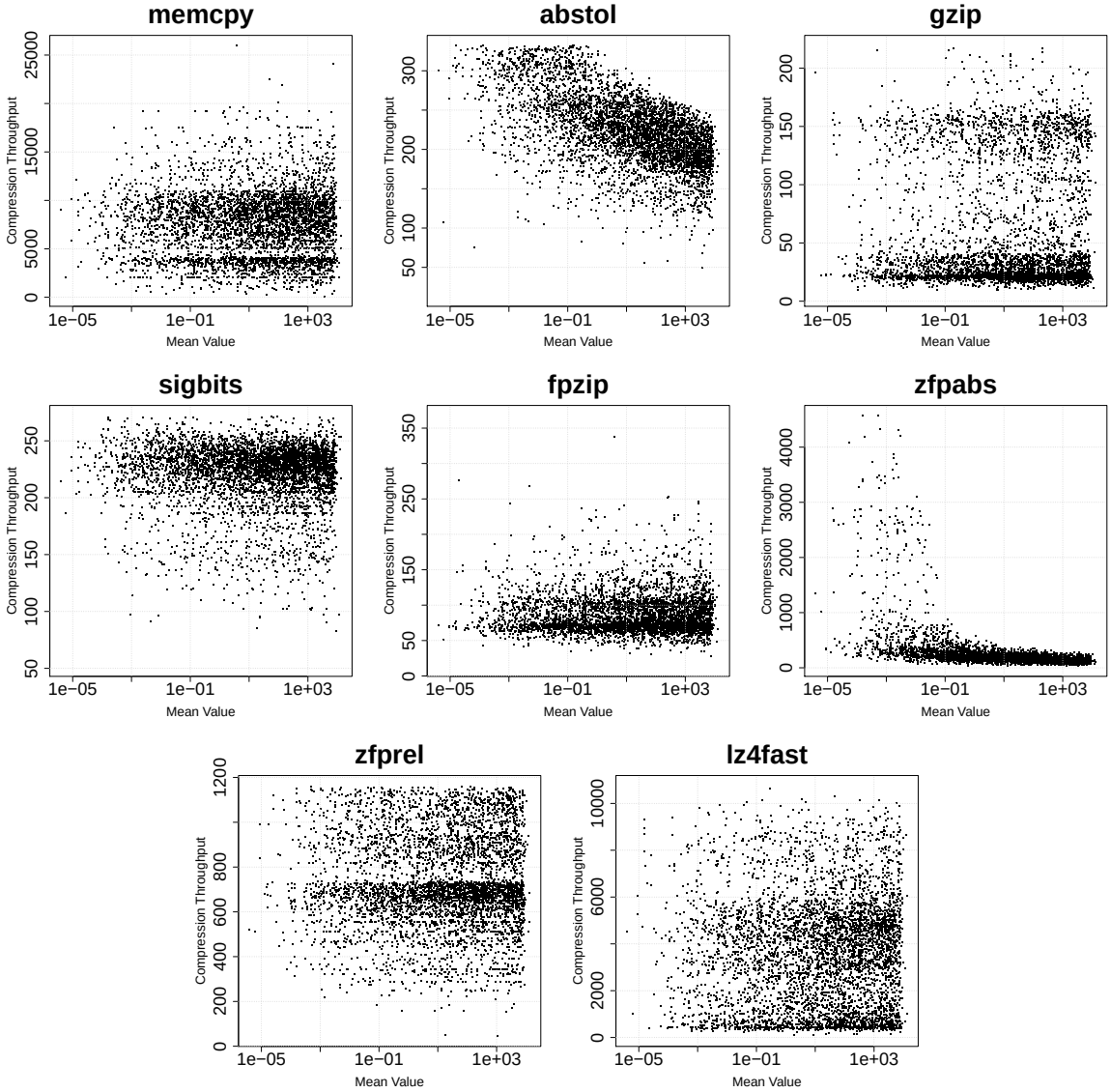


Figure B.7.: Dependencies of Compression Throughput and Mean Value

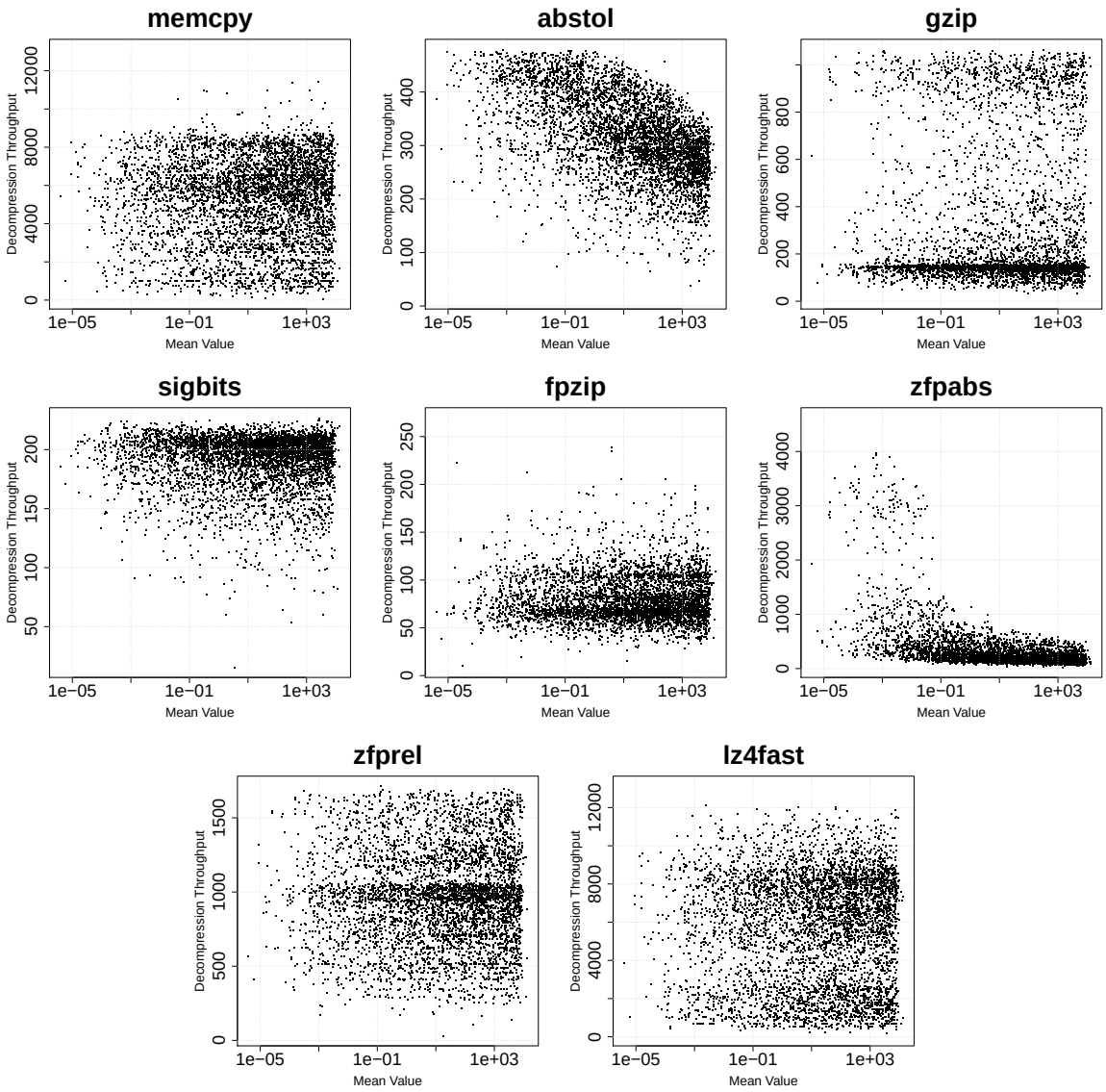


Figure B.8.: Dependencies of Decompression Throughput and Mean Value

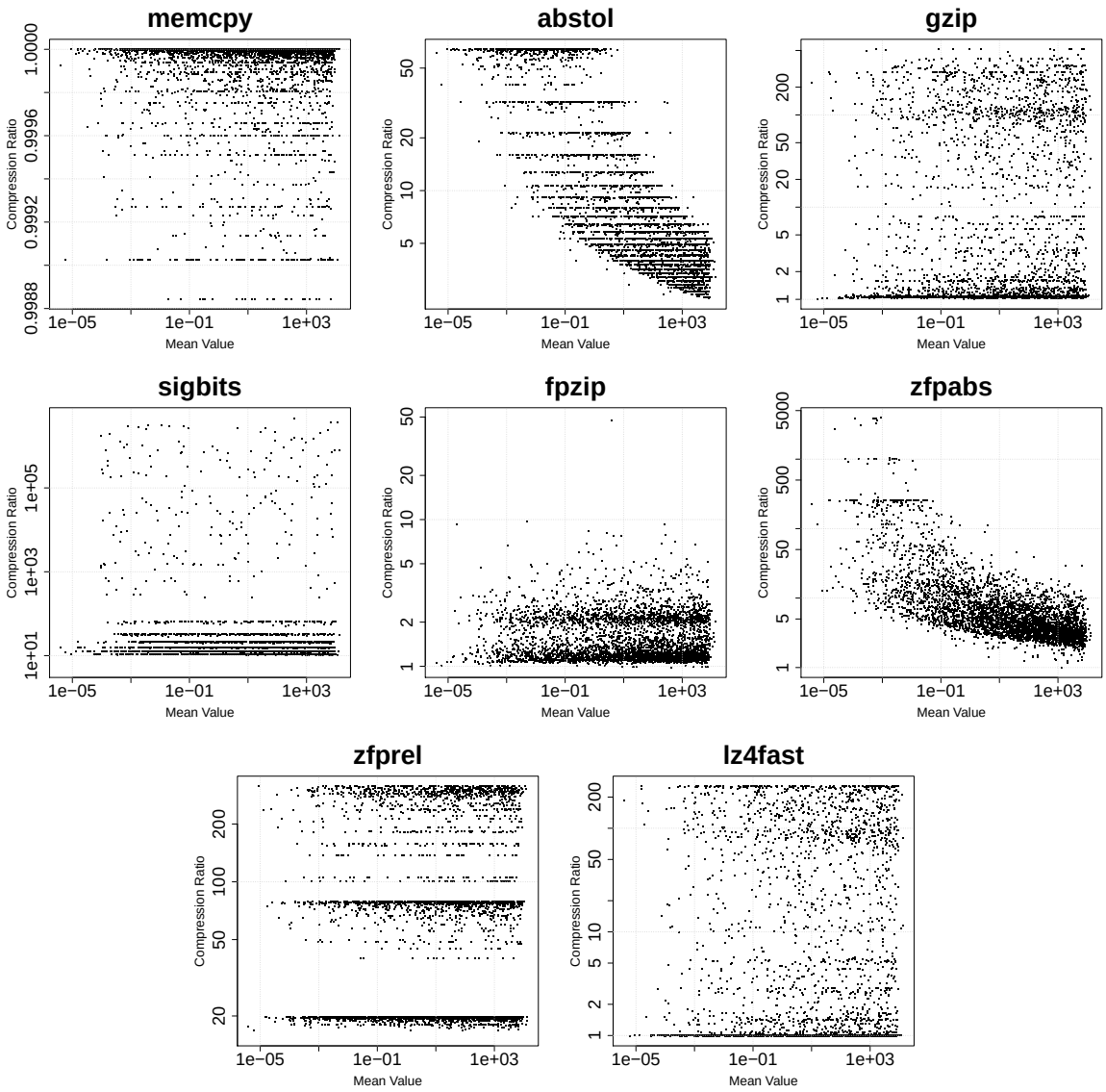


Figure B.9.: Dependencies of Compression Ratio and Mean Value

B.4. Standard Deviation

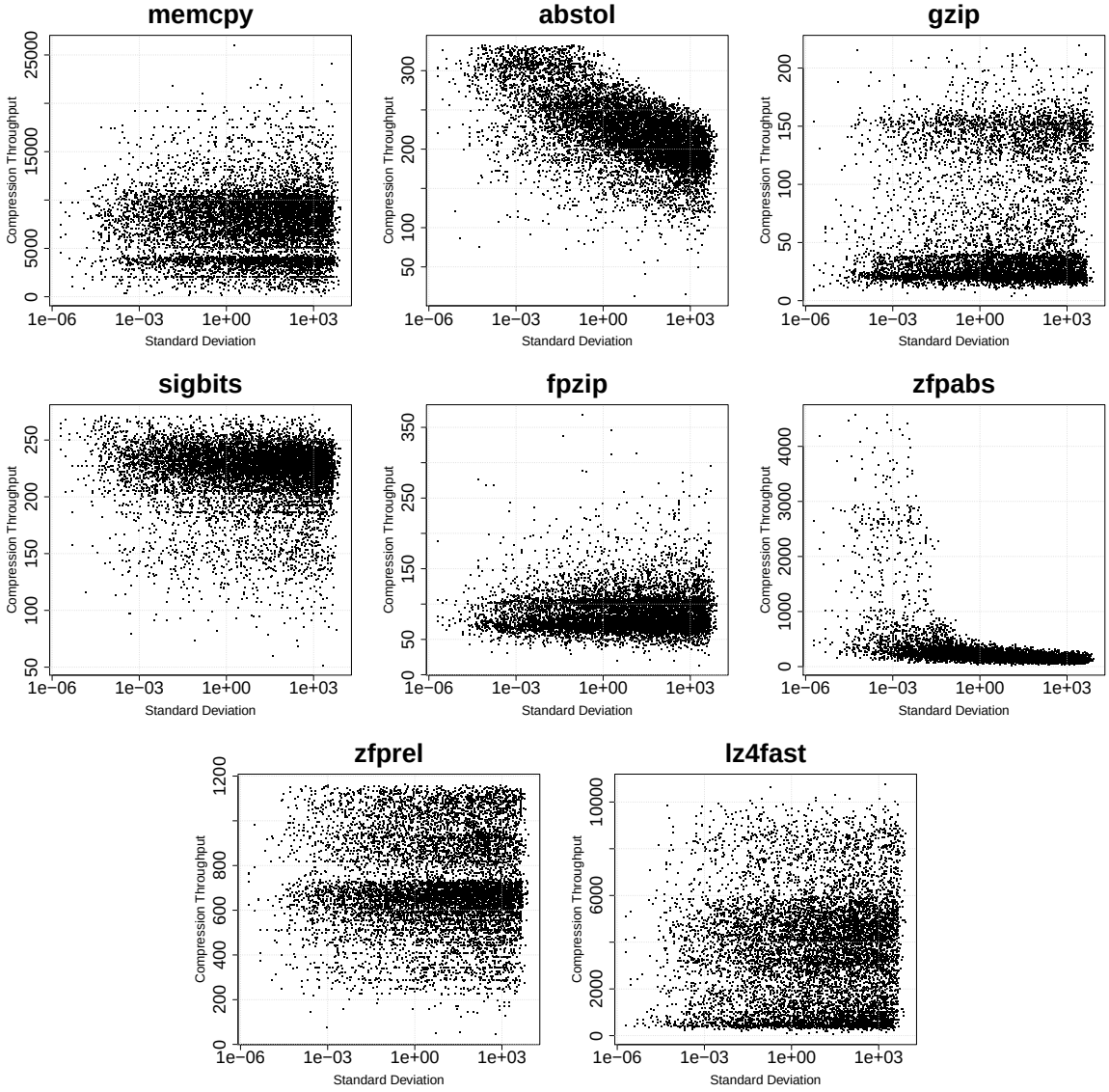


Figure B.10.: Dependencies of Compression Throughput and Standard Deviation

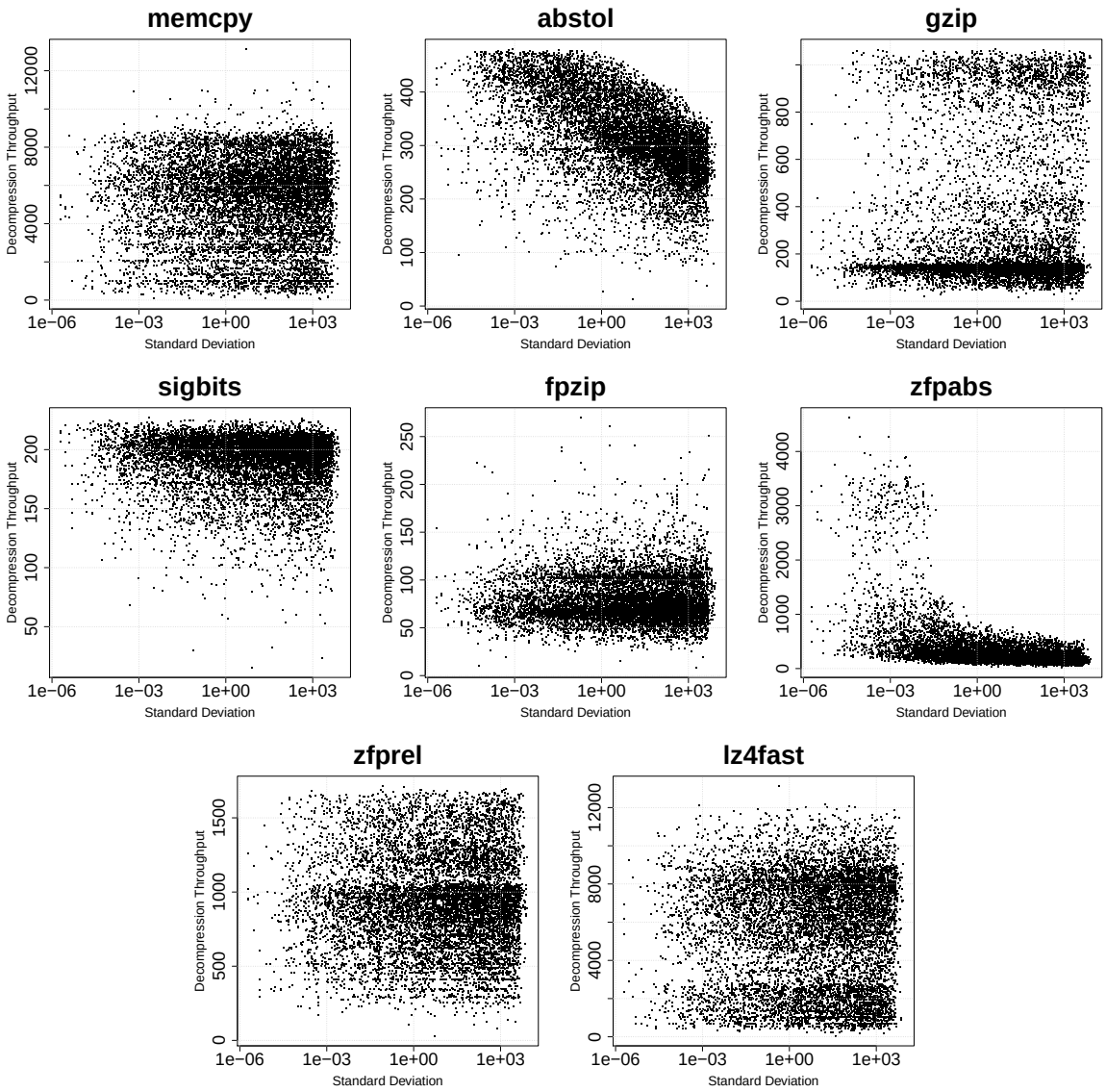


Figure B.11.: Dependencies of Decompression Throughput and Standard Deviation

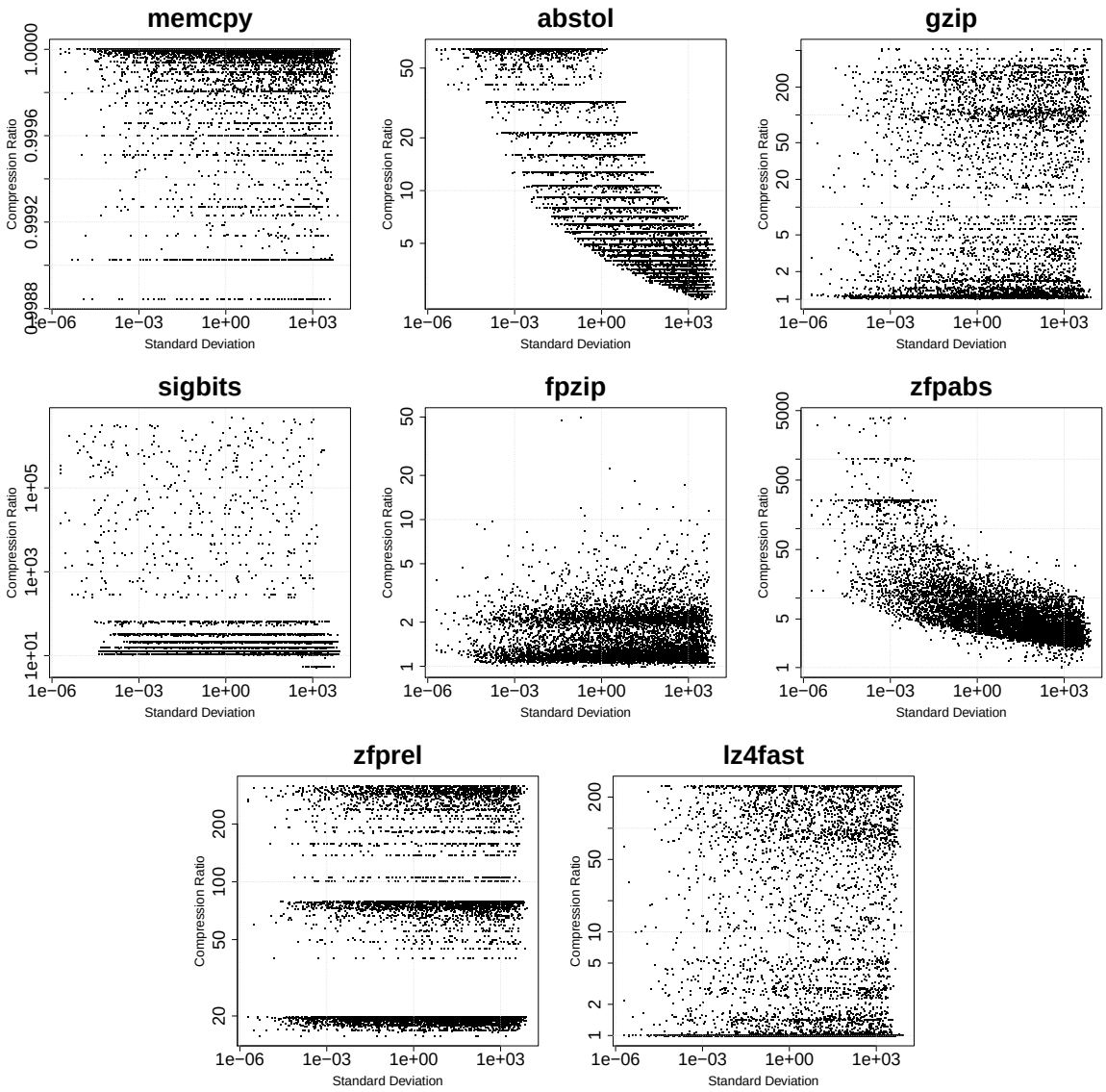


Figure B.12.: Dependencies of Compression Ratio and Standard Deviation

B.5. Maximum Step Size

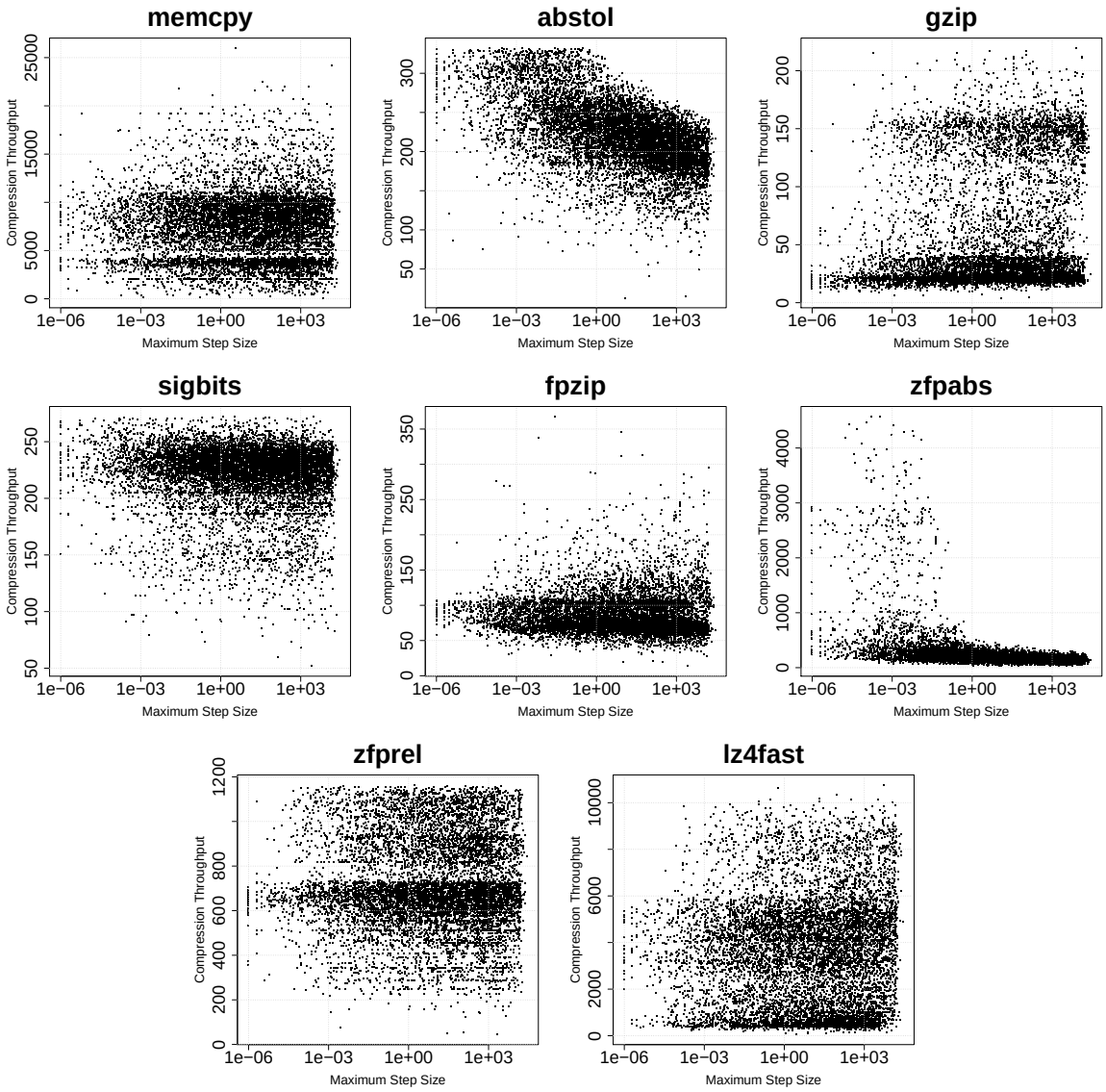


Figure B.13.: Dependencies of Compression Throughput and Maximum Step Size

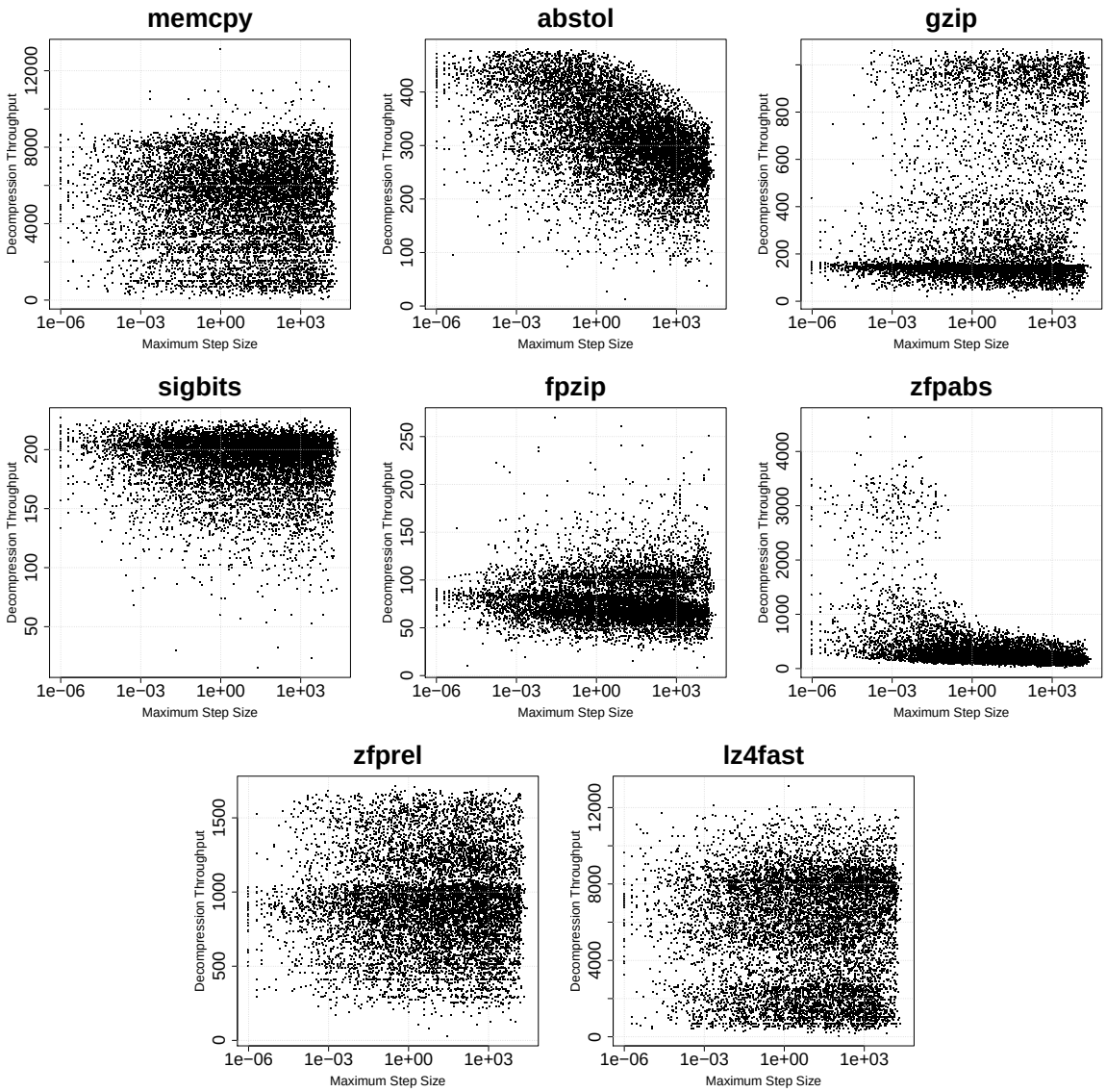


Figure B.14.: Dependencies of Decompression Throughput and Maximum Step Size

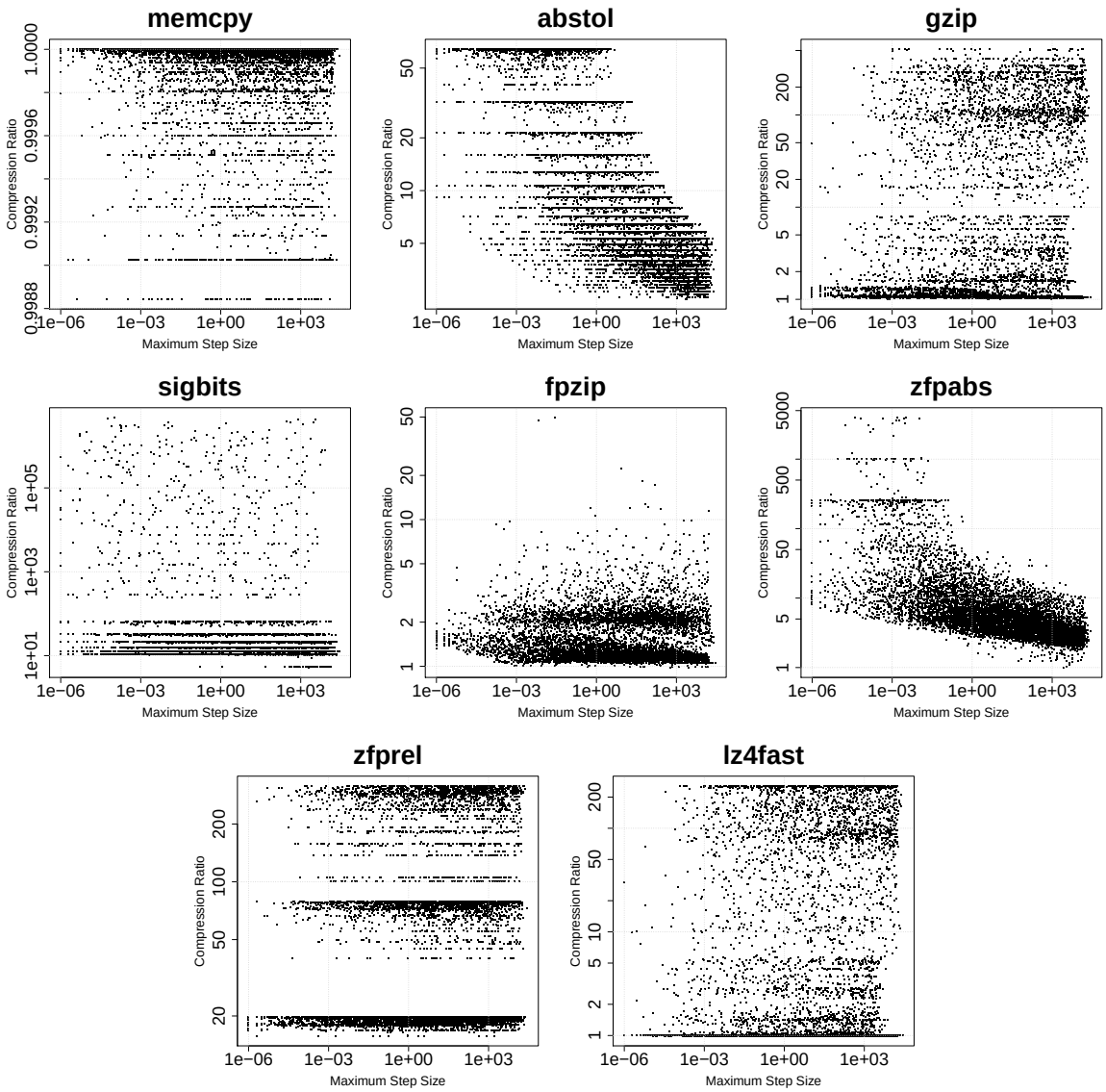


Figure B.15.: Dependencies of Compression Ratio and Maximum Step Size

B.6. Absolute Error Tolerance

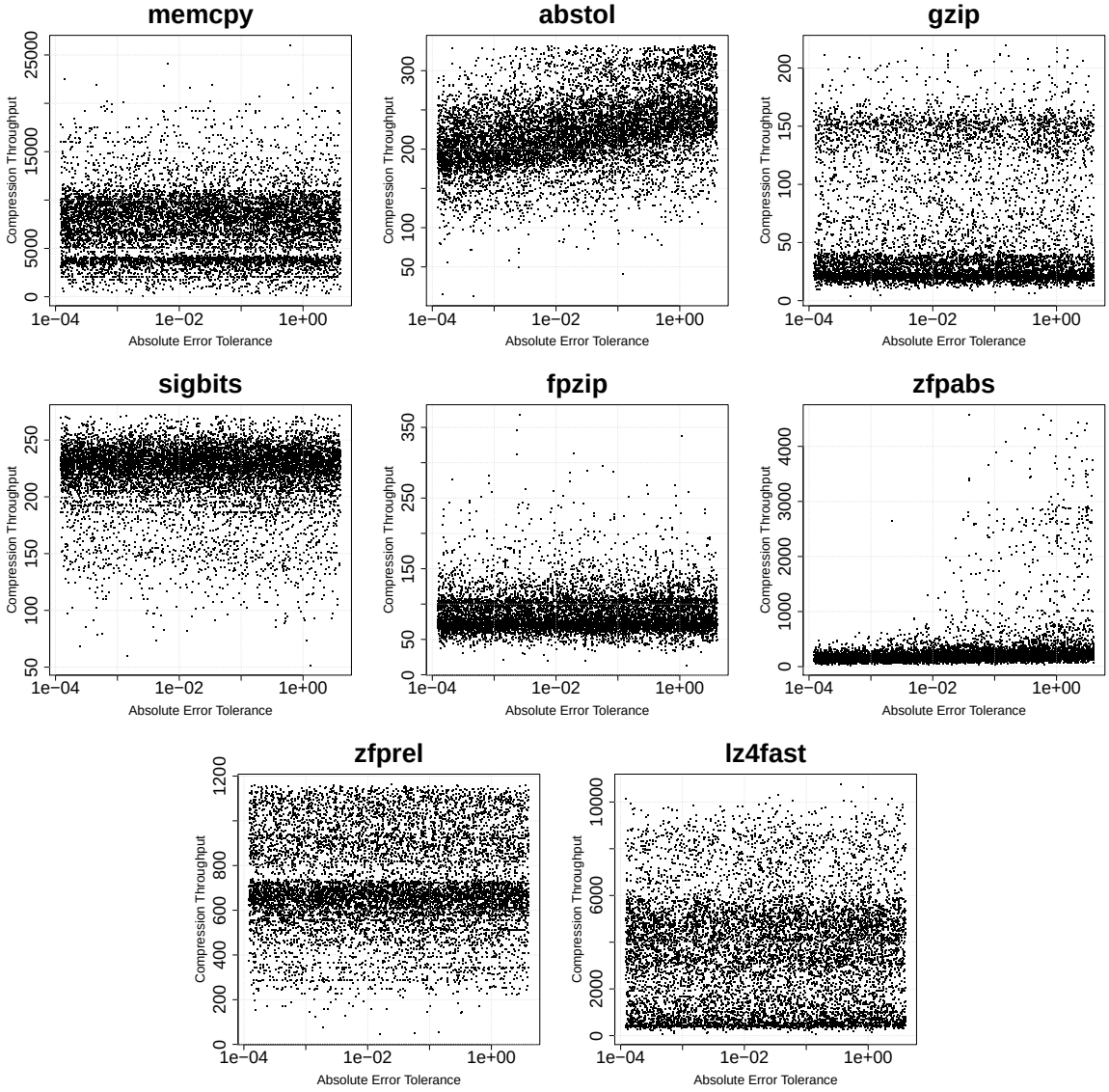


Figure B.16.: Dependencies of Compression Throughput and Absolute Error Tolerance

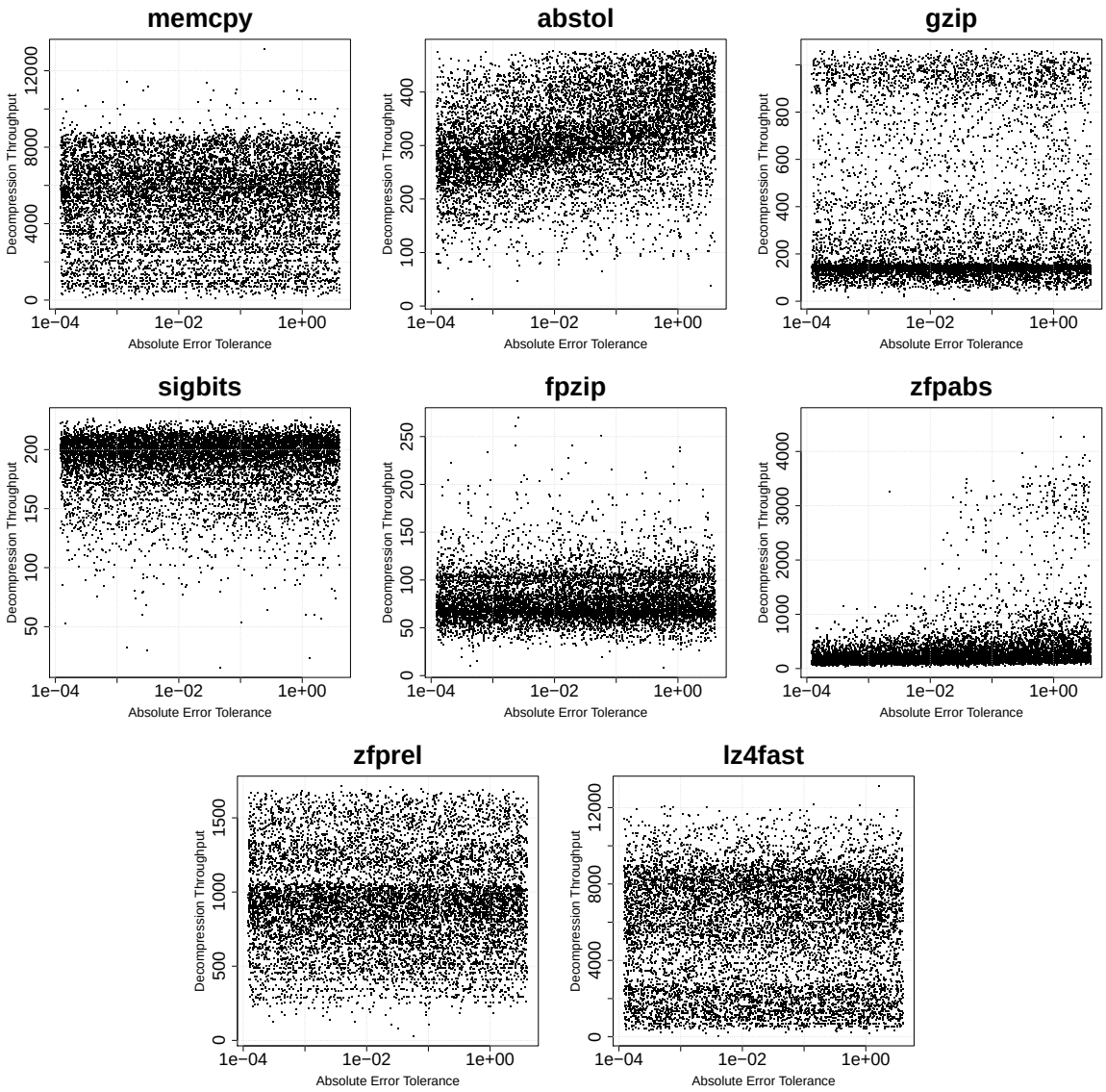


Figure B.17.: Dependencies of Decompression Throughput and Absolute Error Tolerance

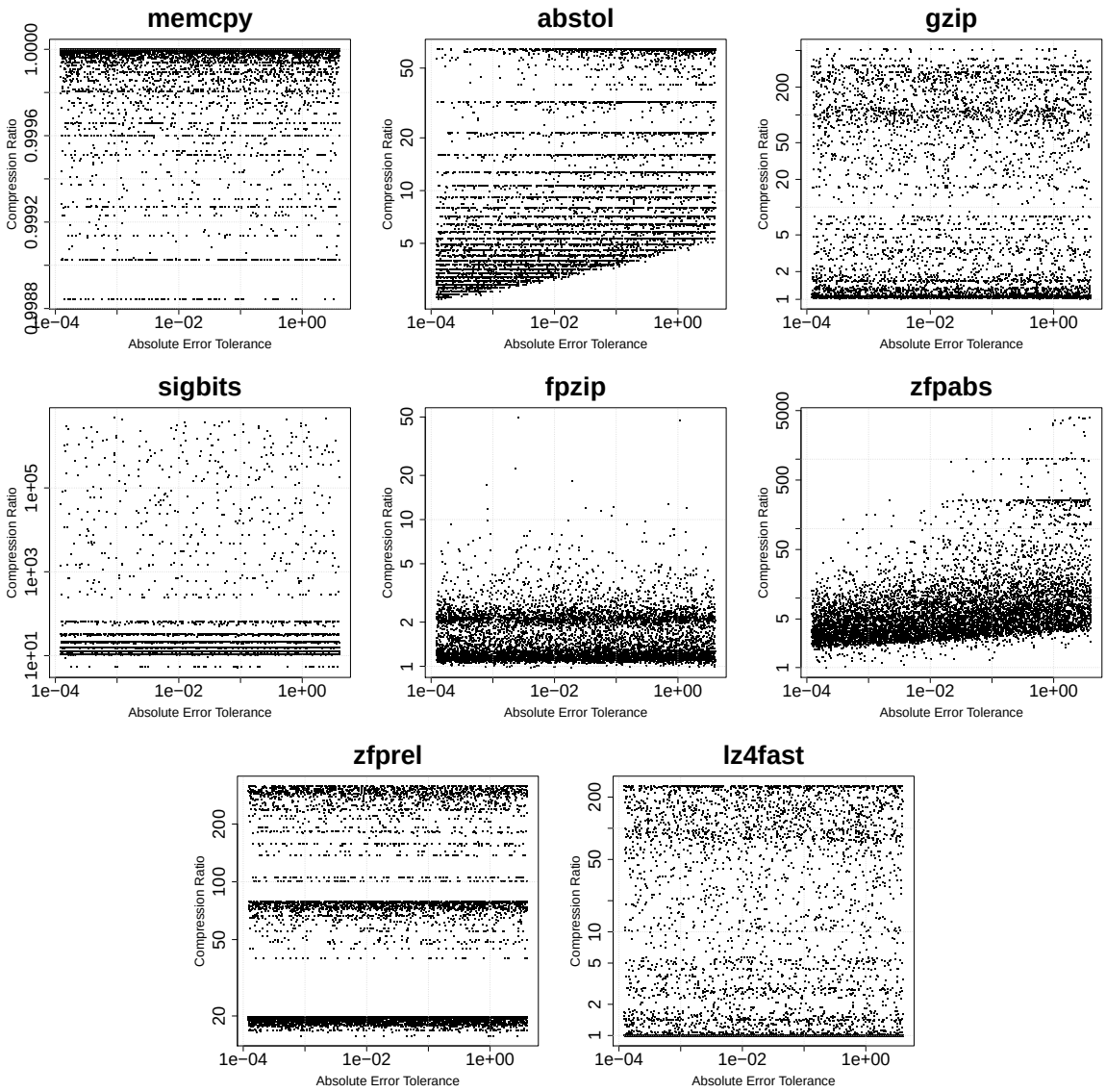


Figure B.18.: Dependencies of Compression Ratio and Absolute Error Tolerance

B.7. Relative Error Tolerance

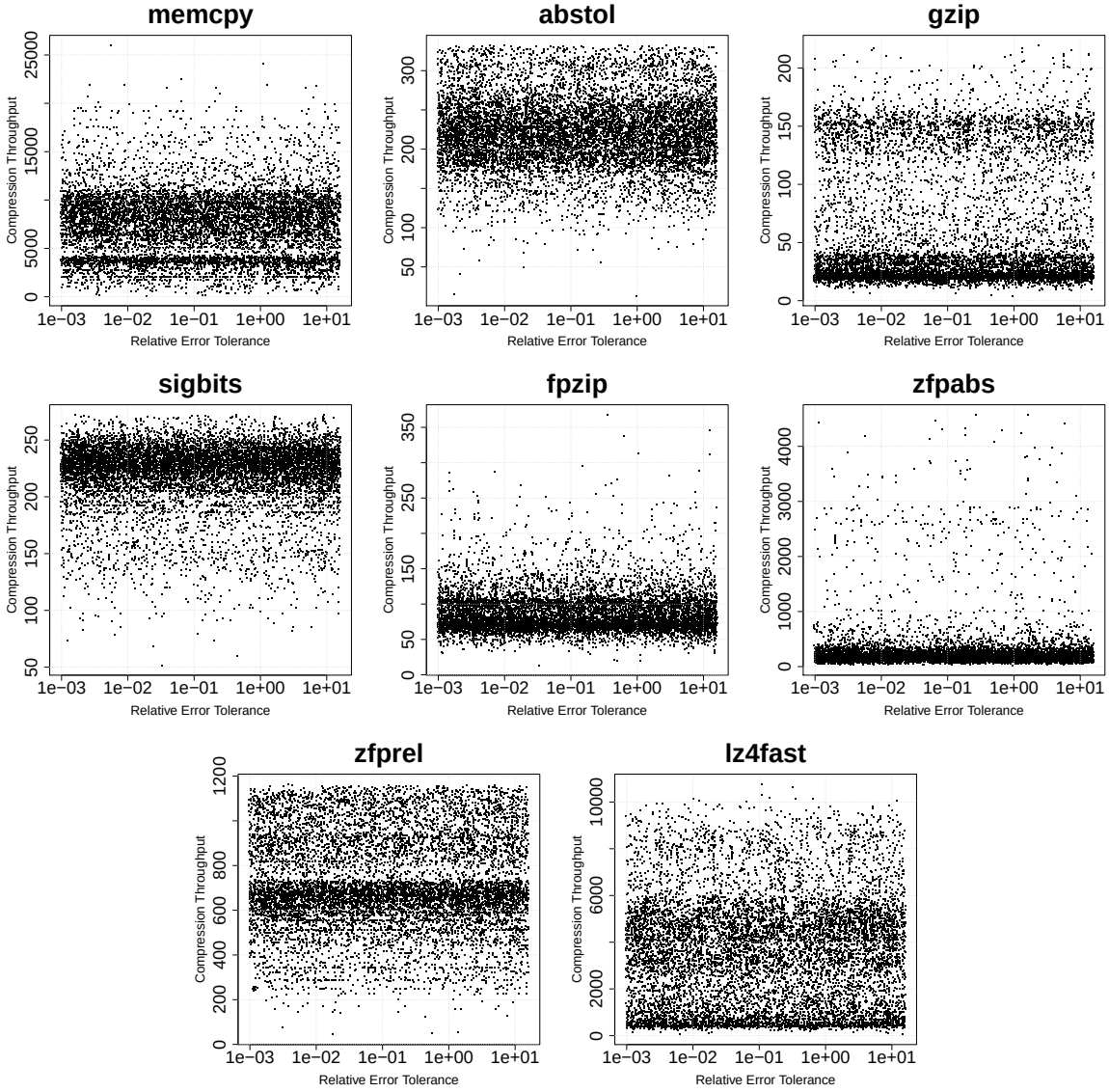


Figure B.19.: Dependencies of Compression Throughput and Relative Error Tolerance

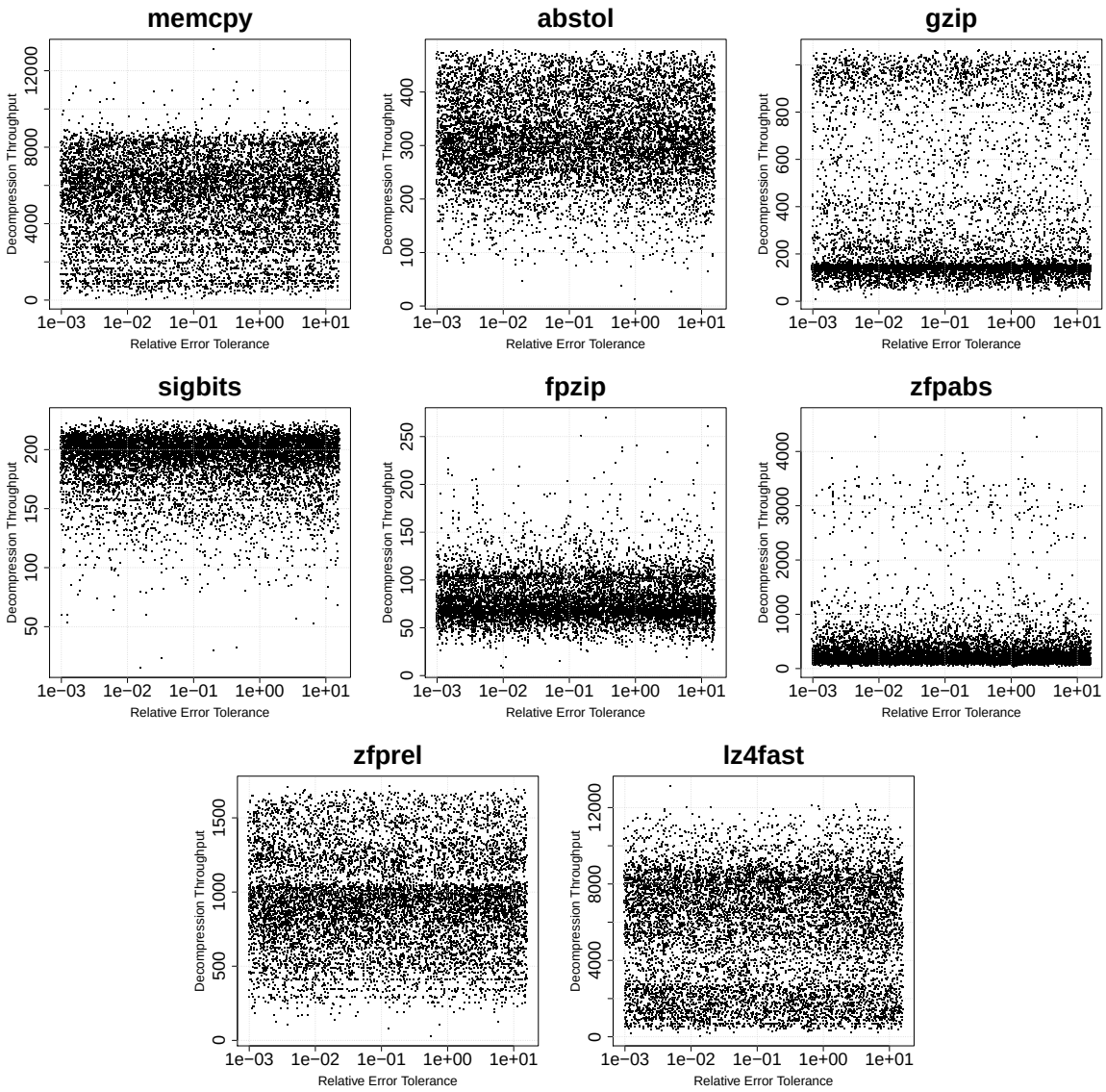


Figure B.20.: Dependencies of Decompression Throughput and Relative Error Tolerance

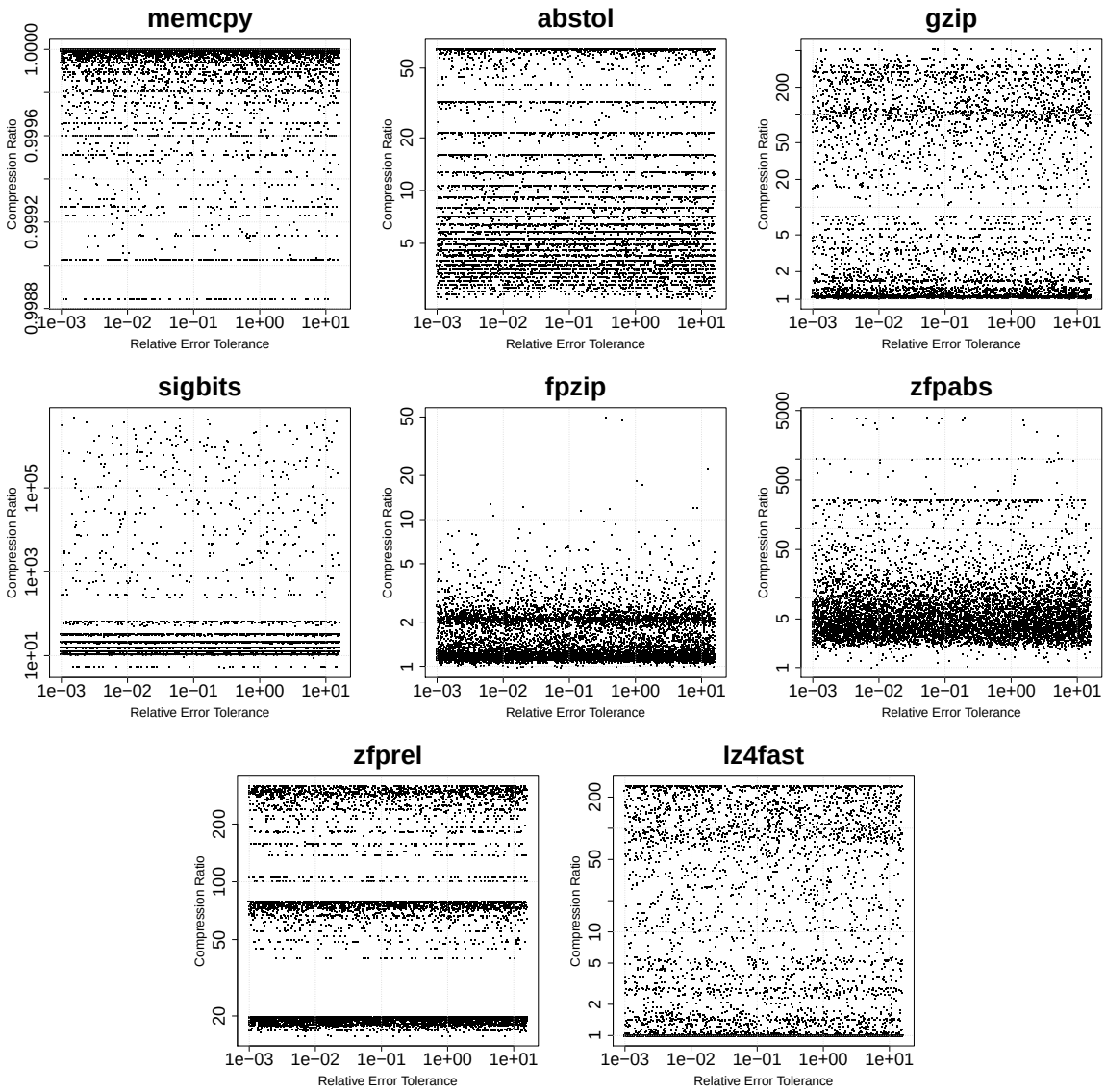


Figure B.21.: Dependencies of Compression Ratio and Relative Error Tolerance

B.8. Mean Value, Standard Deviation and Maximum Step Matrix

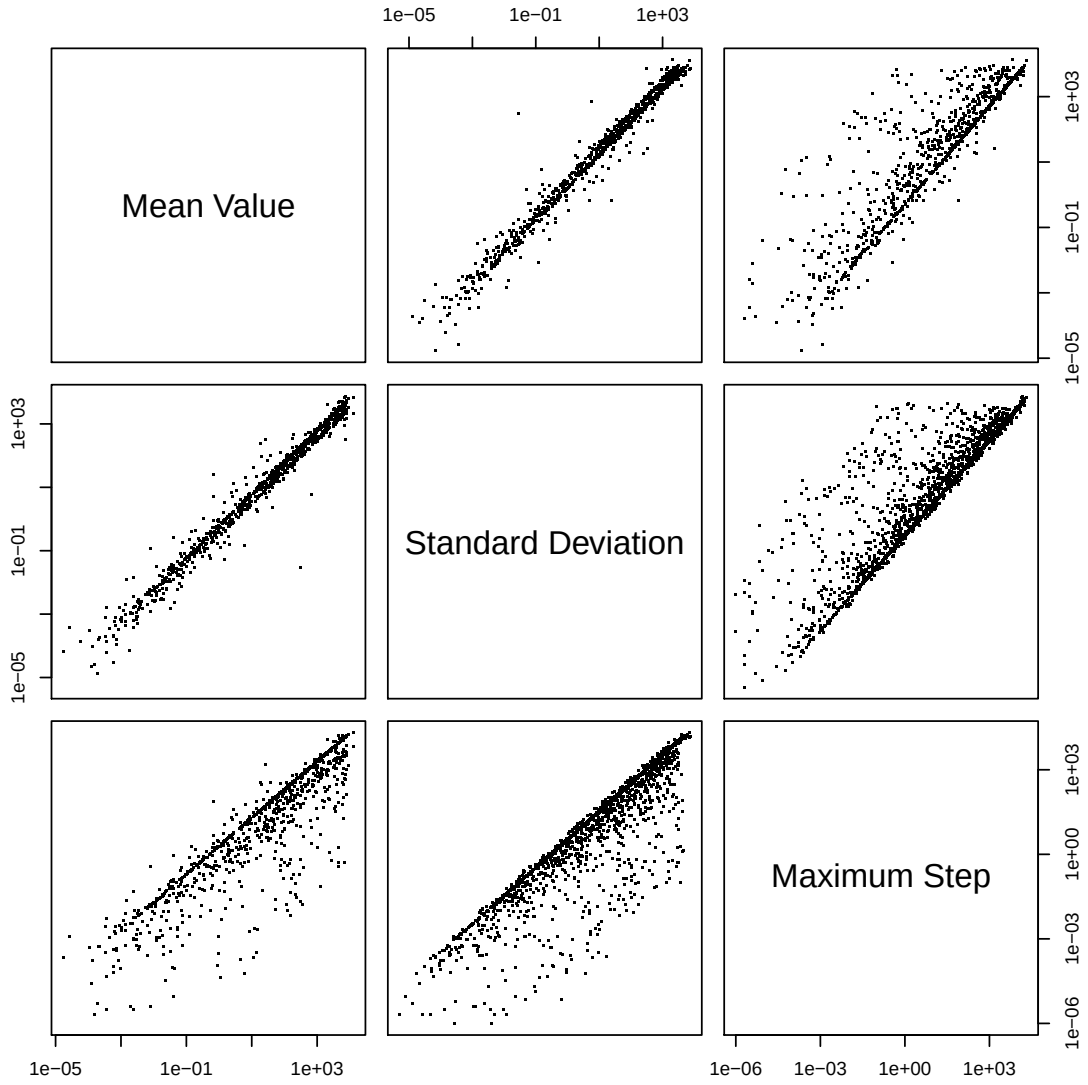


Figure B.22.: Inter-Dependencies of Mean Value, Standard Deviation and Maximum Step

C. RMSEs by Model Complexity

C.1. Linear Model Errors by Polynomial Degree

C.1.1. Compression Throughput

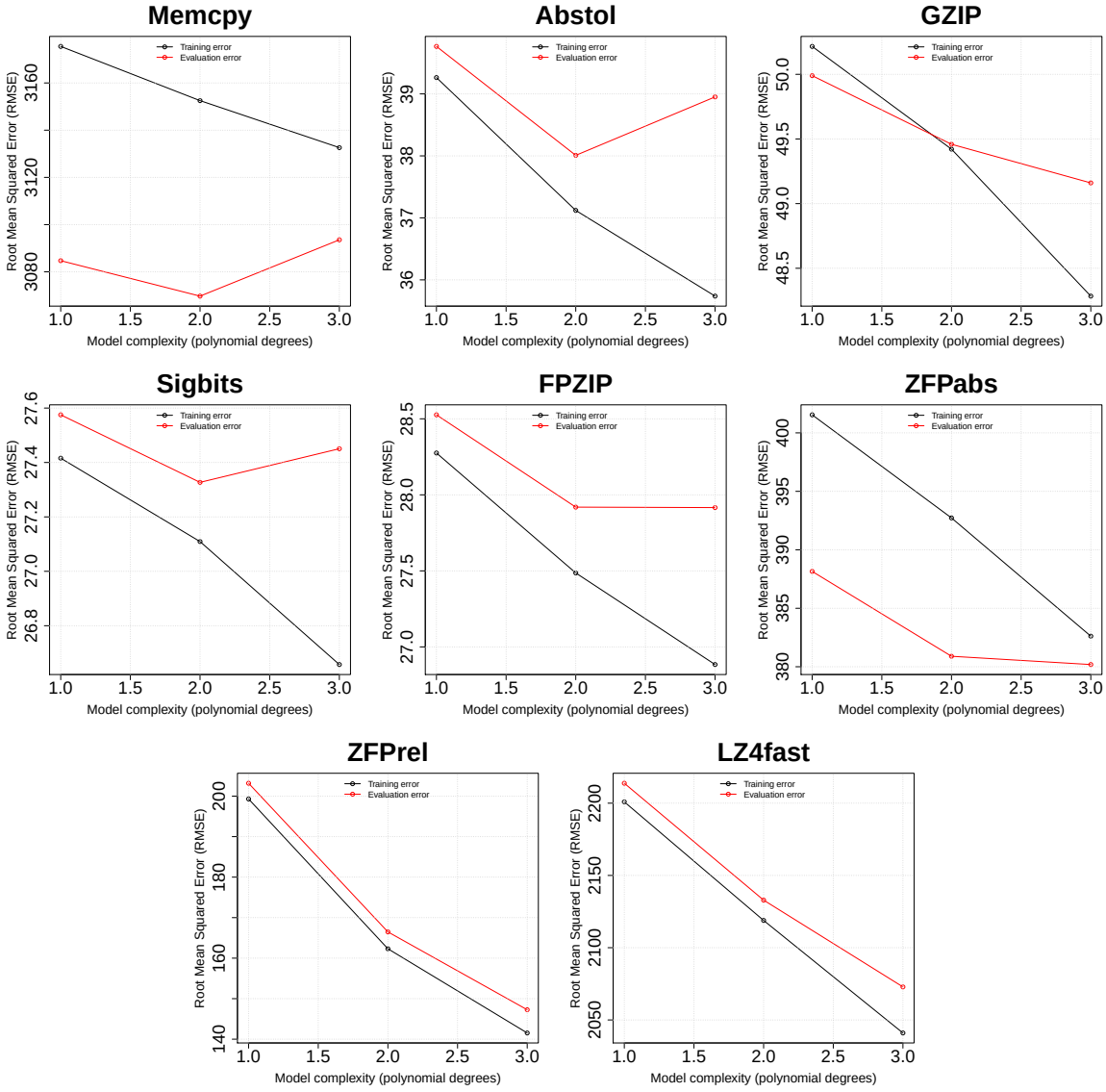


Figure C.1.: RMSEs of Compression Throughput

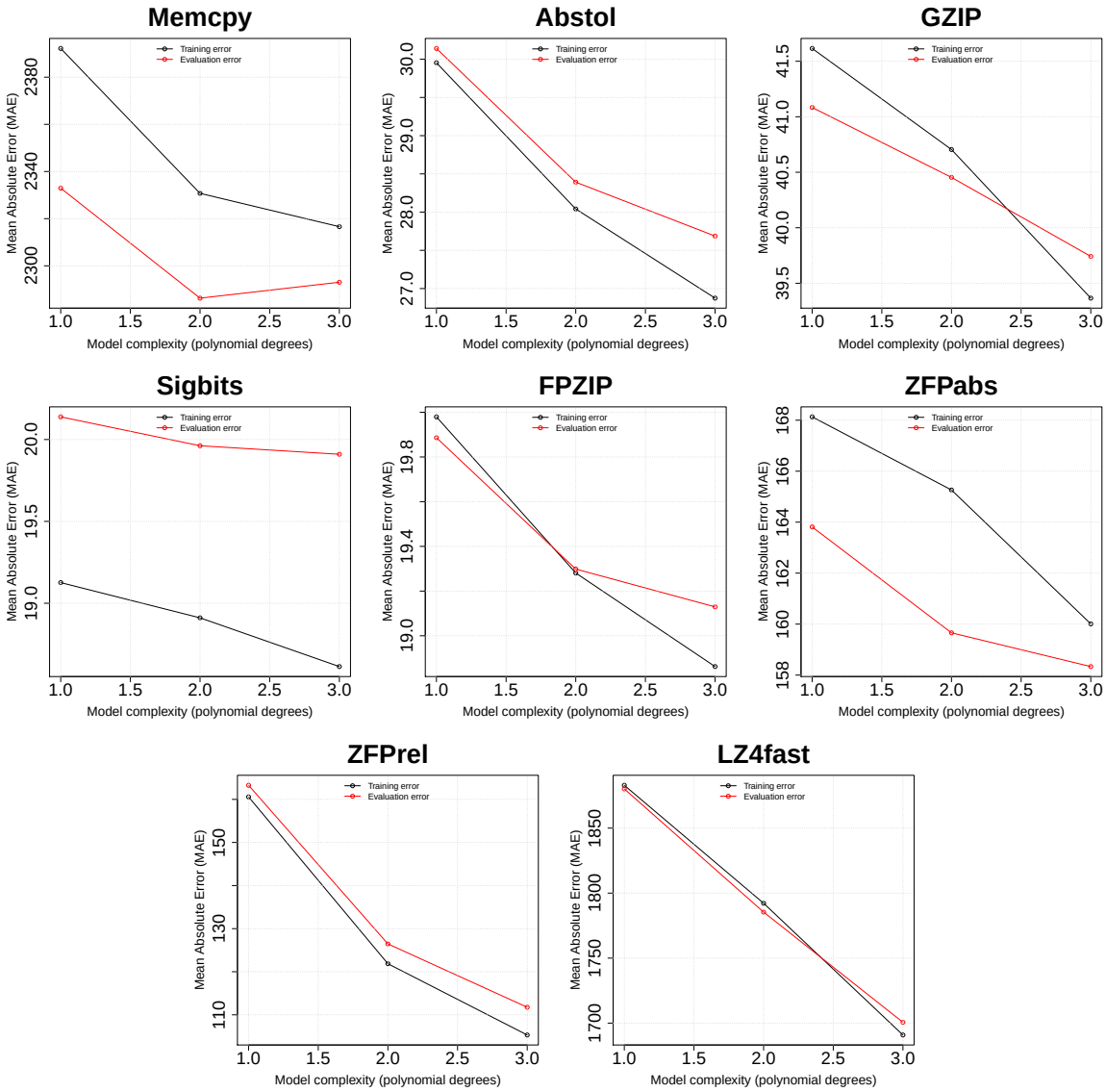


Figure C.2.: MAEs of Compression Throughput

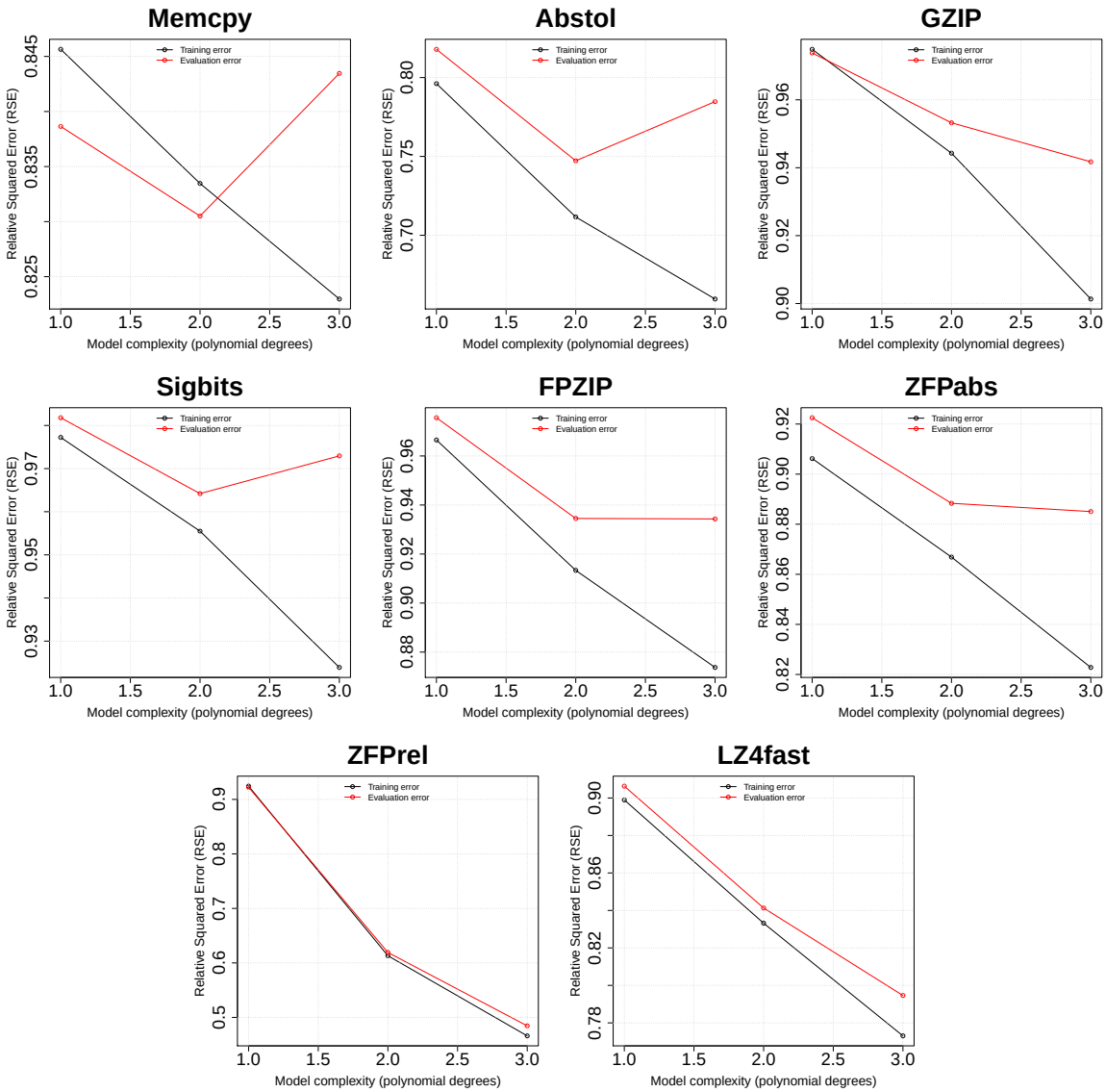


Figure C.3.: RSEs of Compression Throughput

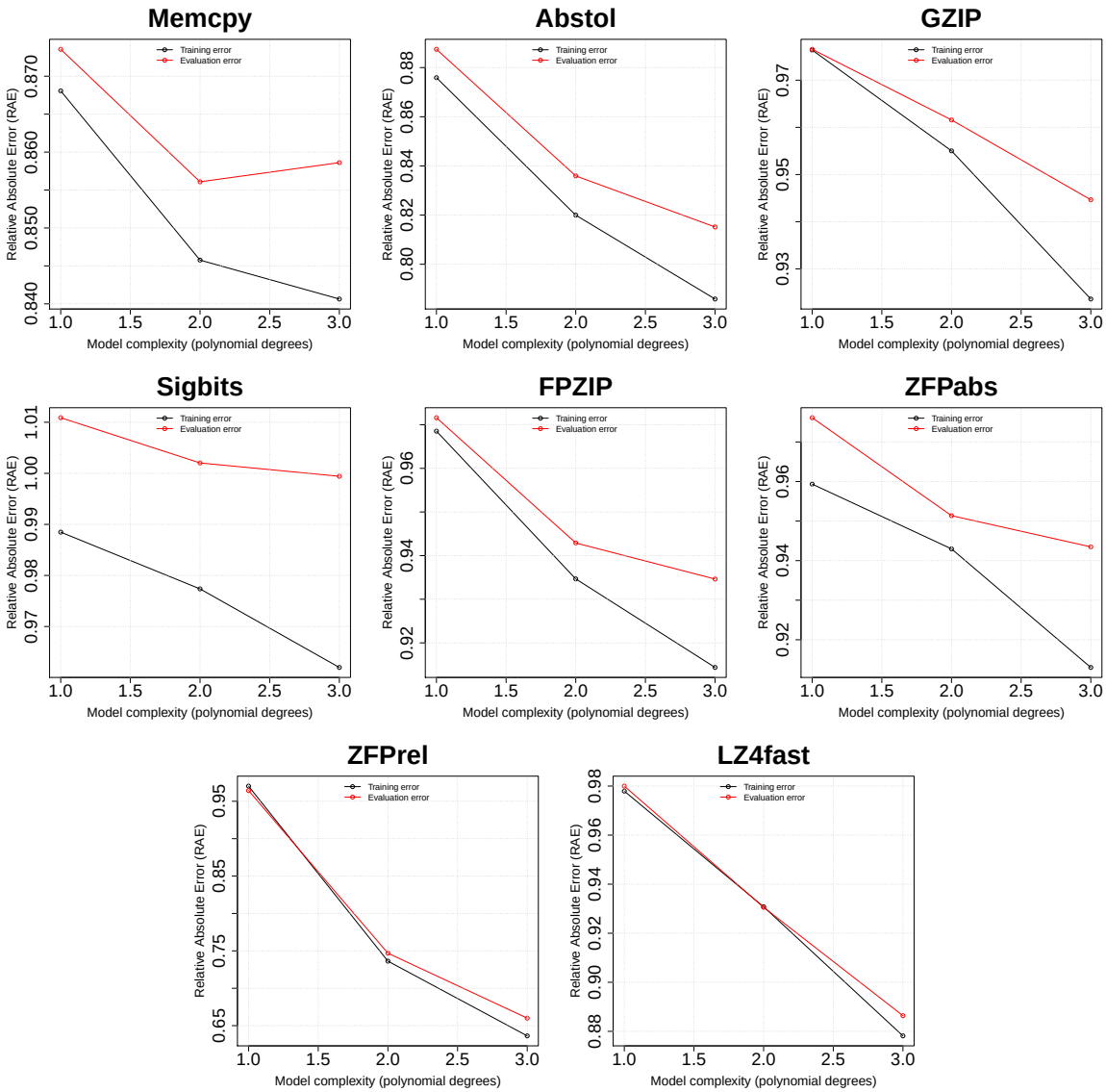


Figure C.4.: RSEs of Compression Throughput

C.1.2. Decompression Throughput

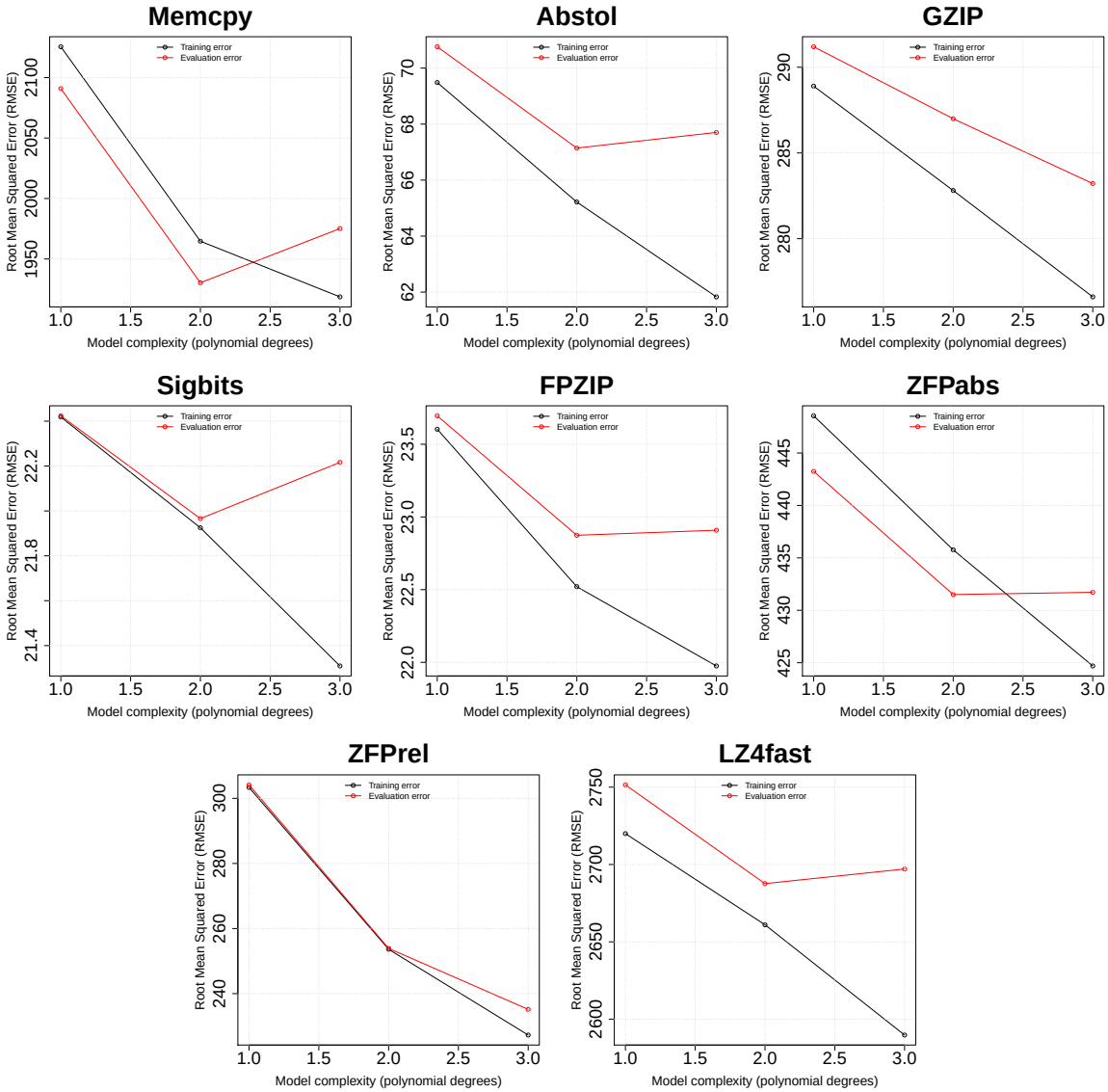


Figure C.5.: RMSEs of Decompression Throughput

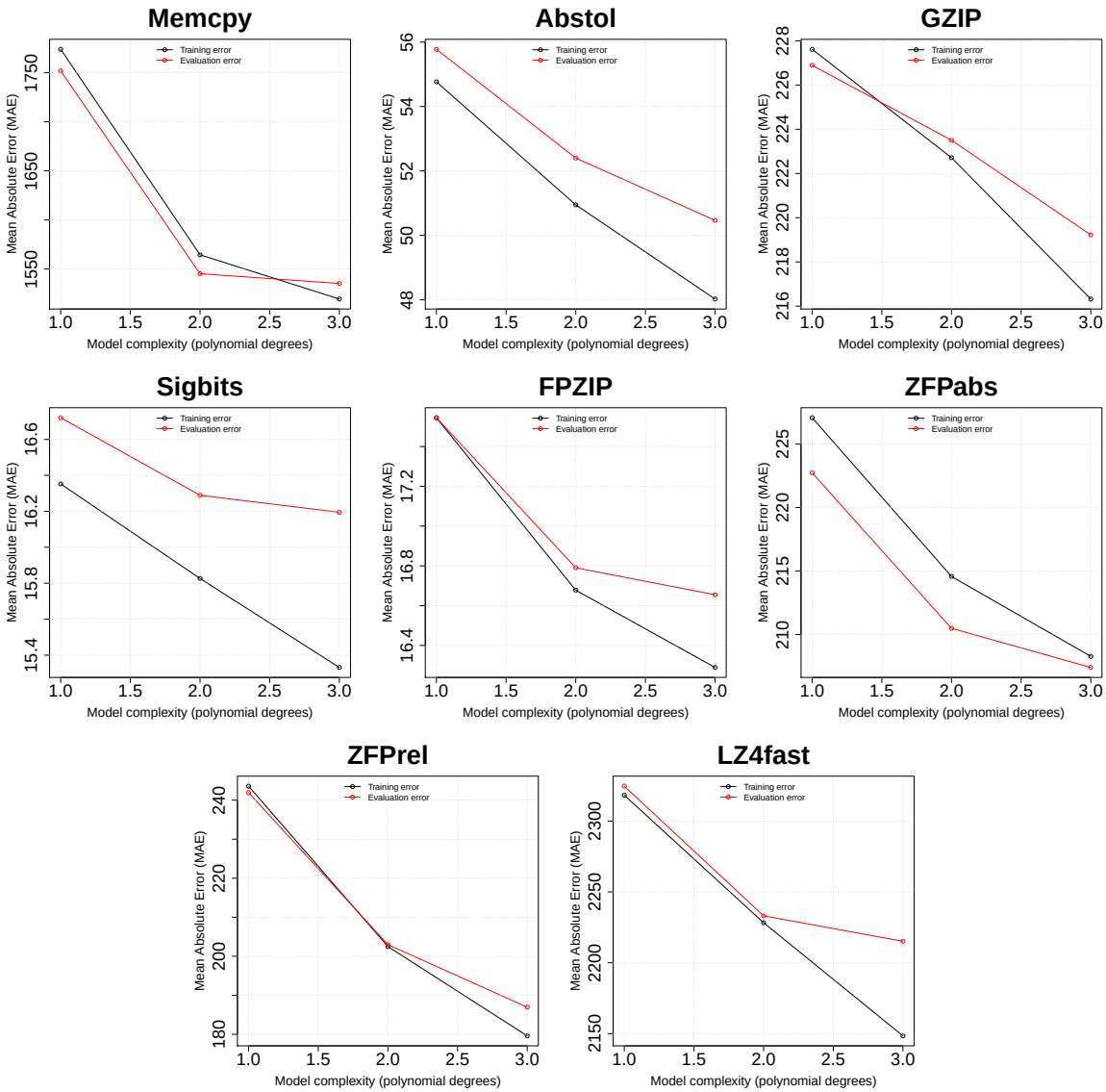


Figure C.6.: MAEs of Decompression Throughput

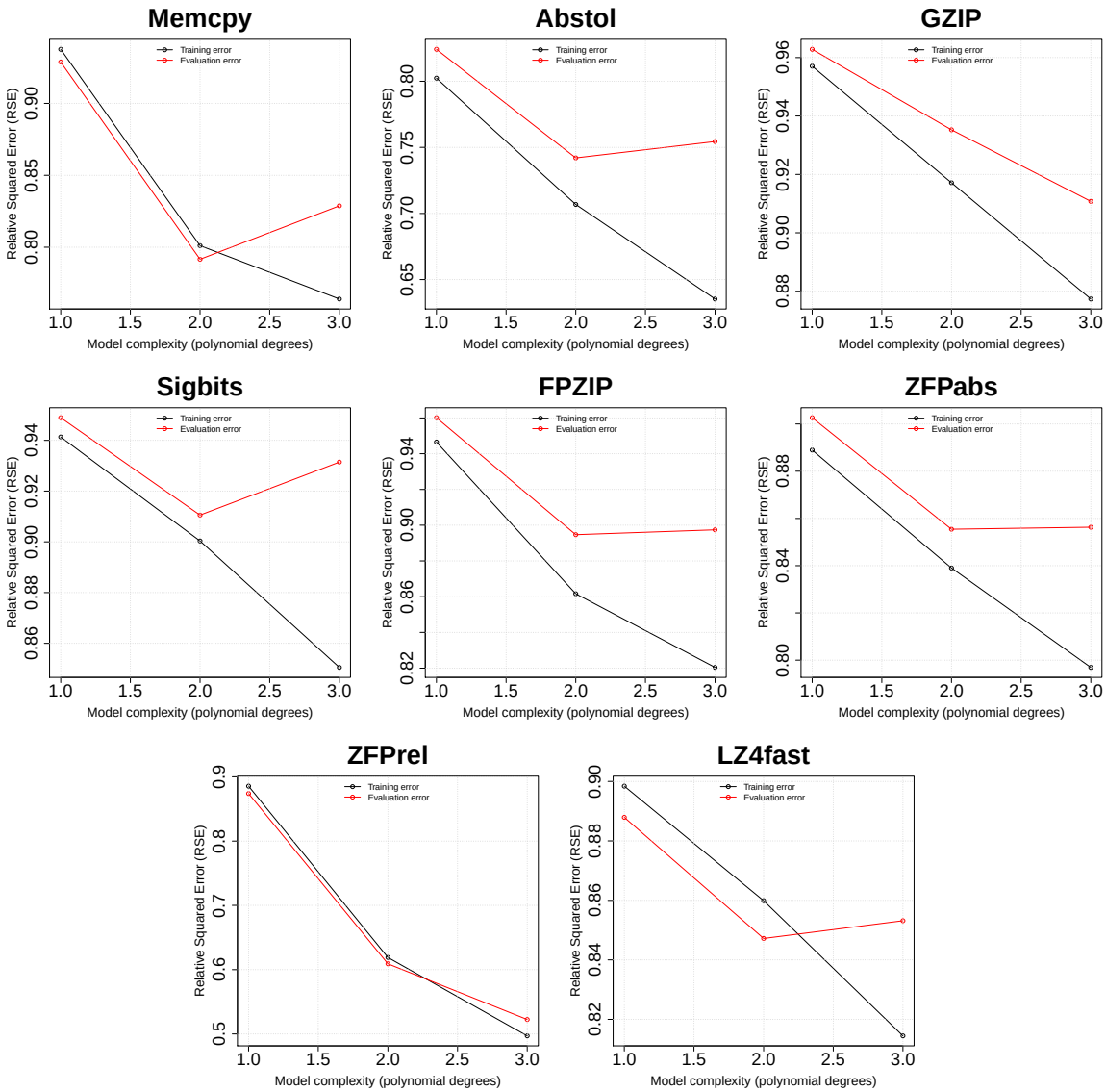


Figure C.7.: RSEs of Decompression Throughput

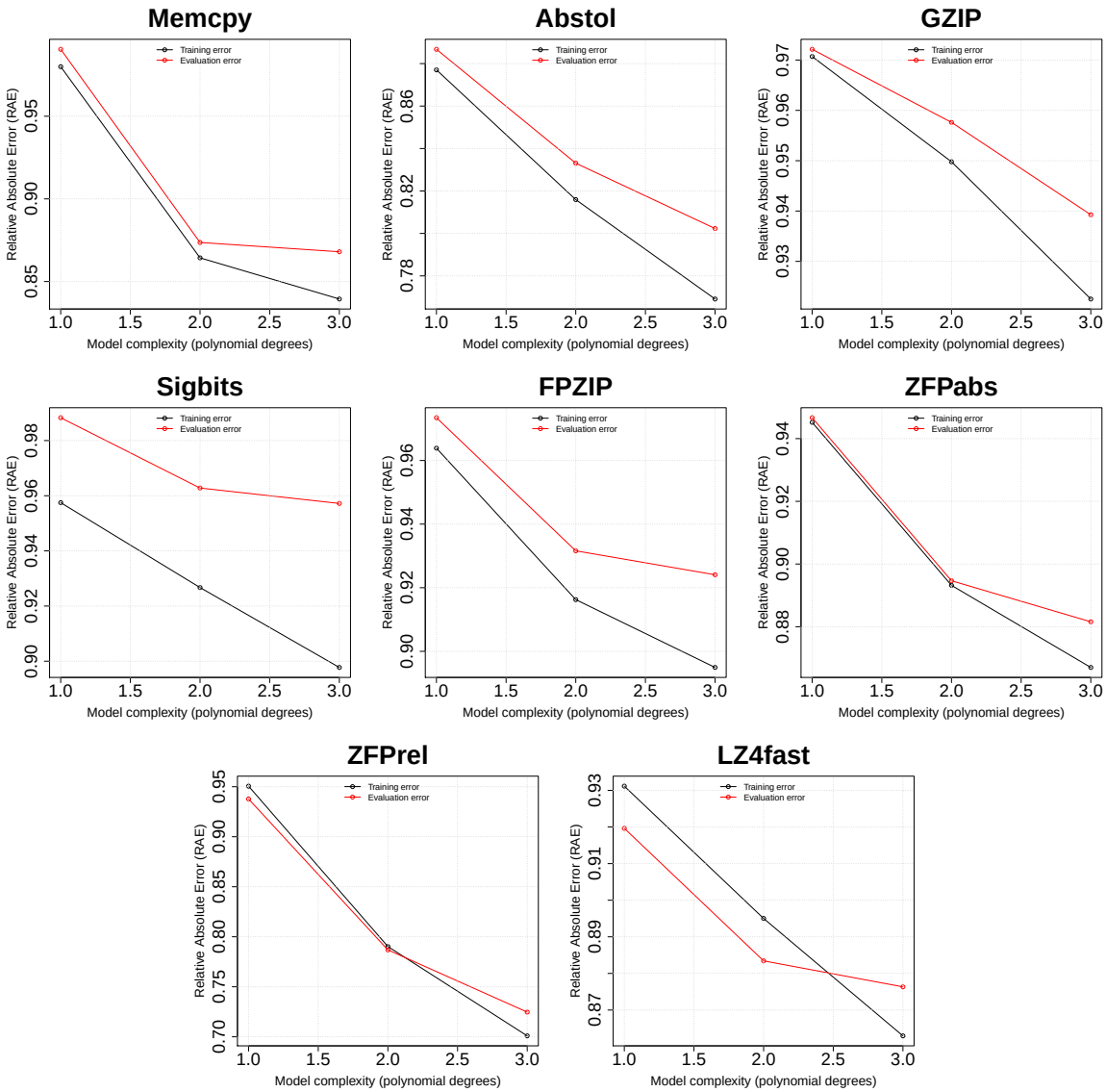


Figure C.8.: RSEs of Decompression Throughput

List of Figures

List of Tables

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich die vorliegende Arbeit im Studiengang B.Sc. Informatik selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel – insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen – benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus Veröffentlichungen entnommen wurden, sind als solche kenntlich gemacht. Ich versichere weiterhin, dass ich die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht habe und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ich bin damit einverstanden, dass meine Abschlussarbeit in den Bestand der Fachbereichsbibliothek eingestellt wird.

Ort, Datum	Unterschrift