

1-18-2023

Comparing the Performance of Different Machine Learning Models in the Evaluation of Solder Joint Fatigue Life Under Thermal Cycling

Jason Scott Ross
Portland State University

Follow this and additional works at: https://pdxscholar.library.pdx.edu/open_access_etds



Part of the [Mechanical Engineering Commons](#)

Let us know how access to this document benefits you.

Recommended Citation

Ross, Jason Scott, "Comparing the Performance of Different Machine Learning Models in the Evaluation of Solder Joint Fatigue Life Under Thermal Cycling" (2023). *Dissertations and Theses*. Paper 6358.
<https://doi.org/10.15760/etd.3425>

This Thesis is brought to you for free and open access. It has been accepted for inclusion in Dissertations and Theses by an authorized administrator of PDXScholar. Please contact us if we can make this document more accessible: pdxscholar@pdx.edu.

Comparing the Performance of Different Machine Learning Models in the
Evaluation of Solder Joint Fatigue Life Under Thermal Cycling

by
Jason Scott Ross

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Mechanical Engineering

Thesis Committee:
Sung Yi, Chair
Hormoz Zareh
Chien Wern

Portland State University
2023

© 2022 Jason Scott Ross

Abstract

Predicting the reliability of board-level solder joints is a challenging process for the designer because the fatigue life of solder is influenced by a large variety of design parameters and many nonlinear, coupled phenomena. Machine learning has shown promise as a way of predicting the fatigue life of board-level solder joints. In the present work, the performance of various machine learning models to predict the fatigue life of board-level solder joints is discussed. Experimental data from many different solder joint thermal fatigue tests are used to train the different machine learning models. A web-based database for storing, sharing, and uploading data related to the performance of electronics materials, the Electronics Packaging Materials Database (EPMD), has been developed and used to store and serve the training data for the present work. Data regression is performed using artificial neural networks, random forests, gradient boosting, extreme gradient boosting (XGBoost), and adaptive boosting with neural networks (AdaBoost). While previous works have studied artificial neural networks as a way to predict the fatigue life of board-level solder joints, the results in this paper suggest that machine learning techniques based on regression trees may also be useful in predicting the fatigue life of board-level solder joints. This paper also demonstrates the need for a large collection of curated data related to board-level solder joint reliability, and presents the Electronics Packaging Materials Database to meet that need.

Dedication

I dedicate this to Kaitlyn. Thank you for your love, support, kindness, patience,
and understanding.

Table of Contents

Abstract	i
Dedication	ii
List of Tables	v
List of Figures	vi
Glossary	viii
1. Introduction	1
2. Failure mechanisms of board-level solder joints	8
3. Data Sources	12
4. Feature Selection	14
5. Regression	17
5.1. Data Retrieval	18
5.2. Data Pre-processing	18
5.2.1. Missing parameter imputation	19
5.2.2. Data point bootstrapping	19
5.2.3. Feature Selection	20
5.2.4. Missing Value Rejection	20
5.2.5. Small Category Rejection	21
5.2.6. Encoding categorical features	21
5.2.7. Scaling numerical features	22
5.3. Model Evaluation	22
5.4. Models	23
5.4.1. Decision Trees	23
5.4.2. Random Forest	25
5.4.3. Gradient Boosting	26
5.4.4. Extreme Gradient Boosting	28
5.4.5. Artificial Neural Network	29
5.4.6. AdaBoost	33
6. Results	35
7. Conclusions	59

8. References	60
Appendix A. Computing parameters of the Weibull distribution	74
Appendix B. Database architecture	77
Appendix C. Plot Digitization	84
Appendix D. Data Values	89
Appendix E. Nested K-Fold Cross-Validation	96
Appendix F. Backpropagation	103

List of Tables

5.1.	Parameter tuning values for Random Forest.	26
5.2.	Parameter tuning values for Gradient Boosting and Extreme Gradient Boosting	28
5.3.	Examples of activation functions used in neural networks.	30
5.4.	Parameter tuning values for artificial neural network.	33
5.5.	Parameter tuning values for AdaBoost with Neural Network	34
6.1.	Hyperparameter combinations used for each fold for the neural network model.	37
6.2.	Hyperparameter combinations used for each fold for the AdaBoost model	37
6.3.	Hyperparameter combinations used for each fold for the random forest model	38
6.4.	Hyperparameter combinations used for each fold for the gradient boosting model	38
6.5.	Hyperparameter combinations used for each fold for the extreme gradient boosting model	38
6.6.	% MAE for different models for the least difficult to predict, median difficulty, and most difficult to predict experiments.	40

List of Figures

2.1.	A typical failure rate plot plotted with a transformation of the form $f(x) = \log(x)$ on the horizontal axis and $g(y) = \log(-\log(1 - y))$ on the vertical axis. Data was generated for illustration purposes only.	11
2.2.	Top: die and substrate at ambient temperature. Bottom: die and substrate expand different amounts when heated resulting in large shear strains developed in solder balls.	11
3.1.	Example of a BGA solder grid. A 4x5 array, with 3 grid positions unpopulated. The number of pad columns is 4; the number of pad rows is 5; the number of pads is 17.	13
4.1.	Factors influencing the reliability of solder joints (Yi and Jones 2019), adapted.	15
5.1.	Model training and evaluation workflow	18
5.2.	Example decision tree which predicts vehicle fuel efficiency from vehicle weight and vehicle horsepower.	24
5.3.	Illustration of artificial neural network consisting of $n - 1$ dense hidden layers with dropout.	32
6.1.	Distribution of % errors of fatigue life. σ is the standard deviation of % error. “95% CI” is the 95% confidence interval for % error.	36
6.2.	Average % error computed for each experiment. “Experiment Id” corresponds with the “Id” column in Table D.1.	39
6.3.	Random forest regression on least difficult experiment to model (Ricky Lee et al. 2002).	41
6.4.	Gradient boosting regression on least difficult experiment to model (Ricky Lee et al. 2002).	42
6.5.	Extreme gradient boosting regression on least difficult experiment to model (Ricky Lee et al. 2002).	43
6.6.	Artificial neural network regression on least difficult experiment to model (Ricky Lee et al. 2002).	44
6.7.	AdaBoost regression on least difficult experiment to model (Ricky Lee et al. 2002).	45
6.8.	Random forest regression on median modeling difficulty experiment (T.-K. Lee et al. 2010).	46
6.9.	Gradient boosting regression on median modeling difficulty experiment (T.-K. Lee et al. 2010).	47

6.10. Extreme gradient boosting regression on median modeling difficulty experiment (T.-K. Lee et al. 2010).	48
6.11. Artificial neural network regression on median modeling difficulty experiment (T.-K. Lee et al. 2010).	49
6.12. AdaBoost regression on median modeling difficulty experiment (T.-K. Lee et al. 2010).	50
6.13. Random forest regression on most difficult experiment to model (Syed et al. 2008).	51
6.14. Gradient boosting regression on most difficult experiment to model (Syed et al. 2008).	52
6.15. Extreme gradient boosting regression on most difficult experiment to model (Syed et al. 2008).	53
6.16. Artificial neural network regression on most difficult experiment to model (Syed et al. 2008).	54
6.17. AdaBoost regression on most difficult experiment to model (Syed et al. 2008).	55

Glossary

N The number of examples in a set.

\mathbf{X} An array of unlabeled examples.

\mathbf{x}_i An unlabeled example indexed by i .

\mathbf{x} An unlabeled example.

\mathbf{y} An array of labels.

$\hat{\mathbf{y}}$ An array of predictions.

\hat{y}_i A single prediction with index i .

\hat{y} A single prediction.

y_i A single label with index i .

y A single label.

Activation Function A function applied to the output of a neuron in a neural network, often to change a linear response into a nonlinear response.

Feature An input variable to a machine learning model.

Hyperparameter A parameter used to control the training process of a machine learning model. Examples include maximum depth of a decision tree, or number of neurons in an artificial neural network.

Label A measured “result” associated with an observation, e.g. if pressure and temperature are used to predict wind velocity, a measured wind velocity would be a “label”.

Labeled Example A correlated set of one or more features and a label corresponding with a single observation, e.g. if pressure and temperature are used to predict wind velocity, a correlated set of measured pressure, temperature, and wind velocity would be a “labeled example”.

Learner A machine learning model, usually in the context of an ensemble of several models.

Loss Function A function which takes a label and a prediction and returns a positive scalar value which quantifies the error.

Prediction A predicted “result” associated with an observation as produced by a machine learning model.

Unlabeled Example A collection of one or more features corresponding with a single observation, e.g. if pressure and temperature are used to predict wind velocity, a correlated set of measured pressure and temperature would be an “unlabeled example”.

Weibull Distribution A probability distribution often used to model the reliability of mechanical components.

1. Introduction

The design of board-level solder joints is a major factor influencing the cost, reliability, and sustainability of electronic devices. Board-level solder joints provide a mechanical, electrical, and thermal interface between the printed circuit board (PCB) and various electronic packages. Due to differences in the coefficient of thermal expansion between the package and the PCB, repetitive mechanical strains develop in the solder connection which can lead to fatigue failure. A failed connection can cause failure of the entire device, with potentially catastrophic consequences: for instance, cracking of a solder joint initiated a sequence of events that lead to the total loss of an Airbus A320-216 aircraft and all passengers and crew (Komite Nasional Keselamatan Transportasi 2014). As electronic devices become smaller and more interconnected, and as these devices are increasingly relied on for safety-critical applications, the prediction of fatigue life of board-level solder joints becomes an increasingly important problem.

Multiple factors influence the reliability of board-level solder joints including solder material, under-bump metallurgy, electrical parameters, and physical dimensions. Packaging engineers must specify design parameters from a large configuration space in order to ensure that board-level solder joints will meet the reliability requirements for the product. The choice of design parameters can be very difficult for the designer because different design parameters can interact in unexpected and complex ways.

One example of how different design and service parameters interact is the process of formation and growth of intermetallic compounds (IMCs). IMCs form due to chemical reactions between the solder alloy and the solder pad metallization layers.

The formation of IMCs in a solder is necessary in order to achieve wetting of the solder to the base metal (Tegehall 2006). However, IMC layers are generally brittle, and fracture is often observed in the IMC layer with lead-free solder alloys (Sona and Prabhu 2013; Tegehall 2006). IMC growth can also reduce joint strength due to the formation of Kirkendal voids by diffusion of atoms into the IMCs (Tegehall 2006). In addition, Z. Huang, Conway, and Thomson (2007) demonstrated that joint scale influences IMC growth and found that smaller joints have thicker IMC layers. Also, Hua Zhong and Yi (1999) and Berthou et al. (2009) analyzed how thermal aging and thermal cycling influenced IMC growth rate and found that thermal cycling can cause substantial changes in the IMC microstructure. Additionally, the formation of IMCs is influenced by solder alloy composition, solder pad metallization, and surface finish (Tegehall 2006). In order to properly manage the formation of IMCs, the designer must select solder materials, surface treatments, fluxes, and solder process parameters such that IMCs are able to form during the soldering step but don't grow too much afterwards, while considering the thermal environment of the joint in service and the physical dimensions of the joint. This alone is difficult and requires the consideration of a large number of factors, but the formation of IMCs is only one of many mutually interacting mechanisms that must be managed to ensure adequate connection life.

Another phenomenon which can be difficult to predict is the failure mechanism itself. The failure mechanism of board-level solder joints is often a combination of plastic fatigue and creep. The rate of change of plastic strain and creep depend on the load, the elastic modulus, and the creep rate of the material. However, the yield point, elastic modulus, and creep rate are temperature-dependent. Therefore, the *response* to a change in thermal loading is dependent on the thermal load itself and therefore the response to the thermal load is nonlinear. G. Chen et al. (2019) developed a unified

constitutive model for the response of lead-free solder to thermal and mechanical loads. Their model used 12 empirically-determined material parameters which they identified for SAC305 solder alloy. Y. Chen, Jin, and R. Kang (2017) investigated the relation between creep damage and fatigue damage for the prediction of solder joint life. They proposed a model for combined creep and fatigue damage where each damage mechanism is summed to compute a total damage. Their model considered a two-stage creep degradation rate where the rate would increase if the cumulative damage exceeded a certain threshold. Their model used 10 empirically-determined parameters for their reliability model in addition to 6 material parameters. Thus, even knowing the thermally-induced mechanical loading on a solder joint in service, the prediction of fatigue failure in board-level solder joints involves a number of empirical constants and solving a nonlinear problem, which presents a significant challenge for the designer.

Accelerated testing is often used to predict the fatigue life of solder joints. JEDEC specifies standard testing conditions for accelerated thermal testing (JEDEC Solid State Technology Association 2009). However, this testing is expensive and time consuming. Fatigue testing involves preparing large numbers of specimens by soldering packages onto PCBs and wiring them in such a way that failure of soldered connections can be detected (one strategy is to “daisy chain” the connections so that they all form a continuous series circuit). During accelerated thermal testing, the specimens are placed inside environmental chambers and subjected to repeated thermal cycling. This thermal cycling typically takes thousands of hours to complete. Once enough samples have failed, the designer can interpret the data and plot the fatigue life using various software.

Machine learning has shown promise as a way to predict the fatigue life of board-

level solder joints. Subbarayan, Li, and Mahajan (1996) used a neural network to optimize solder joint manufacturing process parameters. They used finite element modeling (FEM) to generate the training data for their neural network. Their finite element modeling process consisted of two discrete steps: a shape-prediction step in which the solidified shape of the solder ball was predicted using a finite element formulation of the minimum energy equation; and a fatigue-life-prediction step in which the geometry found in the shape-prediction step was analyzed using FEM and the fatigue life was predicted from a maximum energy dissipation model. Their neural network consisted of a single hidden neuron, four input neurons, and one output neuron. They performed a total of 8 finite element simulations to generate training data. Yi and Jones (2019) used a dense, fully connected neural network to predict failure rate. They trained their model using physical testing data on chip resistors from Collins, Punch, and Coyle (2012). They considered different pad surface finishes in their results. They found that their neural network could predict the failure rate at a given number of cycles for a given surface finish more accurately than a Weibull model. Yuan and C.-C. Lee (2020) used recurrent neural networks and long short-term memory neural networks to predict thermal cycling performance using training data generated by finite element modeling. They modeled the physical behavior of the solder using a temperature-dependent, elastic-plastic material model and predicted failure using accumulated plastic strain. Their data set consisted of 81 finite element model examples. Samavatian et al. (2020) used a novel artificial neural network topology in which both features as well as convolutional operations on features were fed into several dense, fully-connected layers to predict thermal cycling performance. They used a time-temperature-dependent creep-fatigue rainflow counting algorithm (Samavatian 2020) to predict the fatigue life of solder joints,

with training data generated in a finite element modeling program. Their data set consisted of 450 finite element model examples. T.-C. Chen, Alazzawi, et al. (2021) used a neural network with 3 dense, fully-connected hidden layers and 50 neurons per hidden layer with solder joint geometry and thermal cycle parameters as features to predict the number of cycles to failure. They used the Garofalo-Arrhenius model, a hyperbolic sine creep constitutive equation of the form

$$\dot{\varepsilon}_{cr} = C_1 [\sinh(C_2\sigma)]^{C_3} e^{-\frac{C_4}{T}} \quad (1.1)$$

where $C_{1...4}$ are experimentally-determined constants, σ is the von Mises stress, T is the temperature, and $\dot{\varepsilon}_{cr}$ is the creep strain rate. They used a fatigue lifetime model of the form

$$N_f = \frac{1}{C\varepsilon_{acc}} \quad (1.2)$$

where C is a material constant and ε_{acc} is the accumulated creep strain per cycle. Their analysis considered variations in dwelling temperature, dwelling time, and solder joint volume of $\pm 20\%$ and two different solder joint shapes. They performed physical testing on three different solder joint volumes as well as three different temperature ramp rates to validate the results of their FE model. T.-C. Chen, Opuencia, et al. (2022) followed up upon this analysis by expanding the parameter space to include more solder alloys and more geometry parameters. This work used a radial basis neural network (RBNN) with a single hidden layer using a Gaussian radial function as the activation function. Chou, Chiang, and Liang (2019) used an artificial neural network trained on examples generated using finite element models with various upper and lower pad diameters, chip thicknesses, and stress buffer layer thicknesses. They considered several neural networks with different numbers of hidden layers and

neurons per layer. They used an empirical Coffin-Manson plastic strain model of the form

$$N_f = C (\Delta\varepsilon_{eq}^{pl})^{-\eta} \quad (1.3)$$

to predict the fatigue life of the joint where N_f is the cycles to failure, C and η are empirical constants, and $\Delta\varepsilon_{eq}^{pl}$ is the equivalent plastic strain per cycle. Their analysis only considered SAC305 solder.

To the knowledge of the authors, only Yi and Jones (2019) exclusively used data from physical testing to train machine learning models to predict board-level solder joint fatigue life, and, to the knowledge of the authors, only artificial neural networks have been used to date to predict the performance of board-level solder joints. Traditional machine learning methods such as decision trees, support vector machines, and clustering have shown significant promise for data mining tasks (Wu et al. 2008). Artificial neural networks have become more useful for regression and classification tasks since 2008, however, this progress has relied on the availability of large, well-labeled data sets, e.g. ImageNet (Deng et al. 2009; Brunton and Kutz 2019). Traditional learning strategies have a few advantages over neural networks: they are often faster to train, they are often less “opaque” in the sense that the importance of different features may be inspected, and they may handle missing values automatically. While there exists a wide variety of experimental data related to the reliability of board-level solder joints in thermal fatigue, there does not yet exist a similar large, well-labeled data set to facilitate training of more sophisticated machine learning models.

This paper presents a comparison of several different machine learning models applied to the problem of predicting the fatigue life of solder joints. Traditional machine learning models based on regression trees are compared with artificial neural networks. The data set used in this paper was collected from published thermal fatigue testing

data. 286 candidate experiments are included in the data set, collected from 48 papers. To facilitate future research efforts, a web-based database has been developed, the Electronics Packaging Materials Database, for the collection and dissemination of data related to the performance of electronics materials.

2. Failure mechanisms of board-level solder joints

Various fatigue damage models have been proposed for the purpose of evaluating the fatigue life of solder joints. These models may be classified by the damage mechanisms used in the model. Damage mechanisms used in solder fatigue life models include plastic and creep strain, energy, and damage. One of the most common plastic-strain models is the Coffin-Manson model which has the general form

$$\frac{\Delta\varepsilon_p}{2} = \varepsilon'_f (2N_f)^c \quad (2.1)$$

where $\Delta\varepsilon_p$ is the plastic strain amplitude, ε'_f is the fatigue ductility coefficient, N_f is the number of cycles to failure, and c is the fatigue ductility exponent. Energy-based models typically predict fatigue life based on energy under the stress-strain hysteresis loop. Damage-based fatigue models are based on fracture, creep, or fatigue damage mechanisms (W. W. Lee, Nguyen, and Selvaduray 2000).

The failure mechanism considered in this paper is thermal cycling fatigue. In board-level solder joints, a component is soldered to a substrate where the component and substrate have different coefficients of thermal expansion (CTE). When the component and substrate are subjected to changes in temperature, they expand or contract at different rates as shown in Fig. 2.2. However, the solder joint mechanically couples them together. This results in stresses being developed in the solder joint as well as the package and substrate. The strain in the solder joints due to the thermal load is approximated by

$$\gamma = \frac{u_p - u_s}{t_a} \quad (2.2)$$

where γ is the shear strain, u_p is the displacement of the package due to thermal stress, u_s is the displacement of the substrate due to thermal stress, and t_a is the separation between the package and the substrate. There is an implicit relationship between u_s and u_p since the displacement of the package and displacement of the substrate are coupled by the solder joint. Under assumptions of linear material behavior, the average shear stress developed in the solder balls is of the form

$$\tau = \frac{G(\alpha_p - \alpha_s)\Delta T}{t_a\beta} \frac{\sinh(\beta x)}{\cosh(\beta L)} \quad (2.3)$$

The parameter β is defined as

$$\beta = \sqrt{\frac{G(E_s t_s + E_p t_p)}{E_p E_s t_a t_p t_s}} \quad (2.4)$$

α_p, α_s are the coefficients of thermal expansion of the package and substrate, respectively; E_p, E_s are Young's Modulus of the package and substrate, respectively; t_a, t_p, t_s are the thicknesses of the solder connection, package, and substrate, respectively; L is half the diagonal dimension of the package; x is the distance to a particular solder ball where the shear stress is measured; G is the shear modulus of the solder ball/die attach; and ΔT is the temperature variation.

The material parameters in Eq. (2.3) and Eq. (2.4) are not generally constant, and in fact E and G can vary significantly within the operating temperature range of a board-level device. Additionally, the assumption of linear elastic behavior is not generally satisfied. Large thermal cycles can exceed the elastic limit of the solder, so the shear stress will not generally be a linear function of the strain. Additionally, the relation between stresses/strains in the solder joints and the fatigue life of the joints is difficult to predict, because it depends on many factors including the mi-

crostructure of the solder, where the microstructure can be influenced by e.g. alloy composition, under-bump metallization, flux, reflow temperature, and thermal history. The Weibull distribution has been shown to be useful for modeling the fatigue life of solder joints from empirical data. The Weibull distribution uses a hazard rate h_t of the form

$$h(t) = C_1 t^{C_2} \quad (2.5)$$

which can be understood as a power-law hazard rate where C_1 and C_2 are constant parameters and t is a measure of the fatigue life, e.g. time to failure or number of cycles to failure. The Weibull distribution is more commonly parameterized as

$$h(t) = \frac{\beta}{\theta} \left(\frac{t}{\theta} \right)^{\beta-1} \quad (2.6)$$

such that the parameter θ represents the number of cycles where the reliability is $\frac{1}{e}$ and β is a parameter influencing the variability of fatigue life (Johnson, Kotz, and Balakrishnan 1994). The reliability function is

$$R(t) = e^{-\left(\frac{t}{\theta}\right)^\beta} \quad (2.7)$$

The parameters β, θ may be estimated from experimental data using the procedure described in Appendix A. A typical failure rate plot approximating a Weibull distribution generated with random data is presented in Fig. 2.1. The plotted line represents the failure rate predicted by the Weibull distribution with parameters computed by a least-squares fit on the data. The slope of the line is related to the parameter β and the horizontal position of the point $F(t) = 1 - \frac{1}{e}$ is equal to the parameter θ .

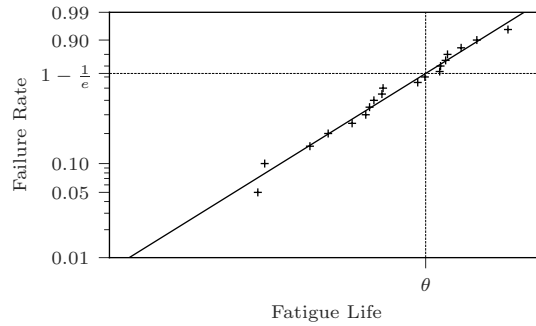


Figure 2.1.: A typical failure rate plot plotted with a transformation of the form $f(x) = \log(x)$ on the horizontal axis and $g(y) = \log(-\log(1 - y))$ on the vertical axis. Data was generated for illustration purposes only.

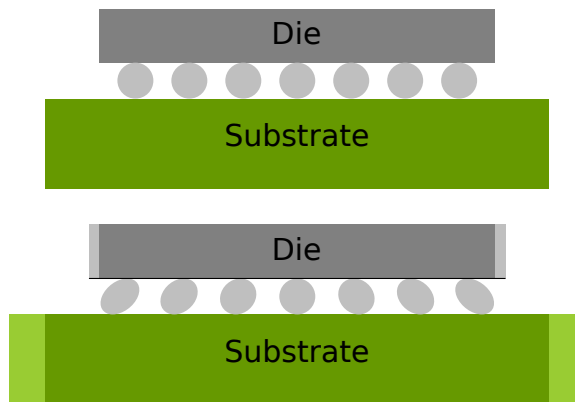


Figure 2.2.: Top: die and substrate at ambient temperature. Bottom: die and substrate expand different amounts when heated resulting in large shear strains developed in solder balls.

3. Data Sources

Data was collected from a corpus of research on solder joint fatigue. The data was stored in an SQLite database and retrieved using a REST API (Fielding 2000). The SQLite database engine was selected to provide sufficient functionality for research needs in a reasonable amount of time. Additional details about the database architecture are described in Appendix B. The data was collected as part of an ongoing project to produce a public web portal for manufacturers and researchers to store and query data related to the performance of solder joints. Table D.1 lists the parameters used in the analysis for each experiment. The procedure for collecting fatigue life data from research papers is described in Appendix C.

Fig. 3.1 demonstrates how solder grid parameters were recorded. Frequently, pad arrays are not fully populated and pads are missing from grid locations for e.g. ensuring rotational alignment of the component. When recording information about the grid array for these components, the recorded number of rows and columns includes all rows and columns in the grid array whether or not they are fully populated. The same applies for the width and length of the array.

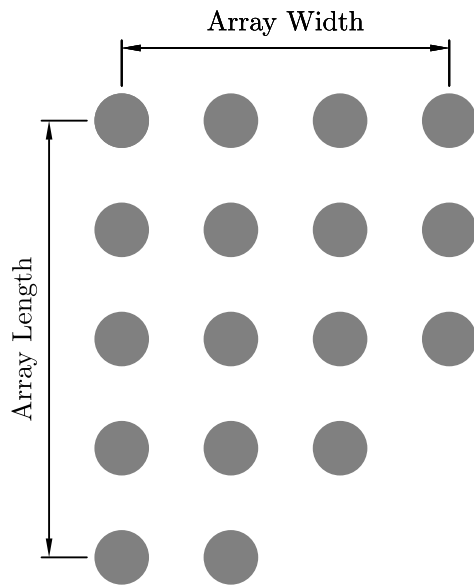


Figure 3.1.: Example of a BGA solder grid. A 4x5 array, with 3 grid positions unpopulated. The number of pad columns is 4; the number of pad rows is 5; the number of pads is 17.

4. Feature Selection

The choice of which features to include in a machine learning model has significant effects on the performance of the model. Including more features can improve model performance by providing more information about each example from which to predict an accurate label. However, the more features are included, the more ways exist for spurious correlations in the data to negatively influence the model. In addition, increasing the number of features may reduce the number of examples that can be included in the data set because many examples are missing information for some of the features. Also, increasing the number of features increases model complexity which can increase the time required to train models. Yi and Jones (2019) identify factors influencing the life of solder joints (Fig. 4.1). Due to limitations in the available data, only a subset of these factors could be included in the training data in the present work. In order to select which features would be included in this study, the space of all combinations of features in the corpus was exhaustively explored by training an XGBoost model on each subset of features and scoring it on the entire data set using hold-one-out cross validation. XGBoost was chosen for this analysis because it was fast to train and this analysis required exploring a very large number of combinations. The set of features which were found to perform the best using this method correspond to well-understood physical mechanisms for solder joint fatigue: the pad array length l_A and width w_A are related to L in Eq. (2.3) by the Pythagorean theorem $L = \sqrt{l_A^2 + w_A^2}$; the solder joint height t_a is often strongly correlated with the pitch, which is the ratio of array size to array count; the physical properties of the package E_p, α_p, t_p are usually correlated with the package type and array size; the

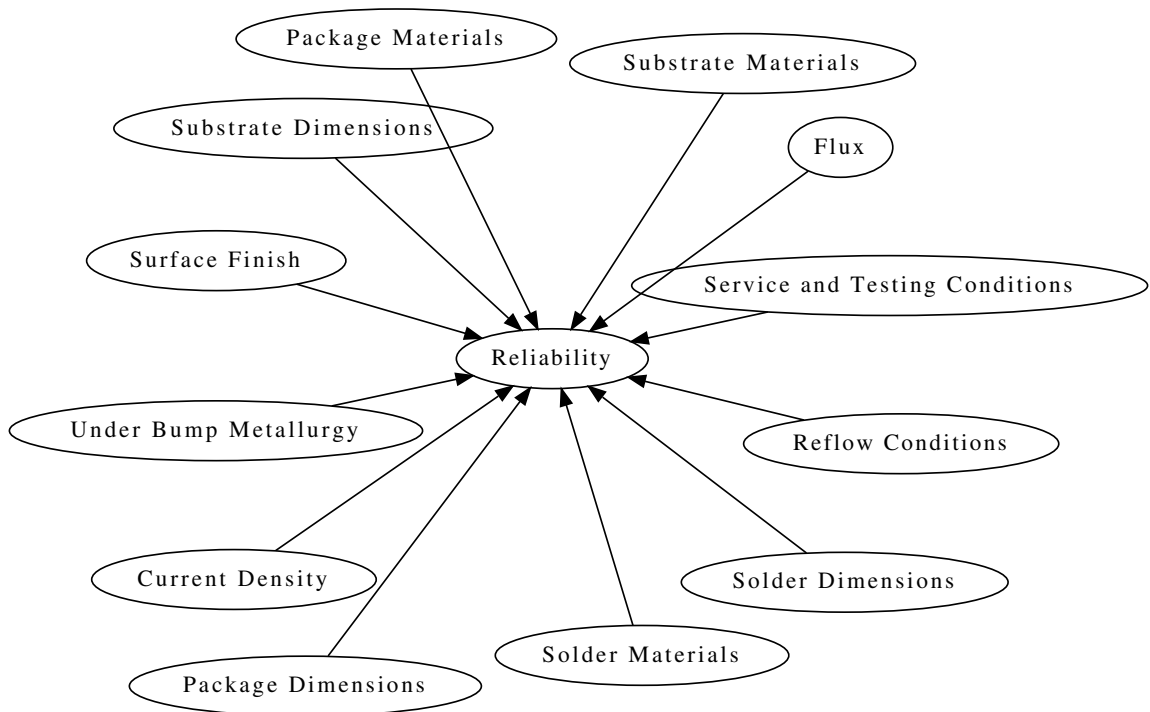


Figure 4.1.: Factors influencing the reliability of solder joints (Yi and Jones 2019), adapted.

physical properties of the solder are correlated with the temperature and the solder material; and finally, aging time and temperature as well as board surface finish are known to influence fatigue life (T.-K. Lee et al. 2010; Collins, Punch, and Coyle 2012).

5. Regression

Data retrieval, pre-processing, and model evaluation proceeds as illustrated in Fig. 5.1. Data is retrieved from the database using the REST API. The data is then pre-processed to clean and transform the data into a structure suitable for each model. Finally, the models are trained and scored on the pre-processed data.

ML models were trained to predict the logarithm of fatigue life. The logarithm was used so that the percent error in fatigue life would be minimized. Given a fatigue life t and a predicted life \hat{t} , the percent error $\epsilon_{\%}$ is

$$\epsilon_{\%} \equiv 100 \frac{\hat{t} - t}{t} \quad (5.1)$$

The error in the logarithm of fatigue life ϵ_{\log} is defined as

$$\epsilon_{\log} \equiv \log \hat{t} - \log t \quad (5.2)$$

Eq. (5.1) is related to Eq. (5.2) by

$$\epsilon_{\%} = 100e^{\epsilon_{\log}} - 100 \quad (5.3)$$

Inspecting equation Eq. (5.3) shows that if ϵ_{\log} is zero, so is $\epsilon_{\%}$, furthermore, as ϵ_{\log} increases, so does $\epsilon_{\%}$, and as ϵ_{\log} decreases, so does $\epsilon_{\%}$. Therefore, a model which minimizes ϵ_{\log} should also minimize $\epsilon_{\%}$.

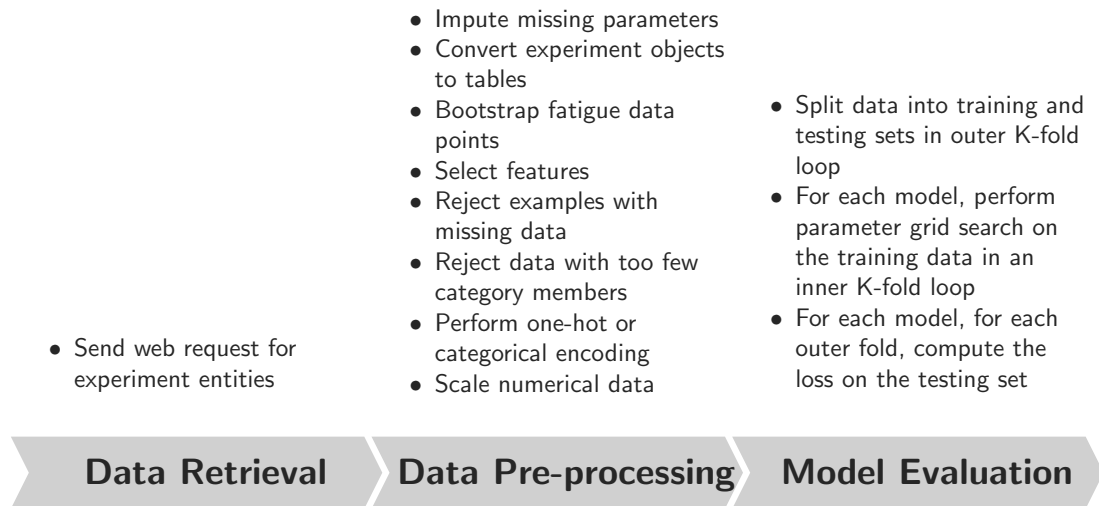


Figure 5.1.: Model training and evaluation workflow

5.1. Data Retrieval

The user searches for experiments using the REST API or web interface. Experiments are represented as data structures which contain lists of attribute-value pairs that represent entries in Table D.1. Experiments for which a fatigue life plot was reported also contain a table of values taken from the fatigue life plot as described in Appendix C.

5.2. Data Pre-processing

Data pre-processing consists of operations on the collected data in order to remove bad or missing data, perform feature selection, and transform and scale the data to be more suitable for training.

5.2.1. Missing parameter imputation

The “parameter imputation” step involves computing parameter values from other parameter values directly. The following parameters are computed during this step.

Number of pins If an array of pins is rectangular, and the number of rows and columns are known, the total number of pins is the product of the number of rows and the number of columns.

Pin array dimension If the number of rows of pins, the number of columns of pins, and the pitch of the pins is known, then the pin array dimensions are found by multiplying the pitch by the number of rows or columns.

5.2.2. Data point bootstrapping

For each experiment in the corpus, there are different numbers of samples used to collect fatigue life data. As a result, the fatigue life plots for each experiment have different numbers of points. This presents a problem for analysis since an experiment with a small number of points contributes fewer examples to the data set than an experiment with a large number of points. This can result in larger errors for experiments with fewer points. To mitigate this issue, the “series bootstrapping” step bootstraps the data series associated with each experiment to an equal length by randomly sampling 30 points from the series with equal probability. In this way, each experiment is represented in equal proportion in the training set. Each point in the bootstrapped series along with all associated attributes of the given experiment are joined into a data table listing all experimental attributes as well as fatigue life and

failure rate for each data point in the entire corpus. This step results in “flattening” the (`attributes`, `series`) data structures into rows in a data table where each row is a labeled example and the label is the fatigue life.

5.2.3. Feature Selection

The “feature selection” step selects which attributes are to be included in the analysis. The features selected in this step are listed in Table D.1.

5.2.4. Missing Value Rejection

For each experiment in the corpus, different experimental attributes are reported. Frequently, experimental information is completely missing and cannot be computed from other information about the experiment, e.g. board surface treatment may be completely unreported. This presents a problem during analysis because the model used may not allow any missing data, or the model may perform worse if examples with too much missing data are included in the data set. The “missing value rejection” step removes rows in the data table where too many values are missing. The number of values which may be missing depends on the choice of model. For models which do not have native handling of missing values, e.g. random forests or artificial neural networks, any row with missing values is removed at this step. All models used except for XGBoost did not allow any missing values. The XGBoost model rejected rows where 20% or more of the features were missing.

5.2.5. Small Category Rejection

Some experiments in the corpus represent the only known member of a category. For example, there may be only one experiment which uses a new exotic solder material: the performance of this solder material provides no new information about the performance of other solder materials and vice versa, therefore, including this material in the data set can only hurt model performance. The “small category rejection” step removes experiments which include categories that are not represented by enough experiments. The minimum number of experiments representing a category was 2.

5.2.6. Encoding categorical features

The machine learning models used in this analysis require numerical data for all features. Two different strategies were used to handle categorical data. For the XGBoost model, categorical features were encoded as integers, e.g. if the feature “package type” included the variants “BGA”, “FlexBGA”, and “LGA”, “BGA” might be encoded as the number 1, “FlexBGA” might be encoded as the number 2, and “LGA” might be encoded as the number 3. This strategy was used only with the XGBoost model because XGBoost includes semantics for identifying features as categorical and interpreting them correctly during training.

Encoding variants of a category as different integers is not a good strategy for models without semantics for identifying categorical features because the model may erroneously interpret the relationship between category codes as meaningful. For example, using the aforementioned encoding strategy, “BGA” and “FlexBGA” may

be interpreted as more similar than “BGA” and “LGA”, but in reality, no such relationship exists. To solve this problem, a strategy known as “one-hot” encoding is employed. With one-hot encoding, a categorical feature with n variants is represented by an array of n numbers, where each entry in the array is zero except for the entry corresponding to the integer value of the category. For instance, using the previous example for categorical encoding, “BGA” might be encoded as the array $[1, 0, 0]$, “FlexBGA” might be encoded as the array $[0, 1, 0]$, and “LGA” might be encoded as the array $[0, 0, 1]$. With the one-hot encoding strategy, no relationship between the different variants of a categorical feature is implied by the encoding because each variant is encoded as a separate feature.

5.2.7. Scaling numerical features

With the artificial neural network model, numerical data was scaled to the range $[0, 1]$ in order to improve model performance. This was done because the range of values a feature can take does not imply anything about how strongly it influences the output. By preemptively scaling numerical data, the model did not have to “train out” built-in biases due to the different value ranges of the different features. This step was not used for the regression-tree-based models because the range of values a feature takes does not influence the output of a regression tree.

5.3. Model Evaluation

Nested cross-validation was used to partition the data into multiple training, testing, and validation sets (Refaeilzadeh, Tang, and H. Liu 2009). Nested cross-validation

was used because it limits variance in the model due to the arbitrary choice of testing data; in nested cross-validation, the *entire* data set is used for testing. In the nested cross-validation algorithm used, all data from an experiment was treated as an indivisible group such that data from a single experiment would appear either entirely in the training set or entirely in the testing set. The nested cross-validation algorithm is described in detail in Appendix E.

5.4. Models

Two classes of models were considered. The first class is based on regression trees, and includes a random forest model, a gradient boosting model, and an XGBoost model. The second class is based on artificial neural networks and includes a dense fully-connected neural network model and an AdaBoost model which uses several dense fully-connected neural networks as base learners.

5.4.1. Decision Trees

Decision trees are a family of machine learning algorithms which make predictions by traversing a *tree*, a type of data structure which encodes the relationship between features and labels as a kind of flowchart. Decision trees work in a similar fashion to the children’s game “20 questions”. Each parent (i.e. non-leaf) node encodes a question such as “is the number of pins less than 100”. Each node (except leaves) contains a split feature and split threshold. In the example “is the number of pins less than 100”, the split feature would be “number of pins” and the split threshold would be 100. A prediction is made when a leaf in the tree is reached. The value

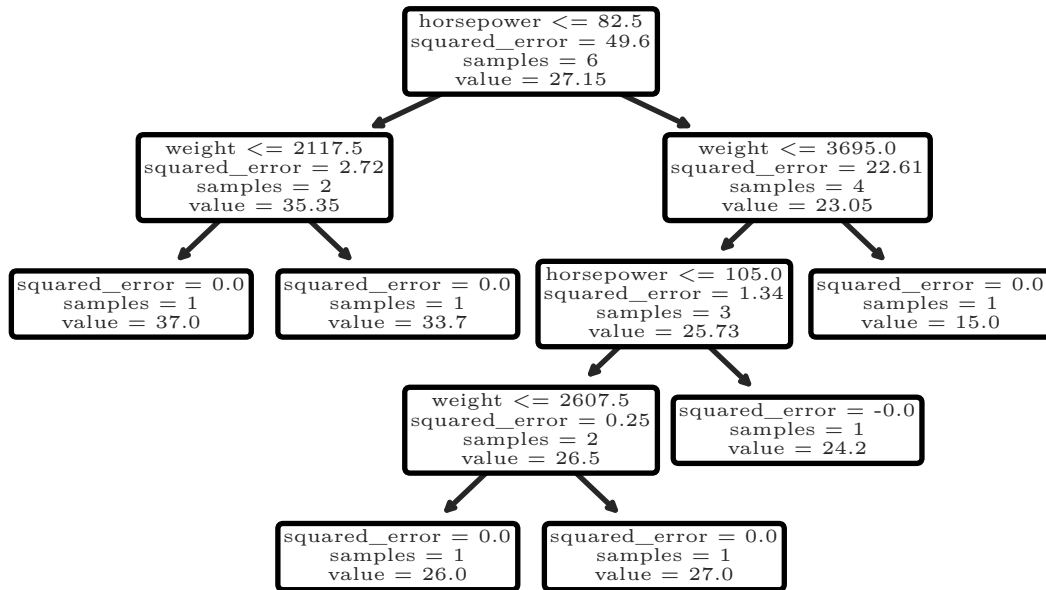


Figure 5.2.: Example decision tree which predicts vehicle fuel efficiency from vehicle weight and vehicle horsepower.

predicted is the value stored in the leaf. Figure Fig. 5.2 shows an example of a decision tree used to predict motor vehicle fuel economy from weight and horsepower. In this example, a vehicle with 100 horsepower that weighs 2500 lbs would have a predicted fuel economy of 26 miles per gallon. Decision trees are constructed using a greedy algorithm where each node splits the data so that the variance in the subgroups is minimized. Each node can split the data on any of the features. The best feature and the best split threshold are chosen for each node. Decision trees are the basis for some of the ensemble methods used in this paper but not by themselves because they have a tendency to over-fit the data.

5.4.2. Random Forest

Random forests are ensembles of decision trees that are trained on random bootstrapped training sets prepared from the training data. Traditionally, the bootstrapped sets are the same size as the training data but only contain observations from two-thirds of the training data. The remaining one-third is selected by randomly resampling from the first two-thirds. Multiple bootstrapped sets are prepared in this fashion and each set is used to train a different decision tree (this process of generating multiple *bootstrapped* datasets and *aggregating* them to form a stronger learner is called *bagging*). In addition, each decision tree only uses a random subset of the features to split the data (Breiman 2001).

The random forest algorithm was used to predict fatigue life. Parameters for the grid-search optimization are listed in Table 5.1. All combinations of the listed parameters were used in the grid search. “Maximum depth” represents the maximum depth of the decision trees in the forest, 1 being a stump. The “minimum samples to split a node” are the number of samples that must be assigned to a node in order for the node to be a candidate for splitting. The “minimum samples for a leaf” is the minimum number of samples to be assigned to a leaf after a split is performed in order for the split to be allowed; if fewer samples would be assigned to a child leaf then the node will not be split. The “maximum split features” is the fraction of features that are considered when splitting; a random subset of the features are chosen if less than 1.

Table 5.1.: Parameter tuning values for Random Forest.

Maximum Depth	Minimum Samples to Split a Node	Minimum Samples for a Leaf	Maximum Split Features
1	2	1	0.01
2	4	3	0.046
5	10	10	0.22
13			1
31			

5.4.3. Gradient Boosting

Gradient boosting is a machine learning algorithm developed by Friedman (2001) in which multiple weak learners (e.g. decision trees) are combined into a single strong learner. Gradient boosting proceeds as follows. First, an initial scalar prediction ρ is found which minimizes

$$\sum_{i=1}^N L(y_i, \rho) \quad (5.4)$$

where $L(y, \hat{y})$ is a loss function. An example loss function is the squared error $L(y, \hat{y}) = (y - \hat{y})^2$, which is minimized when \hat{y} equals the mean of y . The first strong learner is defined as

$$F_0(\mathbf{x}_i) = \rho \quad (5.5)$$

Subsequent strong learners are computed greedily using the following algorithm:

$$F_m(\mathbf{x}_i) = F_{m-1}(\mathbf{x}_i) + \rho_m h(\mathbf{x}_i; \mathbf{a}_m) \quad (5.6)$$

where $h(\mathbf{x}_i; \mathbf{a}_m)$ is a weak learner with parameters \mathbf{a}_m and ρ_m is a scalar parameter to be computed.

The parameters \mathbf{a}_m in Eq. (5.6) are found by minimizing

$$\sum_{i=1}^N \left[-\frac{\partial L(y_i, F_{m-1}(\mathbf{x}_i))}{\partial F_{m-1}(\mathbf{x}_i)} - \beta h(\mathbf{x}_i; \mathbf{a}_m) \right]^2$$

Thus, the parameters \mathbf{a}_m form a best least-squared approximation of some multiple β of the gradient of the loss function for all examples in the data set (in other words, \mathbf{a}_m is a vector which points in the direction along the gradient of the loss function in parameter space). The scalar parameter ρ_m is found by minimizing

$$\sum_{i=1}^N L(y_i, F_{m-1}(\mathbf{x}_i) + \rho_m h(\mathbf{x}_i; \mathbf{a}_m))$$

once the parameters \mathbf{a}_m have been found. In other words, at each step m we compute parameters \mathbf{a}_m such that weak learner $h(\mathbf{x}; \mathbf{a}_m)$ is proportional to a least-squared approximation of the gradient of the loss function at the previous step. This weak learner is then scaled by a constant ρ_m and added to the previous best guess F_{m-1} such that the new strong learner minimizes the loss function.

When using the squared error as the loss function, the problem becomes finding ρ_m, \mathbf{a}_m which minimize

$$\sum_{i=1}^N [y_i - F_{m-1}(\mathbf{x}_i) - \rho_m h(\mathbf{x}_i; \mathbf{a}_m)]^2$$

i.e., compute ρ_m, \mathbf{a}_m which best approximate the residuals $y_i - F_{m-1}(\mathbf{x}_i)$ in a least-squared sense. In other words, when using squared error as the loss function, gradient boosting trains the weak learners such that each weak learner approximates the negative of the error of the previous strong learner.

The gradient boosting algorithm used in this paper is from the Scikit-Learn project

Table 5.2.: Parameter tuning values for Gradient Boosting and Extreme Gradient Boosting

Maximum Depth	Learning Rate	Subsample Ratio
3	0.032	0.1
5	0.01	0.18
10	0.0032	0.32
17		0.56
31		1

(Pedregosa et al. 2011). The model uses regression trees as the weak learners. Squared error was used as the loss function. The parameters used in the grid search algorithm to tune the gradient boosting model are listed in Table 5.2. “Maximum depth ” is the maximum depth of the decision trees used in the ensemble model. “Learning rate” is a constant value used for ρ in Section 5.4.3. “Subsample ratio” is the fraction of samples used to fit the individual learners in the ensemble; if this value is less than 1, then stochastic gradient boosting is used.

5.4.4. Extreme Gradient Boosting

Extreme gradient boosting (XGBoost) attempts to improve upon gradient boosting using sparsity-aware algorithms and weighted quantile sketch for approximate tree learning (T. Chen and Guestrin 2016). XGBoost handles missing values in the data natively by using a *default direction* for each node in the tree. If a value is missing, it is split in the default direction. The best default direction for each node is another parameter to be trained in the decision tree. Parameters used for the grid search algorithm to tune the XGBoost model are listed in table Table 5.2.

5.4.5. Artificial Neural Network


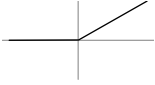


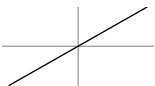
The concept of the neural network was first developed by McCulloch and Pitts (1943). Neural networks were initially developed as a way of formalizing the neural activity in the brain. In their original formulation, neurons would have a binary activation state (i.e. either “on” or “off”, with no values in between) and each neuron would be activated by having a sufficient number of impinging neurons being “on”. Rosenblatt (1958) formulated the *perceptron*, the first *artificial* neural network. In Rosenblatt’s formulation, the perceptron is analyzed as a hypothetical model for the nervous system. Mathematically, an m -layer perceptron can be written as

$$\hat{\mathbf{y}} = f_m\left(\mathbf{A}_m f_{m-1}\left(\cdots \mathbf{A}_2 f_1\left(\mathbf{A}_1 \mathbf{X} + \mathbf{b}_1\right) \cdots + \mathbf{b}_{m-1}\right) + \mathbf{b}_m\right) \quad (5.7)$$

where $\mathbf{A}_{1\dots m}$ are matrix parameters called *weights* which control the strength of connections between the layers as illustrated in Fig. 5.3 and $\mathbf{b}_{1\dots m}$ are vector parameters called *biases* which shift the outputs of each layer. The learning task is to compute the optimal $\mathbf{A}_{1\dots m}$ and $\mathbf{b}_{1\dots m}$ such that $\hat{\mathbf{y}} - \mathbf{y}$ minimizes the total loss. The functions $f_{1\dots m}$ are called *activation functions* and serve to transform the output of the neurons. In a critical difference from McCulloch & Pitts’ original formulation where the output of a neuron was either 1 or 0, the activation of a neuron in a perceptron can take on any real value. Some of the activation functions should be non-linear in order for the neural network to be able to approximate non-linear functions. If all activation functions are linear, e.g. $f_i(\mathbf{X}) = \mathbf{B}_i \mathbf{X} + c_i$, Eq. (5.7) reduces to a linear function of \mathbf{X} (Brunton and Kutz 2019). Table 5.3 lists some commonly used activation functions.

A key development that has enabled artificial neural networks to be viable is the

Table 5.3.: Examples of activation functions used in neural networks.

Logistic	$f(x) = \frac{1}{1+e^{-x}}$	
Rectified Linear Unit	$f(x) = \max(x, 0)$	
Hyperbolic Tangent	$f(x) = \tanh(x)$	
Binary Step	$f(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$	
Identity	$f(x) = x$	

backpropagation algorithm (Werbos 1974). The backpropagation algorithm is a strategy for automatic differentiation. The core idea of backpropagation is the chain rule from calculus (Brunton and Kutz 2019). Using backpropagation, the gradient of the loss function with respect to the weights and biases in Eq. (5.7) is computed and the loss function can be iteratively minimized by gradient descent. The backpropagation algorithm is described in more detail in Appendix F. Since minimizing Eq. (5.7) with respect to the weights and biases is a high-dimensional nonlinear optimization problem, many iterations are required in order to optimize the weights and biases.

The particular implementation of gradient descent used in this analysis is the *Adam* algorithm described by Kingma and Ba (2017). The Adam algorithm uses moving averages of the gradient and the squared gradient to adjust the step size at each iteration. The algorithm is as follows: a stepsize parameter α , a regularization parameter

ϵ , and exponential decay rate parameters $\beta_1, \beta_2 \in [0, 1)$ are chosen. Kingma and Ba (2017) recommend $\alpha = 10^{-3}, \beta_1 = 0.9, \beta_2 = 1 - 10^{-3}, \epsilon = 10^{-8}$ as good default values for these parameters. An objective function $f(\boldsymbol{\theta})$ with parameters to be optimized $\boldsymbol{\theta}$ is required ($\boldsymbol{\theta}$ would be e.g. the weights and biases of a neural network). The initial parameter vector $\boldsymbol{\theta}_0$ is given. First and second moment vectors $\mathbf{m}_i, \mathbf{v}_i$ are initialized with $\mathbf{m}_0 = \mathbf{0}, \mathbf{v}_0 = \mathbf{0}$. An iteration index t is initialized at $t = 0$. At each iteration, a gradient vector \mathbf{g}_t is computed:

$$\mathbf{g}_t := \nabla_{\boldsymbol{\theta}} f_t(\boldsymbol{\theta}_{t-1}) \quad (5.8)$$

This gradient vector is used to update the biased first moment estimate vector \mathbf{m}_t and the biased second moment estimate \mathbf{v}_t :

$$\mathbf{m}_t := \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t \quad (5.9)$$

$$\mathbf{v}_t := \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) \mathbf{g}_t \odot \mathbf{g}_t \quad (5.10)$$

Finally, the updated parameter vector $\boldsymbol{\theta}_t$ is computed

$$\boldsymbol{\theta}_t := \boldsymbol{\theta}_{t-1} - \alpha \frac{\sqrt{1 - \beta_2^2}}{1 - \beta_1^t} \frac{\hat{\mathbf{m}}_t}{\epsilon + \sqrt{\hat{\mathbf{v}}_t}} \quad (5.11)$$

where $\sqrt{\hat{\mathbf{v}}_t}$ denotes the element-wise square root. This process of successively updating $\boldsymbol{\theta}_t$ is repeated until the error reaches a sufficiently small value. One of the features of the Adam algorithm is that it has a “momentum” because the vectors $\mathbf{m}_t, \mathbf{v}_t$ respond slowly to changes in the gradient \mathbf{g}_t because the parameters β_1, β_2 are usually close to 1.

The particular artificial neural network implementation used in the present paper

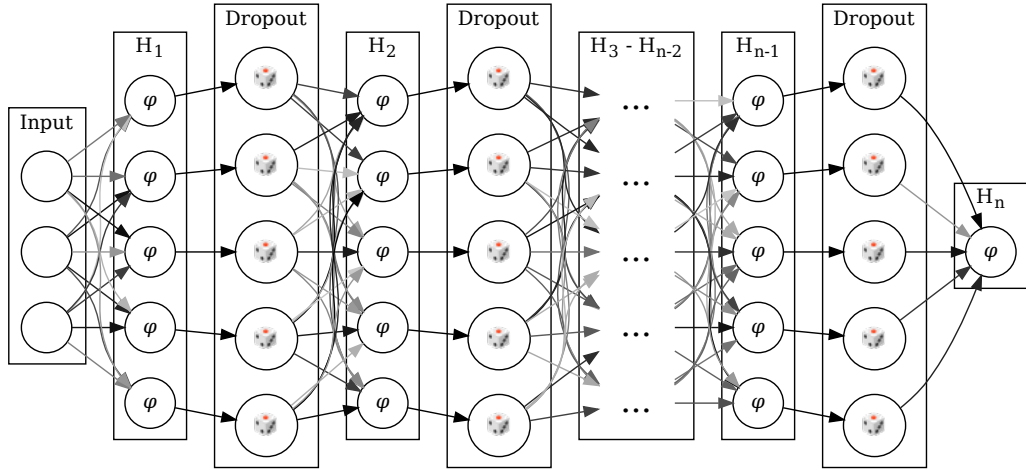


Figure 5.3.: Illustration of artificial neural network consisting of $n - 1$ dense hidden layers with dropout.

is from the TensorFlow software library (TensorFlow Developers 2022). The Adam optimization strategy was used to minimize the mean absolute error. The learning rate α for the Adam optimizer was 0.01, the first moment decay rate β_1 was 0.9, the second moment decay rate β_2 was 0.999, and the regularization term ϵ was 10^{-7} . The Rectified Linear Unit (ReLU) activation function was used for neurons in the hidden layers and the identity activation function was used for the output neuron. An additional *dropout* layer was added after each hidden layer as shown in Fig. 5.3. With dropout, during the training phase, outputs of each neuron are randomly set to zero at a given probability. Dropout is known to reduce over-fitting during training (Brunton and Kutz 2019). Table 5.4 lists the sets of hyperparameters used during the grid search optimization step described in Algorithm 3.

Table 5.4.: Parameter tuning values for artificial neural network.

Number of Neurons	Number of Layers	Dropout Ratio
32	1	0
64	2	0.1
128	3	0.2
256	4	0.4
512		

5.4.6. AdaBoost

AdaBoost is a family of algorithms developed by Freund and Schapire (1997) which use an ensemble of weak learners trained on weighted data where the weights are adjusted based upon errors of previous weak learners in the ensemble. The particular algorithm used herein is from the Scikit Learn library (Pedregosa et al. 2011) which implements a particular AdaBoost algorithm developed by Drucker (1997) called AdaBoost.R2.

In the AdaBoost.R2 algorithm, training is as follows: a vector of sample weights \mathbf{w} is initialized with values $w_{1\dots N} = 1$. For each weak learner t in the ensemble of T weak learners, do the following: compute a vector \mathbf{p} such that $p_i = \frac{w_i}{\sum w_i}$; randomly re-sample the labeled examples into the arrays $(\mathbf{y}^{(t)}, \mathbf{X}^{(t)})$ such that the probability of the i th labeled example being included in the set is p_i ; train the learner t on the re-sampled set $(\mathbf{y}^{(t)}, \mathbf{X}^{(t)})$; for each labeled example (y_i, \mathbf{x}_i) in the original set, compute prediction $\hat{y}_i(\mathbf{x}_i)$ and loss $L_i = L(|\hat{y}_i - y_i|)$; compute weighted average loss $\bar{L} = \sum_{i=1}^N L_i p_i$; compute confidence measure $\beta = \frac{\bar{L}}{1-\bar{L}}$; update the weights $w_i \rightarrow w_i \beta^{(1-L_i)}$; and finally, repeat this process with the newly-computed weights on the next weak learner t in the ensemble. For prediction, a weighted median scheme is used as follows: for each

Table 5.5.: Parameter tuning values for AdaBoost with Neural Network

Number of Estimators	Number of Neurons	Number of Layers	Dropout Ratio
1	32	1	0
2	64	2	0.1
5	128	3	0.2
	256		0.4

trained learner t , compute prediction $h_t(\mathbf{x}_i)$, order these predictions from smallest to largest, then find the first t such that $\sum_{i=1}^t \log(\frac{1}{\beta_i}) \geq \frac{1}{2} \sum_{i=1}^T \log(\frac{1}{\beta_i})$ and use that t as the predictor. The AdaBoost.R2 algorithm thus weighs more “difficult” samples more heavily in the training process, and weighs learners with lower loss scores more heavily in the prediction process.

The AdaBoost model used herein used neural networks with the same architecture discussed in previous sections as the weak learners. The parameters used for the grid search process for the AdaBoost models are listed in Table 5.5.

6. Results

Aggregate performance of the models computed using nested cross-validation is listed in Fig. 6.1. Each subplot represents the distribution of % error for a different model. % error is defined as

$$\% \text{ Error} \equiv 100 \frac{\hat{t} - t}{t} \quad (6.1)$$

where t is the true fatigue life and \hat{t} is the predicted fatigue life for a given example. The labels in each subplot list the mean % error, the standard deviation σ , and the 95% confidence interval of % error. For instance, the label for the neural network model indicates that the mean error of cycles to failure is +3.0%, that the standard deviation of the error was 43.9%, that 2.5% of the errors were greater than +144%, and 2.5% of the errors were less than -63.3%).

For each model, an optimal hyperparameter combination was identified for each outer fold in the nested K-Fold algorithm. Table 6.1 lists the hyperparameter combinations used for the outer folds for the artificial neural network model. Table 6.2 lists the hyperparameter combinations used for the outer folds for the AdaBoost model. Comparing Table 6.1 to Table 6.2, dropout was not used as much in the AdaBoost model, which may be a result of the AdaBoost model using adaptive bootstrapping to reduce overfitting in the estimators. It is notable that in the fourth fold, the AdaBoost model only used a single estimator, which is equivalent to the basic neural network model. However, the number of layers and number of neurons for the fourth fold are different between the AdaBoost model and basic neural network model. This

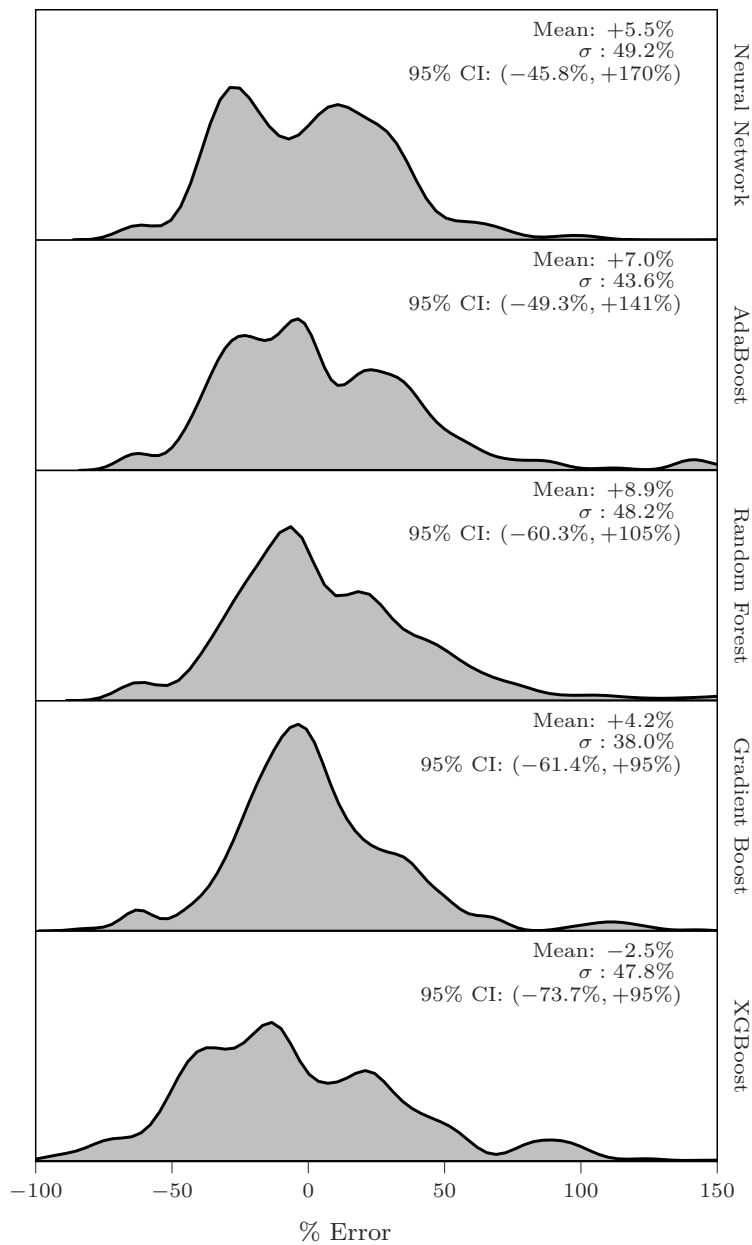


Figure 6.1.: Distribution of % errors of fatigue life. σ is the standard deviation of % error. “95% CI” is the 95% confidence interval for % error.

Table 6.1.: Hyperparameter combinations used for each fold for the neural network model.

Dropout Ratio	Number of Layers	Number of Neurons
0.1	1	256
0.2	1	128
0.1	1	128
0	2	64
0	2	256

Table 6.2.: Hyperparameter combinations used for each fold for the AdaBoost model

Dropout Ratio	Number of Layers	Number of Neurons	Number of Estimators
0	1	256	2
0.1	1	128	2
0	1	128	2
0	2	32	1
0	3	256	5

difference may be a result of the random process used to initialize the weights in the neural networks. The neural-network-based models all used 3 or fewer layers, which may change with additional training epochs. In the decision-tree-based models (Tables 6.3 to 6.5), the maximum tree depth varied across an order of magnitude from 3 to 31. In the boosting models (Tables 6.4 and 6.5), subsample ratio varied across an order of magnitude from 0.1 to 1. In the boosting models, the learning rate varied from 0.0032 to 0.01. The hyperparameter values used in the present work vary significantly depending on how the data is split. Future works using different data sets will likely need to search for optimal hyperparameters for their datasets.

To determine how difficult the results of each experiment were to predict, scores for each experiment in the dataset were computed using nested k-fold cross-validation (Algorithm 4) with hold-one-out splitting ($K =$ the total number of experiments) for

Table 6.3.: Hyperparameter combinations used for each fold for the random forest model

Maximum Depth	Maximum Split Features	Minimum Samples for a Leaf	Minimum Samples to Split a Node
5	0.01	1	4
31	1	1	2
5	1	3	2
13	0.01	1	2
31	0.01	3	10

Table 6.4.: Hyperparameter combinations used for each fold for the gradient boosting model

Learning Rate	Maximum Depth	Subsample Ratio
0.0032	3	0.1
0.0032	17	0.32
0.0032	5	1
0.01	5	1
0.01	3	0.1

Table 6.5.: Hyperparameter combinations used for each fold for the extreme gradient boosting model

Column Sample by Tree	Learning Rate	Maximum Depth	Subsample Ratio
1	0.01	3	0.1
1	0.01	3	0.56
1	0.01	3	0.1
1	0.01	17	1
1	0.01	5	1

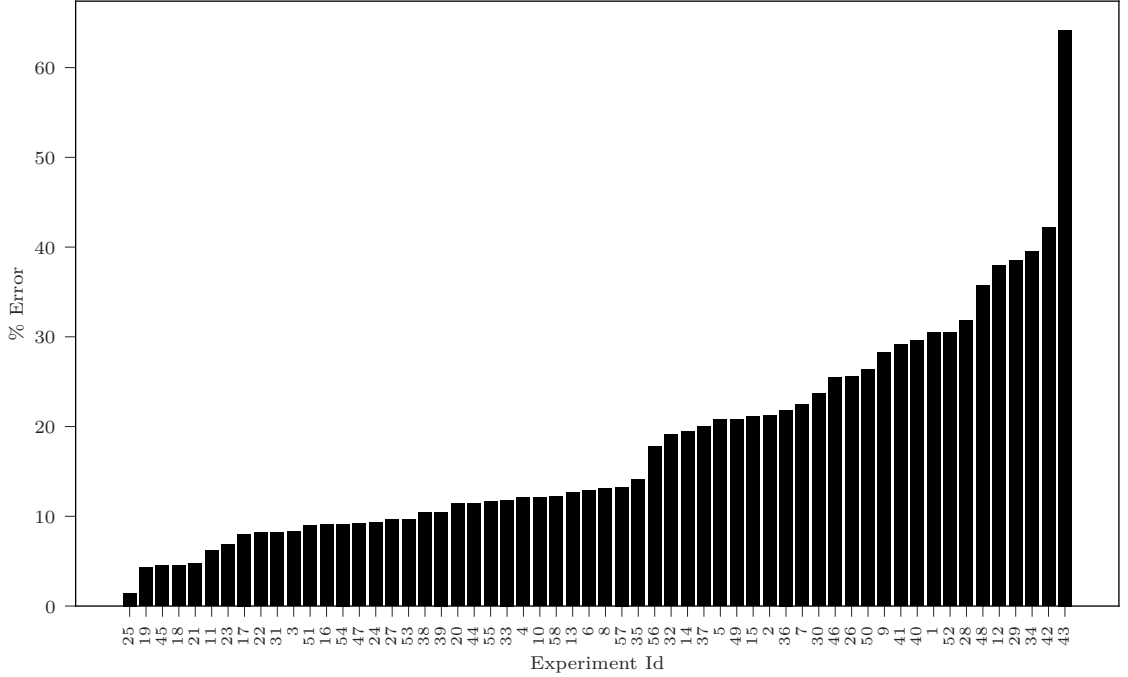


Figure 6.2.: Average % error computed for each experiment. “Experiment Id” corresponds with the “Id” column in Table D.1.

the outer split and $k = 4$ for the inner split. Thus, during ranking, each experiment in the set was “held out” while the remaining experiments were used to train the model with a grid search optimization strategy on the hyperparameters. The scoring function used for this ranking process was the mean absolute error of the logarithm of fatigue life, defined as

$$\text{MAE}_{\log} \equiv \frac{\sum_{i=1}^N |\log \hat{t}_i - \log t_i|}{N} \tag{6.2}$$

where t_i, \hat{t}_i are the true and predicted fatigue lives of the i th data point. The gradient boosting model was used to score each model because it showed good performance and was the fastest to train. Scores for each experiment were sorted and are listed in Fig. 6.2.

Table 6.6.: % MAE for different models for the least difficult to predict, median difficulty, and most difficult to predict experiments.

Model	Least	Median	Most
AdaBoost	3.49	11.7	63.1
Gradient Boost	4.99	10.9	61.8
Neural Network	3.12	10.9	64.1
Random Forest	3.3	8.64	62.1
XGBoost	3.91	15.6	63.2

Once scores for each experiment had been computed, the most difficult, median, and least difficult experiments were selected. Then, the different models used in the analysis were used to predict the fatigue life for each of the most difficult, median, and least difficult experiments. To generate these predictions, each of the experiments to predict was held out from the data set, the remainder of the experiments were used to train each model using a grid search (Algorithm 3), and finally, the trained model was used to predict fatigue lives for the held-out experiment. These results are plotted in Figs. 6.3 to 6.17. The scores for the different models are listed in Table 6.6.

Comparing the models based on decision trees (e.g. Figs. 6.3 to 6.5) to the models based on neural networks (e.g. Figs. 6.6 and 6.7), the models based on neural networks produce considerably smoother fatigue life plots. However, despite the improved appearance, the errors produced by the neural network-based models are not lower than those produced by decision-tree-based models.

Comparing performance on the least difficult experiment to model (Figs. 6.3 to 6.7) the median experiment (Figs. 6.8 to 6.12) and the most difficult experiment to model (Figs. 6.13 to 6.17) it is apparent that the difference in error between models for a given experiment is much less than the difference in error between experiments for a given model. Thus, data quality and size may be a more important consideration than model selection when predicting solder joint fatigue life using machine learning.

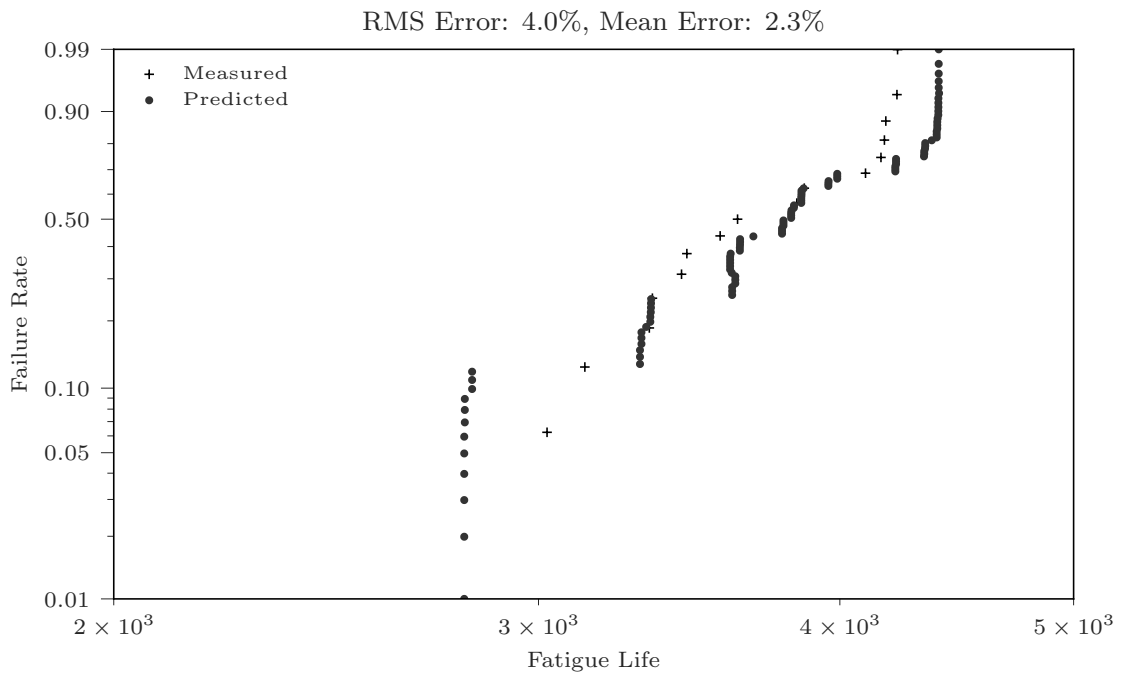


Figure 6.3.: Random forest regression on least difficult experiment to model (Ricky Lee et al. 2002).

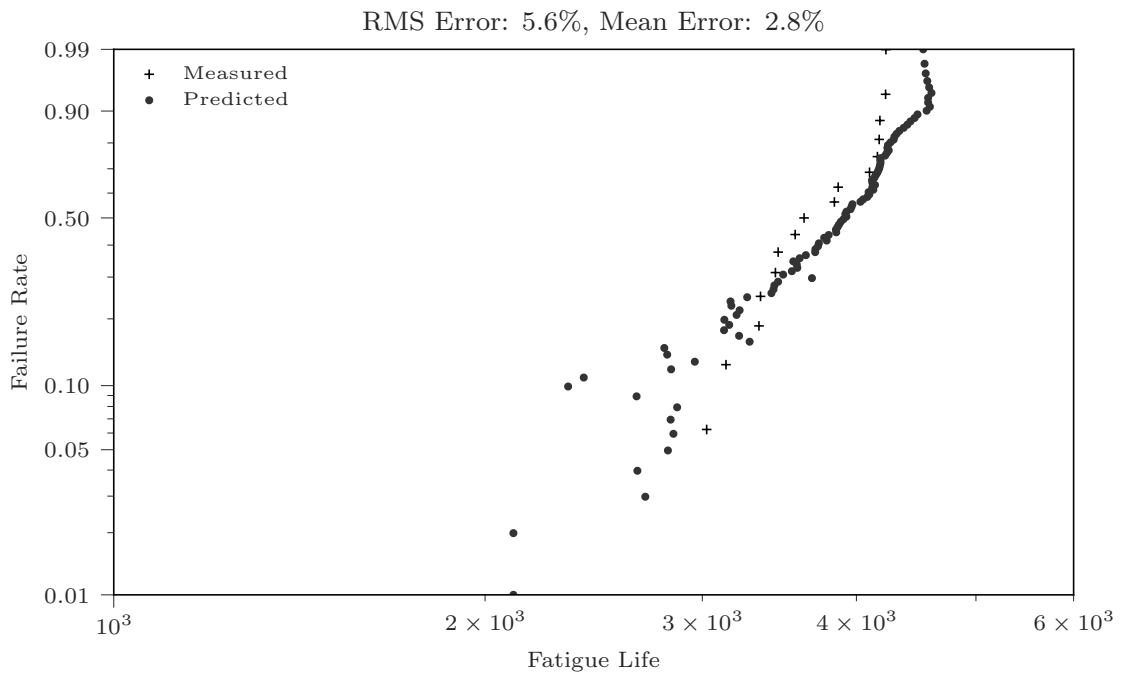


Figure 6.4.: Gradient boosting regression on least difficult experiment to model (Ricky Lee et al. 2002).

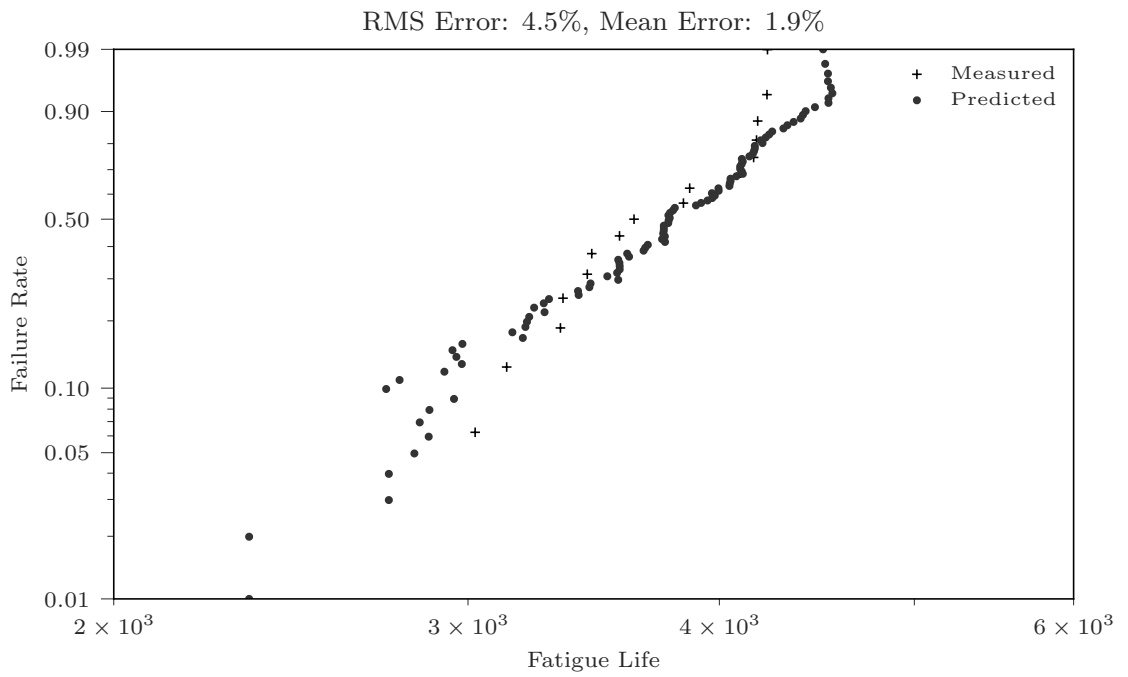


Figure 6.5.: Extreme gradient boosting regression on least difficult experiment to model (Ricky Lee et al. 2002).

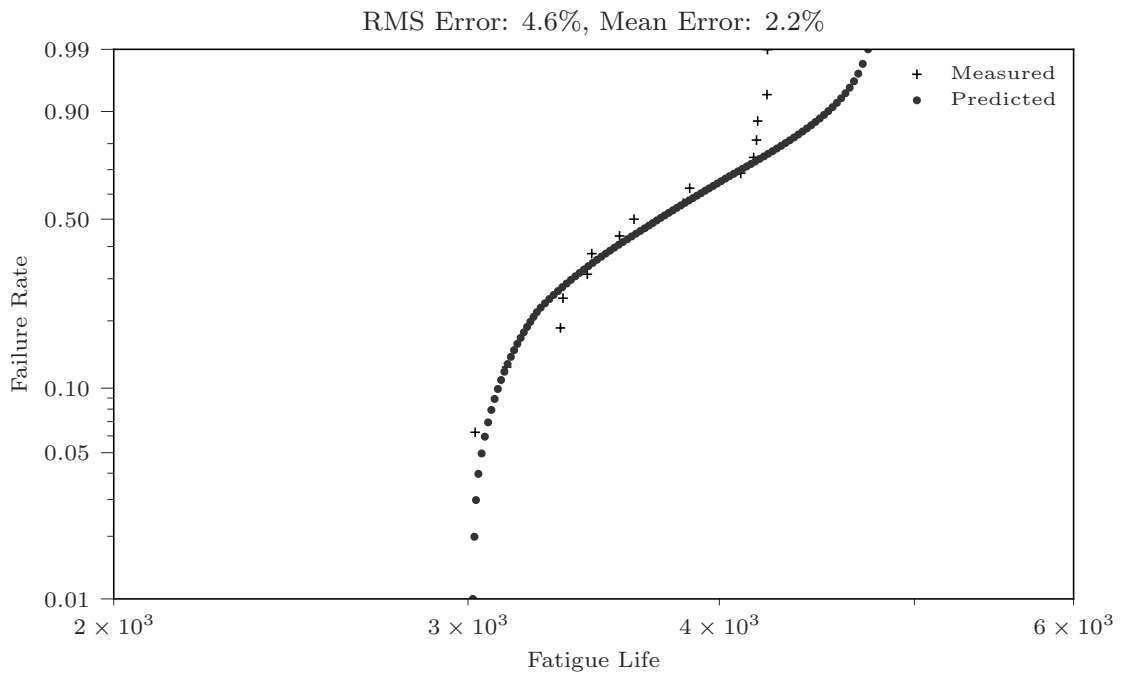


Figure 6.6.: Artificial neural network regression on least difficult experiment to model (Ricky Lee et al. 2002).

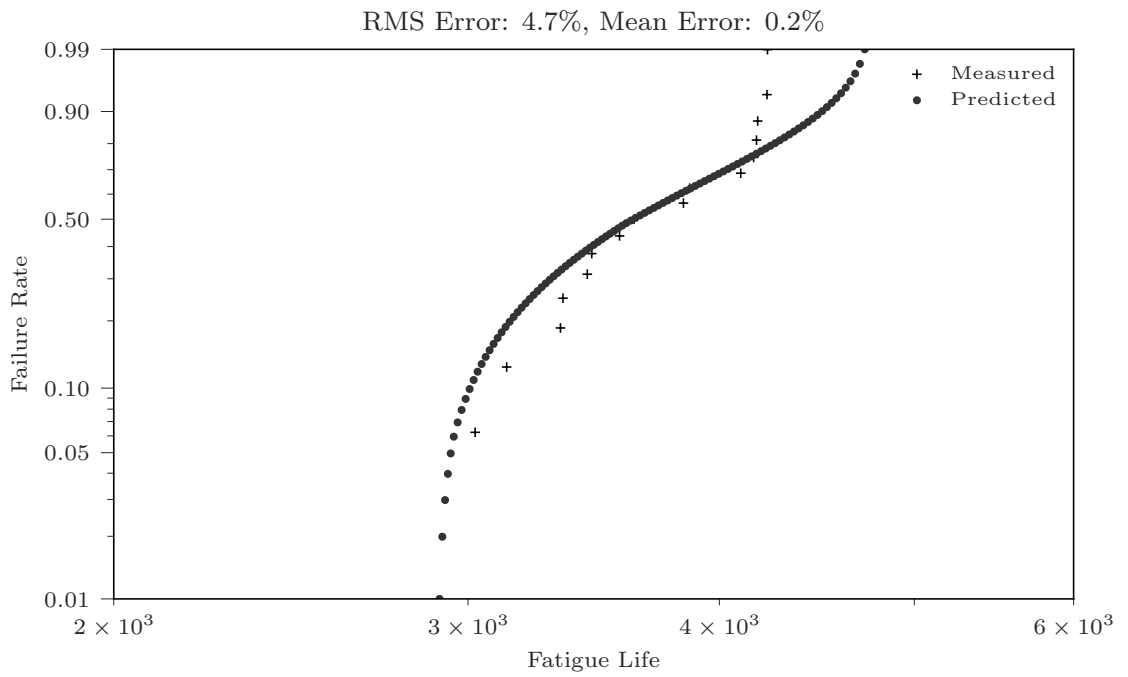


Figure 6.7.: AdaBoost regression on least difficult experiment to model (Ricky Lee et al. 2002).

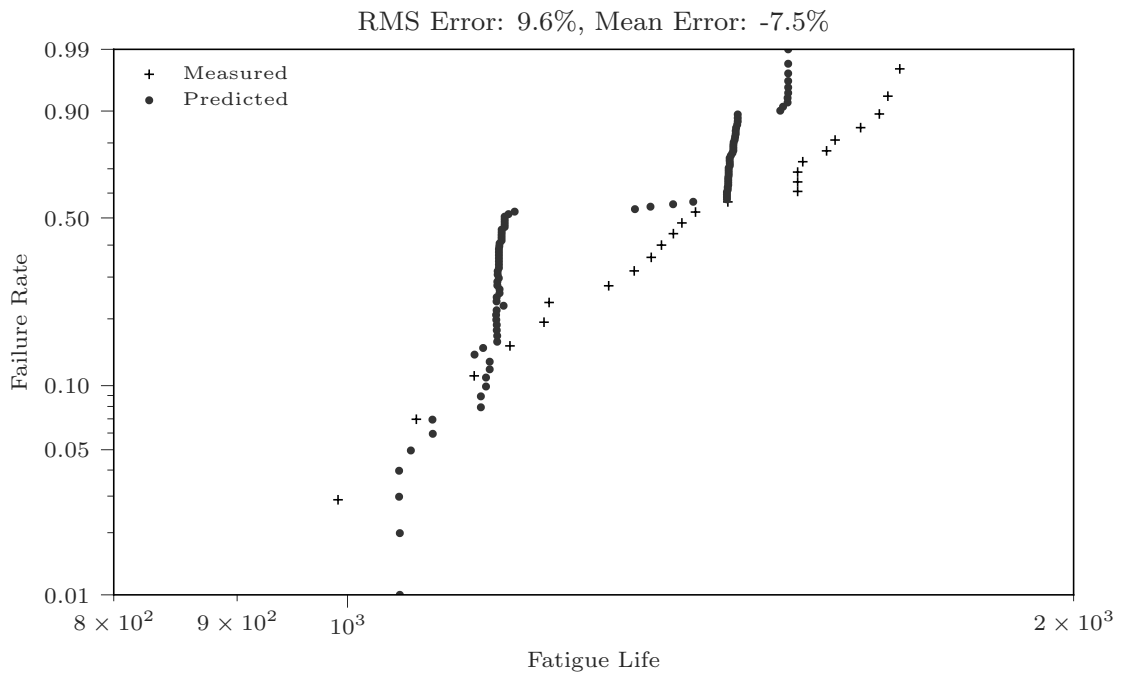


Figure 6.8.: Random forest regression on median modeling difficulty experiment (T.-K. Lee et al. 2010).

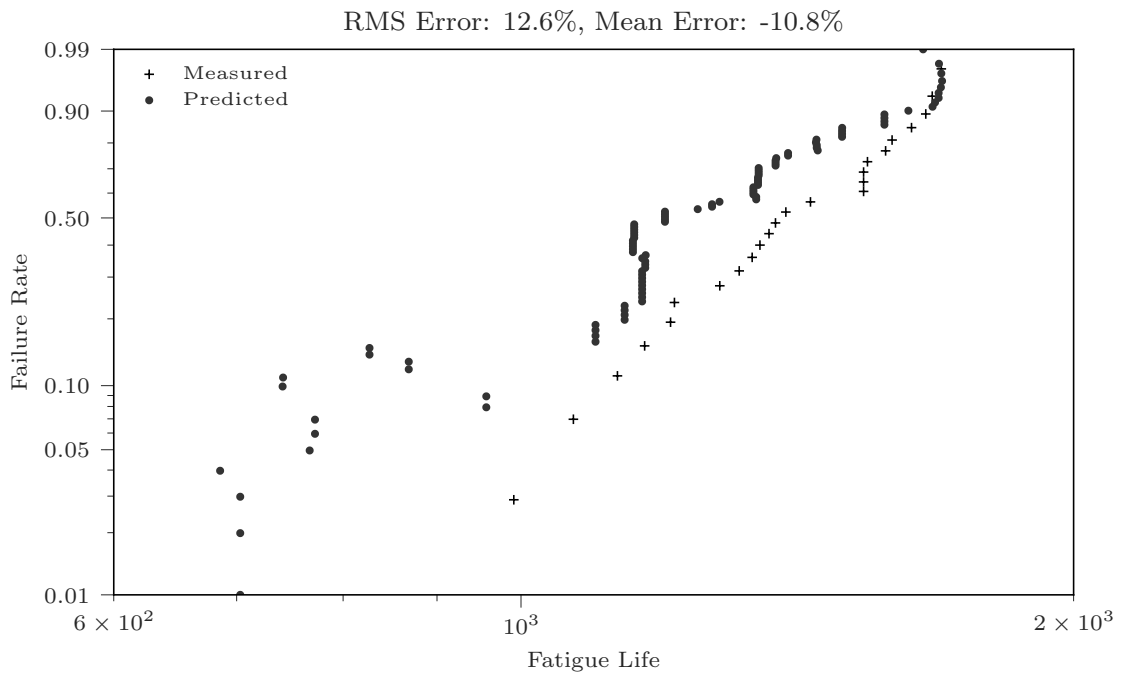


Figure 6.9.: Gradient boosting regression on median modeling difficulty experiment (T.-K. Lee et al. 2010).

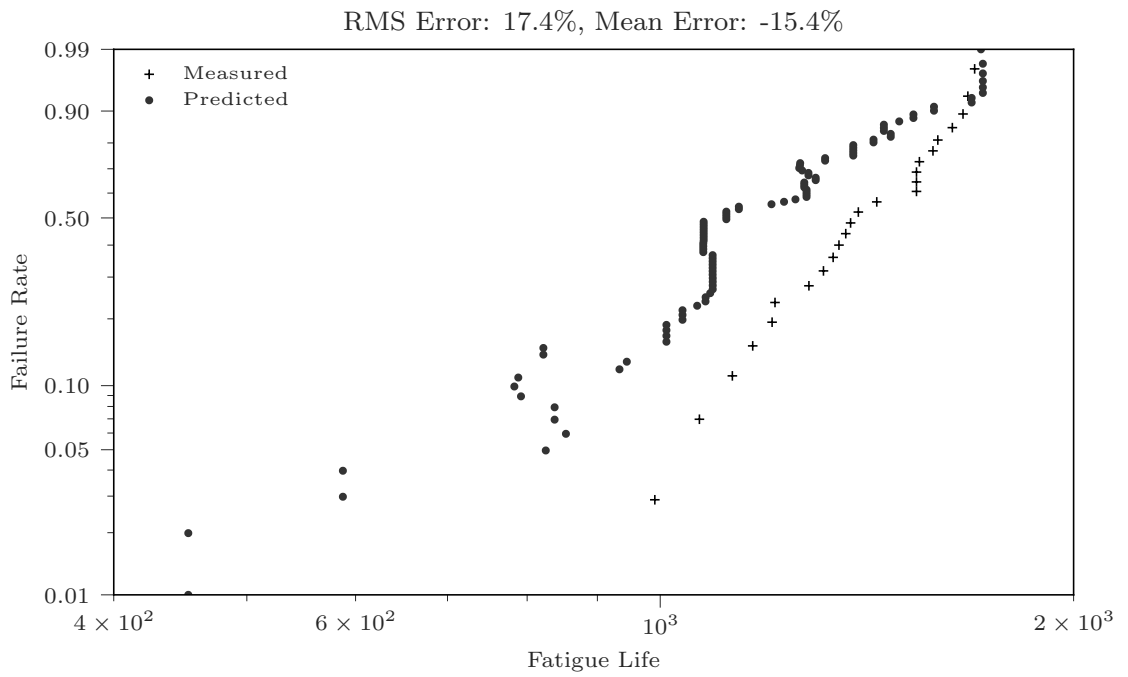


Figure 6.10.: Extreme gradient boosting regression on median modeling difficulty experiment (T.-K. Lee et al. 2010).

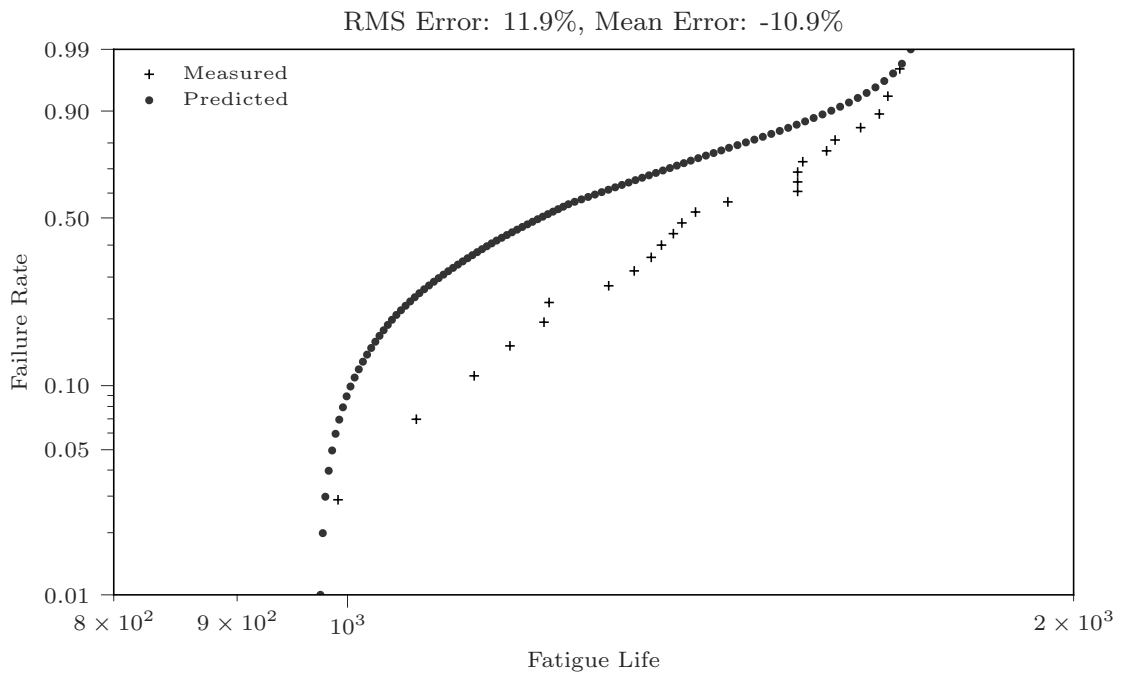


Figure 6.11.: Artificial neural network regression on median modeling difficulty experiment (T.-K. Lee et al. 2010).

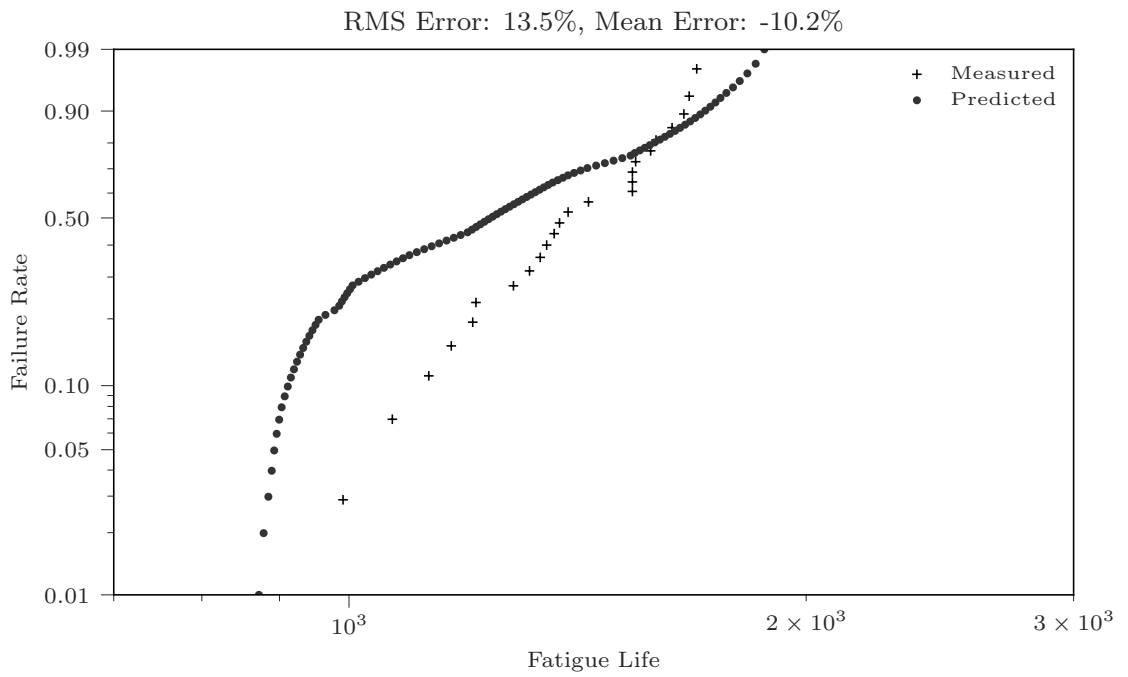


Figure 6.12.: AdaBoost regression on median modeling difficulty experiment (T.-K. Lee et al. 2010).

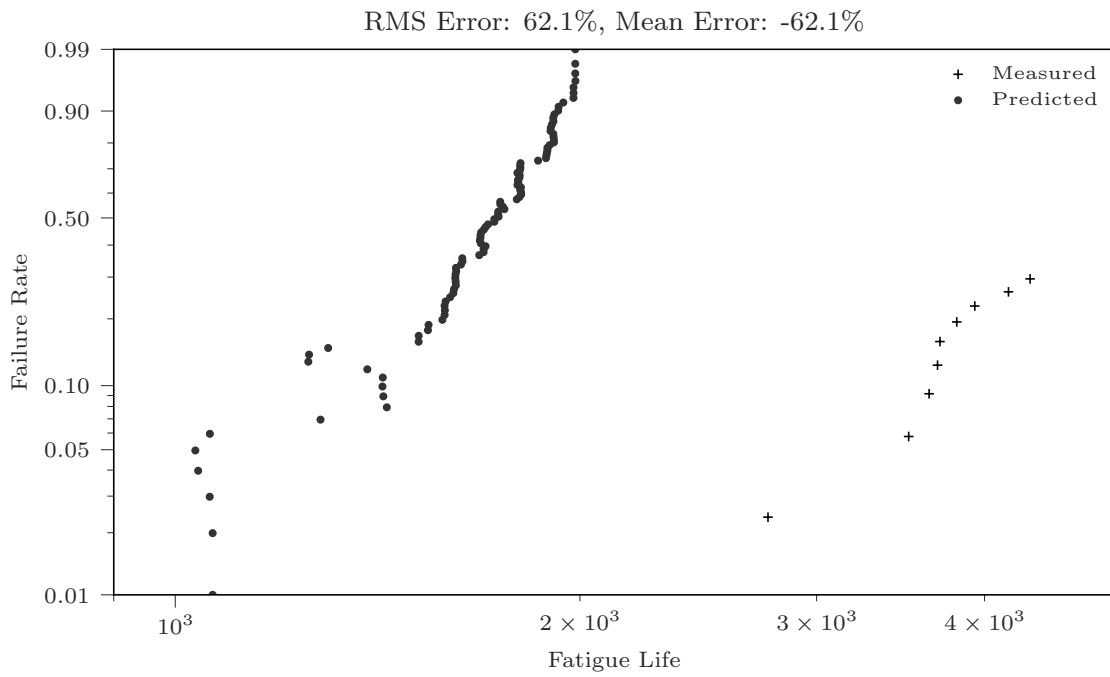


Figure 6.13.: Random forest regression on most difficult experiment to model (Syed et al. 2008).

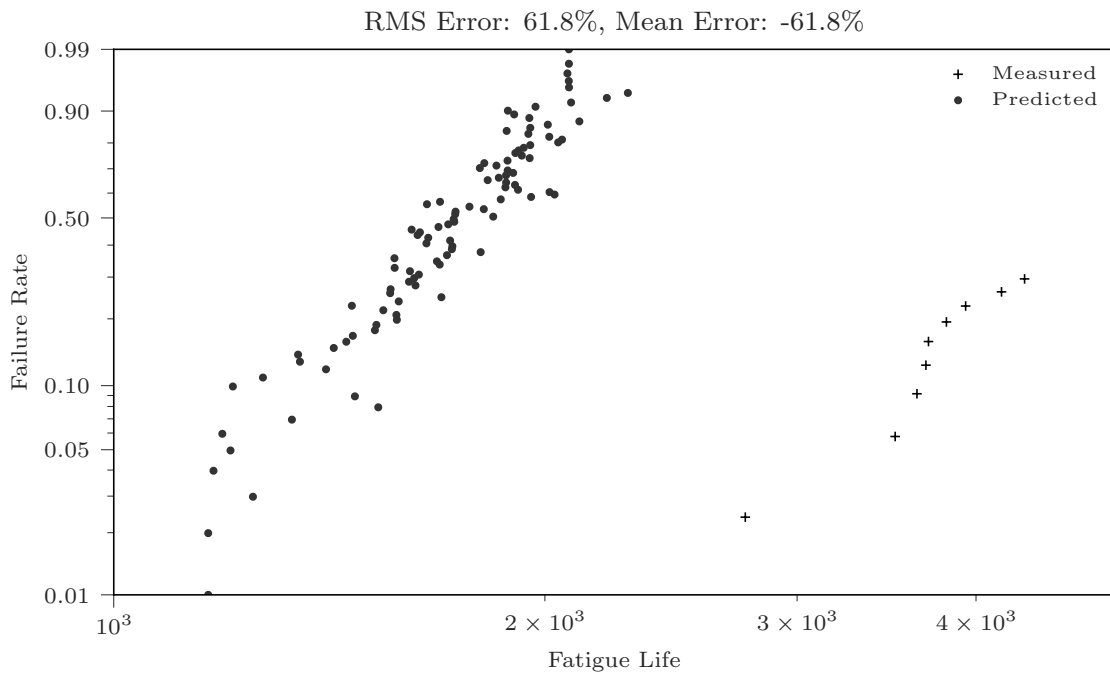


Figure 6.14.: Gradient boosting regression on most difficult experiment to model (Syed et al. 2008).

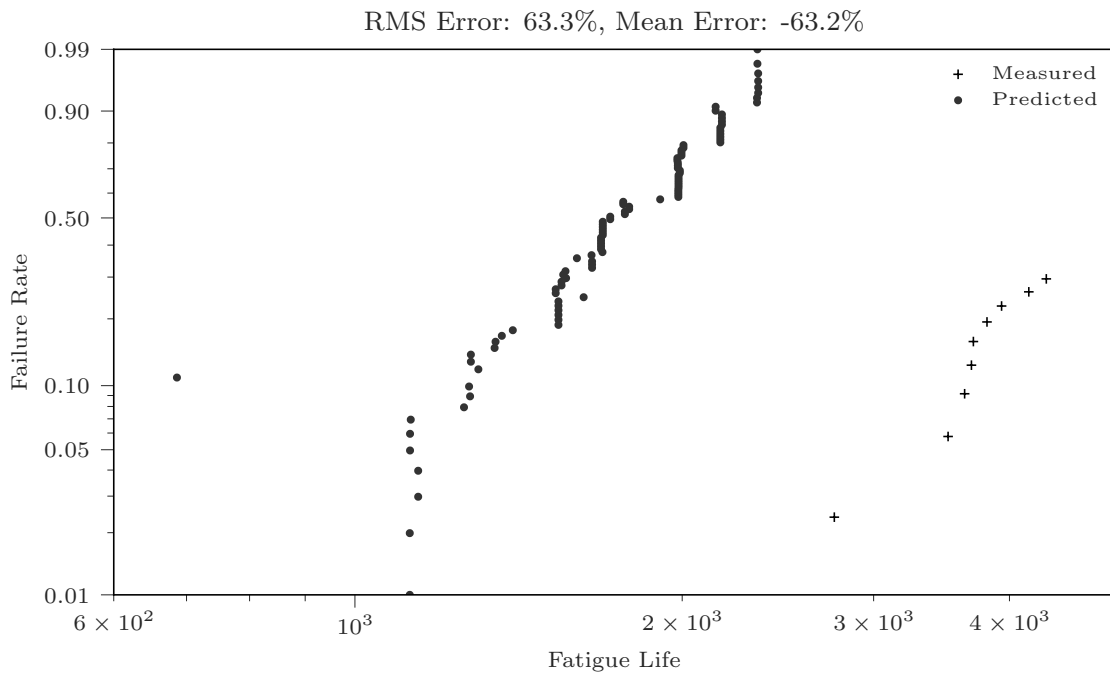


Figure 6.15.: Extreme gradient boosting regression on most difficult experiment to model (Syed et al. 2008).

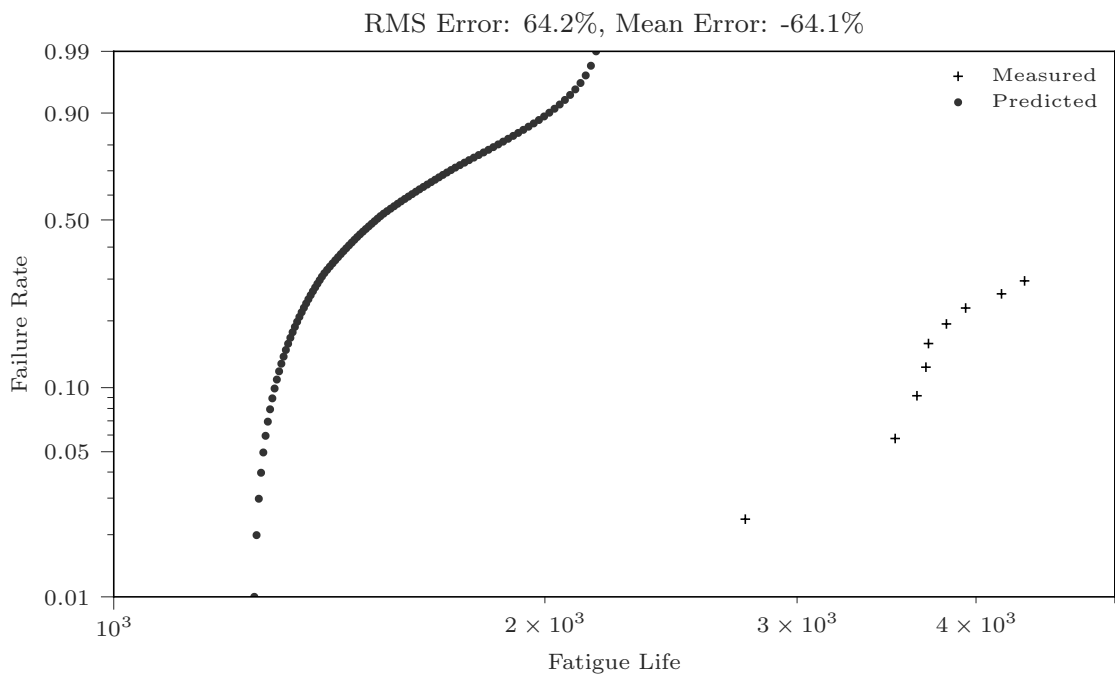


Figure 6.16.: Artificial neural network regression on most difficult experiment to model (Syed et al. 2008).

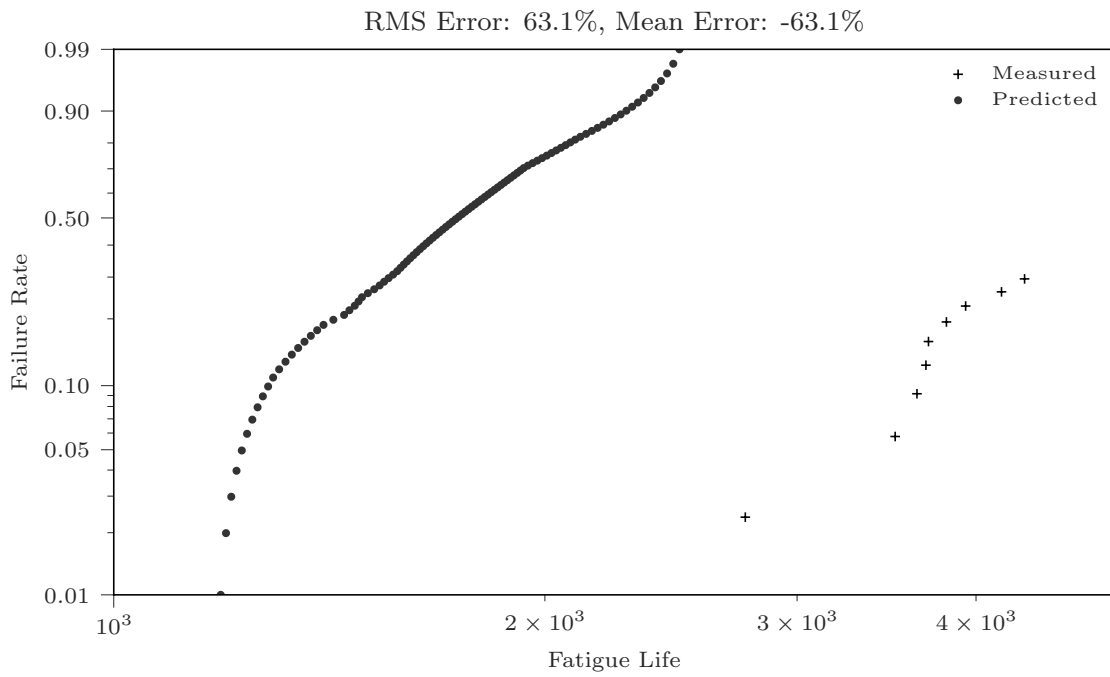


Figure 6.17.: AdaBoost regression on most difficult experiment to model (Syed et al. 2008).

worst-scoring experiment

The most and least difficult experiments to model were both similar to other experiments in the data set, with the exception of additional experimental parameters not included as features in the model. The least difficult experiment to model from Ricky Lee et al. (2002) was part of an ensemble of experiments in which different solder pastes were compared for the same package, solder ball, and test conditions. This paper found that the different solder pastes did not have a very large effect on the fatigue life. The models performed well on this ensemble because any one of the experiments in the ensemble could be well approximated by the others, and because the models did not use solder paste as a feature to “know” that there might any difference between those experiments anyway. The most difficult experiment to model is from Syed et al. (2008). In this paper, different design parameters were studied, including mold compound material, mold cap thickness, package pad surface treatment, solder ball composition, solder volume, package material, and board material were studied. The most difficult experiment to model with identifier 43 was very similar to another experiment with identifier 42 with the exception that experiment 42 used an Ni/Au package pad surface treatment and experiment 43 used an OSP/Cu package pad surface treatment. Package pad surface treatment was not used as a feature in the present analysis. Experiment 43 had approximately 1.5 times the fatigue life of experiment 42. Since the features included in the models were identical for experiment 42 and 43, there was no way for the models to distinguish them. In addition, not all of the packages studied in Syed et al. (2008) are provided with a complete physical specification. The parameters for the package used in experiment 43 have been assumed to be similar to another experiment from that same paper not included in the present analysis, but with a 0.5mm pitch instead of a 0.4mm pitch.

However, since the actual physical specification for the package used in experiment 43 was not provided, it's possible that other features were different as well.

The “best” model for a given application depends on the desired outcome. The gradient boosting model had the smallest standard deviation and 95% confidence interval. Comparing the basic neural network model to the neural network with AdaBoost, the AdaBoost model did not demonstrate substantially better performance than the basic neural network model. Comparing the neural-network-based models to the decision-tree-based models, the decision-tree-based models were more conservative and had smaller confidence intervals. Gradient boosting had the lowest standard deviation of error and the smallest confidence interval of all models considered. However, as the amount and quality of training data increases, the relative performance of the models may change. In particular, since decision-tree-based models are not able to extrapolate outside the range of training data, they may be outperformed by other models which do not have the same limitation, such as artificial neural networks.

Training time is another important consideration for model selection. The training time for the regression-tree-based models was an order of magnitude faster than the training time for the neural-network-based models. This is because neural networks are trained using an iterative algorithm which converges gradually, while decision trees and the ensemble methods considered use greedy algorithms which don't require iterations to converge.

Due to limitations in the available training data, many important features such as thermal ramp rate, solder joint height, and package pad surface treatment were excluded from consideration. By increasing the size of the training set, future works may be able to include those features in their analysis as more data is included in the Electronics Packaging Materials Database.

All of the models used for predicting fatigue life seemed to struggle at very high or very low failure rates. In the case of the models based on regression trees, the base learners are fundamentally unable to extrapolate to failure rates outside the range of observations. Failure rate data is more sparse for failure rates very close to 0 or 1 because those rates can only be accessed by experiments with very large numbers of samples. Unlike regression trees, neural networks do not share the same fundamental limitation of being unable to extrapolate outside the range of labels, however, they also struggled to predict fatigue life at high and low failure rates because the data was sparse in these ranges.

The prediction errors found in the current work are larger than the errors found in previous works (Yuan and C.-C. Lee 2020; Samavatian et al. 2020; T.-C. Chen, Alazawi, et al. 2021; T.-C. Chen, Opuencia, et al. 2022; Chou, Chiang, and Liang 2019). Other works used finite element analysis and various fatigue life models to generate training data, rather than collecting data from physical tests. By synthesizing data in this fashion, many physical variables are completely excluded from the analysis such as the size effects of intermetallic compound formation. Thus, the accuracy of the models in the present work is lower than in previous works because the data is much less uniform and covers a much broader set of parameters.

7. Conclusions

In this work a number of different machine learning strategies were compared for the purpose of predicting the fatigue life of board-level solder joints. The prediction of solder joint fatigue life is challenging because it involves the nonlinear coupling of many phenomena including creep, fatigue, and chemistry. The ability to predict fatigue performance of board-level solder joints can have economic benefits by preventing early failures and by enabling designers to find optimal parameters more easily. This work demonstrates that the performance of traditional learning models based on regression trees may compare favorably to the performance of artificial neural networks in the task of predicting the fatigue life of board-level solder joints when using training data collected from a variety of physical experiments.

While other works using machine learning to predict the fatigue life of solder joints have taken a bottom-up approach to predict the fatigue life from various mathematical models, the work presented in this paper takes a top-down approach and uses experimental results to predict the fatigue life. Both approaches have different benefits and drawbacks, with the bottom-up approach providing very fine-grained data about the theoretical performance of a particular design and the top-down approach revealing broad trends across a large design space. The availability of large open data sets containing solder joint reliability data over a large range of parameters will be important for the development of machine learning frameworks for assisting in the design of board-level solder joints in the future.

8. References

- [1] C. Andersson et al. “Thermal Cycling of Lead-free Sn-3.8Ag-0.7Cu 388PBGA Packages”. In: *Soldering & Surface Mount Technology* 21.2 (Apr. 2009). Ed. by Johan Liu, pp. 28–38. ISSN: 0954-0911. DOI: 10.1108/09540910910947453.
- [2] Babak Arfaei, Sam Mahin-Shirazi, et al. “Reliability and Failure Mechanism of Solder Joints in Thermal Cycling Tests”. In: *2013 IEEE 63rd Electronic Components and Technology Conference*. Las Vegas, NV, USA: IEEE, May 2013, pp. 976–985. ISBN: 978-1-4799-0232-3 978-1-4799-0233-0. DOI: 10.1109/ECTC.2013.6575693.
- [3] Babak Arfaei, L. Wentlent, et al. “Improving the Thermomechanical Behavior of Lead Free Solder Joints by Controlling the Microstructure”. In: *13th InterSociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems*. San Diego, CA, USA: IEEE, May 2012, pp. 392–398. ISBN: 978-1-4244-9532-0 978-1-4244-9533-7 978-1-4244-9531-3. DOI: 10.1109/ITHERM.2012.6231456.
- [4] Periannan Arulvanan, Zhaowei Zhong, and Xunqing Shi. “Effects of Process Conditions on Reliability, Microstructure Evolution and Failure Modes of SnAgCu Solder Joints”. In: *Microelectronics Reliability* 46.2-4 (Feb. 2006), pp. 432–439. ISSN: 00262714. DOI: 10.1016/j.microrel.2005.05.005.
- [5] M. Berthou et al. “Microstructure Evolution Observation for SAC Solder Joint: Comparison between Thermal Cycling and Thermal Storage”. In: *Microelectronics Reliability*. 20th European Symposium on the Reliability of Electron

- Devices, Failure Physics and Analysis 49.9 (Sept. 2009), pp. 1267–1272. ISSN: 0026-2714. DOI: 10.1016/j.microrel.2009.07.040.
- [6] P. Borgesen et al. “Solder Joint Reliability under Realistic Service Conditions”. In: *Microelectronics Reliability* 53.9-11 (Sept. 2013), pp. 1587–1591. ISSN: 00262714. DOI: 10.1016/j.microrel.2013.07.091.
- [7] Leo Breiman. “Random Forests”. In: *Machine Learning* 45.1 (2001), pp. 5–32. ISSN: 08856125. DOI: 10.1023/A:1010933404324.
- [8] Steven L Brunton and J Nathan Kutz. *Data-Driven Science and Engineering*. Cambridge University Press, 2019. ISBN: 978-1-108-42209-3.
- [9] F.X. Che and J.H.L. Pang. “Thermal Fatigue Reliability Analysis for PBGA with Sn-3.8Ag-0.7Cu Solder Joints”. In: *Proceedings of 6th Electronics Packaging Technology Conference (EPTC 2004) (IEEE Cat. No.04EX971)*. Singapore: IEEE, 2004, pp. 787–792. ISBN: 978-0-7803-8821-5. DOI: 10.1109/EPTC.2004.1396715.
- [10] Gang Chen et al. “A New Unified Constitutive Model for SAC305 Solder under Thermo-Mechanical Loading”. In: *Mechanics of Materials* 138 (Nov. 2019), p. 103170. ISSN: 01676636. DOI: 10.1016/j.mechmat.2019.103170.
- [11] Tianqi Chen and Carlos Guestrin. “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. San Francisco California USA: ACM, Aug. 2016, pp. 785–794. ISBN: 978-1-4503-4232-2. DOI: 10.1145/2939672.2939785.

- [12] Tzu-Chia Chen, Fouad Jameel Ibrahim Alazzawi, et al. “Application of Machine Learning in Rapid Analysis of Solder Joint Geometry and Type on Thermomechanical Useful Lifetime of Electronic Components”. In: *Mechanics of Advanced Materials and Structures* (Dec. 2021), pp. 1–9. ISSN: 1537-6494, 1537-6532. DOI: 10.1080/15376494.2021.2014002.
- [13] Tzu-Chia Chen, Maria Jade Catalan Opulencia, et al. “Estimation of Thermo-mechanical Fatigue Lifetime of Ball Grid Solder Joints in Electronic Devices Using a Machine Learning Approach”. In: *Journal of Electronic Materials* 51.7 (July 2022), pp. 3495–3503. ISSN: 0361-5235, 1543-186X. DOI: 10.1007/s11664-022-09635-2.
- [14] Yunxia Chen, Yi Jin, and Rui Kang. “Coupling Damage and Reliability Modeling for Creep and Fatigue of Solder Joint”. In: *Microelectronics Reliability* 75 (Aug. 2017), pp. 233–238. ISSN: 00262714. DOI: 10.1016/j.microrel.2017.03.016.
- [15] Shunfeng Cheng, Chien-Ming Huang, and Michael Pecht. “A Review of Lead-Free Solders for Electronics Applications”. In: *Microelectronics Reliability* 75 (Aug. 2017), pp. 77–95. ISSN: 00262714. DOI: 10.1016/j.microrel.2017.06.016.
- [16] P. H. Chou, K.N. Chiang, and Steven Y. Liang. “Reliability Assessment of Wafer Level Package Using Artificial Neural Network Regression Model”. In: *Journal of Mechanics* 35.6 (Dec. 2019), pp. 829–837. ISSN: 1727-7191, 1811-8216. DOI: 10.1017/jmech.2019.20.
- [17] E F Codd. “A Relational Model of Data for Large Shared Data Banks”. In: *Communications of the ACM* (1970), p. 11.

- [18] Maurice N. Collins, Jeff Punch, and Richard Coyle. “Surface Finish Effect on Reliability of SAC 305 Soldered Chip Resistors”. In: *Soldering & Surface Mount Technology* 24.4 (Sept. 2012), pp. 240–248. ISSN: 0954-0911.
DOI: 10.1108/09540911211262520.
- [19] Jia Deng et al. “ImageNet: A Large-Scale Hierarchical Image Database”. In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. June 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.
- [20] Django Software Foundation. *Django*. Django Software Foundation. Lawrence, Kansas, 2022.
- [21] Harris Drucker. “Improving Regressors Using Boosting Techniques”. In: *Proceedings of the 14th International Conference on Machine Learning*. 1997, pp. 107–115.
- [22] Encode OSS Ltd. *Django Rest Framework*. Encode OSS Ltd. 2022.
- [23] Roy Thomas Fielding. “Architectural Styles and the Design of Network-based Software Architectures”. Doctor of Philosophy in Information and Computer Science. Irvine: University of California, 2000.
- [24] Yoav Freund and Robert E Schapire. “A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting”. In: *Journal of Computer and System Sciences* 55.1 (Aug. 1997), pp. 119–139. ISSN: 00220000.
DOI: 10.1006/jcss.1997.1504.
- [25] Jerome H. Friedman. “Greedy Function Approximation: A Gradient Boosting Machine”. In: *The Annals of Statistics* 29.5 (2001), pp. 1189–1232. ISSN: 0090-5364.

- [26] Frank W. Gayle et al. “High Temperature Lead-Free Solder for Microelectronics”. In: *JOM* 53.6 (June 2001), pp. 17–21. ISSN: 1047-4838, 1543-1851. DOI: 10.1007/s11837-001-0097-5.
- [27] Mohammad Gharaibeh et al. “Experimental and Numerical Investigations of the Vibration Reliability of BGA and LGA Solder Configurations and SAC105 and 63Sn37Pb Solder Alloys”. In: *Soldering & Surface Mount Technology* 31.2 (Apr. 2019), pp. 77–84. ISSN: 0954-0911. DOI: 10.1108/SSMT-07-2018-0020.
- [28] Dominik Herkommer, Jeff Punch, and Michael Reid. “A Reliability Model for SAC Solder Covering Isothermal Mechanical Cycling and Thermal Cycling Conditions”. In: *Microelectronics Reliability* 50.1 (Jan. 2010), pp. 116–126. ISSN: 00262714. DOI: 10.1016/j.microrel.2009.08.008.
- [29] Chong Hua Zhong and Sung Yi. “Solder Joint Reliability of Plastic Ball Grid Array Packages”. In: *Soldering & Surface Mount Technology* 11.1 (Apr. 1999), pp. 44–48. ISSN: 0954-0911. DOI: 10.1108/09540919910254930.
- [30] Meng-Kuang Huang and Chiapnyng Lee. “Board Level Reliability of Lead-free Designs of BGAs, CSPs, QFPs and TSOPs”. In: *Soldering & Surface Mount Technology* 20.3 (June 2008), pp. 18–25. ISSN: 0954-0911. DOI: 10.1108/09540910810885688.
- [31] Zhiheng Huang, Paul P. Conway, and Rachel C. Thomson. “Microstructural Considerations for Ultrafine Lead Free Solder Joints”. In: *Microelectronics Reliability* 47.12 (Dec. 2007), pp. 1997–2006. ISSN: 00262714. DOI: 10.1016/j.microrel.2007.04.013.

- [32] Nilesh Jain, Ashok Bhansali, and Deepak Mehta. “AngularJS: A Modern MVC Framework in JavaScript”. In: *Journal of Global Research in Computer Science* 5.12 (2014), pp. 17–23.
- [33] JEDEC Solid State Technology Association. *JESD22-A104E*. Mar. 2009.
- [34] Norman Lloyd Johnson, Samuel Kotz, and N. Balakrishnan. *Continuous Univariate Distributions*. 2nd ed. Wiley Series in Probability and Mathematical Statistics. New York: Wiley, 1994. ISBN: 978-0-471-58495-7 978-0-471-58494-0.
- [35] S.K. Kang et al. “Evaluation of Thermal Fatigue Life and Failure Mechanisms of Sn-Ag-Cu Solder Joints with Reduced Ag Contents”. In: *2004 Proceedings. 54th Electronic Components and Technology Conference (IEEE Cat. No.04CH37546)*. Las Vegas, NV, USA: IEEE, 2004, pp. 661–667. ISBN: 978-0-7803-8365-4. DOI: 10.1109/ECTC.2004.1319409.
- [36] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *arXiv:1412.6980 [cs]* (Jan. 2017). arXiv: 1412.6980 [cs].
- [37] Komite Nasional Keselamatan Transportasi. *Aircraft Accident Investigation Report*. Aircraft Accident Investigation Report KNKT.14.12.29.04. Komite Nasional Keselamatan Transportasi, Dec. 2014.
- [38] Jae B. Kwak and Soonwan Chung. “Thermal Fatigue Reliability for Cu-Pillar Bump Interconnection in Flip Chip on Module and Underfill Effects”. In: *Soldering & Surface Mount Technology* 27.1 (Feb. 2015), pp. 1–6. ISSN: 0954-0911. DOI: 10.1108/SSMT-03-2014-0008.
- [39] John Lau, Jerry Gleason, et al. “Design, Materials, and Assembly Process of High-density Packages with a Low-temperature Lead-free Solder (SnBiAg)”.

- In: *Soldering & Surface Mount Technology* 20.2 (Apr. 2008), pp. 11–20. ISSN: 0954-0911. DOI: 10.1108/09540910810871520.
- [40] John Lau, Jerry Gleason, et al. “Reliability Test and Failure Analysis of High-density Packages Assembled with a Low-temperature Lead-free Solder (SnBiAg)”. In: *Soldering & Surface Mount Technology* 20.2 (Apr. 2008), pp. 21–29. ISSN: 0954-0911. DOI: 10.1108/09540910810871539.
- [41] John Lau, Nick Hoo, et al. “Reliability Testing and Data Analysis of Lead-free Solder Joints for High-density Packages”. In: *Soldering & Surface Mount Technology* 16.2 (Aug. 2004), pp. 46–68. ISSN: 0954-0911. DOI: 10.1108/09540910410537336.
- [42] Hwa-Teng Lee et al. “Reliability of Sn–Ag–Sb Lead-Free Solder Joints”. In: *Materials Science and Engineering: A* 407.1-2 (Oct. 2005), pp. 36–44. ISSN: 09215093. DOI: 10.1016/j.msea.2005.07.049.
- [43] Tae-Kyu Lee et al. “Impact of Isothermal Aging on Long-Term Reliability of Fine-Pitch Ball Grid Array Packages with Sn-Ag-Cu Solder Interconnects: Surface Finish Effects”. In: *Journal of Electronic Materials* 39.12 (Dec. 2010), pp. 2564–2573. ISSN: 0361-5235, 1543-186X. DOI: 10.1007/s11664-010-1352-8.
- [44] W. W Lee, L. T Nguyen, and G. S Selvaduray. “Solder Joint Fatigue Models: Review and Applicability to Chip Scale Packages”. In: *Microelectronics Reliability* 40.2 (Feb. 2000), pp. 231–244. ISSN: 0026-2714. DOI: 10.1016/S0026-2714(99)00061-X.

- [45] J.B. Libot et al. “Microstructural Evolutions of Sn-3.0Ag-0.5Cu Solder Joints during Thermal Cycling”. In: *Microelectronics Reliability* 83 (Apr. 2018), pp. 64–76. ISSN: 00262714. DOI: 10.1016/j.microrel.2018.02.009.
- [46] Fang Liu, Jiacheng Zhou, and Nu Yan. “Thermal Cycling Effect on the Drop Reliability of BGA Lead-Free Solder Joints”. In: *Soldering & Surface Mount Technology* 29.4 (Sept. 2017), pp. 199–202. ISSN: 0954-0911. DOI: 10.1108/SSMT-03-2017-0007.
- [47] Weiping Liu et al. “Achieving High Reliability Low Cost Lead-Free SAC Solder Joints via Mn or Ce Doping”. In: *2009 59th Electronic Components and Technology Conference*. San Diego, CA: IEEE, May 2009, pp. 994–1007. ISBN: 978-1-4244-4475-5. DOI: 10.1109/ECTC.2009.5074134.
- [48] Jeffery C.C. Lo et al. “Reliability Study of Surface Mount Printed Circuit Board Assemblies with Lead-free Solder Joints”. In: *Soldering & Surface Mount Technology* 20.2 (Apr. 2008), pp. 30–38. ISSN: 0954-0911. DOI: 10.1108/09540910810871548.
- [49] Minhua Lu et al. “Comparison of Electromigration Behaviors of SnAg and SnCu Solders”. In: *2009 IEEE International Reliability Physics Symposium*. Montreal, QC, Canada: IEEE, 2009, pp. 149–154. ISBN: 978-1-4244-2888-5. DOI: 10.1109/IRPS.2009.5173241.
- [50] T.T. Mattila, V. Vuorinen, and J.K. Kivilahti. “Impact of Printed Wiring Board Coatings on the Reliability of Lead-Free Chip-Scale Package Interconnections”. In: *Journal of Materials Research* 19.11 (Nov. 2004), pp. 3214–3223. ISSN: 0884-2914, 2044-5326. DOI: 10.1557/JMR.2004.0436.

- [51] Warren S. McCulloch and Walter Pitts. “A Logical Calculus of the Ideas Immanent in Nervous Activity”. In: *The Bulletin of Mathematical Biophysics* 5.4 (Dec. 1943), pp. 115–133. ISSN: 0007-4985, 1522-9602. DOI: 10.1007/BF02478259.
- [52] Jinhua Mi et al. “Thermal Cycling Life Prediction of Sn-3.0Ag-0.5Cu Solder Joint Using Type-I Censored Data”. In: *The Scientific World Journal* 2014 (2014), pp. 1–11. ISSN: 2356-6140, 1537-744X. DOI: 10.1155/2014/807693.
- [53] Mohammad Motalab et al. “Improved Predictions of Lead Free Solder Joint Reliability That Include Aging Effects”. In: *2012 IEEE 62nd Electronic Components and Technology Conference*. San Diego, CA, USA: IEEE, May 2012, pp. 513–531. ISBN: 978-1-4673-1965-2 978-1-4673-1966-9 978-1-4673-1964-5. DOI: 10.1109/ECTC.2012.6248879.
- [54] O. Nousiainen et al. “Effect of ENIG Deposition on the Failure Mechanisms of Thermomechanically Loaded Lead-free 2nd Level Interconnections in LTCC/PWB Assemblies”. In: *Soldering & Surface Mount Technology* 22.3 (June 2010), pp. 22–35. ISSN: 0954-0911. DOI: 10.1108/09540911011054163.
- [55] Michael Osterman and Abhijit Dasgupta. “Life Expectancies of Pb-free SAC Solder Interconnects in Electronic Hardware”. In: *Journal of Materials Science: Materials in Electronics* 18.1-3 (Dec. 2006), pp. 229–236. ISSN: 0957-4522, 1573-482X. DOI: 10.1007/s10854-006-9017-3.
- [56] Michael Osterman and Michael Pecht. “Strain Range Fatigue Life Assessment of Lead-free Solder Interconnects Subject to Temperature Cycle Loading”. In: *Soldering & Surface Mount Technology* 19.2 (Apr. 2007), pp. 12–17. ISSN: 0954-0911. DOI: 10.1108/09540910710836494.

- [57] H. L. J. Pang et al. “HIGHLY ACCELERATED SOLDER JOINT RELIABILITY TEST USING A THERMO-MECHANICAL DEFLECTION SYSTEM (TMDS)”. In: *Journal of Electronics Manufacturing* 10.01 (Mar. 2000), pp. 49–57. ISSN: 0960-3131. DOI: 10.1142/S0960313100000058.
- [58] Fabian Pedregosa et al. “Scikit-Learn: Machine Learning in Python”. In: *MACHINE LEARNING IN PYTHON* (2011), p. 6.
- [59] Jussi Putaala et al. “Lifetime Prediction and Design Aspects of Reliable Lead-Free Non-Collapsible BGA Joints in LTCC Packages for RF/Microwave Telecommunication Applications”. In: *Soldering & Surface Mount Technology* 26.3 (May 2014), pp. 117–128. ISSN: 0954-0911. DOI: 10.1108/SSMT-07-2013-0018.
- [60] Venkatesh Raghavan et al. “Effects of Pre-Stressing on Solder Joint Failure by Pad Cratering”. In: *2010 Proceedings 60th Electronic Components and Technology Conference (ECTC)*. Las Vegas, NV, USA: IEEE, 2010, pp. 456–463. ISBN: 978-1-4244-6410-4. DOI: 10.1109/ECTC.2010.5490932.
- [61] P. Ratchev, B. Vandeveld, and I. De Wolf. “Reliability and Failure Analysis of Sn-Ag-Cu Solder Interconnections for PSGA Packages on Ni/Au Surface Finish”. In: *IEEE Transactions on Device and Materials Reliability* 4.1 (Mar. 2004), pp. 5–10. ISSN: 1558-2574. DOI: 10.1109/TDMR.2003.822341.
- [62] Payam Refaeilzadeh, Lei Tang, and Huan Liu. “Cross-Validation”. In: *Encyclopedia of Database Systems*. Ed. by Ling Liu and M. Tamer Özsu. Boston, MA: Springer US, 2009, pp. 532–538. ISBN: 978-0-387-35544-3 978-0-387-39940-9. DOI: 10.1007/978-0-387-39940-9_565.
- [63] Shi-Wei Ricky Lee et al. “Assessment of Board Level Solder Joint Reliability for PBGA Assemblies with Lead-free Solders”. In: *Soldering & Surface Mount*

- Technology* 14.3 (Dec. 2002), pp. 46–50. ISSN: 0954-0911.
DOI: 10.1108/09540910210444728.
- [64] Ankit Rohatgi. *WebPlotDigitizer*. May 2022.
- [65] F. Rosenblatt. “The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain”. In: *Psychological Review* (1958), pp. 65–386.
- [66] Vahid Samavatian. “A Systematic Approach to Reliability Assessment of DC-DC Power Electronic Converters”. PhD thesis. University of Tehran, 2020.
- [67] Vahid Samavatian et al. “Correlation-Driven Machine Learning for Accelerated Reliability Assessment of Solder Joints in Electronics”. In: *Scientific Reports* 10.1 (Dec. 2020), p. 14821. ISSN: 2045-2322.
DOI: 10.1038/s41598-020-71926-7.
- [68] A. Schubert et al. “Fatigue Life Models for SnAgCu and SnPb Solder Joints Evaluated by Experiments and Simulation”. In: *53rd Electronic Components and Technology Conference, 2003. Proceedings*. May 2003, pp. 603–610.
DOI: 10.1109/ECTC.2003.1216343.
- [69] J. Seyyedi and J. Padgett. “Ceramic Chip Scale Package Solder Joint Reliability”. In: *Soldering & Surface Mount Technology* 13.3 (Dec. 2001), pp. 7–11. ISSN: 0954-0911. DOI: 10.1108/09540910110407351.
- [70] Dhafer Abdulameer Shnawah, Mohd Faizul Mohd Sabri, and Irfan Anjum Badrud-din. “A Review on Thermal Cycling and Drop Impact Reliability of SAC Solder Joint in Portable Electronic Products”. In: *Microelectronics Reliability* 52.1 (Jan. 2012), pp. 90–99. ISSN: 00262714.
DOI: 10.1016/j.microrel.2011.07.093.

- [71] Mrunali Sona and K. N. Prabhu. “Review on Microstructure Evolution in Sn–Ag–Cu Solders and Its Effect on Mechanical Integrity of Solder Joints”. In: *Journal of Materials Science: Materials in Electronics* 24.9 (Sept. 2013), pp. 3149–3169. ISSN: 0957-4522, 1573-482X. DOI: 10.1007/s10854-013-1240-0.
- [72] G. Subbarayan, Y. Li, and R. L. Mahajan. “Reliability Simulations for Solder Joints Using Stochastic Finite Element and Artificial Neural Network Models”. In: *Journal of Electronic Packaging* 118.3 (Sept. 1996), pp. 148–156. ISSN: 1043-7398, 1528-9044. DOI: 10.1115/1.2792145.
- [73] Jeffrey C. Suhling et al. “Thermal Cycling Reliability of Lead-free Chip Resistor Solder Joints”. In: *Soldering & Surface Mount Technology* 16.2 (Aug. 2004), pp. 77–87. ISSN: 0954-0911. DOI: 10.1108/09540910410537354.
- [74] Ahmer Syed. “Accumulated Creep Strain and Energy Density Based Thermal Fatigue Life Prediction Models for SnAgCu Solder Joints”. In: *2004 Proceedings. 54th Electronic Components and Technology Conference (IEEE Cat. No.04CH37546)*. Las Vegas, NV, USA: IEEE, 2004, pp. 737–746. ISBN: 978-0-7803-8365-4. DOI: 10.1109/ECTC.2004.1319419.
- [75] Ahmer Syed. “Reliability of Lead-Free Solder Connections for Area-Array Packages”. In: *IPC SMEMA Council APEX 2001*. 2001, p. 9.
- [76] Ahmer Syed et al. “Impact of Package Design and Materials on Reliability for Temperature Cycling, Bend, and Drop Loading Conditions”. In: *2008 58th Electronic Components and Technology Conference*. May 2008, pp. 1453–1461. DOI: 10.1109/ECTC.2008.4550168.

- [77] Sami Tapani Nurmi and Eero Olavi Ristolainen. “Reliability of Tin-lead Balled BGAs Soldered with Lead-free Solder Paste”. In: *Soldering & Surface Mount Technology* 14.2 (Aug. 2002), pp. 35–39. ISSN: 0954-0911. DOI: 10.1108/09540910210427808.
- [78] Per-Erik Tegehall. *Review of the Impact of Intermetallic Layers on the Brittleness of Tin-Lead and Lead-Free Solder Joints*. Technical Report 06/07. 2006, p. 65.
- [79] TensorFlow Developers. *TensorFlow*. Zenodo. May 2022. DOI: 10.5281/ZENODO.4724125.
- [80] P. Towashiraporn et al. “Predictive Reliability Models through Validated Correlation between Power Cycling and Thermal Cycling Accelerated Life Tests”. In: *Soldering & Surface Mount Technology* 14.3 (Dec. 2002), pp. 51–60. ISSN: 0954-0911. DOI: 10.1108/09540910210444737.
- [81] Bart Vandeveldde et al. “Thermal Cycling Reliability of SnAgCu and SnPb Solder Joints: A Comparison for Several IC-packages”. In: *Microelectronics Reliability* 47.2-3 (Feb. 2007), pp. 259–265. ISSN: 00262714. DOI: 10.1016/j.microrel.2006.09.034.
- [82] Paul Werbos. *Beyond Regression : New Tools for Prediction and Analysis in the Behavioral Sciences*. Harvard University, 1974.
- [83] Xindong Wu et al. “Top 10 Algorithms in Data Mining”. In: *Knowledge and Information Systems* 14.1 (Jan. 2008), pp. 1–37. ISSN: 0219-1377, 0219-3116. DOI: 10.1007/s10115-007-0114-2.

- [84] Xuejun Fan, G. Rasier, and V.S. Vasudevan. “Effects of Dwell Time and Ramp Rate on Lead-Free Solder Joints in FCBGA Packages”. In: *Proceedings Electronic Components and Technology, 2005. ECTC '05*. Vol. 2. Lake Buena Vista, FL, USA: IEEE, 2005, pp. 901–906. ISBN: 978-0-7803-8906-9. DOI: 10.1109/ECTC.2005.1441379.
- [85] Se Young Yang, Ilho Kim, and Soon-Bok Lee. “A Study on the Thermal Fatigue Behavior of Solder Joints Under Power Cycling Conditions”. In: *IEEE Transactions on Components and Packaging Technologies* 31.1 (Mar. 2008), pp. 3–12. ISSN: 1557-9972. DOI: 10.1109/TCAPT.2007.906294.
- [86] Ming-Chih Yew et al. “Reliability Analysis of a Novel Fan-out Type WLP”. In: *Soldering & Surface Mount Technology* 21.3 (June 2009), pp. 30–38. ISSN: 0954-0911. DOI: 10.1108/09540910910970394.
- [87] Sung Yi and Robert Jones. “Machine Learning Framework for Predicting Reliability of Solder Joints”. In: *Soldering & Surface Mount Technology* 32.2 (Aug. 2019), pp. 82–92. ISSN: 0954-0911, 0954-0911. DOI: 10.1108/SSMT-04-2019-0013.
- [88] Cadmus C. A. Yuan and Chang-Chi Lee. “Solder Joint Reliability Modeling by Sequential Artificial Neural Network for Glass Wafer Level Chip Scale Package”. In: *IEEE Access* 8 (2020), pp. 143494–143501. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3014156.

Appendix A. Computing parameters of the Weibull distribution

The failure rate $F(t)$ of a Weibull distribution with life parameter θ and shape parameter β is given by

$$F(t) = 1 - e^{-\left(\frac{t}{\theta}\right)^\beta} \quad (\text{A.1})$$

Eq. (A.1) can be rearranged as follows:

$$\log\left[-\log(1 - F(t))\right] = \beta \log(t) - \beta \log(\theta) \quad (\text{A.2})$$

Given a vector of measured failure rates \mathbf{F} and a vector of corresponding fatigue lives \mathbf{t} , the parameters θ, β are computed as follows.

$$\text{Let } y_i := \log\left[-\log(1 - F_i)\right] \quad (\text{A.3})$$

$$\text{Let } \hat{y}_i(\beta, \theta) := \beta \log(t_i) - \beta \log(\theta) \quad (\text{A.4})$$

$$\text{Let } \epsilon_i(\beta, \theta) := y_i - \hat{y}_i(\beta, \theta) \quad (\text{A.5})$$

We seek parameters β, θ such that the sum of squared residuals $\sum_{i=1}^N \epsilon_i^2$ is minimized.

Eq. (A.4) may be rewritten as a matrix operation

$$\text{Let } \mathbf{A} := \begin{bmatrix} 1 & \log t_1 \\ \vdots & \vdots \\ 1 & \log t_N \end{bmatrix} \quad (\text{A.6})$$

$$\text{Let } \mathbf{b} := \begin{bmatrix} -\beta \log \theta \\ \beta \end{bmatrix} \quad (\text{A.7})$$

$$\hat{\mathbf{y}} = \mathbf{A}\mathbf{b} \quad (\text{A.8})$$

The residual defined in Eq. (A.5) can then be written in matrix form as

$$\boldsymbol{\epsilon} = \mathbf{y} - \mathbf{A}\mathbf{b} \quad (\text{A.9})$$

The sum of squared residuals may be written in matrix form as

$$\sum_{i=1}^N \epsilon_i = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} \quad (\text{A.10})$$

Eq. (A.9) can be substituted into Eq. (A.10):

$$\boldsymbol{\epsilon}^T \boldsymbol{\epsilon} = (\mathbf{y} - \mathbf{A}\mathbf{b})^T (\mathbf{y} - \mathbf{A}\mathbf{b}) \quad (\text{A.11})$$

$$= \mathbf{y}^T \mathbf{y} - \mathbf{y}^T \mathbf{A}\mathbf{b} - \mathbf{b}^T \mathbf{A}^T \mathbf{y} + 2\mathbf{b}^T \mathbf{A}^T \mathbf{A}\mathbf{b} \quad (\text{A.12})$$

Taking the derivative of Eq. (A.12) with respect to \mathbf{b}^T and solving for \mathbf{b} to find the

minimum yields

$$0 = -2\mathbf{A}^\top \mathbf{y} + 2\mathbf{A}^\top \mathbf{A} \mathbf{b} \quad (\text{A.13})$$

$$\mathbf{A}^\top \mathbf{A} \mathbf{b} = \mathbf{A}^\top \mathbf{y} \quad (\text{A.14})$$

$$\mathbf{b} = \begin{bmatrix} -\beta \log \theta \\ \beta \end{bmatrix} = (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top \mathbf{y} \quad (\text{A.15})$$

$$\theta = e^{-\frac{b_2}{b_1}} \quad (\text{A.16})$$

Appendix B. Database architecture

The Electronics Packaging Materials Database provides a *REST API* in order to enable users to create, retrieve, update, and delete records using scripts. The REST API concept was originally developed by Fielding (2000). *REST* stands for *Representational State Transfer*. It is an architectural style for web services. REST is based upon a client-server architecture in which user interface concerns are separated from data storage concerns. In the REST paradigm, client-server communication is *stateless*, i.e. each request from the client to the server is complete and does not rely on additional context stored on the server, thus the client is solely responsible for maintaining the session state. REST provides a uniform interface for various services and data. REST enables *resources* to be communicated in different *representations* specified by metadata. A REST *resource* is an abstract notion of *a piece of information that can be named* which could include a document, a table, or some kind of service (e.g. the current conversion rate from USD to yen). A REST implementation provides methods for representing resources in different formats, e.g. JSON or HTML.

The term *API* stands for *Application Programming Interface* and refers to a set of methods that can be called by an external application. The REST API of the Electronics Packaging Materials Database provides methods to create, retrieve, update, and delete records of experiments performed on materials. The API is self-documenting: a url endpoint is provided for a specification document which describes the objects, methods, and endpoints implemented by the API. This document can

be consumed by various software to automatically generate an API client software developer’s kit for the end-user in their desired language.

The underlying data model used for storage in the Electronics Packaging Materials Database is a *relational database*. The concept of the relational data model was developed by Codd (1970). The relational database model stores *records* in *tables* related by *foreign keys*. Hierarchical data where a single *parent* object has several *child* objects may be represented by a foreign key relation wherein each child record has a field containing the *primary key* of the parent. The structure of the database, including the definitions of its tables, is referred to as its *schema*.

In the Electronics Packaging Materials Database, the basic unit of information is an *entity*. Fig. B.1 demonstrates how an experiment entity is represented. Each of the blocks represent a row in a database table and the arrows represent foreign key relations. In the figure, there is an *experiment* entity on the far right with the label “Example Experiment” and a `category_id` field with the value 314. The `category_id` column has a foreign key relation to the category table. The category with id 314 has the label “Experiment”. In this way, the experiment is assigned to a category. The “Attribute” block in the left of the figure represents an attribute for the number of pins on a chip. It has an “id” field with the value 265 (this is used to uniquely identify this attribute within the database), a “label” field with the value “Number of Pins” (this is used to identify the attribute to the user), a “datatype” field with the value “integer” (this is used to indicate that this attribute should only be used with integer values), and a `category_id` field, which enforces that this attribute may only be used with entities having the category “Experiment”. The “Integer Value” block in the top center of the figure is used to assign a value of the “Number of Pins” attribute to the “Example Experiment” entity.

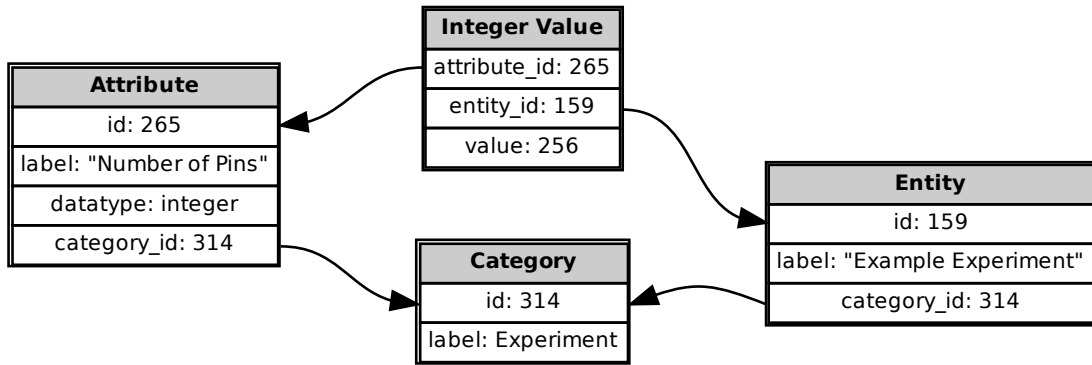


Figure B.1.: Example of an experiment with a “Number of Pins” attribute.

Fig. B.2 demonstrates how a material can be represented in the database.

Fig. B.3 demonstrates how an “Entity Value” can be used to represent a relation between two entities. In this example, the experiment and material entities are related by an entity value which uses the “Material” attribute.

The next abstraction layer is the *object relational mapper* (ORM). This provides a mapping from database relational objects to Python objects using the Django web framework (Django Software Foundation 2022). With the ORM, database objects composed of hierarchical data (e.g. experiments) can be operated upon as if they were a single hierarchical object with the ORM managing the appropriate queries to the underlying database tables, e.g. the “category” attribute of an “entity” object can be queried directly to retrieve the related category record, eliding an SQL statement similar to

```
SELECT *
FROM entity
JOIN category
ON entity.category_id = category.id
WHERE entity.id = 42;
```

The ORM also provides the implementation of the REST API for the Electronics Packaging Materials Database using the Django Rest Framework (Encode OSS Ltd.

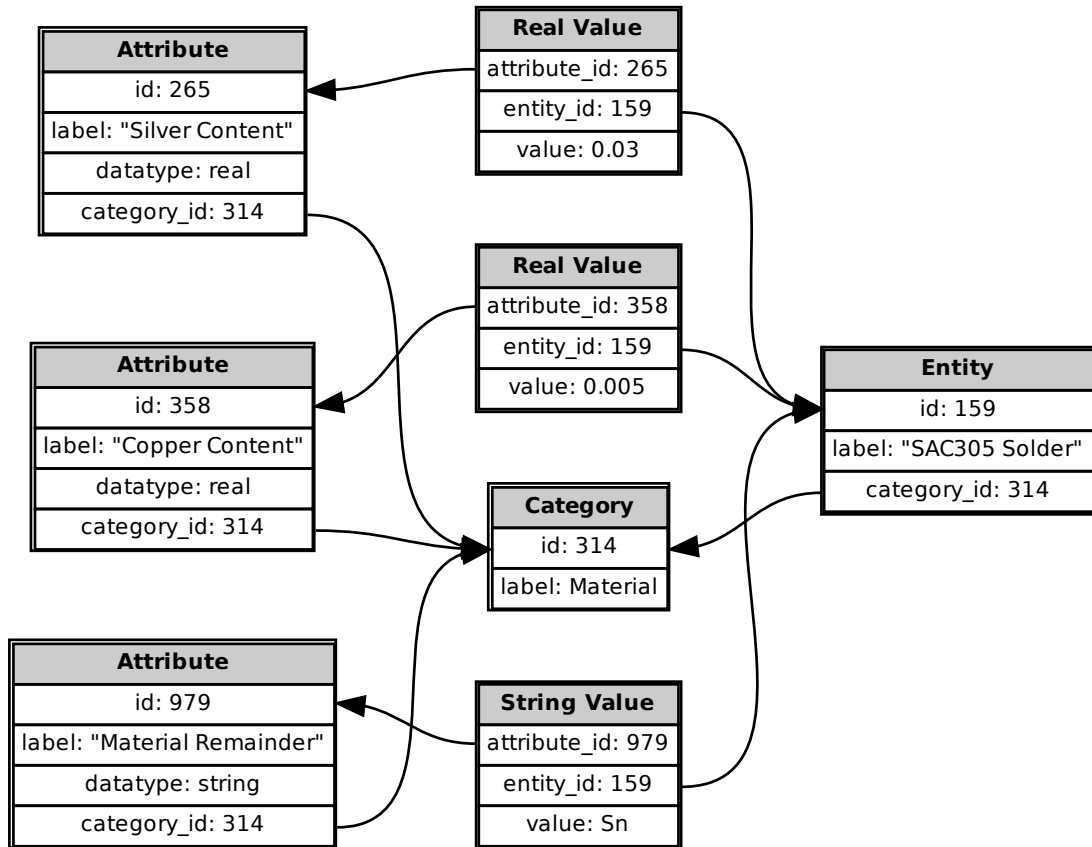


Figure B.2.: Example of how a material entity is represented in the database.

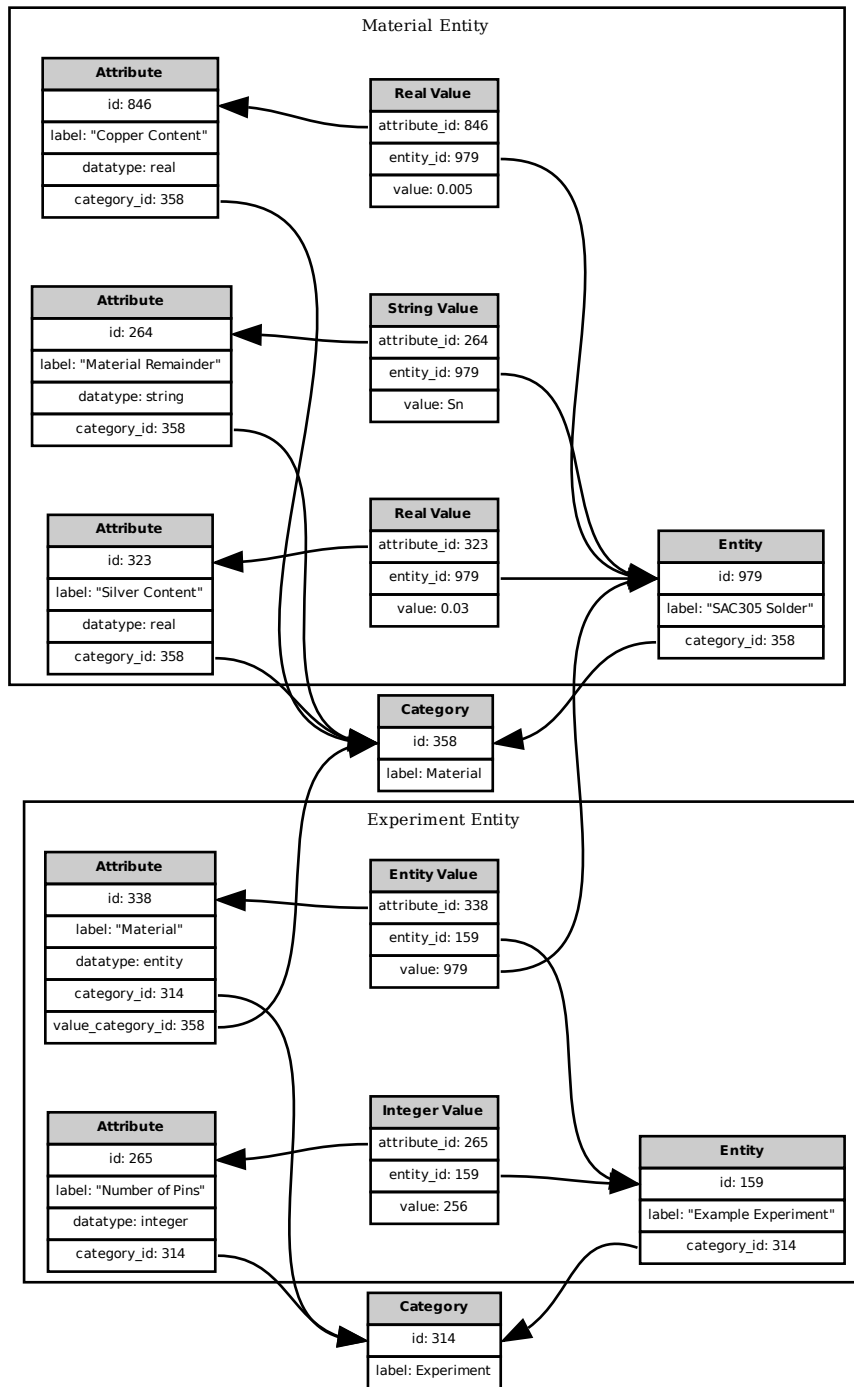


Figure B.3.: Example of an experiment with a “Number of Pins” attribute and a “Material” attribute.

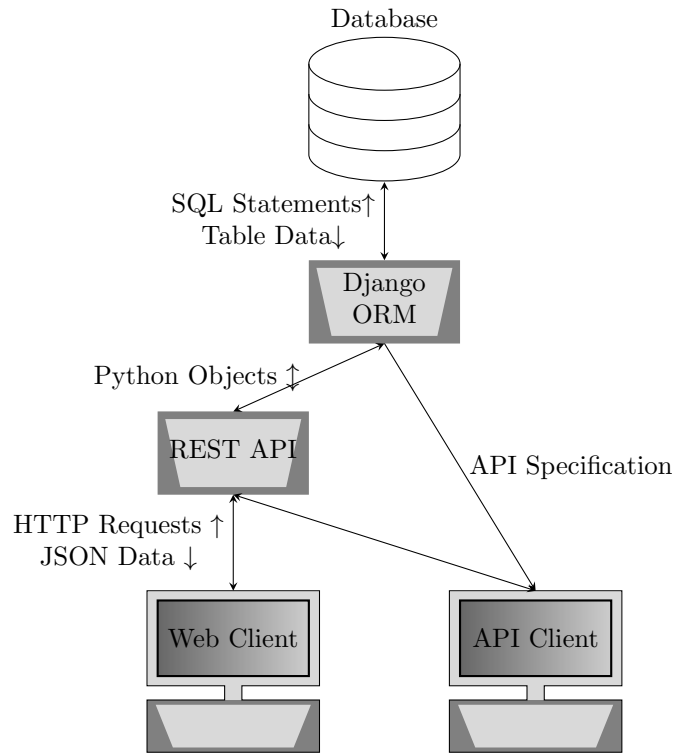


Figure B.4.: Block diagram of database, REST API, and clients.

2022). The architecture of the Electronics Packaging Materials Database website is shown in Fig. B.4.

The web site uses client-side JavaScript to implement a REST client in the browser and handles all HTTP requests through the REST API. The browser-based REST client uses the Angular framework to provide interactive features (Jain, Bhansali, and Mehta 2014).

By using the API specification document, much of the client-side JavaScript including logic for object specification, endpoints, and forms can be generated automatically. By using the REST API as the interface for both browser-based and command-line-

based clients, consistency in features, permissions, and accessibility is assured and the total testing and vulnerability surface area of the application is minimized.

Appendix C. Plot Digitization

The WebPlotDigitizer software can be used to collect data from published charts and add the data to the Electronics Packaging Materials Database (Rohatgi 2022). To begin digitizing a plot, users proceed as follows.

1. The user locates a plot of failure rate vs. cycles to failure, similar to what is shown in Fig. C.1.
2. The user loads the plot in the WebPlotDigitizer interface as shown in Fig. C.2.
3. The user digitizes the plot as shown in Fig. C.3.
4. Reliability plots frequently use a scale transformation of the form

$$f(x) = \log[-\log(1 - x)] \quad (\text{C.1})$$

for the y axis. This transformation is not currently supported by WebPlotDigitizer, so instead, the user must perform the transformation themselves. This transformation can be performed using Python as demonstrated in Listing C.1.

5. The user takes the transformed data and adds it as an attribute of the experiment in the Electronics Packaging Materials Database as shown in Fig. C.4.

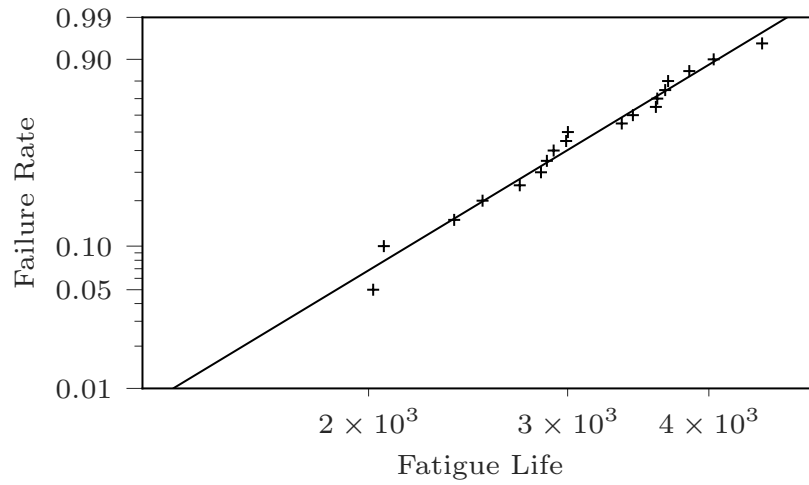


Figure C.1.: Example plot, generated for illustration purposes

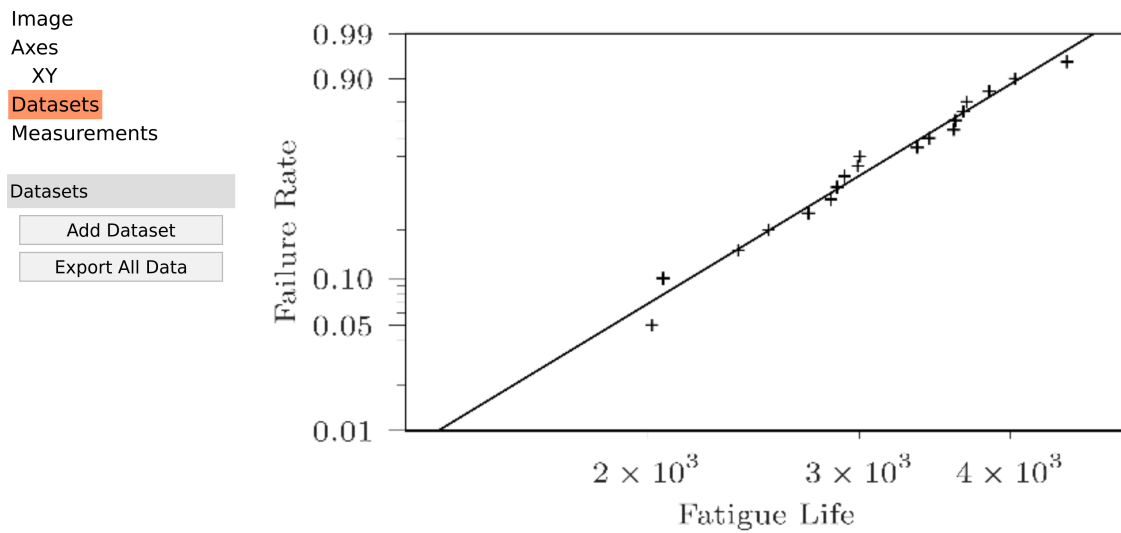


Figure C.2.: Example plot loaded into WebPlotDigitizer

Listing C.1.: Python 3 code to perform failure rate transformation.

```
1 import csv
2 from io import StringIO
3 from math import log, exp
4 point_1 = float(input("Input value of first calibration point: "))
5 point_2 = float(input("Input value of second calibration point: "))
6 # Compute the transformed values of the calibration points
7 g1 = log(-log(1 - point_1))
8 g2 = log(-log(1 - point_2))
9 # Compute parameters of a linear transformation from screen coordinates
10 # to transformed data coordinates
11 a = (point_2 - point_1) / (g2 - g1)
12 b = (point_1 * g2 - point_2 * g1) / (g2 - g1)
13 print(
14     "Enter data in comma-separated value format. Empty line to end."
15 )
16 lines = [] # Build a list of lines
17 while True: # Get lines from user one at a time
18     line = input()
19     if not line.strip():
20         break # Stop getting lines if user enters a blank line
21     lines.append(line)
22 data = "\n".join(lines) # Convert list of lines into a string
23 # Parse the input lines as Comma-Separated Value (CSV) data
24 reader = csv.reader(StringIO(data))
25 # Build output by transforming input data
26 output = []
27 for row in reader:
28     # Convert rows of CSV data from strings to numbers
29     cycles, scale_failure_rate = map(lambda x: float(x.strip()), row)
30     # Remove linear transformation of screen coordinates
31     # from failure rate data
32     transformed_failure_rate = (scale_failure_rate - b) / a
33     # Apply inverse Weibull transformation to failure rate data
34     true_failure_rate = 1 - (exp(-exp(transformed_failure_rate)))
35     output.append([cycles, true_failure_rate])
36 output_buffer = StringIO()
37 writer = csv.writer(output_buffer)
38 for row in output:
39     writer.writerow(row)
40 print(output_buffer.getvalue())
```

Image
 Axes
 XY
 Datasets
 ■ Default Dataset
 Measurements

Dataset

Axes: XY

Display Color

Rename Dataset

Delete Dataset

Edit Point Groups

View Data

Clear Data

Data Points: 19

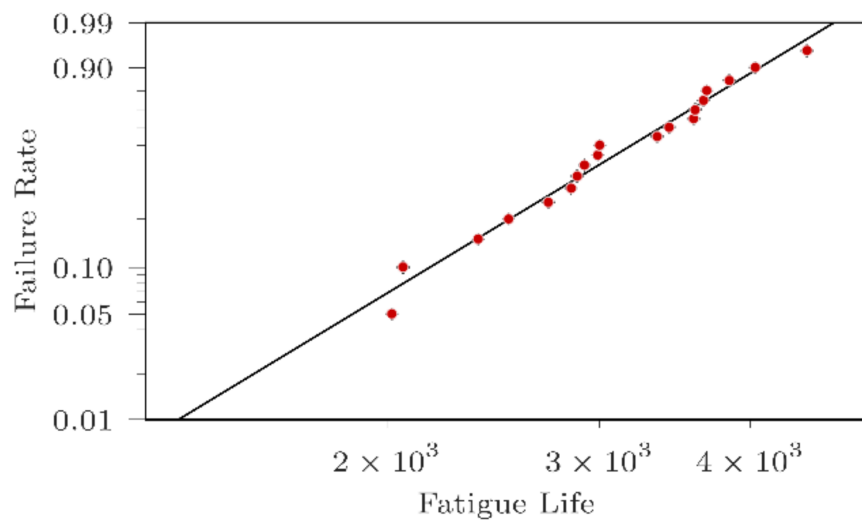
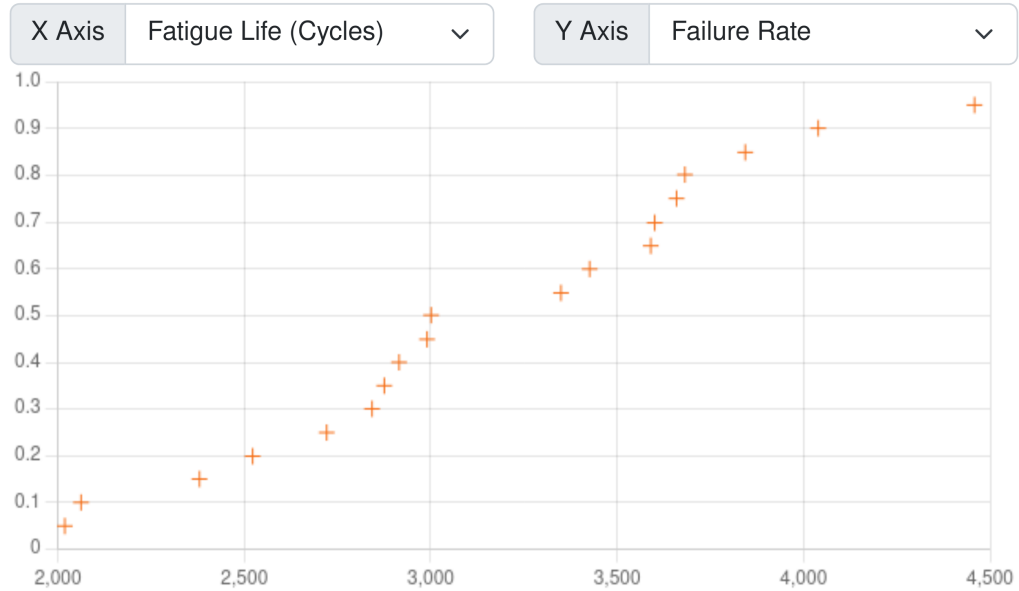


Figure C.3.: Example data digitized in WebPlotDigitizer

Fatigue Data

Chart



CSV Data

, (Comma)

```
fatigue_life, failure_rate
2018.364606851374, 0.05003441979384704
2061.878612958447, 0.10045244438011613
2379.5411741139233, 0.15050087802340617
2522.130633940557, 0.1992741038916177
2720.745710321908, 0.2499265818573193
2842.476058201239, 0.30070133529129983
```

Figure C.4.: Example data loaded into the Electronics Packaging Materials Database

Appendix D. Data Values

Values used for the experiments included in the analysis are listed in Table D.1. The ranges of the different values are listed in Table D.2. For categorical features, all variants are listed in curly brackets. For numerical features, the range is indicated by the minimum and maximum values enclosed in square brackets.

Table D.1.: Data values used for machine learning

Id	Reference	Package Type	Solder Ball Material	Number of Pads	Board Surface Treatment	Max Cycle Temperature (°C)	Min Cycle Temperature (°C)	Aging Temperature (°C)	Aging Time (s)	Pad array length	Pad array width	Number of pad rows	Number of pad columns
1	[43]	BGA	Sn-3.0Ag-0.5Cu	432	OSP	100	0	100	3.6e+06	12.4	12.4	31	31
2	[43]	BGA	Sn-3.0Ag-0.5Cu	432	OSP	100	0	100	3.6e+06	12.4	12.4	31	31
3	[43]	BGA	Sn-3.0Ag-0.5Cu	432	OSP	100	0	100	1.8e+06	12.4	12.4	31	31
4	[43]	BGA	Sn-3.0Ag-0.5Cu	432	OSP	100	0	100	1.8e+06	12.4	12.4	31	31
5	[43]	BGA	Sn-3.0Ag-0.5Cu	432	OSP	100	0	100	3.6e+06	12.4	12.4	31	31
6	[43]	BGA	Sn-3.0Ag-0.5Cu	432	OSP	100	0	100	3.6e+06	12.4	12.4	31	31
7	[43]	BGA	Sn-3.0Ag-0.5Cu	432	OSP	100	0	100	1.8e+06	12.4	12.4	31	31
8	[43]	BGA	Sn-3.0Ag-0.5Cu	432	OSP	100	0	100	1.8e+06	12.4	12.4	31	31
9	[76]	BGA	Sn-1.0Ag-0.5Cu	360	OSP	125	-55	0	0	9.2	9.2	23	23
10	[76]	BGA	Sn-1.0Ag-0.5Cu	360	OSP	125	-55	0	0	9.2	9.2	23	23
11	[75]	FluXBGA	Sn-4.0Ag-0.5Cu	144	OSP	125	-40	0	0	9.6	9.6	12	12
12	[75]	FluXBGA	Sn-4.0Ag-0.5Cu	144	OSP	100	0	0	0	9.6	9.6	12	12

Table D.1 Continued: Data values used for machine learning

Id	Reference	Package Type	Solder Ball Material	Number of Pads	Board Surface Treatment	Max Cycle Temperature (°C)	Min Cycle Temperature (°C)	Aging Temperature (°C)	Aging Time (s)	Pad array length	Pad array width	Number of pad rows	Number of pad columns
13	[75]	FleXBGA	Sn-37Pb	144	OSP	125	-40	0	0	9.6	9.6	12	12
14	[75]	FleXBGA	Sn-37Pb	144	OSP	100	0	0	0	9.6	9.6	12	12
15	[28]	PBGA	Sn-3.0Ag-0.5Cu	100	OSP	125	-40	0	0	8	8	10	10
16	[63]	PBGA	Sn-37Pb	316	ENIG	125	-40	0	0	25.4	25.4	20	20
17	[63]	PBGA	Sn-37Pb	313	ENIG	125	-40	0	0	31.75	31.75	25	25
18	[63]	PBGA	Sn-3.9Ag-0.6Cu	316	ENIG	125	-40	0	0	25.4	25.4	20	20
19	[63]	PBGA	Sn-3.9Ag-0.6Cu	313	ENIG	125	-40	0	0	31.75	31.75	25	25
20	[63]	PBGA	Sn-3.9Ag-0.6Cu	316	ENIG	125	-40	0	0	25.4	25.4	20	20
21	[63]	PBGA	Sn-3.9Ag-0.6Cu	313	ENIG	125	-40	0	0	31.75	31.75	25	25
22	[63]	PBGA	Sn-3.9Ag-0.6Cu	316	ENIG	125	-40	0	0	25.4	25.4	20	20
23	[63]	PBGA	Sn-3.9Ag-0.6Cu	313	ENIG	125	-40	0	0	31.75	31.75	25	25
24	[63]	PBGA	Sn-3.9Ag-0.6Cu	316	ENIG	125	-40	0	0	25.4	25.4	20	20

Table D.1 Continued: Data values used for machine learning

Id	Reference	Package Type	Solder Ball Material	Number of Pads	Board Surface Treatment	Max Cycle Temperature (°C)	Min Cycle Temperature (°C)	Aging Temperature (°C)	Aging Time (s)	Pad array length	Pad array width	Number of pad rows	Number of pad columns
25	[63]	PBGA	Sn-3.9Ag-0.6Cu	313	ENIG	125	-40	0	0	31.75	31.75	25	25
26	[77]	BGA	Sn-36Pb-2.0Ag	151	NiAu	125	-40	0	0	9	9	18	18
27	[77]	BGA	Sn-36Pb-2.0Ag	151	OSP	125	-40	0	0	9	9	18	18
28	[77]	BGA	Sn-3.8Ag-0.7Cu	151	NiAu	125	-40	0	0	9	9	18	18
29	[77]	BGA	Sn-3.8Ag-0.7Cu	151	OSP	125	-40	0	0	9	9	18	18
30	[30]	BGA	Sn-3.0Ag-0.5Cu	576	ENIG	125	-40	0	0	30.48	30.48	24	24
31	[30]	CSP	Sn-3.0Ag-0.5Cu	48	ENIG	125	-40	0	0	4.9	4.9	7	7
32	[30]	QFP	Sn-58Bi	208	ENIG	125	-40	0	0	7.5	7.5	15	15
33	[30]	TSOP	Sn-58Bi	28	ENIG	125	-40	0	0	3	3	6	6
34	[30]	BGA	Sn-3.0Ag-0.5Cu	576	OSP	125	-40	0	0	30.48	30.48	24	24
35	[30]	CSP	Sn-3.0Ag-0.5Cu	48	OSP	125	-40	0	0	4.9	4.9	7	7
36	[30]	QFP	Sn-58Bi	208	OSP	125	-40	0	0	7.5	7.5	15	15

Table D.1 Continued: Data values used for machine learning

Id	Reference	Package Type	Solder Ball Material	Number of Pads	Board Surface Treatment	Max Cycle Temperature (°C)	Min Cycle Temperature (°C)	Aging Temperature (°C)	Aging Time (s)	Pad array length	Pad array width	Number of pad rows	Number of pad columns
37	[30]	TSOP	Sn-58Bi	28	OSP	125	-40	0	0	3	3	6	6
38	[47]	TFBGA	Sn-1.0Ag-0.5Cu	244	OSP	125	-40	150	900000	9	9	18	18
39	[76]	BGA	Sn-1.0Ag-0.5Cu	360	OSP	125	-55	0	0	11.5	11.5	23	23
40	[76]	BGA	Sn-1.0Ag-0.5Cu	360	OSP	125	-55	0	0	9.2	9.2	23	23
41	[47]	TFBGA	Sn-3.0Ag-0.5Cu	244	OSP	125	-40	150	900000	9	9	18	18
42	[76]	BGA	Sn-3.0Ag-0.5Cu	360	OSP	125	-55	0	0	11.5	11.5	23	23
43	[76]	BGA	Sn-3.0Ag-0.5Cu	360	OSP	125	-55	0	0	11.5	11.5	23	23
44	[75]	FleXBGA	Sn-4.0Ag-0.5Cu	144	OSP	125	-40	0	0	9.6	9.6	12	12
45	[75]	FleXBGA	Sn-4.0Ag-0.5Cu	144	OSP	125	-55	0	0	9.6	9.6	12	12
46	[75]	FleXBGA	Sn-37Pb	144	OSP	125	-40	0	0	9.6	9.6	12	12
47	[75]	FleXBGA	Sn-37Pb	144	OSP	125	-55	0	0	9.6	9.6	12	12
48	[77]	BGA	Sn-36Pb-2.0Ag	151	NiAu	125	-40	0	0	9	9	18	18

Table D.1 Continued: Data values used for machine learning

Id	18	18	18	18	18	24	7	15	6	31	31
Reference	[77]	[77]	[77]	[47]	[30]	[30]	[30]	[30]	[30]	[43]	[43]
Package Type	BGA	BGA	BGA	TFBGA	BGA	CSP	QFP	TSOP	BGA	BGA	
Solder Ball Material	Sn-36Pb-2.0Ag	Sn-36Pb-2.0Ag	Sn-36Pb-2.0Ag	Sn-37Pb	Sn-37Pb	Sn-37Pb	Sn-37Pb	Sn-37Pb	Sn-37Pb	Sn-3.0Ag-0.5Cu	Sn-3.0Ag-0.5Cu
Number of Pads	151	151	151	244	576	48	208	28	432	432	
Board Surface Treatment	OSP	NiAu	OSP	OSP	ENIG	ENIG	ENIG	ENIG	OSP	OSP	
Max Cycle Temperature (°C)	125	125	125	125	125	125	125	125	100	100	
Min Cycle Temperature (°C)	-40	-40	-40	-40	-40	-40	-40	-40	0	0	
Aging Temperature (°C)	0	0	0	150	0	0	0	0	0	0	
Aging Time (s)	0	0	0	900000	0	0	0	0	0	0	
Pad array length	9	9	9	9	30.48	4.9	7.5	3	12.4	12.4	
Pad array width	9	9	9	9	30.48	4.9	7.5	3	12.4	12.4	
Number of pad rows	18	18	18	18	24	7	15	6	31	31	
Number of pad columns	18	18	18	18	24	7	15	6	31	31	

Table D.2.: Ranges of data values used for machine learning

Attribute	Values
Number of pad columns	[6, 31]
Number of pad rows	[6, 31]
Pad array width	[3, 33.02]
Pad array length	[3, 33.02]
Aging Time (s)	[0, $3.6e + 06$]
Aging Temperature ($^{\circ}\text{C}$)	[0, 150]
Min Cycle Temperature ($^{\circ}\text{C}$)	[-55, 0]
Max Cycle Temperature ($^{\circ}\text{C}$)	[100, 125]
Board Surface Treatment	{ENIG, I/Ag, Immersion Ag, Ni/Au, OSP, Sn-Pb HASL}
Number of Pads	[28, 900]
Solder Ball Material	{Sn-0.7Cu, Sn-1.0Ag-0.5Cu, Sn-1.0Ag-0.5Cu-0.02Ce, Sn-1.0Ag-0.5Cu-0.05Mn, Sn-1.2Ag-0.5Cu-0.05Ni, Sn-2.1Ag-0.9Cu, Sn-2.3Ag-0.5Cu-0.2Bi, Sn-2.5Ag-0.9Cu, Sn-3.0Ag-0.5Cu, Sn-3.4Ag-0.7Cu, Sn-3.5Ag, Sn-3.5Ag-0.5Cu, Sn-3.5Ag-51Cu, Sn-3.87Ag-0.7Cu, Sn-3.8Ag-0.7Cu, Sn-3.9Ag-0.6Cu, Sn-36Pb-2.0Ag, Sn-37Pb, Sn-4.0Ag-0.5Cu, Sn-4.0Ag-1.0Cu, Sn-58Bi}
Package Type	{BGA, CBGA, CLCC, CSP, Chip Resistor, FBGA, LGA, LTCC, PBGA, QFN, QFP, SBGA, TARRY, TFBGA, TSOP, WLP}
Failure Rate	[0.01679, 0.99017]
Fatigue Life	[32, 11545]

Appendix E. Nested K-Fold Cross-Validation

To understand *nested* K-fold cross-validation, the reader must first be familiar with K-fold cross-validation. The K-fold cross-validation algorithm is presented in Algorithm 2. In K-fold cross-validation, k models will be trained and tested, each time using 1 of the k testing partitions as described in Algorithm 1. For each testing partition, the complement of the partition is used to train the model. The trained model is then scored on the testing data. Since the k testing sets were created by partitioning the entire data set, each example in the data set is used for testing exactly once. The nested K-fold cross-validation algorithm uses a grid-search hyperparameter optimization procedure described in Algorithm 3. The grid-search optimization procedure searches a grid of model hyperparameters and uses Algorithm 2 to compute scores for each “point” (i.e. parameter combination) in the grid. The parameter combination with the best score is returned. Nested K-fold cross-validation splits the data into K folds, performs Algorithm 3 over each fold to find the optimal hyperparameter combination for each fold, and computes a score on the testing data for each fold using that optimal combination of hyperparameters.

The nested K-fold cross-validation algorithm is illustrated in Fig. E.1. In the figure, 4 “outer folds” are marked with braces. Within each outer fold, a grid-search optimization is performed on 3 “inner folds”.

Fig. E.2 depicts the nested cross-validation process. In an outer loop, data is split into different train/test partitions. Within the outer loop, a hyperparameter loop iterates through all combinations of hyperparameters. The training/validation data from the outer loop and the hyperparameter combination from the hyperparame-

Algorithm 1 K-Fold Splitting algorithm. This algorithm returns lists of “validation” and “training” sets. Each “validation” set consists of disjoint sets of indices that form a partition of S . Each “training” set consists of sets of indices that are the complement of the corresponding validation set.

1: **function** KFOLDSPLIT($k \in \mathbb{Z}^+$, $S = \{s | s \in \mathbb{Z}^+\}$) \triangleright k is the number of folds \triangleright S is a set of indices to split

Require: $k \leq |S|$

2: $V \leftarrow \{V_i | 1 \leq i \leq k \wedge V_i \subset S\}$ \triangleright Define a set of k subsets of S for validation

Require: $V_i \cap V_j = \emptyset \iff i \neq j$ \triangleright the subsets are disjoint

Require: $\bigcup_{m=1}^k V_m = S$ \triangleright the union of the subsets is S

Require: $|V_i| \approx |V_j|$ \triangleright subsets are approximately the same length

3: $T \leftarrow \{T_i | 1 \leq i \leq k \wedge T_i = S \setminus V_i\}$ \triangleright Define a set of k subsets of S for training

4: **return** V, T

5: **end function**

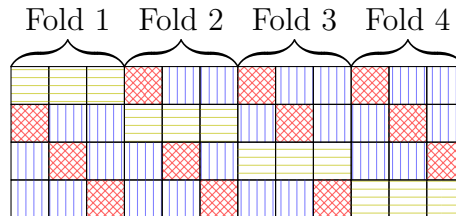





Figure E.1.: Illustration of nested cross-validation using 4 outer folds and 3 inner folds. : Outer testing fold; : Inner validation fold; : Training fold.

Algorithm 2 K-Fold scoring algorithm. This algorithm takes a learner and computes a score on k splits of the data. k is the number of folds. S is the set of indices of the features and labels. \mathcal{X} is a list of features which is indexed by S . \mathcal{Y} is a list of labels which is indexed by S . `model` is the machine learning model to evaluate. `model` shall have a `train` method which takes a list of features and labels and performs the training on them. `model` shall have a `predict` method which takes a list of features and returns a list of labels predicted by the machine learning algorithm. `scorer` is a function that takes two sets of labels and returns a scalar which represents some measure of error between the two sets. Examples of scorers include the Mean Absolute Error, Mean Squared Error, etc.

1: **function** KFOLDScore($k \in \mathbb{Z}^+$, $S = \{s \in \mathbb{Z}^+\}$, $\mathcal{X} = \{X_i | i \in S\}$, $\mathcal{Y} = \{Y_i | i \in S\}$,
`model`, `scorer` $\in \{f : a \subset Y, b \subset Y \mapsto \mathbb{R}\}$)

Require: $k \leq |S|$

2: $\sigma \leftarrow 0$ ▷ Initialize aggregated error
3: $V, T \leftarrow \text{KFOLDSPLIT}(k, S)$
4: **for** $i \leftarrow 1 \dots k$ **do** ▷ Iterate over the folds
5: $X_{\text{train}} \leftarrow \{X_j | j \in T_i\}$ ▷ Define training features
6: $Y_{\text{train}} \leftarrow \{Y_j | j \in T_i\}$ ▷ Define training labels
7: `model.train`($X_{\text{train}}, Y_{\text{train}}$) ▷ Train the model on the training labels and
features
8: $X_{\text{validate}} \leftarrow \{X_j | j \in V_i\}$ ▷ Define testing features
9: $Y_{\text{validate}} \leftarrow \{Y_j | j \in V_i\}$ ▷ Define testing labels
10: $Y_{\text{predict}} \leftarrow \text{model.predict}(X_{\text{validate}})$ ▷ Use the trained model to predict
validation labels
11: $\sigma \leftarrow \sigma + \text{scorer}(Y_{\text{validate}}, Y_{\text{predict}})$ ▷ Score the predicted validation labels
against the known labels
12: **end for**
13: **return** $\frac{\sigma}{k}$ ▷ Return the average score
14: **end function**

Algorithm 3 Grid search hyperparameter optimization algorithm. This algorithm takes a learner and a set of parameters and returns the combination of parameters which give the best score over k folds of the data. \mathcal{P} is a set of sets of parameters for `model` indexed by $i \in I$. Each member P_i of \mathcal{P} is a set of values for a single parameter of `model`. For example, a dense, fully-connected neural network might have a parameter for the number of hidden layers and a parameter for the number of neurons per layer. If the number of layers takes the range $\{2, 3\}$ and the number of neurons per layer takes the range $\{32, 64\}$, then e.g. $P_1 = \{2, 3\}$, $P_2 = \{32, 64\}$, $I = \{1, 2\}$. Remaining parameters are the same as Algorithm 2. The cartesian product $\times_{i \in I} P_i$ would then be $\{\{2, 32\}, \{2, 64\}, \{3, 32\}, \{3, 64\}\}$. `model` must have a `set_params` method defined.

```

1: function GRIDSEARCH(  $k \in \mathbb{Z}^+$ ,  $S = \{s \in \mathbb{Z}^+\}$ ,  $\mathcal{X} = \{X_i | i \in S\}$ ,  $\mathcal{Y} = \{Y_i | i \in S\}$ ,
   model, scorer  $\in \{f : a \subset Y, b \subset Y \mapsto \mathbb{R}\}$ ,  $\mathcal{P} = \{P_j | j \in I\}$ )
2:    $\sigma_{\text{best}} \leftarrow \infty$ 
3:    $p_{\text{best}} \leftarrow \emptyset$ 
4:   for all  $p_i \in \times_{i \in I} P_i$  do  $\triangleright$  Iterate over the cartesian product of members of  $\mathcal{P}$ 
5:     model.set_params( $p_i$ )
6:      $\sigma \leftarrow \text{KFOLDSCORE}(k, S, \mathcal{X}, \mathcal{Y}, \text{model}, \text{scorer})$ 
7:     if  $\sigma < \sigma_{\text{best}}$  then  $\triangleright$  Always true if this is the first combination
8:        $\sigma_{\text{best}} \leftarrow \sigma$ 
9:        $p_{\text{best}} \leftarrow p_i$ 
10:    end if
11:  end for
12:  return  $p_{\text{best}}, \sigma_{\text{best}}$ 
13: end function

```

Algorithm 4 Nested K-Fold Scoring. K is the number of outer folds for testing data, k is the number of inner folds to use with Algorithm 3. Remaining parameters are the same as Algorithm 3. Returns a list of best scores found for each of the k folds.

```

1: function NESTEDKFOLDScore(  $K \in \mathbb{Z}^+$ ,  $k \in \mathbb{Z}^+$ ,  $S = \{s \in \mathbb{Z}^+\}$ ,  $\mathcal{X} = \{X_i | i \in S\}$ ,  $\mathcal{Y} = \{Y_i | i \in S\}$ , model, scorer  $\in \{f : a \subset Y, b \subset Y \mapsto \mathbb{R}\}$ ,  $\mathcal{P} = \{P_j | j \in I\}$ )
2:    $V, T \leftarrow \text{KFOLDSPPLIT}(K, S)$  ▷ Get training and testing index sets
3:   scores  $\leftarrow []$  ▷ Initialize an empty list of scores
4:   for  $i \leftarrow 1 \dots K$  do
5:      $X_{\text{train}} \leftarrow \{X_j | j \in T_i\}$  ▷ Define training features
6:      $Y_{\text{train}} \leftarrow \{Y_j | j \in T_i\}$  ▷ Define training labels
7:      $p_{\text{best}}, \sigma_{\text{best}} \leftarrow \text{GRIDSEARCH}(k, T_i, X_{\text{train}}, Y_{\text{train}}, \text{model}, \text{scorer}, \mathcal{P})$ 
8:     model.set_params( $p_{\text{best}}$ )
9:      $X_{\text{test}} \leftarrow \{X_j | j \in V_i\}$  ▷ Define testing features
10:     $Y_{\text{predict}} \leftarrow \text{model.predict}(X_{\text{test}})$  ▷ Use the trained model to predict
    validation labels
11:     $Y_{\text{test}} \leftarrow \{Y_j | j \in V_i\}$  ▷ Define testing labels
12:     $\sigma \leftarrow \text{scorer}(Y_{\text{test}}, Y_{\text{predict}})$  ▷ Score the model with best parameters
13:    scores.append( $\sigma$ ) ▷ Append the score on the validation fold to the list
    of scores
14:   end for
15:   return scores
16: end function

```

ter loop are passed to an inner training loop within the hyperparameter loop. In the inner training loop, the training/validation data is repeatedly split into different training and validation partitions. The model is trained on the current training partition using the current hyperparameter combination, then scored on the current validation partition. The scores from each inner training loop cycle are aggregated in the hyperparameter loop to compute a composite score for the current hyperparameter combination. The hyperparameter combination with the best composite score is selected in the outer loop. The training and validation data in the outer loop is then used to train the model with the “best” hyperparameter combination, the model is scored against the test split, and finally an aggregate score for all test splits is reported.

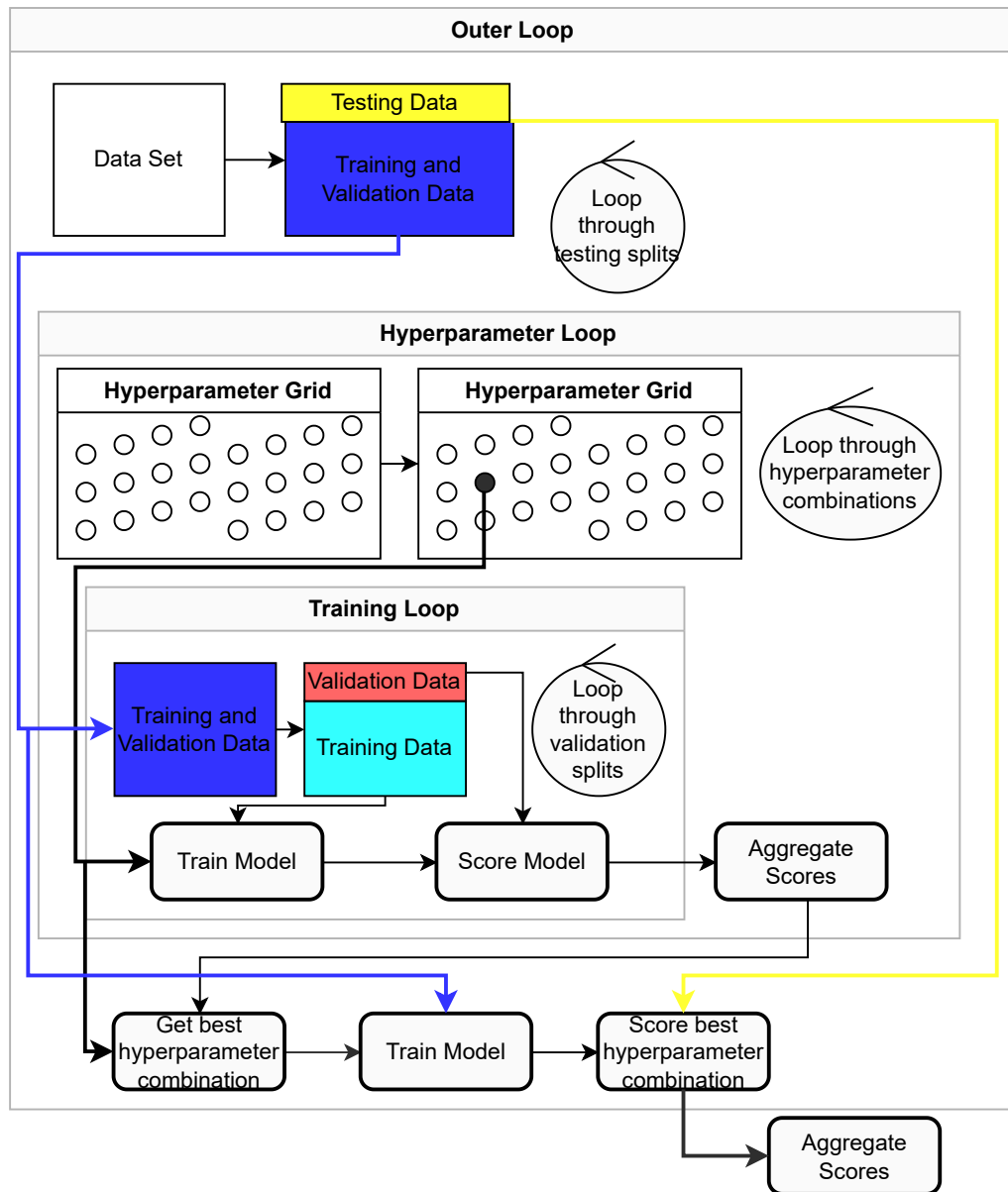


Figure E.2.: Diagram illustrating the nested cross-validation process.

Appendix F. Backpropagation

Consider a simplified version of Eq. (5.7):

$$\hat{\mathbf{y}} = f_2(\mathbf{A}_2 f_1(\mathbf{A}_1 \mathbf{x} + b_1) + b_2) \quad (\text{F.1})$$

We may wish to find $\mathbf{A}_1, \mathbf{A}_2, b_1, b_2$ to minimize some loss function

$$C = L(\mathbf{y} - \hat{\mathbf{y}}) \quad (\text{F.2})$$

where \mathbf{y} is a vector of labels. We can represent Eq. (F.2) as a table as shown in Table F.1. In this way, the expression is decomposed into a list of operations which are composed of the results of other operations. To perform backpropagation, an ancillary table (Table F.2) is constructed where each row represents a derivative of a term from Table F.2 with respect to one of its operands. The chain rule from calculus recursively defines the derivative of x_i with respect to x_j as

$$\frac{\partial x_i}{\partial x_j} = \sum_{k \in s_i} \frac{\partial x_i}{\partial x_k} \frac{\partial x_k}{\partial x_j} \quad (\text{F.3})$$

where the set s_i contains the indices of terms which depend on x_i . We can use the chain rule to populate an additional table (Table F.3) from the terms in Table F.2. This table then contains all derivatives of C with respect to each term of its terms. By performing the computations in this fashion, the number of calculation steps required to compute arbitrary derivatives is minimized.

Table F.1.: Representation of Eq. (F.2) as a table.

Identifier	Expression	Operation	Major Source	Minor Source
x_1	$L(\mathbf{y} - f_2(\mathbf{A}_2 f_1(\mathbf{A}_1 \mathbf{x} + b_1) + b_2))$	L	x_2	-
x_2	$\mathbf{y} - f_2(\mathbf{A}_2 f_1(\mathbf{A}_1 \mathbf{x} + b_1) + b_2)$	difference	x_3	x_4
x_3	\mathbf{y}	given	-	-
x_4	$f_2(\mathbf{A}_2 f_1(\mathbf{A}_1 \mathbf{x} + b_1) + b_2)$	f_2	x_5	-
x_5	$\mathbf{A}_2 f_1(\mathbf{A}_1 \mathbf{x} + b_1) + b_2$	sum	x_6	x_{14}
x_6	$\mathbf{A}_2 f_1(\mathbf{A}_1 \mathbf{x} + b_1)$	matmul	x_7	x_8
x_7	\mathbf{A}_2	parameter	-	-
x_8	$f_1(\mathbf{A}_1 \mathbf{x} + b_1)$	f_1	x_9	-
x_9	$\mathbf{A}_1 \mathbf{x} + b_1$	sum	x_{10}	x_{13}
x_{10}	$\mathbf{A}_1 \mathbf{x}$	matmul	x_{11}	x_{12}
x_{11}	\mathbf{A}_1	parameter	-	-
x_{12}	\mathbf{x}	given	-	-
x_{13}	b_1	parameter	-	-
x_{14}	b_2	parameter	-	-

Table F.2.: Derivatives of Table F.1.

Identifier	Operation	Expression
$x_{1,2}$	$\frac{\partial x_1}{\partial x_2}$	$x_2 \rightarrow L'(x_2)$
$x_{2,3}$	$\frac{\partial x_2}{\partial x_3}$	$x_3, x_4 \rightarrow 1$
$x_{2,4}$	$\frac{\partial x_2}{\partial x_4}$	$x_3, x_4 \rightarrow -1$
$x_{4,5}$	$\frac{\partial x_4}{\partial x_5}$	$x_5 \rightarrow f_2'(x_5)$
$x_{5,6}$	$\frac{\partial x_5}{\partial x_6}$	$x_6, x_{14} \rightarrow 1$
$x_{5,14}$	$\frac{\partial x_5}{\partial x_{14}}$	$x_6, x_{14} \rightarrow 1$
$x_{6,7}$	$\frac{\partial x_6}{\partial x_7}$	$x_7, x_8 \rightarrow x_8$
$x_{6,8}$	$\frac{\partial x_6}{\partial x_8}$	$x_7, x_8 \rightarrow x_7$
$x_{8,9}$	$\frac{\partial x_8}{\partial x_9}$	$x_9 \rightarrow f_1'(x_9)$
$x_{9,10}$	$\frac{\partial x_9}{\partial x_{10}}$	$x_{10}, x_{13} \rightarrow 1$
$x_{9,13}$	$\frac{\partial x_9}{\partial x_{13}}$	$x_{10}, x_{13} \rightarrow 1$
$x_{10,11}$	$\frac{\partial x_{10}}{\partial x_{11}}$	$x_{11}, x_{12} \rightarrow x_{12}$
$x_{10,12}$	$\frac{\partial x_{10}}{\partial x_{12}}$	$x_{11}, x_{12} \rightarrow x_{11}$

Table F.3.: Representation of the chain rule applied to Eq. (F.2) using terms defined in Table F.2.

Identifier	Symbol	Operation
$x_{1,3}$	$\frac{\partial C}{\partial \mathbf{y}}$	$x_{1,2}x_{2,3}$
$x_{1,4}$	$\frac{\partial C}{\partial \hat{\mathbf{y}}}$	$x_{1,2}x_{2,3}$
$x_{1,5}$	-	$x_{1,4}x_{4,5}$
$x_{1,6}$	-	$x_{1,5}x_{5,6}$
$x_{1,7}$	$\frac{\partial C}{\partial A_2}$	$x_{1,6}x_{6,7}$
$x_{1,8}$	-	$x_{1,6}x_{6,8}$
$x_{1,9}$	-	$x_{1,8}x_{8,9}$
$x_{1,10}$	-	$x_{1,9}x_{9,10}$
$x_{1,11}$	$\frac{\partial C}{\partial \mathbf{A}_1}$	$x_{1,10}x_{10,11}$
$x_{1,12}$	$\frac{\partial C}{\partial \mathbf{x}}$	$x_{1,10}x_{10,12}$
$x_{1,13}$	$\frac{\partial C}{\partial b_1}$	$x_{1,9}x_{9,13}$
$x_{1,14}$	$\frac{\partial C}{\partial b_2}$	$x_{1,5}x_{5,14}$