IN-SENSOR THIN-FILM MAGNETICS WITH RESERVOIR COMPUTING

Matteo Pearce
Langwith College
University of York

May 2024

4^{th} Year Project Final Report for degree of MEng in Electronic Engineering with Nanotechnology

Contents

1 Abstract

Random search exploration in Python of system parameters for thin ferromagnetic films (Co, Fe, Ni), simulated with the Vampire atomistic simulator and used as reservoirs according to the Reservoir Computing machine learning paradigm. Properties such as Gilbert magnetic damping and parameters such as input scaling, film dimensions and readout discretisation are trialled in different combinations to assess the effects on NRMSE in predicting the NARMA10 algorithm, injected into the film as a z-component magnetic field intensity. An appraisal of the compatibility of this technology with biomagnetic signals as a feasibility study into the application of such films for in-situ biosensors. Exploration of novel implementations such as body-temperature test conditions, equal input scaling and uniform signal injection are trialled as cornerstone concepts for the suggested use-case, with the latter constituting a significant roadblock. Film dimensions emerge as a clear strategy for mitigating errors, and pico-Tesla signals display good compatibility with the ferromagnetic mediums.

2 Acknowledgements

I would like to thank my pastoral supervisor Steven Johnson for his support, both academic and moral. His unbridled enthusiasm and love for the craft has fuelled my appreciation for my chosen path since day dot.

I would like to thank Martin Trefzer and Tian Gan for assisting me in this project. Without their invaluable input, enthusiasm, immeasurable patience and frequent reassurance, this project would not have been possible and I would have descended into madness.

The Viking cluster was used during this project, which is a high performance computer facility provided by the University of York. I are grateful for computational support from the University of York, IT Services and the Research IT team.

3 Introduction

Technological progress assumes many guises, each professing to belong to the realm of the unexplored, whereby progress and novelty are implied to be inextricably linked. Yet the reality is that the true face of progress is relentless optimisation, exhausting all possible tricks and quirks to perfect established methods. It is only when met with the brick wall of a technology's theoretical limit that we are forced to re-invent the wheel. For years Moore's law has remained a steadfast heuristic, however progress through miniaturisation is fast reaching unbreakable physical limits, setting the scene for the next bout of desperate innovation. Computational intelligence is rapidly becoming a technological pillar of the modern world, but poses problems of complexity and power consumption that cannot be overcome purely by reducing the dimensions of its constituent parts. Among the myriad branches of machine learning research, some avenues are exploring unconventional computing, defining a paradigm that is not based on the Von Neumann Architecture, wherein data filters through the system, modulated by its inherent structures and properties. This type of approach reduces the operational complexity and energy requirements of the computational system, by virtue of its innate passivity and suitability to task-specific optimisation. The pursuit of topological sophistication could prove to be a winning strategy when the already waning benefits of down-scaling transistors are exhausted.

3.1 Reservoir Computing

The premise of machine learning is built upon the imitation of our own neural structures. Configuring our electronic building blocks to emulate the behaviour of our neurons and synapses has proven to be a very effective tool and has found applications in all areas of society. What we have so far struggled to replicate is the power-efficiency of our own biology. Through observations of how the different sensory inputs to our brain marry up to form concise recognition of phenomena, a neural architecture emerged comprised of a passive filter and an active output stage. It was stipulated that if the dynamics of the filter are sufficiently complex and non-linear, that a simple linear learning rule is sufficient for training the output to infer the inputs to the system. This principle forms the basis of reservoir computing. There is no requirement for the reservoir to be comprised of physical or metaphysical neurons and in fact any physical substrate is eligible if the captured dynamics are sufficiently complex. By embedding computation in the physics of a substrate, it becomes clear that there is potential for extremely low power and lightning fast processing.

3.2 Nanomagnetics

The single most valuable attribute of a computer is its speed and today we enjoy processing power capable of billions of operations per second. In reservoir computing, the speed of computation is dictated by the dynamical properties of the reservoir and their response-times to external perturbation. Physical reservoirs embed computation in the causal wave of interactions that stem from disturbing the stability of their current state, such as dropping an object into water and observing the ripples. Magnetic nanostructures can be used in this way by measuring variations in the state of magnetic polarisation across different regions, as caused by fluctuating external magnetic fields. This mechanism hinges on the exchange interactions between electronic spins of adjacent atoms, boasting atomic dimensions and femtosecond dynamics. These structures therefore lend themselves to lightning fast, extremely low power computing at the nanoscale.

3.3 Magnetism in Biology

Many of the basic mechanisms that underpin cellular metabolism rely on the flux of ions. This movement of charged particles creates fluctuating magnetic fields millions of times weaker than the magnetic field of the earth, which is itself only a mere handful of micro-Tesla. It is only relatively recently that we are capable of directly measuring some of these fields, for example those emanating from the polarisation currents in the axons of the neurons in our brains. This requires state of the art machines relying on incredibly advanced technologies that operate superconductors close to absolute zero. The difficulty arises from the faintness of these signals, exacerbated by the cumulative interference of each field emanating from on average 86 billion different sources. Current methods rely on deconstructing the measured macroscopic landscape of magnetic emanation, inferring by complex retrospection the probable sources and their contribution. We are adept at measuring electric phenomena in the body, both at the organ and cellular level, and subsequently, magnetic considerations can be inferred. Directly capturing these faint magnetic signals is challenging and is an exploit that is either limited to research or to complex diagnostic instrumentation.

3.4 Hypothesis and Objectives

This paper explores the feasibility of using thin magnetic films as reservoirs in the context of physiologically produced magnetic fields. The research question is therefore:

Research Question: Can a single layer of perceptrons be trained with the time-varying magnetic polarisations of arbitrarily defined regions of a nanomagnetic substrate and reliably infer the input fields that caused them?

There are many challenges associated with the suggested use case of this fledgling technology, and it is the exploration of techniques for overcoming these that forms the basis of this feasibility study. These challenges arise from biology itself as well as forced deviations from Reservoir Computing canon inherent to a realistic implementation. The main objectives are:

Objective 1: Assess the feasibility of reducing the prediction inaccuracies that arise from thermal noise at 309.65K through design choices such as film material and dimensions.

Objective 2: Assess whether applying equal input to all cells in the film compromises its complex dynamics, as this is representative of a real-world scenario where the entire film would be exposed to the same magnetic field.

Objective 3: Assess whether equal input scaling to all cells in the film compromises its complex dynamics, as this is representative of a more realistic scenario where the film would be coated with an attenuating or amplifying layer, applied homogeneously to the entire film.

Objective 4: Assess the medium's compatibility with fields of the magnitude produced by physiological processes.

4 Background

4.1 Machine Learning

Machine learning is a subset of artificial intelligence that deploys an algorithmic approach to enhance performance on specific tasks by learning from data, without the need to explicitly program desired functionality. When correctly implemented, self-tuning algorithms have proved extremely capable at notoriously difficult tasks such as pattern recognition, trend prediction and complex decision making [1]. The process of training a model for a particular task is called fitting. There is an implicit trade-off between bias and variance when a model is fitted to a dataset. This is inextricably linked to model complexity, whereby more complex models have greater variance and lower bias. This translates into a greater capacity for generalisation, whereas less complex models with higher bias and lower variance have a higher propensity to over-fit data [2]. Designing such systems therefore requires careful consideration and falls within the remit of an optimisation problem. Explaining the entirety of the subject would warrant an entire book, so I will focus on the topics most pertinent to the Reservoir Computing paradigm.

4.1.1 The Perceptron

Neural Networks encompass a variety of techniques and algorithms, each suited to different types of data and tasks. The key difference with conventional computer architectures is the shift from serial operation to distributed parallelism. They are inspired by biology and were first developed to try and imitate the processing centres of the human brain. The brain's biological building blocks are neurons, which communicate with each other by via cascades of polarity inversions. These signals are mediated electrochemically by synapses and transmitted via dendrites to the cell, affecting the voltage of the cell wall. If this voltage is large enough, the neuron "fires" and sends its own polarity inversion wave down its output channel, the axon [3]. We have sought to emulate this mechanism, with configurable electronic units linked in large neural networks. Although many architectures exist, the simplified behaviour of a neuron with multiple connections can be summarised by a Perceptron-style model, shown in figure 1. It is important to note that integrate-and-fire architectures more closely emulate biological neurons, but are more complex. For our basic Perceptron, input signals are scaled by a connection weight, summed and fed to an activation function [4]. In the simplest models the activation function is a simple comparison to a threshold, outputting a logic 1 if exceeded and a 0 if not. Generally, the activation function is a non-linear equation such as the hyperbolic tangent and

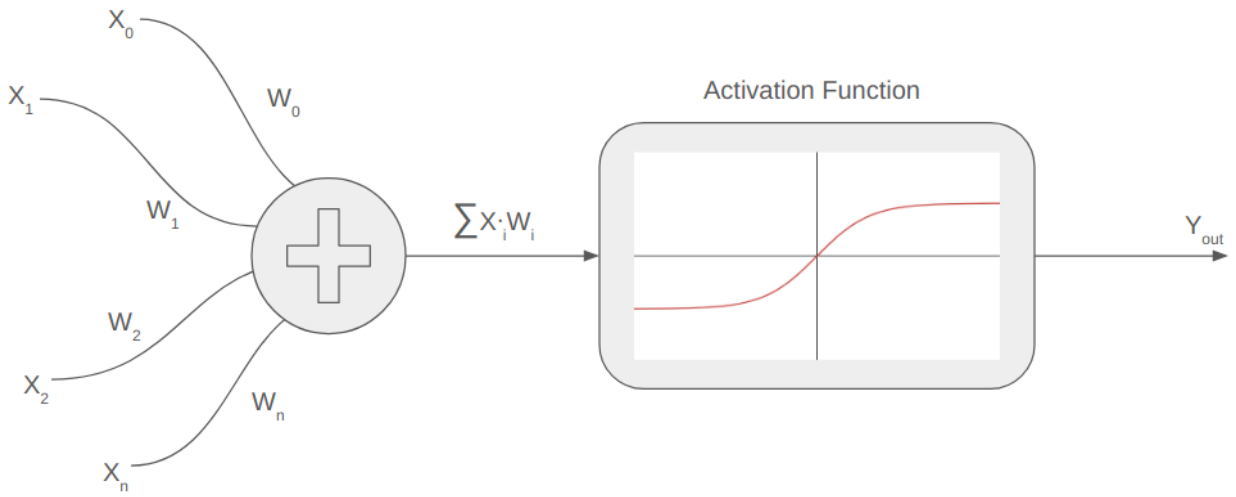


Figure 1: A Perceptron-based neuron. The inputs are multiplied by their respective connection weights and then summed together. This value is then the argument of the activation function, which is typically chosed from Tanh, Sigmoid or ReLu.

the output is a real number [5]. The mutable components of this assembly are the weights of the input connections. By passing signals through a network of Perceptrons and comparing the results to a desired output, a learning rule is applied to tweak the weights of the connections, gradually iterating with each new time-step towards a smaller discrepancy with the training data.

4.1.2 Supervised Learning

Supervised learning is one of the most widely used techniques in machine learning and the most intuitive to comprehend. A model is trained using labelled data consisting of input-output pairs, representing a desired transformation of the input to produce the output. This allows the algorithm to learn the relationship between input and output labels, enabling it to make predictions on new, unseen data [6]. There are two types of tasks that can be tackled with a supervised learning approach:

- *Regression*: prediction of numerical values. An example is predicting house prices based on features such as size, number of bedrooms and location. The algorithm learns to predicts its price from these input features. To evaluate the performance of a regression model the most common metrics are Mean Squared Error (MSE), Root Mean Squared Error (RMSE) and R-squared (coefficient of determination) [7].

- *Classification*: categorisation of data. An example is email spam detection based on fea-

tures extracted from an email such as sender, subject and content. the algorithm learns to predict whether the email is spam from these input features. To evaluate the performance of a classification model the most common metrics are accuracy, precision, recall, F1-score and area under the Receiver Operating Characteristic curve (ROC) [8].

Many of the most common supervised learning algorithms, such as Decision Trees, Random Forests, Support Vector Machines (SVM), k-Nearest Neighbours (kNN) and Neural Networks are capable of both regression and classification tasks provided proper setup.

4.1.3 Learning Rules

Focusing on Regression algorithms, the first step to implementing a learning rule is to quantify the error. The error of a prediction can be simply defined as the difference between observed output y and desired output d, for an individual unit k in a layer of K perceptrons with a single training sample n:

$$\epsilon_k(n) = d_k(n) - y_k(n) \tag{1}$$

The observed output is obtained after the inputs to the perceptron have been scaled by the connection weights, summed and passed through the activation function. for a perceptron with J connections, activation function ϕ and denoting inputs as v, the observed output is given by:

$$y_k(n) = \phi\left(\sum_{j=1}^{J} \omega_{kj} \cdot v_j\right) \tag{2}$$

One of the most common learning rules is the Least Squares (LS) method, which defines a cost or loss function as the sum of the square of all the individual error terms [9]. A simple explanation of the why we take the square of the error is that it allows us to use differentiation as part of the toolkit for solving. Simply having a linear term entails variables vanishing upon partial differentiation. The cost function for the layer of K perceptrons is:

$$E(n) = \sum_{k=1}^{K} \epsilon_k^2 \tag{3}$$

The aim is to reduce $E(n)$ with every subsequent training sample n by adjusting the perceptron input weights. There are many optimisation techniques for solving this, with the most common method being gradient descent. This technique is used to minimise the cost function by iteratively moving in the direction of the steepest descent of the function's gradient [10]. Often, a learning rate scaling term is included to stop the weights changing too quickly. The adjustment for the j^{th} connection weight in perceptron k is given by:

$$\Delta\omega_{kj} = -\eta \cdot \frac{\partial E(n)}{\partial \omega_{kj}} \tag{4}$$

This equation can fairly simply be solved with the chain rule for differentiation, yielding:

$$\Delta\omega_{kj} = \eta \cdot \epsilon_k(n) \cdot \phi^{'} \left(\sum_{j=1}^{J} \omega_{kj} \cdot v_j \right) \cdot v_j \tag{5}$$

A well known limitation of the LS estimator for regression is the negative impact of collinearity. Ridge Regression (RR) on the other hand, provides a means of addressing this problem without removing variables from the original set of independent variables [11]. It involves the introduction of some bias into the regression equation to reduce the variance of the weights. The bias is introduced by the use of a regularisation parameter $\alpha \geq 0$ which controls the extent to which the ridge estimates differ from the least square estimate [12]. The overall effect is that it reduces over-fitting, especially where collinearity is present. The cost function for RR can be defined as a combination of a least squares term and a regularisation term:

$$E(n) = \sum_{k=1}^{K} \epsilon_k^2 \quad + \quad \alpha \cdot \sum_{k=1}^{K} \sum_{j=1}^{J} \omega_{kj}^2 \tag{6}$$

Similarly to (3), the weights can be found with gradient descent as well as many other optimisation algorithms. The regularisation term is defined as the squared euclidean norm of the weights, scaled by the regularisation parameter. From this formalisation it is clear that LS is a special case of RR where $\alpha = 0$. Using gradient descent as formalised in 4, the adjustment for the jth connection weight in perceptron k is given by:

$$\Delta\omega_{kj} = \eta \cdot \left[\epsilon_k(n) \cdot \phi' \left(\sum_{j=1}^{J} \omega_{kj} \cdot v_j\right) \cdot v_j - 2\alpha \cdot \omega_{kj}\right] \tag{7}$$

The inclusion of the $2\alpha \cdot \omega_{kj}$ term penalises large weights by reducing them proportionally to their size. The choice of α is an optimisation problem in its own right and can be determined heuristically with methods like cross-validation, that trials multiple values and selects the best one for the particular task.

4.1.4 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are a perceptron based architecture that excels at preserving the temporal dependencies of the input data, making it well suited to complex time-varying tasks such as speech-recognition, time-series prediction and natural language processing [13]. This is achieved by deviating from typical neuron inter-connectivity schemes that tend to be unidirectional. By adding feedback lines to neurons in previous layers, the processing of a datum can span several time-steps, generating in effect an artificial memory [14]. They are particularly apt at capturing long-term trends in addition to capabilities beyond pure prediction, such as capturing subtle changes in continuous cycles and detecting anomalies. Examples of this are inferring the state of operation of a human brain from EEG scans [15], and unearthing

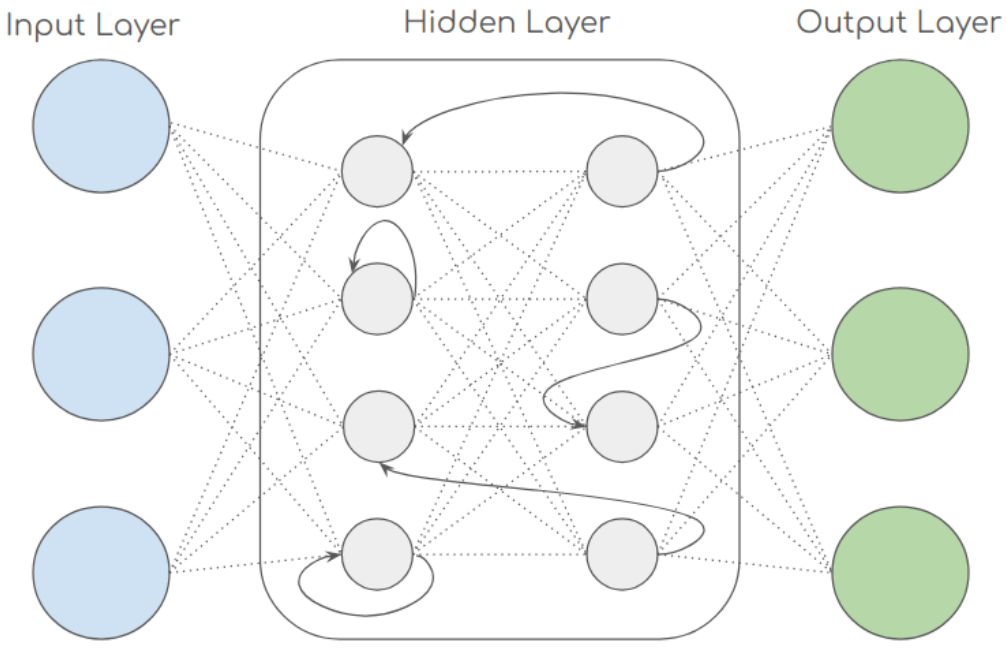


Figure 2: Recurrent Neural Network diagram, flowing from left to right. Feedback lines are created within the hidden layer, delaying signal propagation and unearthing the data's temporal dependencies.

previously undetected operational anomalies from aircraft in-flight data [16]. RNNs fall under the category of Artificial Neural Networks [17], the most well known being the Feed-Forward architecture. Despite the state of the hidden layer being largely unknown to the designer, the weights are still adjusted by a complex process called back-propagation. The weights of each perceptron's connections are adjusted as a function of the layer preceding it, starting with the output layer and propagating in opposition to the usual direction of processing [18]. The feedback lines in RNNs make this a particularly complex process as the the time-delay between different signal pathways during back-propagation can cause gradients to either grow exponentially large (exploding gradients) or decay exponentially small (vanishing gradients) [19]. Many convoluted solutions to this problem exist, yet it was the concurrent discovery of two analogous simple resolutions that gave rise to the field of Reservoir Computing.

4.2 Reservoir Computing

The origins of Reservoir Computing (RC) took inspiration from the observed relationship between the cortical and sub-cortical parts of the brain. It was discovered that structurally similar corticostriatal circuits within the brain were responsible for both spatio-temporal tasks as well as higher cognitive function and language [20]. A model based on these finding was proposed by Dominey et al., comprised of neurons in the prefrontal cortex with stable recurrent connections and adjustable connections from prefrontal cortex neurons to those in the striatum [21]. The first computational models based on this concept were devised almost concurrently: the Echo State Network (ESN) proposed by Jaeger in 2001 [22], and the Liquid State Machine (LSM) proposed by Maass in 2002 [23]. The basic principle in both models is to separate the model into two distinct phases, a randomly connected and recurrent passive filter stage connected via adjustable weighted connections to an output layer. In LSMs, the filter stage is comprised of a vast number of spiking neurons, a network of sufficient size and complexity to possess a fading memory, encapsulated in the pointwise-separation property. The ESN utilised perceptron-based neurons with sigmoidal or hyperbolic tangent activation functions and similarly feature a fading memory, called the Echo State Property (ESP). It is clear that barring slight differences in setup the two models are both implementations of the same foundational concepts, and as such they were unified by Verstraeten into a single framework: Reservoir Computing [24]. It is a simple and cost-effective paradigm of RNN that requires no back-propagation, relying on the complex dynamics of the filter stage (the reservoir), and a trained single neuron output layer (the readout).

4.2.1 Reservoir Computing Basics

Like the RNN on which it is based, RC excels at capturing the temporal dependencies present within data, which it achieves despite using a simple linear regression algorithm to train the readout. These qualities are direct consequences of the the two fundamental properties that a reservoir must possess:

- *Fading Memory*: The recurrent connections within the reservoir dilates the processing time of inputs beyond the time-step that separates them. This entails that the processing of any input datum is inextricably linked to the history of data that preceded it. This effect can be amplified by artificially introducing leakage into the state equation of the reservoir nodes [25].

- *High Dimensionality*: it is the inherent complexity and non-linearity of the reservoir that allows for patterns to be extracted with a simple linear regression. The transformation of mapping linearly inseparable features into a higher-dimensional space makes them linearly separable at the readout [26].

Patterns within the input stream are learned from the complex dynamics of the reservoir. As mentioned, the reservoir itself is random and immutable: only the output weights are configurable in response to training. Much like the corticostriatal circuits in our brains, the RC architecture is very powerful at multitasking, as separate clusters of readout neurons could be trained to infer different input data dynamics [27, 28].

A typical RC setup contains three layers: input, reservoir and readout. The passage of inputs from the input layer to the reservoir is moderated by a fixed and random input weight matrix $\boldsymbol{W_{in}}$ of size $N \times i$, where i denotes the number of input neurons and N the number of neurons in the reservoir. It is worth noting that not all inputs need necessarily be connected to every neuron in the reservoir, as values in $\boldsymbol{W_{in}}$ can be 0. The reservoir matrix $\boldsymbol{W_{res}}$ is square of shape $N \times N$ and the trainable output weights matrix $\boldsymbol{W_{out}}$ to j output neurons is of size $j \times N$, as shown in Figure 3. Denoting a time-varying input u at sample time n, the internal states $\boldsymbol{x_{res}}$ updates are given by:

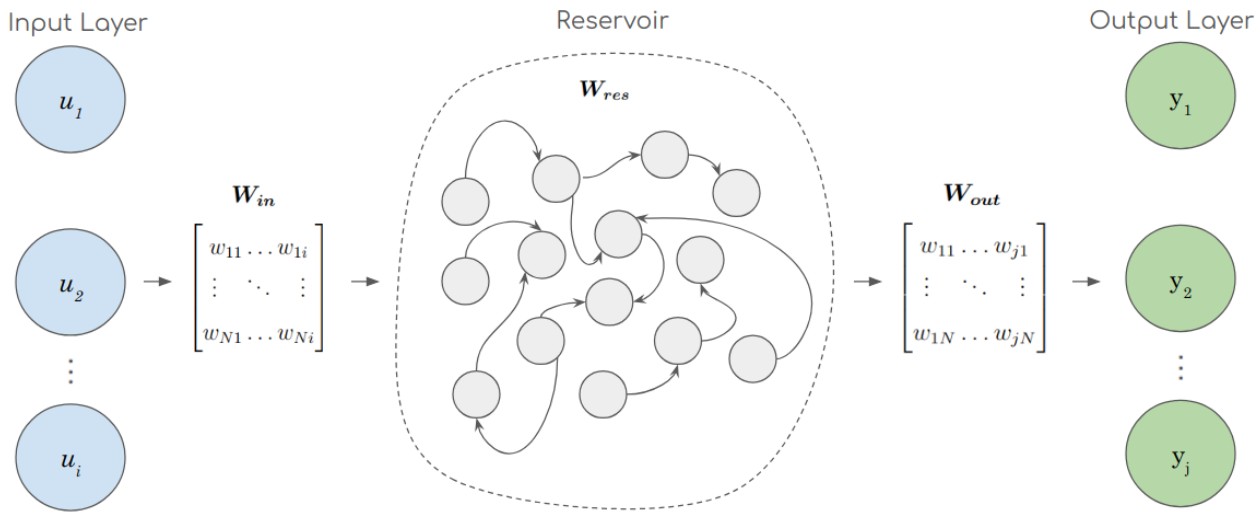


Figure 3: Example of RC architecture. The reservoir is a black-box dynamical system with random connectivity. The broad definitions of what can constitute a reservoir imply that any physical substrate with sufficient non-linear complexity and fading memory is permissible within the framework.

$$\boldsymbol{x_{res}}(n) = \phi\left(\alpha \cdot \boldsymbol{W_{in}}(u(n)) + \beta \cdot \boldsymbol{W_{res}} \cdot \boldsymbol{x_{res}}(n-1)\right) \tag{8}$$

where ϕ is the neuron activation function, which is typically sigmoidal or the hyperbolic tangent. α and β are scaling parameters which for controlling the relationships of contribution to the new state of the new input versus the previous state. For the first input datum, $\boldsymbol{x_{res}}(n-1)$ is simply the randomly initialised internal states. Training often includes a washout period, where an arbitrary number of initial values are discarded such that the memory of the initial state is "forgotten" and does not influence the regression training [29]. ESN implementations typically feature artificial leakage mediated by an extra term called the leakage rate. Denoting the leakage rate as l, the state update equation becomes:

$$\boldsymbol{x_{res}}(n) = l \cdot \phi\left(\alpha \cdot \boldsymbol{W_{in}}(u(n)) + \beta \cdot \boldsymbol{W_{res}} \cdot \boldsymbol{x_{res}}(n-1)\right) + (1-l) \cdot \boldsymbol{x_{res}}(n-1) \tag{9}$$

Where equation (8) is a special case when $l = 1$. Denoting the output as y, the output for a single training sample n is simply:

$$y(n) = \boldsymbol{W_{out}} \cdot \boldsymbol{x_{res}}(n) \tag{10}$$

4.2.2 Training

To determine $\boldsymbol{W_{out}}$ linear regression as stated in section 4.1.3 on learning rules, is sufficient. There are two main distinctions for training the output layer based on using each datapoint sequentially or a set extruded *a priori*. Online training is a dynamic approach for refining the output weights of a reservoir model. It utilises iterative algorithms like Recursive Least Squares (RLS) or Adaptive Moment Estimation (ADAM) to adjust readout weights continuously during training. This method allows the reservoir model to adapt to input signal variations and enhance predictions in real-time. This approach has the added benefit of having low memory requirements as it only needs to save one snapshot of the reservoir states in memory at any given time. However, online learning is susceptible to overfitting due to washout, particularly with longer datasets. Offline learning is a more accurate alternative if you can handle the extra computational overhead [30]. The entire dataset is fed through the reservoir, the reservoir states are collated into a vector of matrices, of size $dim(u) \times (N \times N)$. An output matrix is constructed from the readout's output values for every timestep, of size $j \times dim(u)$. Calling $\boldsymbol{X_{res}}$ the vector of reservoir state matrices and $\boldsymbol{Y_{out}}$ the readout value matrix, the linear relationship between inputs and outputs is:

$$\begin{gathered} \boldsymbol{Y_{out}} = \boldsymbol{W_{out}} \cdot \boldsymbol{X_{res}} \\ \boldsymbol{Y_{out}} \in \Re^{\,j \times dim(u)}, \boldsymbol{W_{out}} \in \Re^{\,j \times N}, \boldsymbol{X_{res}} \in \Re^{\,dim(u) \times (N \times N)} \end{gathered} \tag{11}$$

Applying LS to 11 in vector notation yields:

$$\boldsymbol{W_{out}} = \boldsymbol{Y_{out}} \cdot \boldsymbol{X_{res}}^{T} \cdot (\boldsymbol{X_{res}} \cdot \boldsymbol{X_{res}}^{T})^{-1} \tag{12}$$

In section 4.1.3 on learning rules, RR was presented as replacement for LS to mitigate the effects of collinearity. This occurs when columns can be expressed as linear combinations of the other columns, making the matrix rank-deficient and non-invertible [31]. Furthermore, a matrix can be non-invertible if there does not exist a unique solution to the system of equations. As there is clearly an inversion operation in (12), Lukoševičius proposes RR as a better alternative [30]. In vector notation, RR as applied to (11) is given by:

$$\boldsymbol{W_{out}} = \boldsymbol{Y_{out}} \cdot \boldsymbol{X_{res}}^T \cdot (\boldsymbol{X_{res}} \cdot \boldsymbol{X_{res}}^T + \boldsymbol{I}\alpha)^{-1} \tag{13}$$

Where α is the regularisation parameter, as in (6).

4.2.3 Physical Reservoirs

There are no requirements in RC for the reservoir to be a neuron-based system, so long as the high-dimensionality and fading memory are preserved. This naturally lead some to investigate the feasibility of using a physical substrate with surprising success. In a brilliant tour de force of the concept, Fernando and Sojakka proved that a bucket of water could be perturbed with simple movements to elicit complex ripple dynamics, which were captured by a camera and fed to a readout layer that was successfully trained to infer the input movements [32]. They were able to train their system to solve the XOR benchmark problem, which is notably unsolvable by a single layer of perceptrons [33].

4.2.4 Thin Magnetic Films

A whole host of physical mediums have been proposed as suitable candidates for RC [34], yet a clear front-runner is the broad category of nanomagnetic devices as their inherent qualities lend themselves to low-power, compact and fast computing. Spatially distributed arrays of interacting nanomagnetic regions provide a framework for defining multiple input and output dimensions, constituting a system whose dimensionality is equal to the number of N structures. Allwood et al. assess the dynamical response of magnetic reservoirs that exploit the damped, oscillatory motion of magnetic moments according to the Laudau-Lifshitz-Gilbert equation of motion. With frequencies in the Megahertz to Terahertz region and settling times in the order of nanoseconds, ferromagnetic materials are not only well suited to high-speed applications, but naturally satisfy the ESP. In summary, they can be treated as "non-linear activation functions with short-term temporal dependencies" [36]. The validity of this appraisal has been corroborated by simulation work by Dale et al. In the study, a thin magnetic ferromagnetic film divided into macro-regions was simulated in software as the reservoir for an RC implementation [35]. The group demonstrated that simulated thin-films of copper, iron & cobalt of various area and thickness can be used successfully to predict a triumvirate of complex industry standard prediction tasks: Santa Fe Laser, NARMA-10 and NARMA-30. The study found that the materials were competitive against neural networks and typically outperformed them when the reservoir

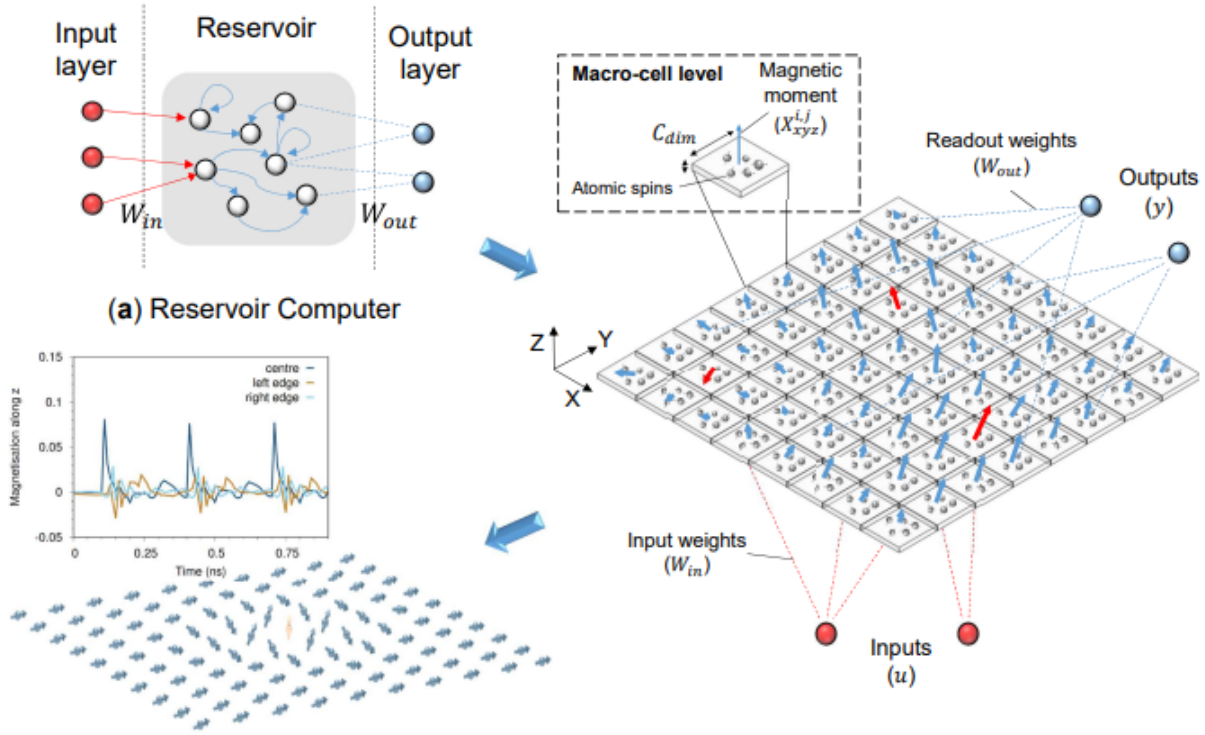


Figure 4: Simulation setup as described in "Reservoir Computing with Thin Magnetic Films" [35]. The thin-film is divided into macro-cells whose state parameter is the magnetic polarisation vector, as a function of x,y,z field components.

size was small. Figure 4 shows a visual representation of the thin-film architecture, its division into macro-cells and resulting array of magnetic polarisations that will serve as the reservoir output.

4.3 The Micromagnetic Model

The micromagnetic model is a theoretical framework used to describe and simulate the behaviour of magnetic materials. It provides insight into the complex interactions between individual magnetic moments (spins) within a material and how these interactions give rise to macroscopic magnetic properties. Micromagnetism is essentially a continuum approximation which allows the calculation of magnetisation structures and magnetisation reversal assuming the magnetisation to be a continuous function of position, and deriving relevant expressions for the important contribution arising from the exchange, magnetostatic and anisotropy energies [37].

4.3.1 The Ising Model

Ernst Ising developed an eponymously named model in the 1920s as a mathematical framework for studying ferromagnetism from a perspective of phase transitions. The model seeks to

describe how small changes in the fundamental parameters governing small scale interactions within a larger complex generate a butterfly effect that plunges the entire system into a new state [38]. These are the fundamental steps to follow to represent a magnetic medium according to the Ising model:

1. The magnetic material is represented as a lattice of discrete magnetic moments - the spins. Each spin can be in one of two states: up or down, corresponding to positive and negative magnetisation directions.

2. Assumptions are made about the overall contributions of magnetic energy to the system. In an idealised formalisation, the only contributors are the nearest neighbour interactions, the energy exchanged between spins located at neighbouring lattice sites, and an external magnetic field.

3. Under these assumptions, the system Hamiltonian can be defined, an equation describing the energy landscape from which other important phenomenological properties such as the Curie temperature can be extrapolated.

Nearest neighbour interactions imply the exchange of energies between literal atomic neighbours, the number of which is dictated by the crystal structure. Different materials crystallise with very specific lattices. Table 4.3.1 contains the most common crystal structures and how many neighbours each atom has.

Lattice name	abbreviation	nearest neighbours
Simple Cubic	SC	6
Body-Centred Cubic	BCC	8
Face-Centred Cubic	FCC	12
Hexagonal Close-Packed	HCP	12
Body-Centered Tetragonal	BCT	8

Table 1: Common crystal structures with number of nearest neighbours [39].

Although a powerful tool for modelling micromagnetism, it is a reductive model for describing isotropic magnetisation and antiferromagnetism. The Heisenberg model developed in the 30s can be seen as a generalisation of the Ising model and provides a more comprehensive description of exchange interactions.

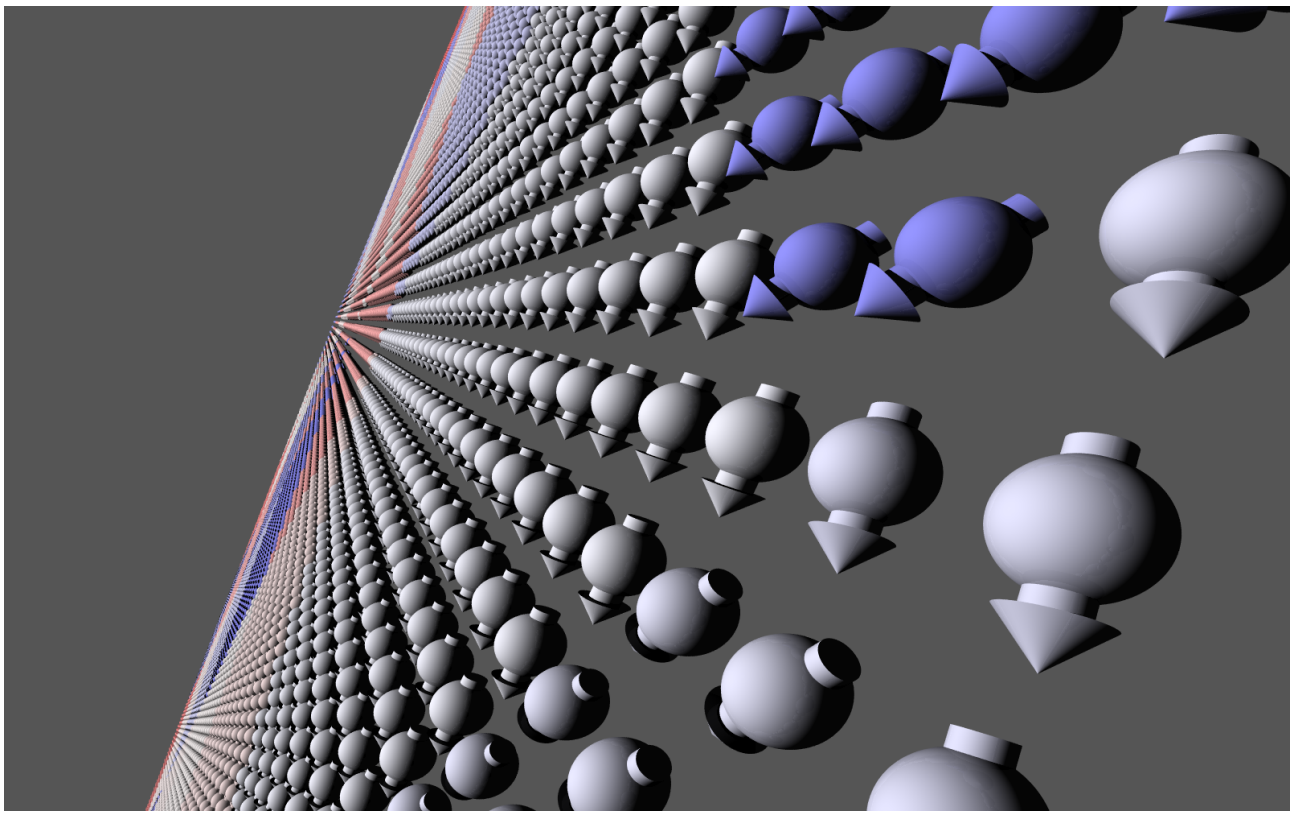


Figure 5: Monolayer of Cobalt atoms simulated in Vampire that uses the Heisenberg model of micromagnetism. Each atom is represented as the direction and magnitude of its overall spin moment.

4.3.2 The Heisenberg Model

The Ising model simplifies the spins to be one-dimensional, with only two possible states neglecting their directionality. In the Heisenberg model developed in the subsequent decade, the magnetic moments are described by three-dimensional vectors and includes quantum mechanical definitions of atomic phenomena [40]. This allows for the consideration of the direction of the spins in addition to their magnitude. Furthermore, the Ising model only considers interactions between nearest neighbours and assumes that spins preferentially align parallel to each other, which precludes any functional notion of antiferromagnetism, characterised by antiparallel spins.

In the Heisenberg model, the exchange interaction arises from the quantum mechanical exchange energy associated with the spatial overlap of electron wave functions between adjacent atoms or ions. When two spins are aligned parallel to each other, the exchange interaction results in a lower energy state, favouring parallel alignment. Many magnetic materials exhibit anisotropy, meaning that the orientation of spins is favoured along specific directions in space. This preference arises from the local crystal environment at the atomic level, leading to what is known as magnetocrystalline anisotropy. The prevalent form of anisotropy in materials is

typically uniaxial, wherein spins align preferentially along a single axis termed the "easy axis" [41]. Crystal structures are seldom uniform and can present regions where the crystal structure changes abruptly, which can be due simply to heterogeneity in the material composition or defects in the structure itself. These separate regions are called grains and the interaction between the magnetic moments of separate grains is called the intergranular exchange [42].

The spin moment Hamiltonian describes the energy landscape of the fundamental spin-dependent interactions at the atomic level. Under the assumption that the effects of potential energy, kinetic energy and electron correlations can be ignored, and that the contributions to energy are given respectively by Heisenberg exchange, uniaxial anisotropy and applied field contributions, the spin Hamiltonian is defined as:

$$\boldsymbol{H} = -\sum_{i<j} J_{ij} \cdot \boldsymbol{S}_i \cdot \boldsymbol{S}_j - K_{ani} \sum_i (\boldsymbol{S}_i \cdot \hat{n})^2 - \mu_B \sum_i \boldsymbol{B}_{ext} \cdot \boldsymbol{S}_i \tag{14}$$

Where J_{ij} is the Heisenberg exchange, $\boldsymbol{S}_i$ is a vector representation of the spin, K_{ani} is the magnetocrystalline anisotropy constant, $\hat{n}$ is the unit vector along the spin easy axis, μ_B is the magnetic moment of an electron called Bohr's Magneton and $\boldsymbol{B}_{ext}$ is the external applied field. The net magnetic field $\boldsymbol{B}_i^{net}$ acting on each spin is given by the first derivative of the spin Hamiltonian with respect to the spin vector:

$$\boldsymbol{B}_i^{net} = \frac{1}{\mu_B} \cdot \frac{\partial \boldsymbol{H}}{\partial \boldsymbol{S}_i} \tag{15}$$

The Landau-Lifshitz-Gilbert (LLG) equation is a fundamental equation in the study of magnetisation dynamics. It describes the time evolution of the spin vector $\boldsymbol{S}_i$ in a magnetic material under the influence of external magnetic fields, magnetic anisotropy, and damping effects [43]. The damping, called Gilbert damping, is understood to be a nonlinear spin relaxation phenomenon and it controls the rate at which magnetisation spins reach equilibrium [44]. Although formulated in the 50s, an *ab initio* formalisation that effectively describes its origin is yet to be found. Several models for predicting Gilbert damping exist, yet none constitute a universally valid characterisation and tend to be phenomenological interpretations. It remains a frequently studied problem to this day, and exploring ways of controlling it in nanomagnetic systems is a key avenue of research [45, 46, 47, 48].

The LLG describing time-variation of $\boldsymbol{S}_i$ is:

$$\frac{\partial \boldsymbol{S}_i}{\partial t} = -\frac{\gamma}{1+\lambda^2}\left[\boldsymbol{S}_i \times \boldsymbol{B}_i^{net} + \lambda \cdot \boldsymbol{S}_i \times \left(\boldsymbol{S}_i \times \boldsymbol{B}_i^{net}\right)\right] \tag{16}$$

where λ is the Gilbert damping and γ is the gyromagnetic ratio defined as the ratio of the magnetic moment to the angular momentum of the particle [49]. Using these equations, a comprehensive understanding of micromagnetism is formed, constituting the basis for simulation of magnetic phenomena at the atomic level.

4.4 Measuring Biomagnetism

Despite electric and therefore magnetic phenomena being suffuse in our physiology, we are surprisingly inept at directly measuring it, but this is largely due to the scale at which they manifest. The magnetic field of the heart is the largest biomagnetic signal generated by the body and is only in the order of 50–100pT [50]. The largest magnetic emanation of the human brain, the alpha rhythm, has a peak-to-peak amplitude of around $2pT$, with other activity being an order of magnitude smaller [51]. Measuring magnetic fields of the brain is an incredibly high-tech and non-trivial endeavour called magnetoencephalography (MEG). MEG instruments consist of an array of superconducting quantum interference devices (SQUIDs) that are highly sensitive to magnetic fields and cooled by liquid Helium to $-271°C$. They are placed in close proximity to the scalp, surrounding the head of the subject and detect the magnetic fields generated by the neuronal currents that penetrate the skull [52]. These devices sample signals in the kilohertz range, offering millisecond scale detail. Similarly, magnetocardiography (MCG) instruments are also based on SQUID technology and have similar sensitivity and sampling rate [53].

5 Methodology

This project is entirely software-based and uses the university of York's in-house software package Vampire, a state-of-the-art atomistic simulator for magnetic nanomaterials developed by Evans et al. [54, 55]. I exploited this to code a random search algorithm in Python which generates an untested configuration of simulation parameters, invokes Vampire, captures the output and uses it to train a perceptron layer with RR. The objective is to find a combination of material and associated physical parameters that can excel as a reservoir at body tempera-

ture with input fields of a magnitude as found in biomagnetism. The principal constraints to this methodology are computing power and time, as larger simulations can take hours or days, and the factorial dependency of combinations can cause the number of permutations to balloon substantially.

Appendix D provides a summary of the main software packages and python libraries used, whereas sections 5.1 and 5.2 cover the principal arms of the project, the atomistic simulation and random search algorithm respectively. The nature of this exploration entails generating a significant amount of data, as all of the input and output data, results, result display and interpretation need to be preserved. Section 5.4 explains the strategies and modus operandi for acquiring a sufficient amount of meaningful data in the time-frame.

5.1 Atomistic Simulation

Vampire runs in a Linux distribution and is called from the command line. It is open source under the GNU General Public License [56] and Matthew Dale added functionality to the package enabling micromagnetic simulation and magnetic field injection into quantised regions - macrocells - of the simulated materials [57]. This was an essential component to the "Reservoir computing with thin-film ferromagnetic devices" paper by Dale et al. [35], without which my project would not be possible using Vampire. It is important to specify that only the z-dimension of the input field vectors are non-zero. Similarly, only the z-dimension of the cell magnetic polarisation is extracted to form the output.

5.1.1 Running Vampire

Vampire is run by invoking a binary executable from the console. No information is passed in this step; instead, Vampire automatically parses the current directory for two files, without which the simulation will not run:

- *Input file*: contains all the simulation parameters, namely:
  - Crystal structure
  - Unit cell size
  - Material file

  - System dimensions
  - Macrocell size
  - Integrator function (LLG etc.)
  - Duration and time steps
  - Temperature
  - Output data and format

- *Material file*: is invoked by the input file and contains all intrinsic properties of an element, crystal or alloy, namely:

  - Gilbert damping constant
  - 2^{nd} order uniaxial anisotropy constant
  - Heisenberg exchange constant j_{ij}
  - Atomic spin moment
  - Curie temperature
  - Temperature rescaling exponent [58]

- *sourcefield file*: a simple text file that contains the input field in Tesla to each macrocell. It is important to note that this is a modification of the source code, and is not compatible with the default Vampire distribution. Each line must be explicitly labelled by a number at the end of each line, and all values are space separated. The first two lines (-2 & -1) are the grid coordinates, from which Vampire infers where to input each field value according to its position in the line. Each line is a timestep and must contain a number of values equal to the number of cells in the film, excluding the endline label. For a 10 timestep simulation of a thin film with 16 cells, the sourcefield.txt file must be 12 lines long, with 17 numbers per line, as in figure 7.

These files are not formatted according to any particular scripting language and use Vampire specific keywords, written in plain ASCII text. See figure 6 for an example, where comments are denoted by #. I included two extra comments at the top of the material file, crystal structure and unit cell size. The search algorithm computes a combination of simulation parameters before modifying the input and material files and invoking vampire. As such it was simpler to keep all of the information inherent to a specific material in once place. The z-dimension

```
# crystal structure = fcc
# unit cell size = 3.524 !A
#===============================================
# Generated material file for input into vampire
#===============================================
#
#-----------------------------------------------
# Number of Materials
#-----------------------------------------------
material:num-materials=1
#-----------------------------------------------
# Material 1 (Ni)
#-----------------------------------------------
material[1]:material-name=Ni
material[1]:damping-constant=0.5
material[1]:second-order-uniaxial-anisotropy-constant=5.47e-26
material[1]:exchange-matrix[1]=2.757e-21
material[1]:atomic-spin-moment=0.606 !muB
material[1]:temperature-rescaling-exponent = 2.322
material[1]:temperature-rescaling-curie-temperature = 635
material[1]:material-element=Ni
material[1]:minimum-height=0.0
material[1]:maximum-height=1.0
material[1]:initial-spin-direction=1,0,0

#material can be specified to be micromagnetic here else it is automatically
#atomistic.
material[1]:micromagnetic-discretisation-enabled
#-----------------------------------------------
```

```
create:crystal-structure = fcc
dimensions:unit-cell-size = 3.524 !A
material:file = Ni.mat

dimensions:system-size-x = 49 !nm
dimensions:system-size-y = 49 !nm
dimensions:system-size-z = 3.524 !A

cells:macro-cell-size = 5 !nm

micromagnetic:discretisation=micromagnetic
micromagnetic:atomistic-steps-per-micromagnetic-step = 1

sim:integrator = llg-heun
micromagnetic:integrator = llg

sim:time-step= 100 !fs
sim:time-steps-increment = 1000
sim:temperature = 309.65
sim:total-time-steps = 500000
sim:applied-field-strength = 0 !T
sim:applied-field-unit-vector = 0, 0, 1
dipole:solver=tensor
sim:program = time-series

sim:cells-source-output = true
config:macro-cells
config:macro-cells-output-rate = 1

output:time-steps
screen:time-steps
output:temperature
output:material-magnetisation
screen:material-magnetisation
```

Figure 6: On the left an example of a material file for Nickel, on the right the associated input input file.

can be less than the unit cell size and Vampire will simply simulate a monolayer of atoms, as is the case in Figure 5. Increasing the system dimensions incurs serious implications for the simulation speed. This particular example, a $50nm \times 50nm \times 0.3524nm$ took roughly $444s$, with a 12-core Intel i7 and 16GB of ddr5 RAM. By contrast, a simple monolayer takes roughly $120s$.

Vampire natively supports parallelisation through compatibility with the OpenMPI framework, as it is primarily aimed at research application which will require high performance clusters to simulate systems larger that a few dozen nanometres. If in the correct directory, Vampire can be easily invoked with bash:

```
./vampire-serial
```

If running with parallelisation, mpirun must be called first:

```
mpirun -np x vampire-parallel
```

where x denotes how many cores to request. there are many flags for customising mpirun calls, the above being the simplest invocation of parallel Vampire. In my simulations, I used the

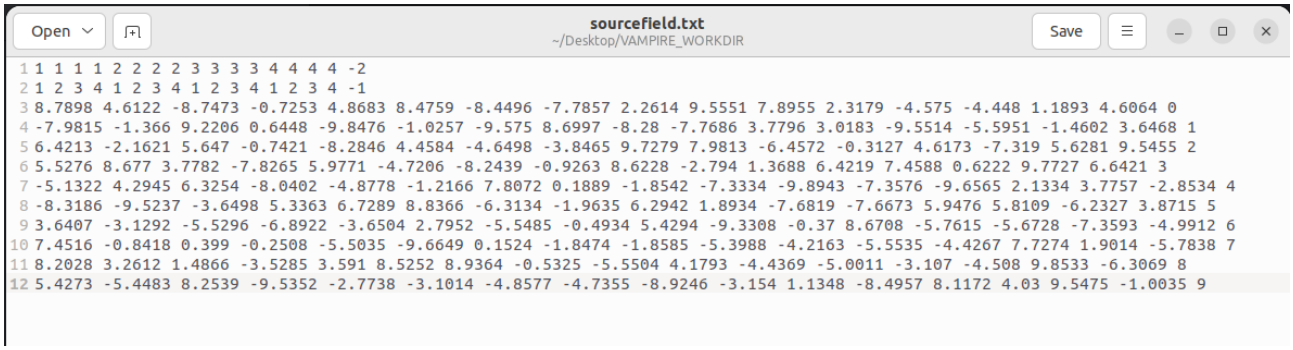


Figure 7: Example of a `sourcefield.txt` file, containing the input fields to each macrocell in the thin film. The first two lines describe a 4×4 lattice of cells simulated for 10 timesteps. Accordingly, there are 16 input values per line, plus the addition endline label.

following call to parallel Vampire:

```
mpirun -np x -map-by ppr:1:core -bind-to core vampire-parallel
```

Which ensures that the parallelisation is distributed amongst available cores such that only one process per core is allowed, resulting in x processes on x cores. Hard coding this mapping mitigates the risk of multiple processes being run on virtual cores within the same processor, potentially creating bottlenecks.

5.1.2 Extracting Data

There are three output files generates by a Vampire simulation:

- *Output file*: this is the default vampire output file containing the requested output parameters (as specified in input file) at every user defined time-step. This file is not used in this project, as the reservoir output file contains all of the necessary magnetic data.

- *Log file*: transcript of Vampire execution containing timestamped entries pertaining to memory allocation, setup steps, errors etc. Only utilised for debugging and testing.

- *Reservoir output*: an important member of Dale's modifications and contains the magnetic polarisation of every cell at every timestep. This is the data that the output layer is trained on. The format is slightly different from the `sourcefield.txt` file, as the `reservoir_output.txt` file does not contain a timestep label, nor any information pertaining to grid. It simply contains as many entries per line as there are cells. The number

of timesteps is always significantly longer than the requested timesteps, I surmise to observe the relaxation dynamics of the film after perturbation.

See section 5.4 for a more detailed description of the exploration, and sections 6 & 7 for an appraisal of the findings.

5.1.3 Test Bed

In this section I will set out the test conditions that remain unchanged throughout the project. These can be separated into two main categories: simulation parameters, defined in the input file and shown in table 5.1.3, and material properties, defined in each materials respective material file and shown in table **??**. The choice of time step and the frequency of input injection is the same as in the setup used by Dale et al. [35], in absence of any other justification. The applied field parameter refers to the option in Vampire to engulf the entire system in a static magnetic field and is not related to the fields injected via sourcefile.

Simulation Parameters	
Integrator	LLG
Dipole solver	Tensor
Time step	$20ps$
Step increment	25
Time steps	$25,000$
Input length	1000
Applied field	$0T$
Field Vector	$(0, 0, 1)$

Table 2: Static simulation parameters used in Vampire simulations.

5.2 Random Search Algorithm

The random search algorithm uses an object-oriented programming (OOP) approach, coupled with function-specific modularity by using a multi-file architecture. The search is embedded in a single class called `MaterialEvolution` containing all of the exploration parameters and associated data as fields. Figure 8 is a basic representation of the workflow, which roughly sets out the class functions. A heuristic description of the structure is that class functions contains simple code, that typically warrants a single call per loop, with more complex or repeatedly called functions defined in separate `.py` files. The following subsection provide a

Material Properties				
property	Ni	Co	Fe	units
crystal structure	fcc	fcc	bbc	-
unit cell size	3.524	2.507	2.866	Å
atomic spin moment	0.606	1.72	2.22	μ_B
exhange energy J_{ij}	$2.757 \cdot 10^{-21}$	$6.064 \cdot 10^{-21}$	$7.050 \cdot 10^{-21}$	$J/link$
2^{nd} order uniaxial anisotropy	$5.470 \cdot 10^{-26}$	$6.690 \cdot 10^{-24}$	$5.650 \cdot 10^{-25}$	$J/atom$
Temperature rescaling exponent	2.322	2.369	2.876	-
Rescaling Curie temperature	635	1395	1049	K

Table 3: Static material properties used in Vampire simulations.

detailed description of some of the functional block represented in Figure 8, with the exception of the training step and data acquisition which have their own sections, 5.3 and 5.4 respectively. There are 19 python scripts in this project, so in order to preserve brevity and legibility their functionality will be explained with diagrams, whereas the entire codebase can be found on my github:

```
https://github.com/MatteoPearce/
```

The repository name is:

```
MEng-Final-Project-In-Sensor-Thin-Film-Magnetic-with-Reservoir-Computing
```

Provided that Vampire is installed as described in appendix E, Python 3.10.12 is installed, and the working directories are setup as described in section 5.2.1, the entire search algorithm can be run with:

```
chmod +x main.py ; ./main.py
```

Changing the sweep parameters has to be done manually as I have not designed the code for the purposes of distribution. The sweep parameters themselves need to be changed in the dictionaries described in section 5.2.2, whereas customisation options such as random scaling, random input locations (2 macrocells instead of all), input signal strength and input time-series length can be set at the bottom of the main with the class constructor. The directory paths will also need to be set accordingly.

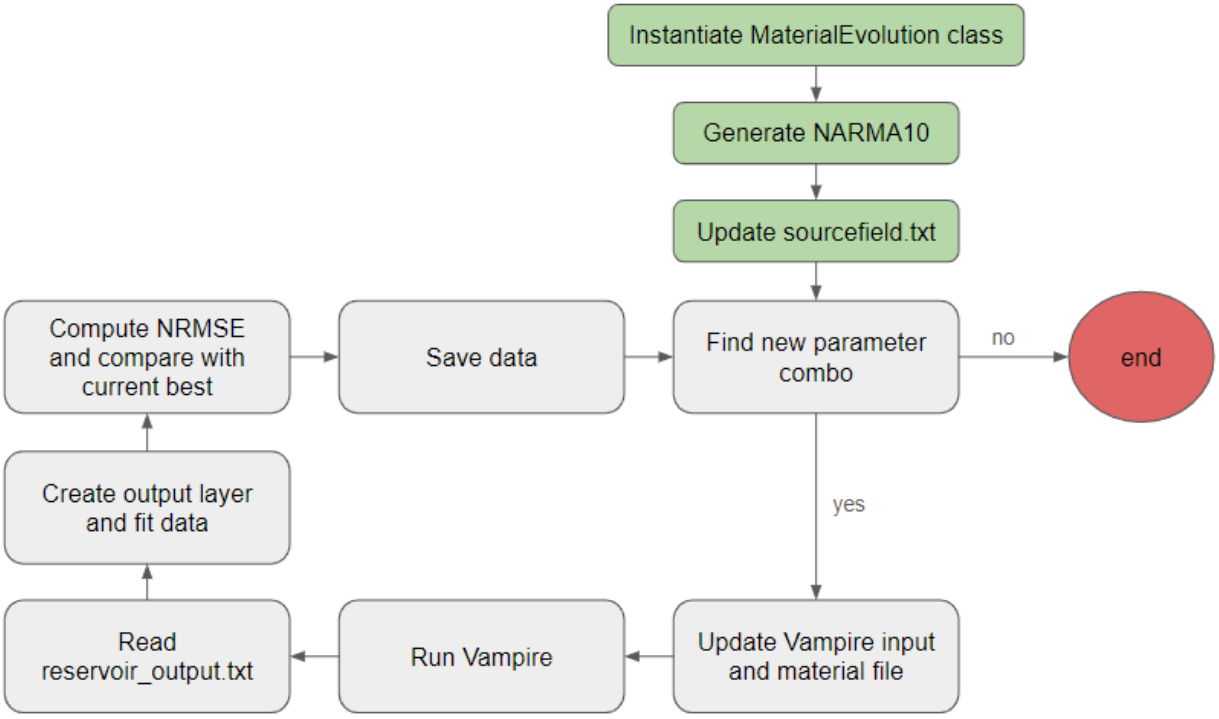


Figure 8: Top-level workflow of the random search algorithm. Green boxes represent the setup phase and grey boxes the loop.

5.2.1 Directory Setup

To run my code a simple file-structure comprised of two main directories is required. The first is the main working directory called `VAMPIRE_WORKDIR/` and contains the following important files:

- `vampire-serial`/`vampire-parallel` binary.
- `VDC` binary.
- Vampire `input.txt` file.
- Vampire material (`.mat`) file.
- Vampire `sourcefield.txt` file.
- `Materials/` directory.

The `Materials/` directory is a library for material files. When a new combination of simulation parameters warrants a change of material, this is simply copied into `VAMPIRE_WORKDIR/` without modifying the contents, unlike the input file. The other main directory is `VAMPIRE_TEST_RESULTS/`, where all retained data is saved. It is worth noting that both these directories can have different names and paths by changing their respective fields in the `MaterialEvolution` class.

5.2.2 Simulation Parameters Selection

The information relative to exploration parameters is contained within three dictionaries, two contain the numbers to select at random for the search whereas the other only contains the units of measurement. This is so that the parameters can be stored as numbers instead of strings, allowing for operations to be carried out on them. At the moment of writing information to the input files, all new information is copied and then converted to string. The names and purpose of the dictionaries are:

- `input_file_parameters`: the keys are the names of Vampire fields as defined in the manual and must be exact as they are written to the input file. The values are lists of parameters to explore. Even if only one value is to be explored for a particular parameter, it must still be in a list.

- `input_file_units`: the keys must be identical to those in `input_file_parameters`, as it contains the unit of measurement as a string of each Vampire simulation parameter.

- `other_sweep_parameters`: same exact syntax as `input_file_parameters`, but kept separate as these parameters are not included in the input file. There are only two of these parameters, which are *field intensity input scaling* and *intrinsic magnetic damping*. The former is applied in steps preceding the simulation and is not a Vampire parameter, whereas information regarding the latter is housed in the material file.

Parameter combinations are selected every iteration and not pre-computed. The class function that selects combinations and verifies whether they are unique is `select_parameters`. The pseudo-code describing the behaviour is in algorithm 1.

All the parameters that make up the unique combination are stored in a separate dictionary `all_sweep_parameters`, so that checking future combinations and writing iteration details to text easier. The `scale_height` function, from `ScaleHeight.py`, ensures that the requested height is either a monolayer or a multiple of the unit cell size of the material. This makes it easier to define a list of film heights to trial, as setting z-dimension to 4 for example, entails that the only permissible multiple of unit cell for each material with cell size of 2.866$\mathring{A}$, for Fe, 3.524$\mathring{A}$ for Ni and 2.507$\mathring{A}$ for Co, is 1. `scale_grid`, from `ScaleGrid.py`, ensures that the

Algorithm 1:
`select_parameters` pseudo-code: find unique simulation parameter combination.

```
while combo not found do
    for key, value in input_file_parameters do
        random value per key
    end
    for key, value in other_sweep_parameters do
        random value per key
    end
    film height = scale_height()
    film dimensions = scale_grid()
    if combo is unique then
        combo found
        save combo
    else
        attempts + = 1
    end
    if attempts > threshold then
        end simulation
    end
end
```

chosen combination of x, y and macrocell dimensions are geometrically conceivable.

5.2.3 Input Time-series

For the entire random search the time-series used as an input to the thin films under simulation is the Nonlinear Auto-Regressive Moving-Average model with exogenous inputs of order 10 (NARMA10) [59]. It is a staple of network benchmarking due to several key attributes:

- *Nonlinearity*: its inherent nonlinearity presents a challenging problem for models to learn and predict. The requirement for high-dimensionality in reservoirs makes NARMA10 an appropriate testbed for evaluating their capabilities.

- *Memory Dependence*: the NARMA10 algorithms has a built-in dependency on past inputs and outputs up to a specified order (10 in this case) which tests a reservoirs fading memory, crucial for tasks like time series prediction.

- *Well Established*: It has a well-defined structure and generation process, making it highly reproducible and the wealth of literature makes comparison of results across different studies easy, constituting a well-explored standard for benchmarking.

`GenerateTimeseries.py` contains `generate_timeseries` which generates the NARMA10 input and output traces. The input u_t is given by:

$$u_t = \hat{x_t} + \mu \cdot x_t \tag{17}$$

Where x_t a time-series of random values pulled from a uniform distribution, in this case between $[-a; a]$ where $a = 1$. $\hat{x_t}$ is the average of this trace, in this case 0, and μ is a scaling factor defined as $\frac{a}{2}$. Typically one would simply use a as the scaling factor in order to utilise the full range. However, in my implementation I get a *RuntimeWarning* due to overflow, hence I opt for $\frac{a}{2}$ which doesn't trigger any errors. The output time-series next step y_{t+1} is defined as:

$$y_{t+1} = 0.3 \cdot y_t + 1.5 \cdot u_t \cdot u_{t-9} + 0.1 + 0.05 \cdot y_t \sum_{k=0}^{9} y_{t-k} \tag{18}$$

It is customary to allow the system to "warm up", providing more reliable traces, but discarding the first N timesteps; in this project, $N = 1000$. Both input and output traces are integral to proper execution of ridge training, as the input trace is fed to the reservoir, and the training set is the NARMA10 output. In this way the ridge is trying to match the NARMA10 algorithm, by mapping the transformation of inputs by spin wave dynamics to the output trace generated by (18). One set of time-series is generated per random search, so that all parameter combinations are tested on the same data.

5.2.4 Updating Input Files

The class function `update_input_files` updates the `input.txt`, `Xx.mat` and `sourcefield.txt` files necessary for Vampire to run. Before writing to file, the parameters are converted to strings and their respective units are added, the number of macrocells per dimensions is calculated and used to create the `sourcefield.txt` headers (first two lines), and the input time-series is multiplied with the scaling term. If random scaling is selected, the time-series is also multiplied with vector of equal length comprised of random numbers between 0 and 1 from a uniform distribution, generated with Numpy's `random.rand()` function. The material file is copied from the `Materials` folder to the parent directory `VAMPIRE_WORKDIR/` and then the magnetic damping within the file is updated. This is the only value within the material file that is modified during

the search. The files are rewritten by scripts in separate `.py` files, namely:

- `SourcefieldFilemaker.py` → `write_sourcefield` → `sourcefield.txt`
- `SelectMaterialFile.py` → `select_material` → copy `Xx.mat`
- `ModifyVampireInputFile.py` → `modify_vampire_input` → `input.txt`
- `UpdateMagneticDamping.py` → `update_damping` → `Xx.mat`

Lastly, the `reservoir_output.txt` file is deleted to avoid the possibility of an older output being used for the ridge training.

5.2.5 Invoking Vampire

The python script `CallVAMPIRE.py` houses the function `call_vampire` for invoking Vampire and allows for several run modes: serial, parallel, debug. The last mode is only available if the vampire distribution is compiled specifically with the correct debug flag, as can be inferred from the `makefile`. Invoking an executable with bash in Python was implemented with the `subprocess` module, with the `call` and `run` functions. These methods receive as arguments strings of commands exactly as they would be written in the terminal. `call` is a procedure and does not return any values, however it prints to screen Vampire's output messages. `run` on the other hand, does not automatically print and instead gives the option to save the output in a variable. The former method is therefore used for a verbose run mode, printing the output which is useful for debugging, whereas the latter is a quicker run mode, with computing resources saved by not having to print the output. Invoking the respective binaries is as explained in section 5.1.1. The sole difference in the programmatic approach is how to determine the number of available CPUs for parallelisation, which is easily done using the `cpu_count()` function from the `multiprocessing` module. There is a heuristically developed mechanism for catching Vampire failures as a function of the simulation duration. A script execution time of less than 1 second is sufficient to deduce that Vampire was unable to create the system and that no simulation has occurred. Vampire can also fail during simulation, however this is harder to determine as it manifests as a returned NaN value for a macrocell polarisation. This is easier to catch at the training stage when the ridge regression fails throwing a *ValueError*.

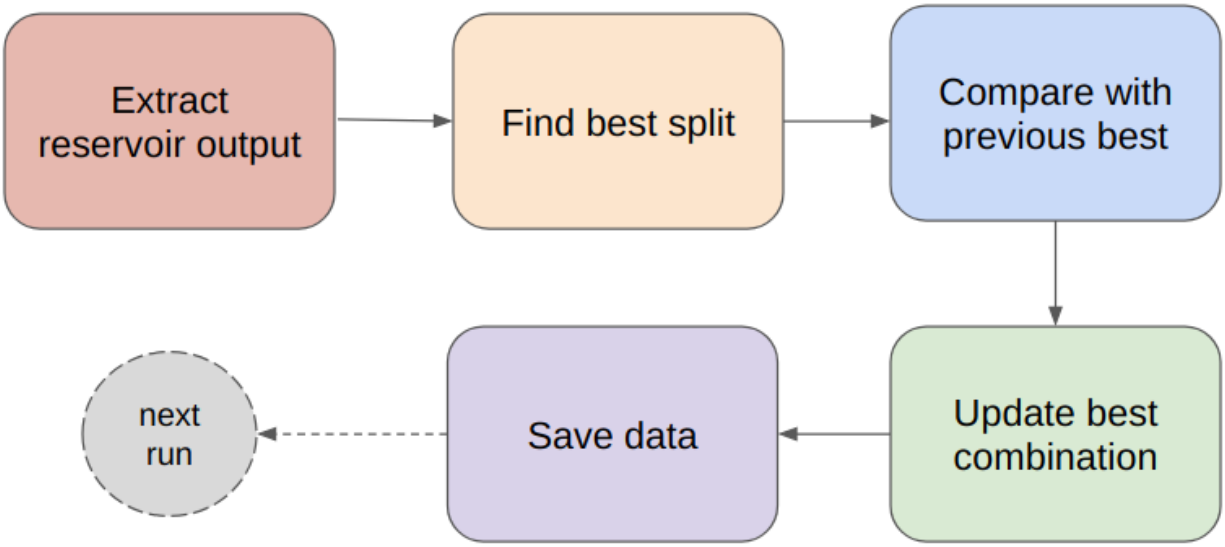


Figure 9: Workflow of the training, comparison and data saving steps.

5.3 Reservoir Computing

The `reservoir_computing` class method handles the data associated with the training and prediction task, including updating the current best combination and calls to `save_data` from `SaveData.py`. The actual training steps are performed in the `RegressionTraining.py` file, discussed in more detail in sections 5.3.1 and 5.3.2. The top-level workflow of the mechanisms described in these sections is shown in in figure 9. The metric of choice for evaluating the performance of a thin film is the Normalised Root Mean Squared Error (NRMSE), defined as:

$$NRMSE = \sqrt{\frac{\sum_{i=0}^{N-1}(y_{target} - y_{observed})^2}{N \cdot variance}} \tag{19}$$

Using a normalised metric makes the data more interpretable, as the order of magnitude of the imput data does not impinge on one's ability to assess the result. The model fitting and NRMSE calculation are achieved with wrapper functions provided by the reservoirpy library. For all iterations, both the training NRMSE, i.e the prediction task when the training data has been fed back into the ridge, and the NRMSE on unseen data are calculated. This does not warrant generating more data, as it is simply split before training such that a portion of the reservoir output is never seen by model until the testing phase. Both successful and unsuccessful runs are saved, so that if any bugs were to be the cause, the conditions can be recreated. For more information on the data acquisition and processing, see section 5.4.

5.3.1 Training Data Split

The functionality described in this section is that of the `train_ridge` function in from the `RegressionTraining.py` module. In this method the reservoir output data is first extracted from the `reservoir_output.txt` file in the working directory. The target used for training is the output from the NARMA10 algorithm and is passed upon invocation. The data is extracted by the `extract_reservoir_IO` function from `ExtractReservoirIO.py`. This method can also extract the sourcefield information directly from the working directory, however this is a legacy mechanism and is currently redundant. RR works very poorly on minute values, which is problematic given that the output from the reservoir is typically in the order of $[\pm 10^{-20}T : \pm 10^{-22}T]$, many orders of magnitude smaller than the fields being injected into the medium. To overcome this, the reservoir output is divided by the signal strength: for sub-alpha signals as an example, the output is divided by 10^{-13}. After this pre-processing step, a loop commences where a range of data splits are trialled and the data is passed to the `train_cycle` function which performs the fitting step and NRMSE calculation, described in section 5.3.2. This logic flow is summarised in algorithm 2.

Algorithm 2:
`train_ridge` pseudo-code: pre-process data, find best split, return best result.

```
reservoir_output = extract_reservoir_IO
reservoir_output / signal_strength
for split in numpy.arange(0.5, 0.95, 0.05) do
    split data
    NRMSE = train_cycle( split data )
    if NRMSE < best_NRMSE then
        best_NRMSE = NRMSE
        best_split = current_split
    end
end
return best_NRMSE
```

The NRMSE of the prediction using the training data, which I will refer to as the training NRMSE, is also saved.

5.3.2 Model Training

The model is trained using RR, hence a ridge parameter, also known as the regularisation parameter, needs to be chosen. This is typically achieved through trial and error, with no definitive rule for its selection. In this particular case I have found that the regularisation

parameter quickly causes the prediction trace to flatline. The training step using reservoirpy is the remit of `train_cycle` in `RegressionTraining.py`. Having received the training and testing datasets, it fits the model with the training data, then runs the testing data and the training data through the model and computes the error, thus acquiring the NRMSE and the training NRMSE. This functionality is described in algorithm 3.

Algorithm 3:
`train_cycle` pseudo-code: fit model and compute NRMSE.

```
ridges = np.linspace(0, macrocells · signal_strength, 10)
for ridge in ridges do
    W_out = target · training_data ^T · (training_data · training_data ^T + I · ridge) ^−1
    prediction = W_out ·testing_data
    re-run =W_out ·training_data
    NRMSE = NRMSE(target, prediction)
    training_NRMSE = NRMSE(training_data, re-run)
    if NRMSE < best_NRMSE then
        best_NRMSE = NRMSE
        best_training_NRMSE = training_NRMSE
        best_prediction = prediction
        best_testing_data = testing_data
    end
end
return best
```

The ridge parameter is one of 10 values equally spaced between 0 and the number of cells times the signal strength. In this way, a selection of ridges which won't cause the prediction to flatline as well as being sufficiently large to mitigate collinearity is trialled. In an ideal world this number would be larger, but the toll on computation time would have been too great. It is worth clarifying that besides the NRMSE, the inclusion of "best" in the variable names does not imply that it was the best observed instance of that variable, but rather that it is associated with the best NRMSE. The model and the weights matrix calculated with RR are deleted to avoid using the same model twice. Considering that 9 data splits are trialled, and that for each split 10 regularisation parameters are trialled, the total number of models trained per search iteration is 90. The number of perceptrons in the ridge is equal to the number of macrocells in the thin film. Therefore, N input timesteps from the NARMA10 series and M macrocells provides an output matrix of $N \times M$. A single timestep produces a vector of size M which needs to be transformed to an output dimension of 1. This entails that the weight matrix calculated with RR is of size $M \times 1$, yielding an output of size $N \times 1$. For a single timestep, $N = 1$ and the output is 1×1 as desired.

5.4 Results Acquisition

In this section the format of the test runs and the ranges of exploration parameters will be set out. All of the design considerations, trials and tribulations of acquiring data will be discussed, providing an overview of the inherent difficulties of the project with an appraisal of the reality that transpired versus the expectation. The principal idea for this project was to develop the random search algorithm, create a library of Vampire material files of ferromagnetic elements and alloys, and define high resolution ranges of exploration parameters. The input data was to be sampled from open source databases of magneto-medical data, acquired from processes such as MEG and MCG. The hope was to unearth an ideal combination of parameters for each material that could excel as a reservoir such that a layer of perceptrons could infer the magnetic inputs, demonstrating a potential for diagnosis. Due to several factors which will be discussed in this subsection, the project did not transpire as initially envisioned. Sections 5.4.1 and 5.4.2 discuss these difficulties in more detail, whereas section explains the measures taken to overcome them and strategy devised in lieu of the ideal scenario.

5.4.1 Materials

At the beginning of the project I was, perhaps unsurprisingly, naïve in my expectation of the level of completeness of the literature on the micromagnetic behaviour of elements. I was convinced that I would find tabulated values of the necessary exchange and anisotropy properties of any and all crystals and alloys. I quickly learned that the choice by Dale et al. to explore Iron, Cobalt and Nickel was strongly influenced by exhaustive corroboration of their micromagnetic properties. I had envision testing materials such as Magnetite, Yttrium garnets and different combinations of multilayer films and alloys comprised of the three ferromagnetic elements explored in this project. This would not be possible due to a lack of data that would be sufficient to eliminate guesswork and that could provide acceptable approximations. The literature was particularly lacking with regards the Heisenberg exchange, and extrapolating this from first principles with the meanfield expression requires the spin-wave stiffness constant, another scantly explored property. Furthermore, the exchange needs to be calculated between all atoms in a lattice, in both directions. For example, a crystal comprised of atoms from two different elements requires defining the exchange from an atom of element A to an atom of element B and vica versa. The inability to define materials with a sufficient degree of completeness combined with other factors discussed in subsequent sections, constrained me to solely considering Iron, Cobalt and Nickel.

5.4.2 Parallelisation

The biggest contributor the time taken in each search is the simulation in Vampire. Increasing the film dimensions and thickness causes the simulation time to balloon significantly, as mentioned briefly in section 5.1.1. A key strategy to overcome this is the use of several cores in parallel to speed up the Vampire simulations. The random search in Python was not designed to be compatible with multi-threading as it enjoys fast execution times. Unfortunately, using Vampire-parallel either does not speed up the simulation or it does so minimally to no great gain. In fact, it was often observed that using parallelisation actually slowed down the simulation in many instances. This could be due to lack of compatibility between the modified Vampire distribution, or even a pitfall of Vampire itself at its current stage of development. This fact is by far the biggest hindrance to the acquisition of exhaustive test results, as there is simply not enough time to test a full range of parameters. Corners inevitably had to be cut, which mostly entailed increasing the resolution of the steps between different values of the exploration parameters in addition to some heuristic jockeying of simulation choices. Section 5.4.3 sets out in detail the ranges of parameters used to acquire the test results discussed in this paper, as well as the use of multiple servers to parallelise random searches, purely by running multiple instances with different searches.

5.4.3 Strategy

In this section I will describe the strategies used to make the most given the lack of computational speed and disposable time, as well as the general approach to acquiring data to try and corroborate or disprove the goals set out in the objectives. Integral to the results of this project was the concurrent use of multiple devices, each running at least one random search. The bulk of the searches were run on the Viking HPC, which is a high performance computer facility provided by the University of York. Though Viking did not afford any increase in the speed of the search, the high number of nodes, processors and memory available allowed me to run many searches in parallel. Furthermore, I made use of the Lovelace server, made available to me by my project supervisor Martin, and the teaching0 server available to me by default as a student at the University. My modus operandi was to use the latter servers to test code functionality and test setups, followed by implementing wider searches acquiring final data on the Viking server. Communication with the servers and transmission of test data to my home computer was achieved with the SSH protocol. To speed up the acquisition of data I ran a search for each material in isolation, requiring therefore three concurrent searches per exploration. Each

set of searches is labelled according to the category of exploration they pertain to, the syntax being a letter and a number. These categories are as follows:

1. Concept Validation: emulate the setup of Dale et al., and explore the effects of equal signal injection, uniform input scaling and body temperature.

2. Biomagnetic Magnitude: film inputs of magnitude equal to alpha and sub-alpha brain rhythms, as well as the magnetic emanations of the heart. The trace is still a NARMA10. All subsequent tests are run with the smallest signal.

3. Film dimensions: using the data from the previous two searches to narrow the exploration ranges, the effects of increasing the (x, y) film dimensions are explored. These are very computationally expensive, hence the need for restricted ranges for other film parameters.

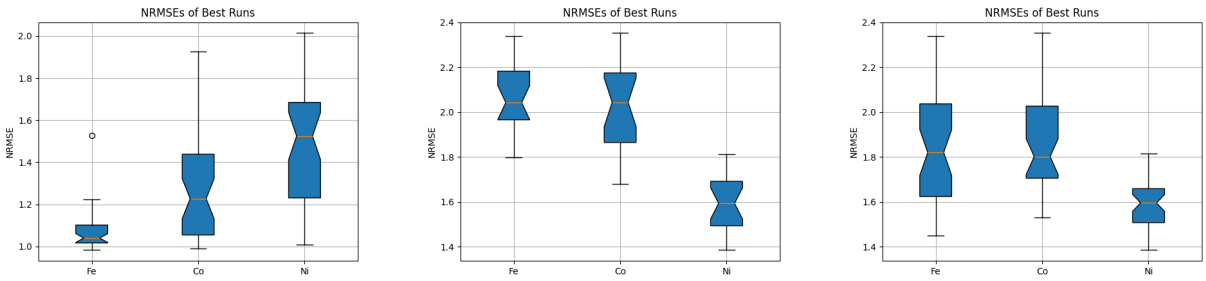
4. Film Height: the best film parameters thus discovered are kept constant and the effects of height are explored.

5. Strip geometries: using the best height and film properties, rectangular configurations are explored as an interesting aside.

All exploration data can be found in appendix B, where the static values are tabulated in B.1 and the exploration ranges in B.2. For a reminder of the test bed setup, see appendix A. To corroborate the validity of each best performing run, the `TrialBest` class from the `TrialBest.py` file tests each best setup many times and plots boxplots showing the range of NRMSEs achieved for each new NARMA10. The setup of these trials is shown in appendix B.3, whereas the results can be found in each respective section of the results appendix C.

6 Technical Chapters

Here will be discussed the effects of different simulation parameters on the NRMSE of a film as a reservoir for the purposes of predicting NARMA10, mirroring the objectives as set out in section 3.4. Due to the short time available and significant computational resources required I was not able to perform exhaustive searches, nor achieve a good NRMSE for the NARMA10 prediction. I will therefore focus on the discrepancies between results and the parameter changes that may have induced them.



(a) A1 (b) A2 (c) A3

Figure 10: Box plots of 40 trials of the best performing system setups per material, belonging to exploration classes A, as described in section 5.4.3 and summarised in appendices B.1 and B.3.

6.1 Body Temperature

Increasing the system temperature amplifies the effects of thermal noise in the system. The expectation therefore, building on observations from previous studies, is for the performance to tangibly worsen. The results displayed in figure 10 defy this prediction, as the differences between A2 & A3 display an improvement is performance. Nickel's NRMSE range has remained virtually unchanged, whilst Iron and Cobalt both have lower median values, at the expense of wider ranges. The discrepancies between materials echos the trend seen in A2, as can been seen by the similarities in the shape of both graphs. This would indicate that the most substantial contributor to the difference in performance between A1 and the rest of the A class is the input injection mode. It is difficult to explain these results, as the underlying physics and reports from other studies would give cause to predict a significant loss of performance, much greater than the scale observed in this project. Perhaps the small dimensions of the system and consequentially small distances between propagating spin waves dampens in some way the effects of the thermal noise, as the larger dynamics dominate. It could also be an effect of the uniform input injection, which may cause the noise to be applied quasi-equally to all parts of the system, thus manifesting as an offset of sorts. Lastly, it could be the case that the negative impact of temperature pales in comparison to that of input injection and hence is less identifiable.

6.2 Input Injection

To be implemented as part of a biosensor, the film would need to be situation near the target physiology. The nanoscale dimensions and proximity to the field source imply that the film would we fully immersed in the emitted fields, being perturbed at all locations approximately

uniformly. Furthermore, the amplification or attenuation of these signals would have to be achieved with a surface lacquer which again would be uniformly spread across the film. To emulate these conditions, the input scaling was applied uniformly and the inputs injected equally in all macrocells, from A2 onwards. A concern at project inception was that applying the inputs in this manner could compromise the dimensionality of the reservoir, as actively driving the whole film would dampen the magnon propagation between macrocells. Assessing the results from figure 10, it is clear that this has transpired. The results in A2 are significantly worse, however this has manifested differently in each material. Interestingly, the range of values for Cobalt and Nickel has improved, with no outliers and tighter boxes, but at the expense of far worse absolute NRMSE. Iron's NRMSE has worsened in tandem with a widening of the box height. Referring to appendix B.3, there is a clear shift from higher to lower values of magnetic damping from A1 to A2. This could suggest a loss of dynamics complexity when driving all cells equally, requiring less damping of the magnetic relaxation. It may well be that applying inputs in this way is unfeasible, narrowing the spectrum of use cases for thin films. Conversely, it could also be the case this can be overcome with an optimal system setup, which would require more exhaustive research.

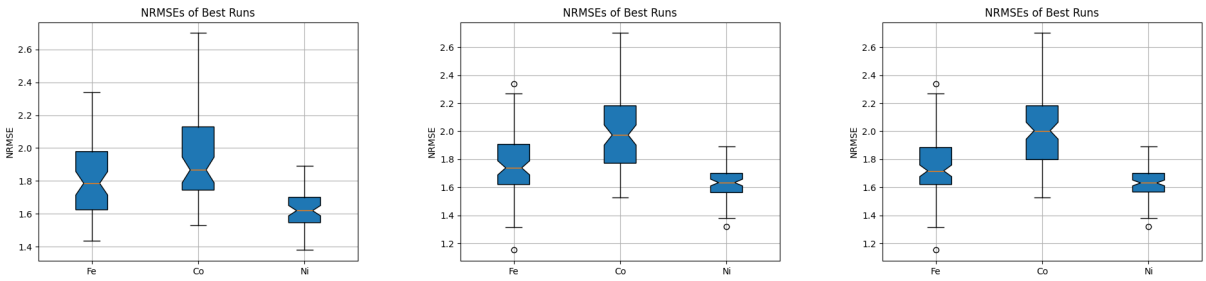
6.3 Biomagnetic Inputs

It is reasonable to expect that the larger the injected signal, the smaller the contribution of thermal noise and therefore the better the prediction results for NARMA10. However, a cursory appraisal of figure 6.3, when compared with A3 in figure 10, reveals that the findings are closely matched. These results are quite mixed and it is difficult to unearth any linear trends, with the idiosyncrasies of each material manifesting in unique responses to the different signals. Nickel once again has an unchanged NRMSE profile, Cobalt's median values improves as the signal strength wanes whereas Iron's median values worsen. It is interesting to note that in the case of the smallest signals there are no outliers, marking the starkest contrast between the graphs. Overall, the general lack of performance change suggests that the exploration of nanomagnetic reservoirs with pico-Tesla scale signals could be a worthwhile endeavour.

6.4 System Dimensions

6.4.1 Film Height

Comparing figures 6.3 and 12, it is clear that increasing the height of the film has a positive impact on the NRMSE, with the sole exception of Nickel, whose previous consistency of results



(a) B1 (b) B2 (c) B3

Figure 11: Box plot of 20 trials of the best performing system setups per material, belonging to exploration classes B, as described in section 5.4.3 and summarised in appendices B.1 and B.3.

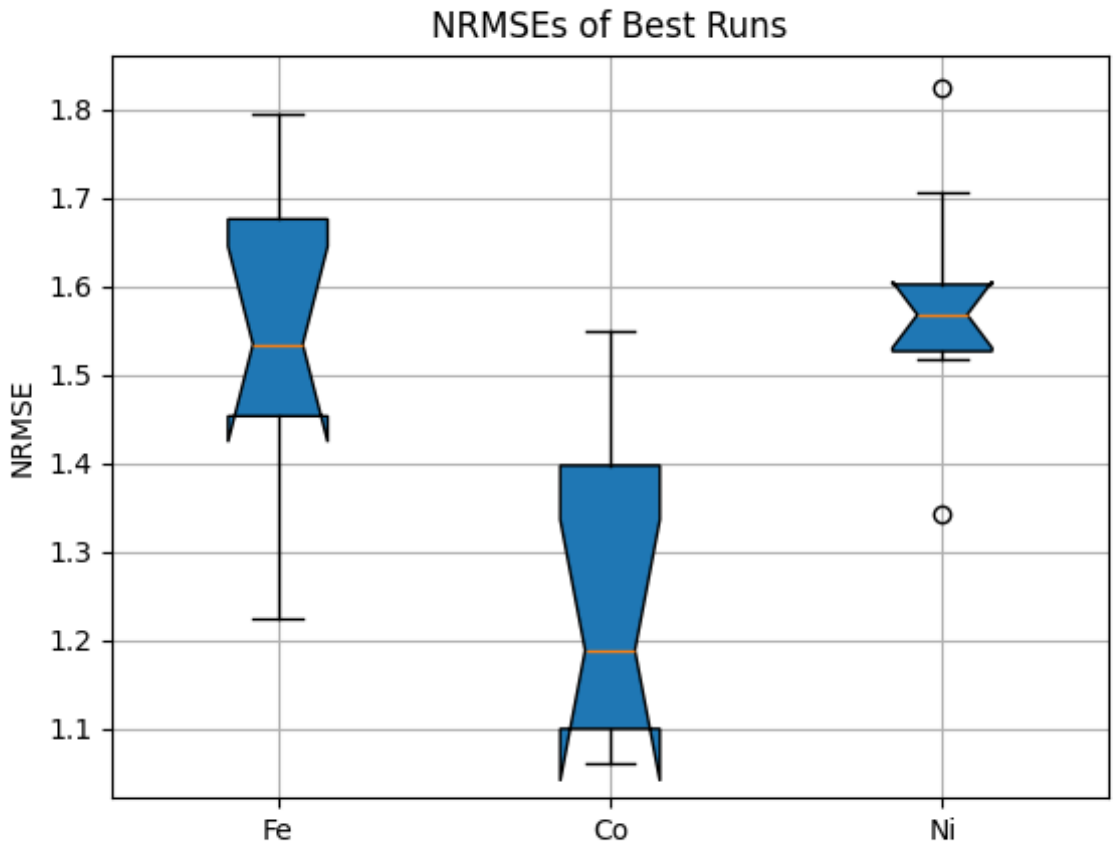
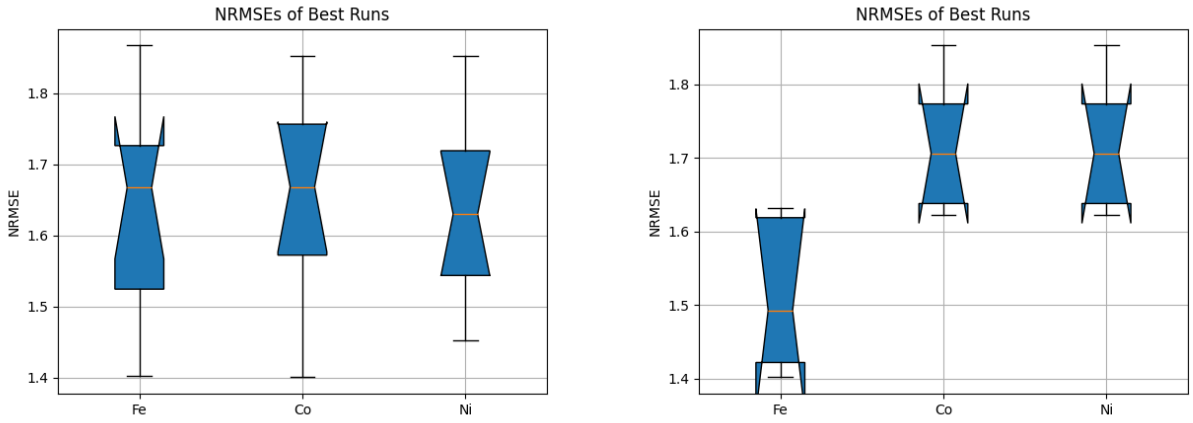


Figure 12: Box plot of 10 trials of the best performing system setups per material, belonging to exploration classes H, as described in section 5.4.3 and summarised in appendices B.1 and B.3.

is not mirrored in this exploration. Nickel's Median NRMSE has cleary worsened and presents a couple of outliers. The H1 graph looks like B1 if mirrored around the x-axis, such that cobalt has seen the biggest net improvement. Repeatability apears to have improved across the board as the ranges are narrower for all materials. Overall, it is reasonable to conclude that increasing the system height is conducive to a better NRMSE.



(a) C1 (b) C2

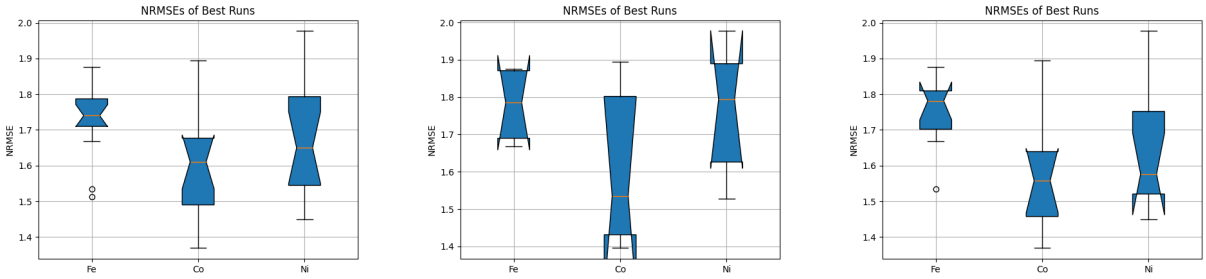
Figure 13: Box plot of 5 trials of the best performing system setups per material, belonging to exploration classes C, as described in section 5.4.3 and summarised in appendices B.1 and B.3.

6.4.2 Film Area

The box plots in figure 6.4.2 shows a marginal improvement in performance, especially in the case of Iron. Nickel appears to once again have a mind of its own as it noticeably worsens as the system dimensions increase. These larger films appear to be robust against outliers and present a fairly narrow range of values. Due to the long simulation times, the increase in area may be too small to unearth a more stark trend in NRMSE. Looking at the graphs in C.3.2, it is very interesting to note, is the presence of minuscule training errors, in contract with the NRMSE on unseen data. This is indicative of very prominent overfitting, perhaps due to the choice of rule for determining the ridge parameter. This would be my first port of call for expanding the exploration, as reducing the discrepancy between training and testing NRMSE has the potential for very good results.

6.4.3 Strip Geometries

The results displayed in figure 6.4.3 are difficult to assess. There are no emergent trends and the performance is either marginally better or worse depending on the material and the strip geometry. The fact that there has not been any significant worsening of results, coupled with the apparent randomness of the NRMSEs, could warrant further exploration in hope of finding an optimal strip setup that may excel at a specific task.



(a) S1 (b) S2 (c) S3

Figure 14: Box plot of 5 trials of the best performing system setups per material, belonging to exploration classes S, as described in section 5.4.3 and summarised in appendices B.1 and B.3.

7 Conclusions

It is clear that due to the inherent computational overhead of this modality of research has proven to be a significant hindrance to the acquisition of exhaustive or unequivocal results. The results induce a mixed bag of optimism and pessimism. On the one hand, thin magnetic films appear to be highly compatible with pico-Tesla scale signals, promising a potential for use with biomagnetic signals of a similar order of magnitude. It is clear that increasing the system height and dimensions improves reservoir performance and should be the primary exploration direction in any future work. Strip geometries have not proved incompatible with the physical RC paradigm, however do not convey any apparent trends and warrant more detailed investigation. Thermal noise due to a body-temperature environment promised to be a large hindrance to the feasibility of the concept, however the equal input injection and homogeneous input scaling proved to be far more significant contributors to loss of predictive performance. It is clear that a far greater body of work is required to fully corroborate or disprove the initial objectives and this project was greatly hampered by the speed of simulation and inability to replicate the initial premise of utilising simulated thin magnetic films as reservoirs.

8 Further Work

The immediate prerogative for anyone seeking to build on this endeavour should focus on a more exhaustive parameter exploration, with greater scope for materials, dimensions and geometries, as well as much finer resolution in magnetic damping and input scaling. Furthermore, I recommend a different modus operandi, as a random search is too computationally expensive when coupled with Vampire. Due to the ineffective nature of parallelised Vampire, one could consider using a different micromagnetic simulator. Additionally, implementing an evolution-

ary algorithm could reduce the number of search iterations and ultimately find a more optimal solution with greater efficiency. If I were to start this project anew, equipped with the power of hindsight, I would have scripted my search to be more compatible with the Viking cluster, in such a way that the search can initialise many simulations in parallel, distributed amongst the available resources. The inherent serial operation of my search algorithm was hugely detrimental in light of the enormous timescale of the simulations. Lastly, it would be beneficial to include a post-state collection leakage algorithm, as described by Dale et al. [35].

9 Statement on Ethics

The project is entirely software based, any and all stated physical design considerations are academically informed conjecture or speculation. Discussion on biological compatibility will adhere strictly and exclusively to scientific concerns. In light of these facts there are no ethical considerations that need to be stated.

References

[1] J. G. Carbonell, R. S. Michalski, and T. M. Mitchell, "An overview of machine learning," *Machine learning*, pp. 3–23, 1983.

[2] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural computation*, vol. 4, no. 1, pp. 1–58, 1992.

[3] J. F. Fulton, "Physiology of the nervous system," 1949.

[4] H.-D. Block, "The perceptron: A model for brain functioning. i," *Reviews of Modern Physics*, vol. 34, no. 1, p. 123, 1962.

[5] K. Gurney, *An introduction to neural networks*. CRC press, 2018.

[6] P. Cunningham, M. Cord, and S. J. Delany, "Supervised learning," in *Machine learning techniques for multimedia: case studies on organization and retrieval*. Springer, 2008, pp. 21–49.

[7] A. V. Tatachar, "Comparative assessment of regression models based on model evaluation metrics," *International Research Journal of Engineering and Technology (IRJET)*, vol. 8, no. 09, pp. 2395–0056, 2021.

[8] Ž. Vujović *et al.*, "Classification model evaluation metrics," *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 6, pp. 599–606, 2021.

[9] S. J. Miller, "The method of least squares," *Mathematics Department Brown University*, vol. 8, no. 1, 2006.

[10] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[11] G. C. McDonald, "Ridge regression," *WIREs Computational Statistics*, vol. 1, no. 1, pp. 93–100, 2009.

[12] M. Rajan, "An efficient ridge regression algorithm with parameter estimation for data analysis in machine learning," *SN Computer Science*, vol. 3, no. 2, p. 171, 2022.

[13] L. Medsker and L. C. Jain, *Recurrent neural networks: design and applications*. CRC press, 1999.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] S. Patnaik, L. Moharkar, and A. Chaudhari, "Deep rnn learning for eeg based functional brain state inference," in *2017 International Conference on Advances in Computing, Communication and Control (ICAC3)*. IEEE, 2017, pp. 1–6.

[16] A. Nanduri and L. Sherry, "Anomaly detection in aircraft data using recurrent neural networks (rnn)," in *2016 Integrated Communications Navigation and Surveillance (ICNS)*. Ieee, 2016, pp. 5C2–1.

[17] J. Zou, Y. Han, and S.-S. So, "Overview of artificial neural networks," *Artificial neural networks: methods and applications*, pp. 14–22, 2009.

[18] B. J. Wythoff, "Backpropagation neural networks: a tutorial," *Chemometrics and Intelligent Laboratory Systems*, vol. 18, no. 2, pp. 115–155, 1993.

[19] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proceedings of the 30th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, S. Dasgupta and D. McAllester, Eds., vol. 28, no. 3. Atlanta, Georgia, USA: PMLR, 17–19 Jun 2013, pp. 1310–1318.

[20] P. F. Dominey and M. A. Arbib, "A cortico-subcortical model for generation of spatially accurate sequential saccades," *Cerebral cortex*, vol. 2, no. 2, pp. 153–175, 1992.

[21] P. Dominey, M. Arbib, and J.-P. Joseph, "A Model of Corticostriatal Plasticity for Learning Oculomotor Associations and Sequences," *Journal of Cognitive Neuroscience*, vol. 7, no. 3, pp. 311–336, 07 1995.

[22] H. Jaeger, "The "echo state" approach to analysing and training recurrent neural networks-with an erratum note," *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, vol. 148, no. 34, p. 13, 2001.

[23] W. Maass, T. Natschläger, and H. Markram, "Real-time computing without stable states: A new framework for neural computation based on perturbations," *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, 2002.

[24] D. Verstraeten, B. Schrauwen, M. d'Haene, and D. Stroobandt, "An experimental unification of reservoir computing methods," *Neural networks*, vol. 20, no. 3, pp. 391–403, 2007.

[25] S. Stepney, "Non-instantaneous information transfer in physical reservoir computing," in *Unconventional Computation and Natural Computation: 19th International Conference, UCNC 2021, Espoo, Finland, October 18–22, 2021, Proceedings 19*. Springer, 2021, pp. 164–176.

[26] S. Fusi, E. K. Miller, and M. Rigotti, "Why neurons mix: high dimensionality for higher cognition," *Current opinion in neurobiology*, vol. 37, pp. 66–74, 2016.

[27] S. Wakabayashi, T. Arie, S. Akita, K. Nakajima, and K. Takei, "A multitasking flexible sensor via reservoir computing," *Advanced Materials*, vol. 34, no. 26, p. 2201663, 2022.

[28] A. Loeffler, R. Zhu, J. Hochstetter, A. Diaz-Alvarez, T. Nakayama, J. M. Shine, and Z. Kuncic, "Modularity and multitasking in neuro-memristive reservoir networks," *Neuromorphic Computing and Engineering*, vol. 1, no. 1, p. 014003, 2021.

[29] M. Cucchi, S. Abreu, G. Ciccone, D. Brunner, and H. Kleemann, "Hands-on reservoir computing: a tutorial for practical implementation," *Neuromorphic Computing and Engineering*, vol. 2, no. 3, p. 032002, 2022.

[30] M. Lukoševičius, *A Practical Guide to Applying Echo State Networks*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 659–686. [Online]. Available: https://doi.org/10.1007/978-3-642-35289-8_36

[31] E. Lesaffre and B. D. Marx, "Collinearity in generalized linear regression," *Communications in Statistics-Theory and Methods*, vol. 22, no. 7, pp. 1933–1952, 1993.

[32] C. Fernando and S. Sojakka, "Pattern recognition in a bucket," in *Advances in Artificial Life*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 588–597.

[33] Z. Yanling, D. Bimin, and W. Zhanrong, "Analysis and study of perceptron to solve xor problem," in *The 2nd International Workshop on Autonomous Decentralized System, 2002.*, 2002, pp. 168–173.

[34] K. Nakajima, "Physical reservoir computing—an introductory perspective," *Japanese Journal of Applied Physics*, vol. 59, no. 6, p. 060501, 2020.

[35] M. Dale, R. F. Evans, S. Jenkins, S. O'Keefe, A. Sebald, S. Stepney, F. Torre, and M. Trefzer, "Reservoir computing with thin-film ferromagnetic devices," 2021.

[36] D. A. Allwood, M. O. Ellis, D. Griffin, T. J. Hayward, L. Manneschi, M. F. Musameh, S. O'Keefe, S. Stepney, C. Swindells, M. A. Trefzer *et al.*, "A perspective on physical reservoir computing with nanomagnetic devices," *Applied Physics Letters*, vol. 122, no. 4, 2023.

[37] J. Fidler and T. Schrefl, "Micromagnetic modelling-the current state of the art," *Journal of Physics D: Applied Physics*, vol. 33, no. 15, p. R135, 2000.

[38] B. A. Cipra, "An introduction to the ising model," *The American Mathematical Monthly*, vol. 94, no. 10, pp. 937–959, 1987.

[39] R. J. Tilley, *Crystals and crystal structures*. John Wiley & Sons, 2020.

[40] G. Joyce, "Classical heisenberg model," *Physical Review*, vol. 155, no. 2, p. 478, 1967.

[41] M. Darby and E. Isaac, "Magnetocrystalline anisotropy of ferro-and ferrimagnetics," *IEEE Transactions on Magnetics*, vol. 10, no. 2, pp. 259–304, 1974.

[42] J.-G. Zhu and H. N. Bertram, "Micromagnetic studies of thin metallic films," *Journal of applied physics*, vol. 63, no. 8, pp. 3248–3253, 1988.

[43] M. Lakshmanan, "The fascinating world of the landau–lifshitz–gilbert equation: an overview," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 369, no. 1939, pp. 1280–1300, 2011.

[44] M. C. Hickey and J. S. Moodera, "Origin of intrinsic gilbert damping," *Physical review letters*, vol. 102, no. 13, p. 137601, 2009.

[45] L. Chen, S. Mankovsky, M. Kronseder, D. Schuh, M. Prager, D. Bougeard, H. Ebert, D. Weiss, and C. H. Back, "Interfacial tuning of anisotropic gilbert damping," *Physical Review Letters*, vol. 130, no. 4, p. 046704, 2023.

[46] M. Haidar, P. Dürrenfeld, M. Ranjbar, M. Balinsky, M. Fazlali, M. Dvornik, R. Dumas, S. Khartsev, and J. Åkerman, "Controlling gilbert damping in a yig film using nonlocal spin currents," *Physical Review B*, vol. 94, no. 18, p. 180409, 2016.

[47] S. Yoshii, M. Müller, H. Inoue, R. Ohshima, M. Althammer, Y. Ando, H. Huebl, and M. Shiraishi, "Significant modulation of gilbert damping in ultrathin ferromagnetic films by altering the surface magnetic anisotropy," *Physical Review B*, vol. 109, no. 2, p. L020406, 2024.

[48] Y. Zhao, Q. Song, S.-H. Yang, T. Su, W. Yuan, S. S. Parkin, J. Shi, and W. Han, "Experimental investigation of temperature-dependent gilbert damping in permalloy thin films," *Scientific reports*, vol. 6, no. 1, p. 22890, 2016.

[49] R. E. Sheriff and D. Williams, "Nuclear gyromagnetic ratios. iii," *Physical Review*, vol. 82, no. 5, p. 651, 1951.

[50] B. J. Roth, "Biomagnetism: The first sixty years," *Sensors*, vol. 23, no. 9, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/9/4218

[51] D. Brenner, S. Williamson, and L. Kaufman, "Visually evoked magnetic fields of the human brain," *Science*, vol. 190, no. 4213, pp. 480–482, 1975.

[52] D. Cohen, "Magnetoencephalography: detection of the brain's electrical activity with a superconducting magnetometer," *Science*, vol. 175, no. 4022, pp. 664–666, 1972.

[53] R. Fenici, D. Brisinda, and A. M. Meloni, "Clinical application of magnetocardiography," *Expert review of molecular diagnostics*, vol. 5, no. 3, pp. 291–313, 2005.

[54] R. F. L. Evans. (2013) Vampire 6 - atomistic simulation of magnetic materials. [Online]. Available: https://vampire.york.ac.uk/

[55] R. F. Evans, W. J. Fan, P. Chureemart, T. A. Ostler, M. O. Ellis, and R. W. Chantrell, "Atomistic spin model simulations of magnetic nanomaterials," *Journal of Physics: Condensed Matter*, vol. 26, no. 10, p. 103202, 2014.

[56] G. Who?, "The GNU general public license v3.0 - GNU project - free software foundation," https://www.gnu.org/licenses/gpl-3.0.en.html, accessed: 2024-5-6.

[57] M. Dale, "Spinspired: Source code related to the EPSRC spinspired project (2018-22). related blog: https://spinspired900669414.wordpress.com/."

[58] R. F. L. Evans, U. Atxitia, and R. W. Chantrell, "Quantitative simulation of temperature-dependent magnetization dynamics and equilibrium properties of elemental ferromagnets," *Phys. Rev. B*, vol. 91, p. 144425, Apr 2015. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevB.91.144425

[59] A. F. Atiya and A. G. Parlos, "New results on recurrent network training: unifying the algorithms and accelerating convergence," *IEEE transactions on neural networks*, vol. 11, no. 3, pp. 697–709, 2000.

[60] “Enterprise open source and linux,” https://ubuntu.com/.

[61] “Welcome to,” https://www.python.org/.

[62] “PyCharm: the python IDE for data science and web development,” https://www.jetbrains.com/pycharm/.

[63] “POV-ray - the persistence of vision raytracer,” https://www.povray.org/.

[64] “OpenSSH,” https://www.openssh.com/.

[65] OpenSSL Foundation, Inc, “/index.html,” https://www.openssl.org/.

[66] “Open MPI: Open source high performance computing,” https://www.open-mpi.org/.

[67] I. Free Software Foundation, “Bash - GNU project - free software foundation,” https://www.gnu.org/software/bash/.

[68] “pip: The python package installer.”

[69] “NumPy documentation — NumPy v1.26 manual,” https://numpy.org/doc/stable/index.html.

[70] “Pillow,” https://pillow.readthedocs.io/en/stable/.

[71] “Matplotlib documentation — matplotlib 3.8.4 documentation,” https://matplotlib.org/stable/.

[72] “Tqdm: :zap: A fast, extensible progress bar for python and CLI.”

A Test Bed

Simulation Parameters	
Integrator	LLG
Dipole solver	Tensor
Time step	$20ps$
Step increment	25
Time steps	$25,000$
Input length	1000
Applied field	$0T$
Field Vector	$(0, 0, 1)$

Material Properties				
property	Ni	Co	Fe	units
crystal structure	fcc	fcc	bbc	-
unit cell size	3.524	2.507	2.866	Å
atomic spin moment	0.606	1.72	2.22	μ_B
exhange energy J_{ij}	$2.757 \cdot 10^{-21}$	$6.064 \cdot 10^{-21}$	$7.050 \cdot 10^{-21}$	$J/link$
2^{nd} order uniaxial anisotropy	$5.470 \cdot 10^{-26}$	$6.690 \cdot 10^{-24}$	$5.650 \cdot 10^{-25}$	$J/atom$
Temperature rescaling exponent	2.322	2.369	2.876	-
Rescaling Curie temperature	635	1395	1049	K

B Exploration Tables

B.1 Static Parameters

Search label	Name	T(K)	Input matrix	Input No.	Signal strength	x×y(nm)	z(unit cells)
Concept Validation	A1	0	random	2	1	50×50	1
	A2	0	uniform	all	1	50×50	1
	A3	309.65	uniform	all	1	50×50	1
Biomagnetic Magnitude	B1	309.65	uniform	all	$1 \cdot 10^{-13}$	50×50	1
	B2	309.65	uniform	all	$2 \cdot 10^{-12}$	50×50	1
	B3	309.65	uniform	all	$75 \cdot 10^{-12}$	50×50	1
System Height	H1	309.65	uniform	all	$1 \cdot 10^{-13}$	50×50	explored
System Dimensions	C1	309.65	uniform	all	$1 \cdot 10^{-13}$	75×75	1
	C2	309.65	uniform	all	$1 \cdot 10^{-13}$	100×100	1
Strip Geometries	S1	309.65	uniform	all	$1 \cdot 10^{-13}$	50×100	1
	S2	309.65	uniform	all	$1 \cdot 10^{-13}$	40×100	1
	S3	309.65	uniform	all	$1 \cdot 10^{-13}$	30×100	1

B.2 Exploration Ranges

Search label	Name	Mat.	Magnetic damping	Input scaling	Mcells	Comb.
Concept Validation	A1	All	0.001 : 0.501, 0.00263	0.1 : 2, 0.525	2.5, 5, 7.5	600
	A2	All	0.001 : 0.501, 0.00263	0.1 : 2, 0.525	2.5, 5, 7.5	600
	A3	All	0.001 : 0.501, 0.00263	0.1 : 2, 0.525	2.5, 5, 7.5	600
Biomagnetic Magnitude	B1	All	0.001 : 0.501, 0.00263	0.1 : 2, 0.525	2.5, 5, 7.5	600
	B2	All	0.001 : 0.501, 0.00263	0.1 : 2, 0.525	2.5, 5, 7.5	600
	B3	All	0.001 : 0.501, 0.00263	0.1 : 2, 0.525	2.5, 5, 7.5	600
Film Height	H1	Co	0.1852 : 0.4747, 0.14475	0.1 : 2, 0.95	2.5, 5	108
	H1	Fe	0.0799 : 0.2642, 0.09215	1.05 : 2, 0.475	2.5, 5	108
	H1	Ni	0.0799 : 0.4221, 0.1711	0.1 : 2, 0.95	5	54
System Dimensions	C1	Co	0.351 : 0.451, 0.05	0.5 : 1.6, 0.55	2.5, 5, 7.5	75
	C1	Fe	0.001 : 0.501, 0.1255	0.1 : 2, 0.525	2.5, 5, 7.5	75
	C1	Ni	0.001 : 0.501, 0.1255	0.1 : 2, 0.525	2.5, 5, 7.5	75
	C2	Co	0.001 : 0.501, 0.1255	0.1 : 2, 0.525	2.5, 5, 10	75
	C2	Fe	0.001 : 0.501, 0.1255	0.1 : 2, 0.525	2.5, 5, 10	75
	C2	Ni	0.001 : 0.501, 0.1255	0.1 : 2, 0.525	2.5, 5, 10	75
Strip Geometries	S1	All	0.001 : 0.501, 0.1255	0.1 : 2, 0.525	2.5, 5, 10	225
	S2	All	0.001 : 0.501, 0.1255	0.1 : 2, 0.525	2.5, 5, 10	225
	S3	All	0.001 : 0.501, 0.1255	0.1 : 2, 0.525	2.5, 5, 10	225

B.3 Best Run Configurations

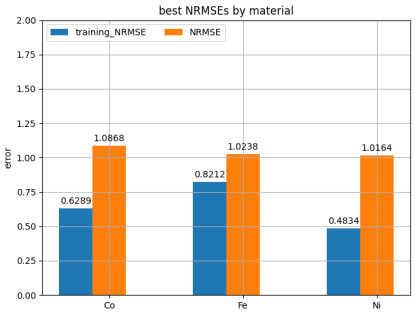
Search label	Name	Material	Magnetic damping	Input scaling	Mcells	Trials
Concept Validation	A1	Co	0.501	0.575	5	40
	A1	Fe	0.501	0.1	5	40
	A1	Ni	0.3694	0.1	5	40
	A2	Co	0.209	0.1	2.5	20
	A2	Fe	0.2115	0.1	2.5	20
	A2	Ni	0.0799	0.1	5	20
	A3	Co	0.2642	0.1	5	20
	A3	Fe	0.0799	1.05	5	20
	A3	Ni	0.501	0.575	5	20
Biomagnetic Magnitude	B1	Co	0.3957	2.0	2.5	20
	B1	Fe	0.2642	1.05	5	20
	B1	Ni	0.3431	1.05	5	20
	B2	Co	0.4484	1.05	2.5	20
	B2	Fe	0.1852	0.1	5	20
	B2	Ni	0.2642	1.05	5	20
	B3	Co	0.4484	0.1	2.5	20
	B3	Fe	0.1589	0.1	5	20
	B3	Ni	0.3957	1.05	5	20

Search label	Name	Material	Magnetic damping	Input scaling	Mcells	Height	Trials
Film Height	H1	Co	0.4747	0.1	5	15.042	10
	H1	Fe	0.0799	1.05	5	11.464	10
	H1	Ni	0.0799	1.05	5	7.048	10

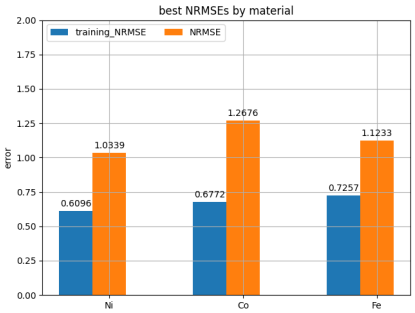
Search label	Name	Material	Magnetic damping	Input scaling	MCells	Trials
System Dimensions	C1	Co	0.501	0.1	5	5
	C1	Fe	0.126	2.0	7.5	5
	C1	Ni	0.126	1.05	7.5	5
	C2	Co	0.501	0.1	10	5
	C2	Fe	0.376	0.1	5	5
	C2	Ni	0.501	0.1	10	5
Strip Geometries	S1	Co	0.501	1.05	10	5
	S1	Fe	0.126	0.1	5	5
	S1	Ni	0.251	2.0	5	5
	S2	Co	0.501	0.1	10	5
	S2	Fe	0.126	2.0	5	5
	S2	Ni	0.501	2.0	5	5
	S3	Co	0.501	1.05	10	5
	S3	Fe	0.126	0.575	5	5
	S3	Ni	0.501	0.575	10	5

C Results

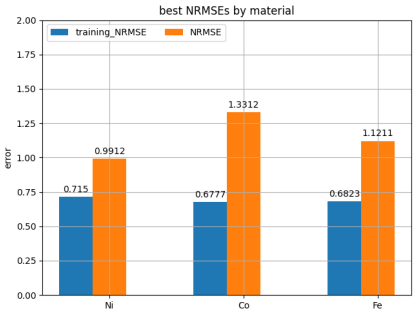
C.1 Concept Validation Results



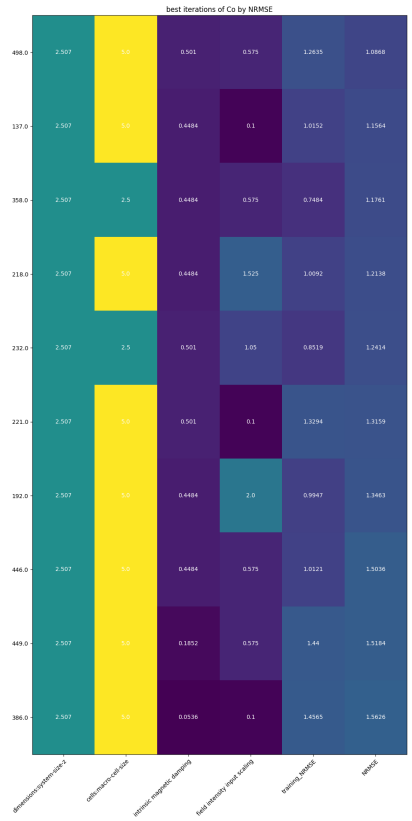
(a) A1



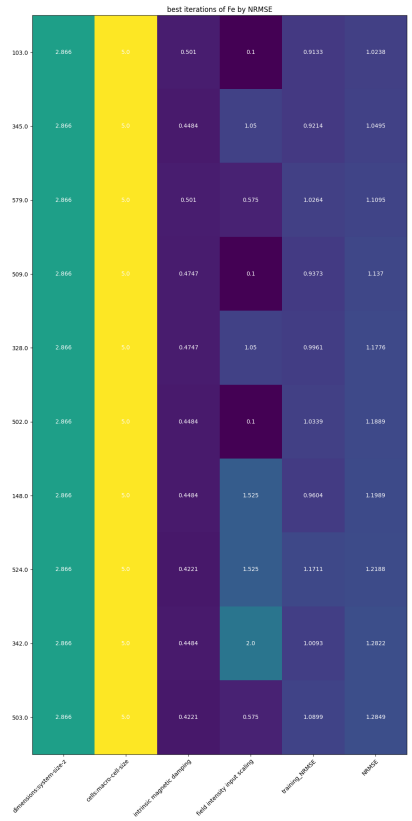
(b) A2



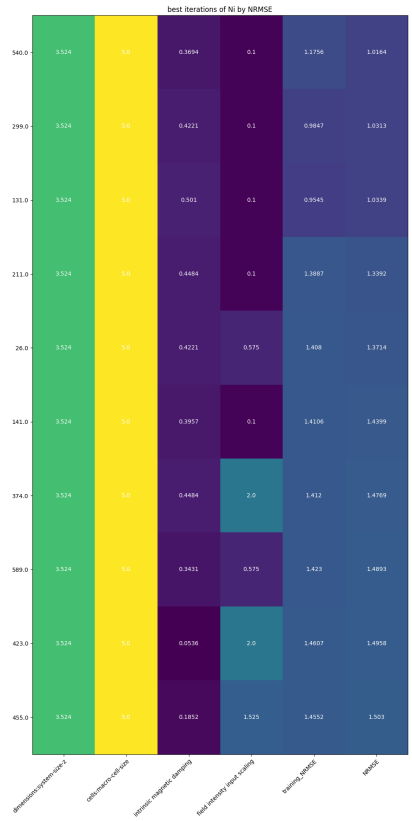
(c) A3



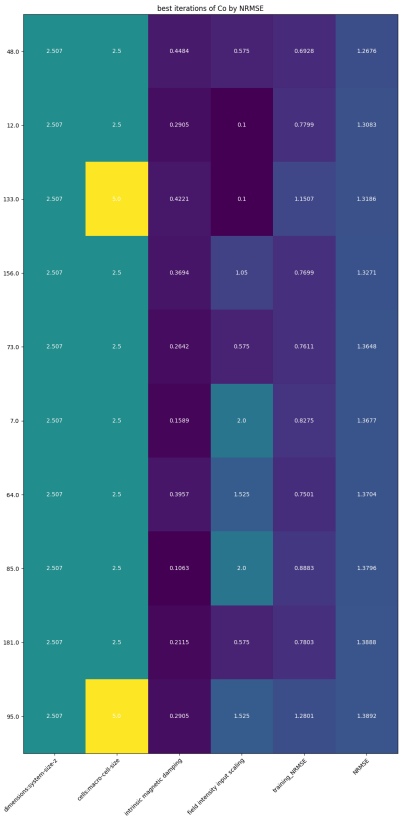
(a) A1: Co



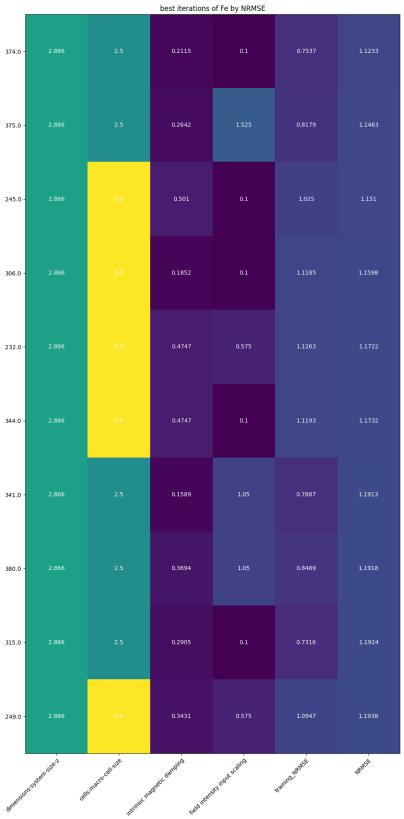
(b) A1: Fe



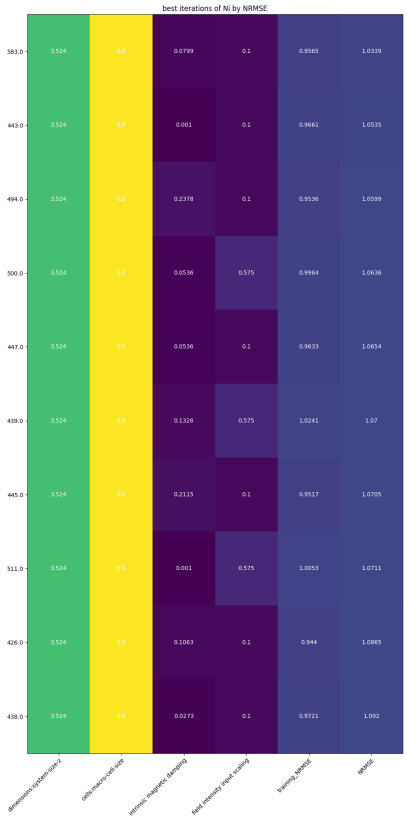
(c) A1: Ni



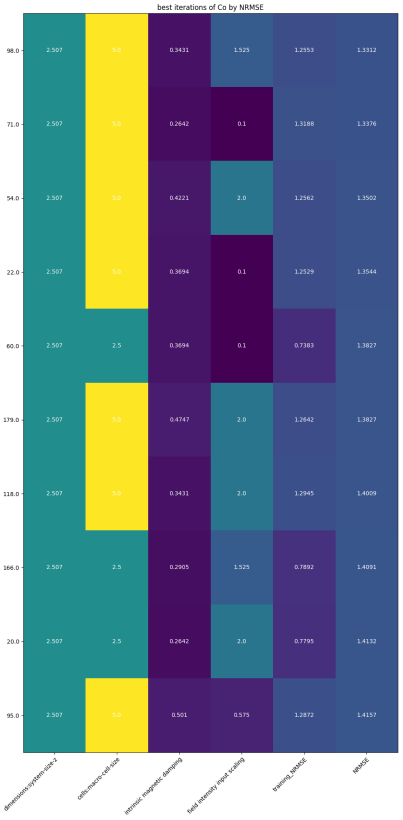
(a) A2: Co



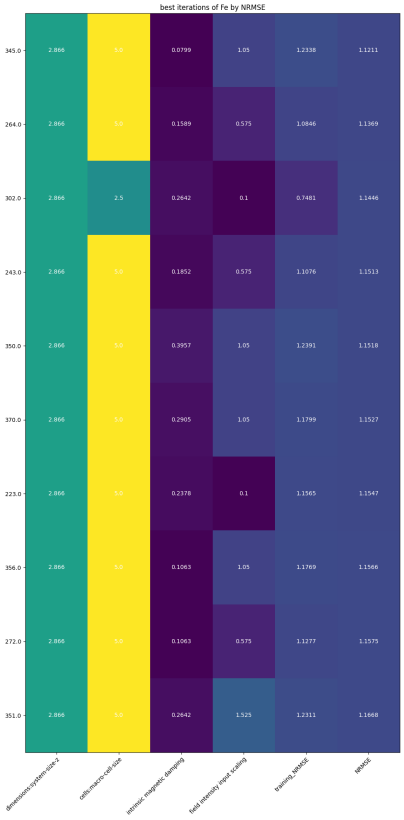
(b) A2: Fe



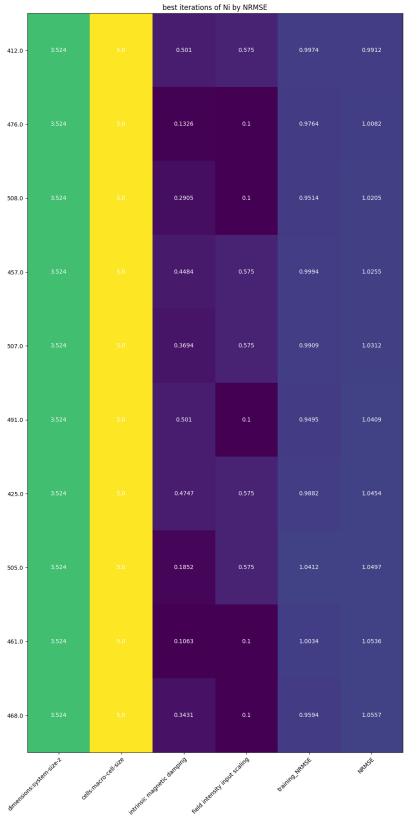
(c) A2: Ni



(a) A3: Co

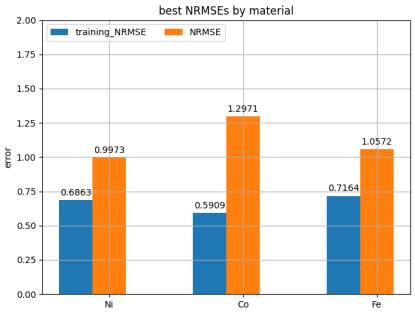


(b) A3: Fe

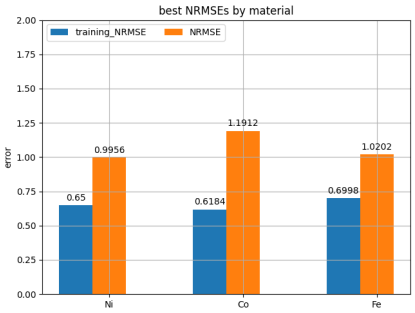


(c) A3: Ni

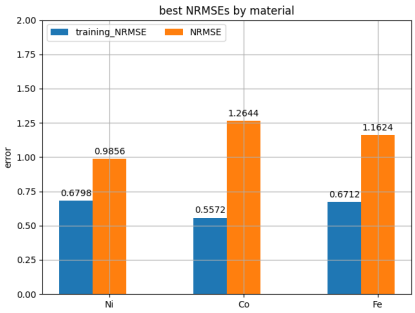
C.2 Biomagnetic Magnitude Results



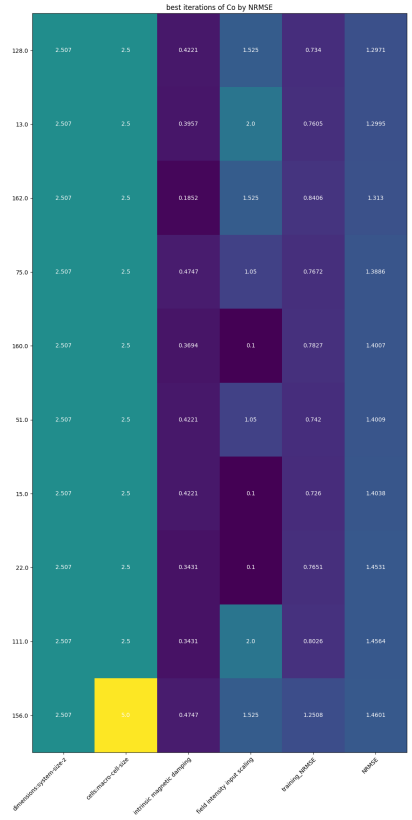
(a) B1



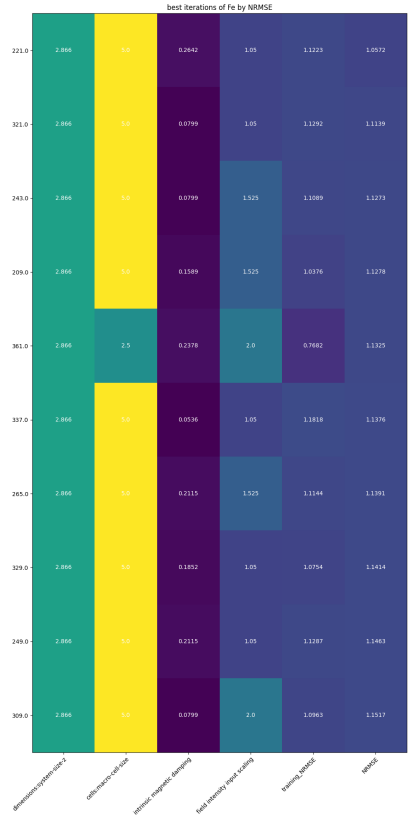
(b) B2



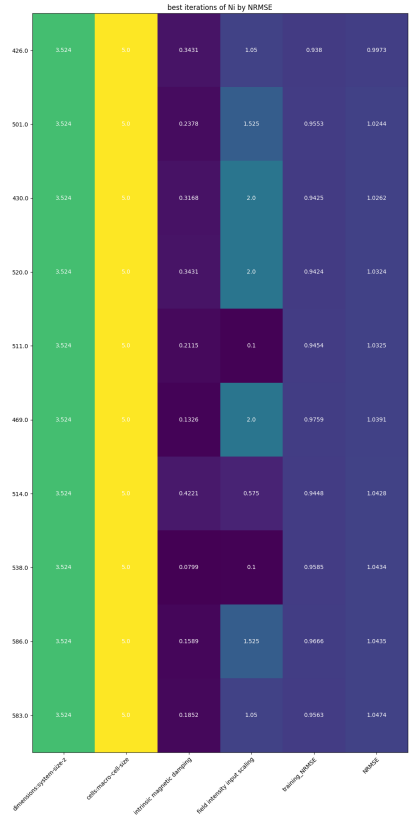
(c) B3



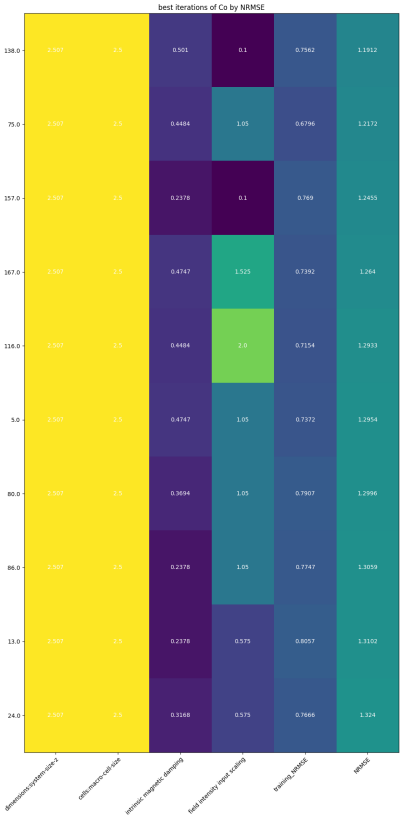
(a) B1: Co



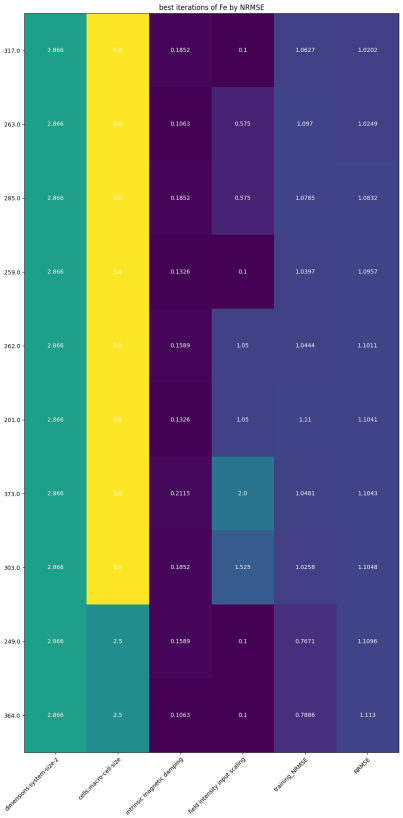
(b) B1: Fe



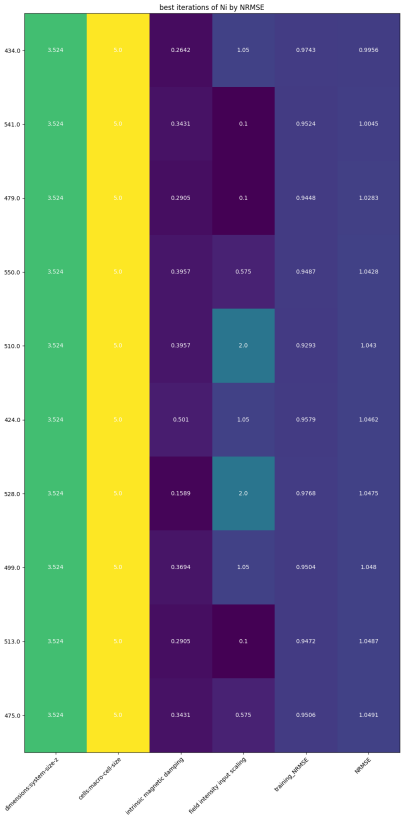
(c) B1: Ni



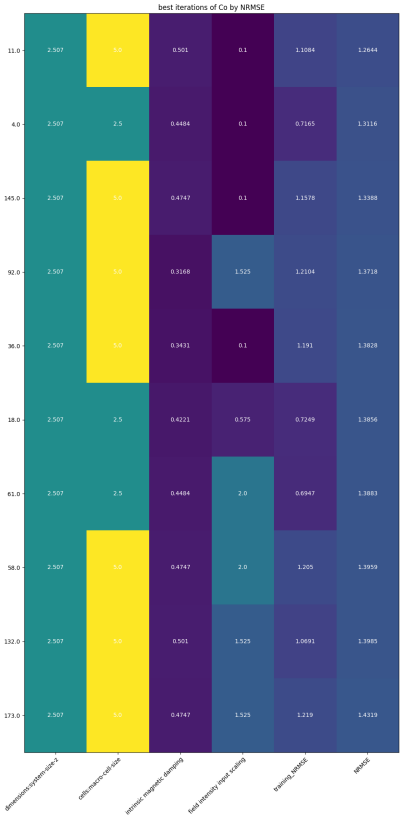
(a) B2: Co



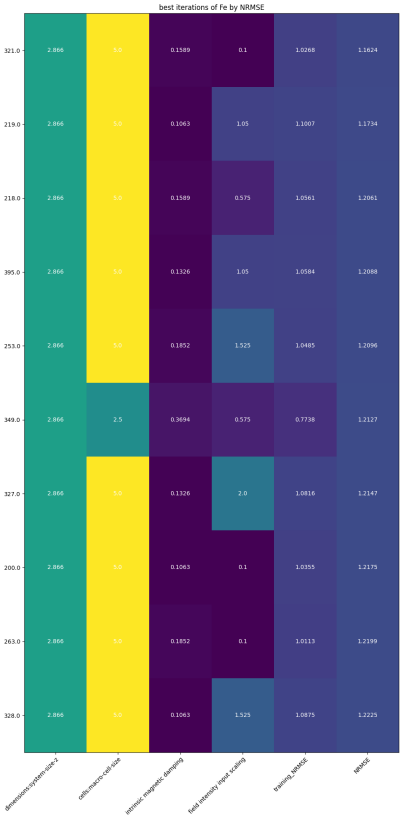
(b) B2: Fe



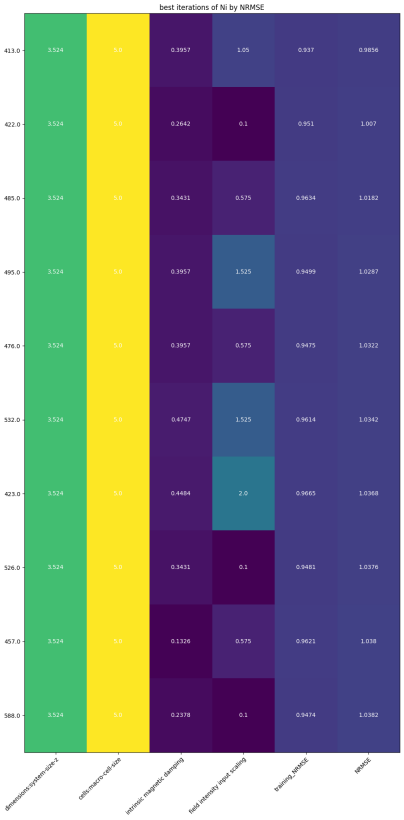
(c) B2: Ni



(a) B3: Co



(b) B3: Fe



(c) B3: Ni

C.3 System Dimensions Results

C.3.1 Z Dimension

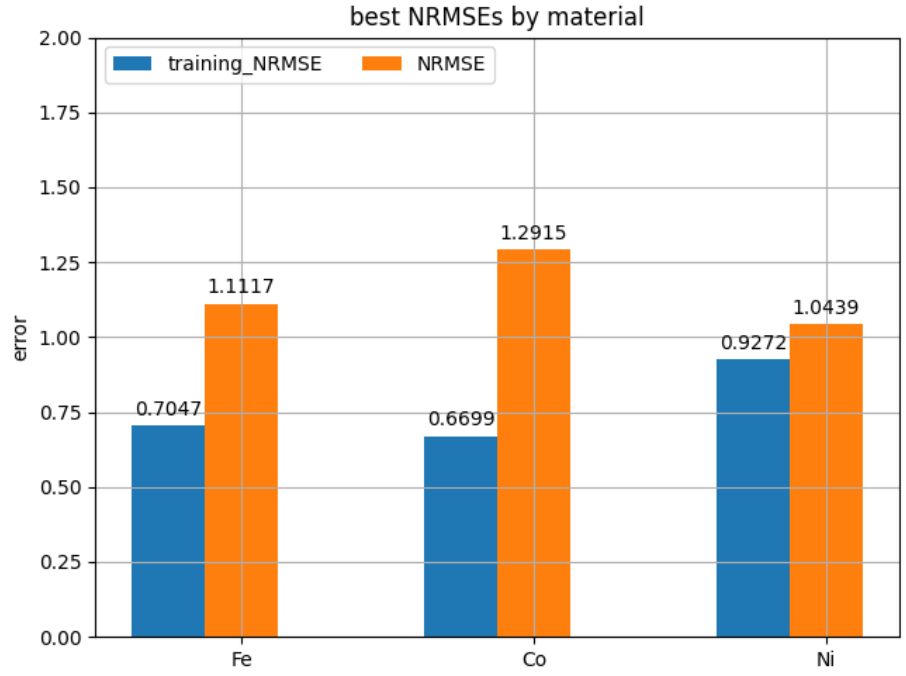
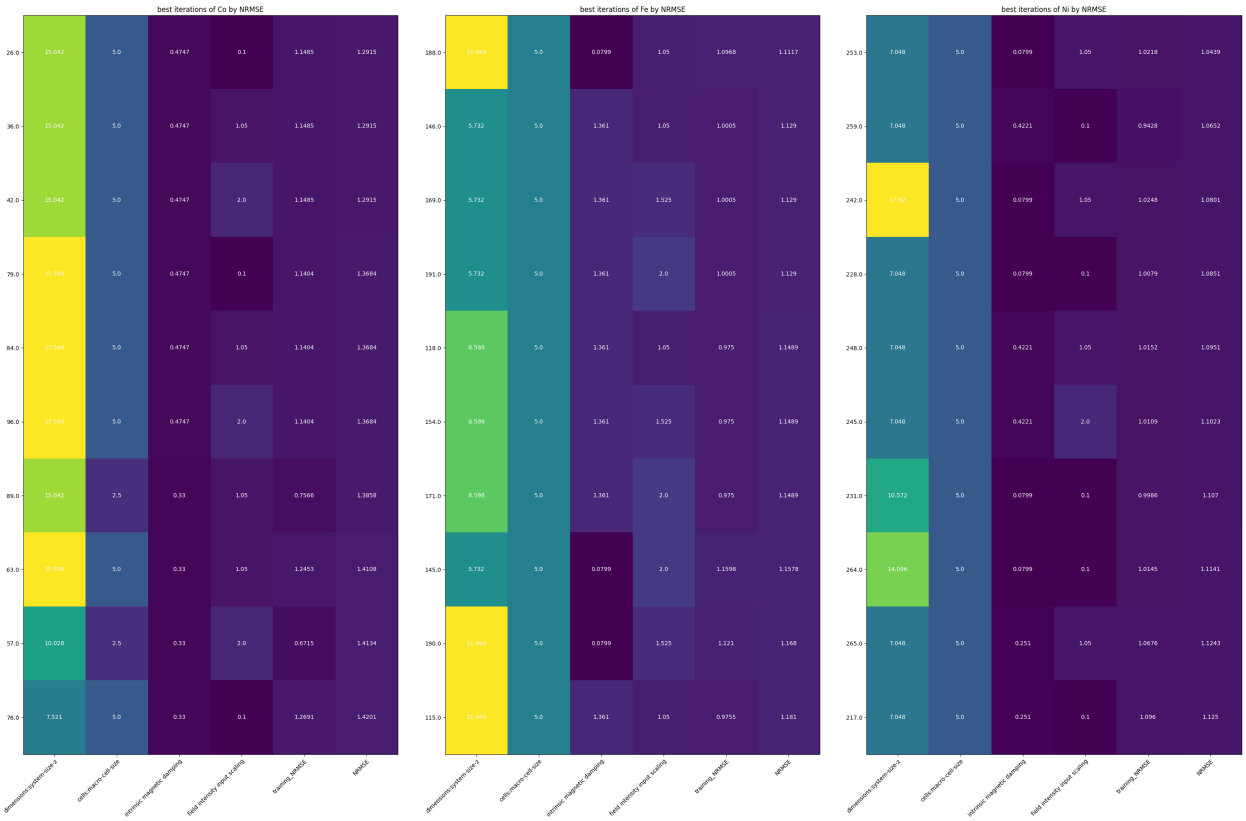
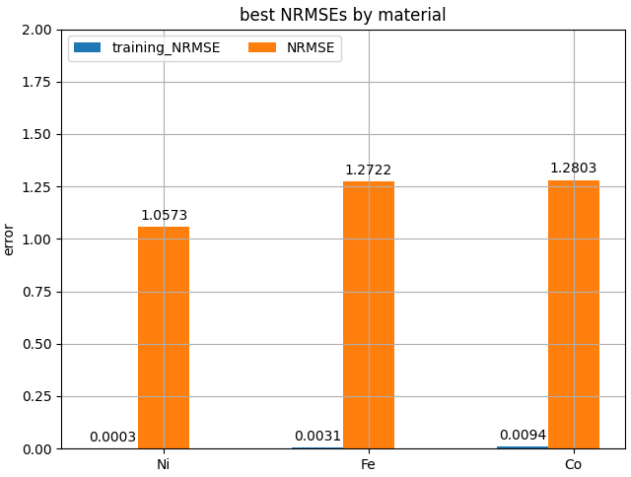


Figure 23: H1

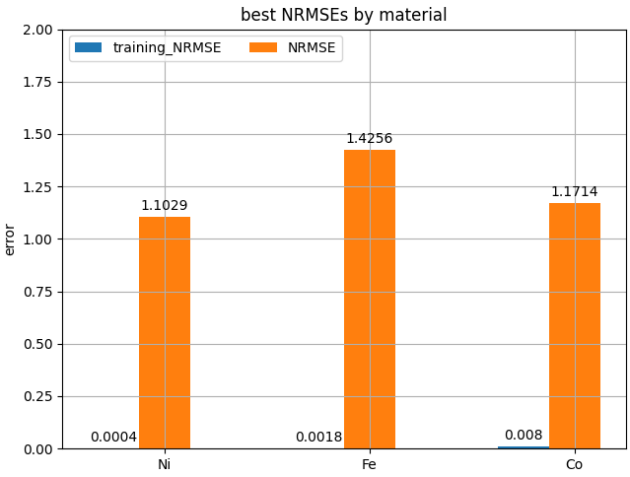


(a) H1: Co (b) H1: Fe (c) H1: Ni

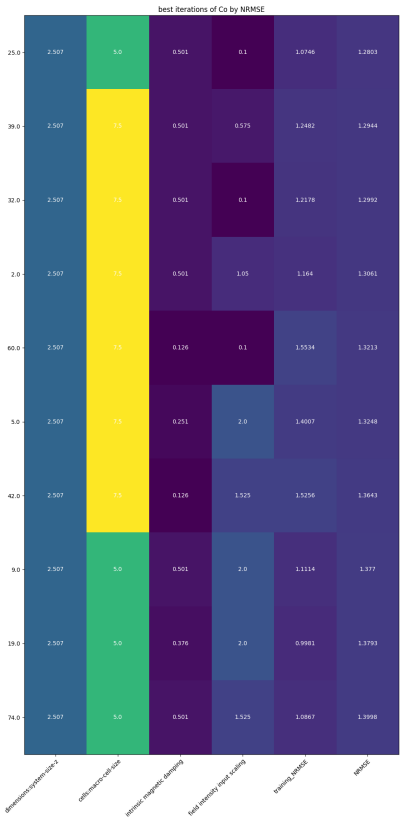
C.3.2 X,Y Dimensions



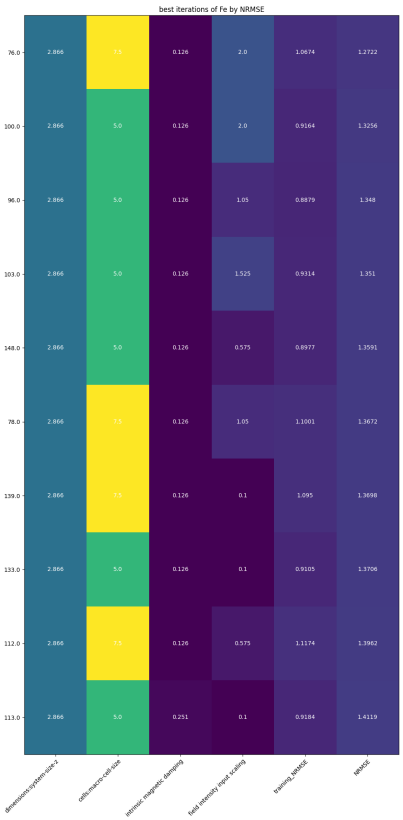
(a) C1



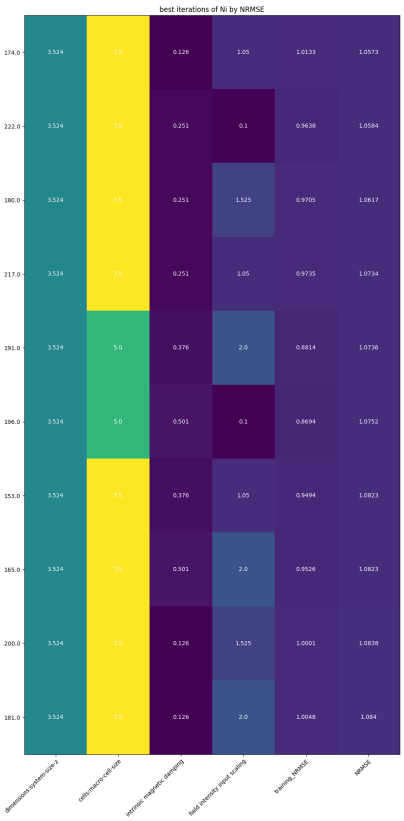
(b) C2



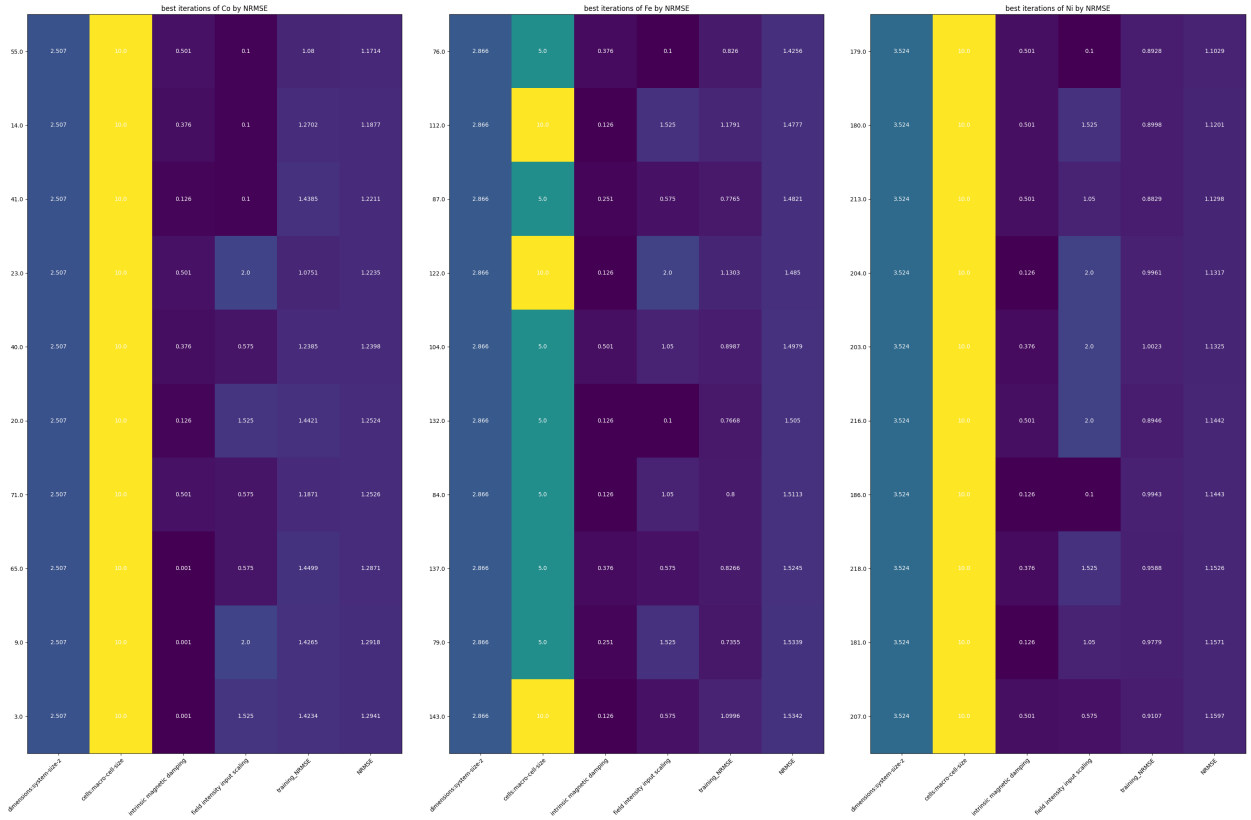
(a) C1: Co



(b) C1: Fe

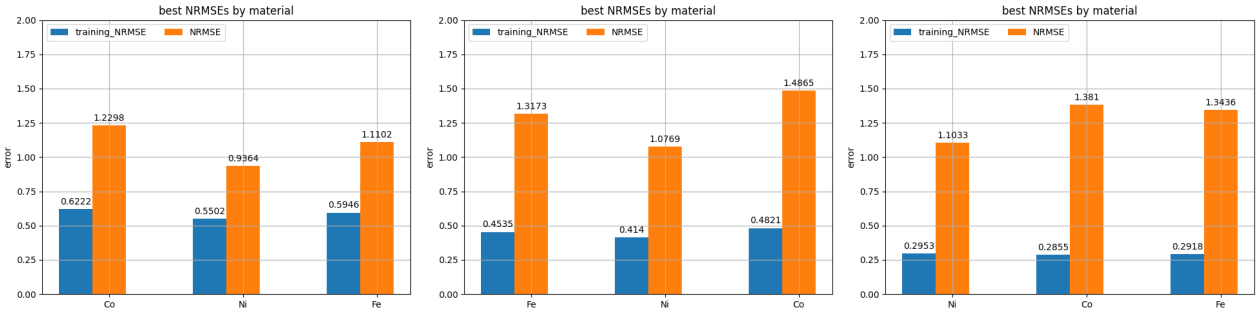


(c) C1: Ni

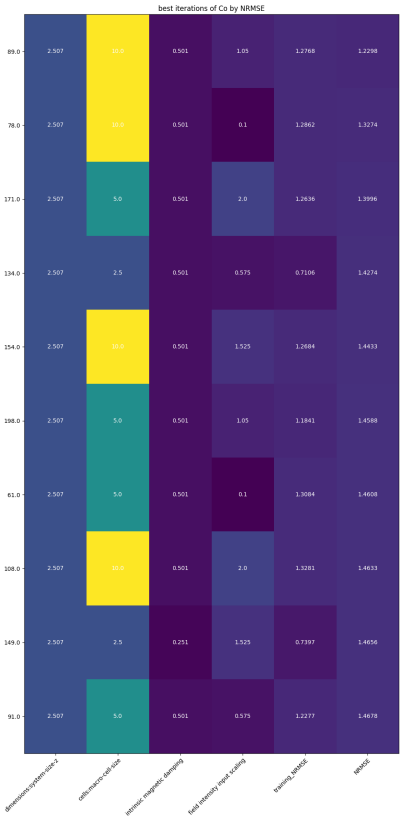


(a) C2: Co (b) C2: Fe (c) C2: Ni

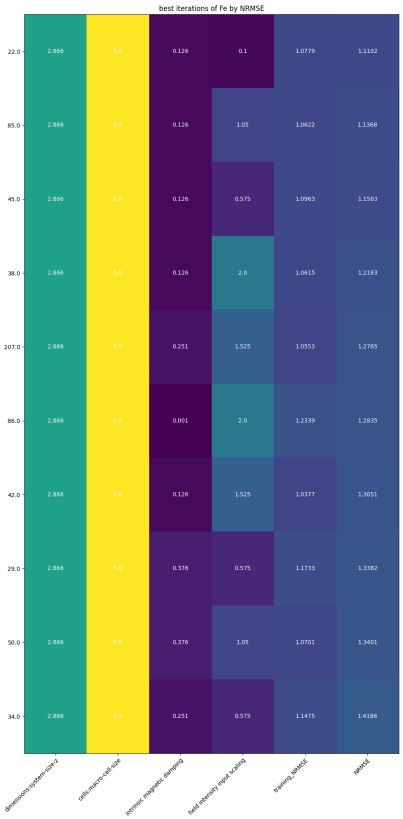
C.4 Strip Geometries Results



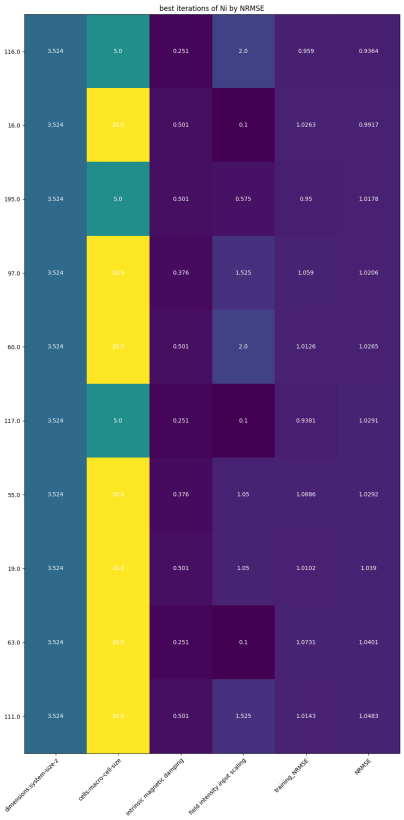
(a) S1 (b) S2 (c) S3



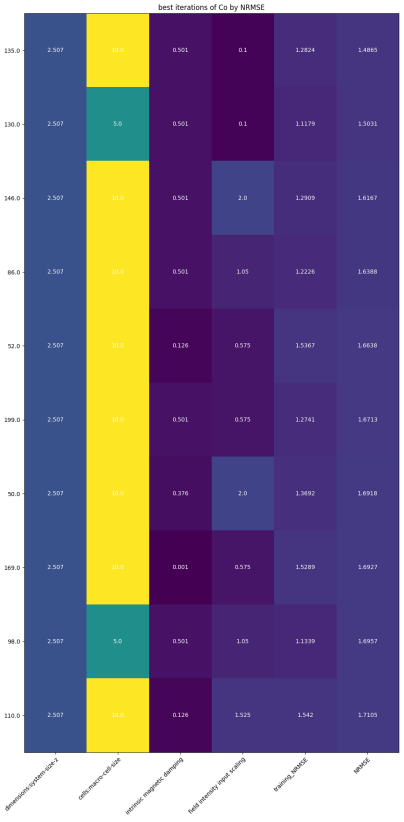
(a) S1: Co



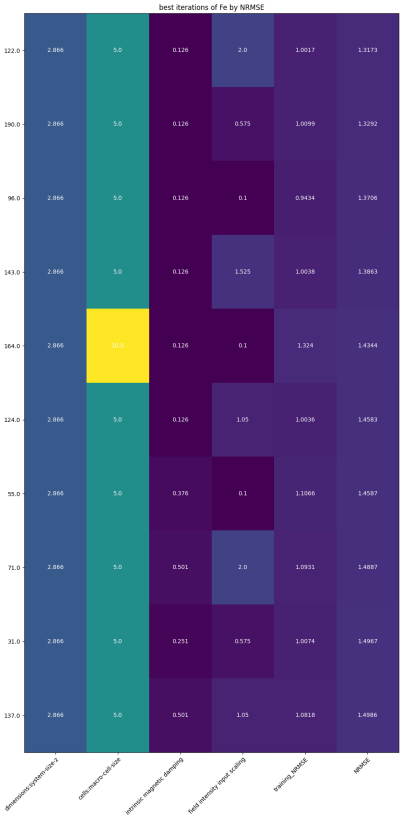
(b) S1: Fe



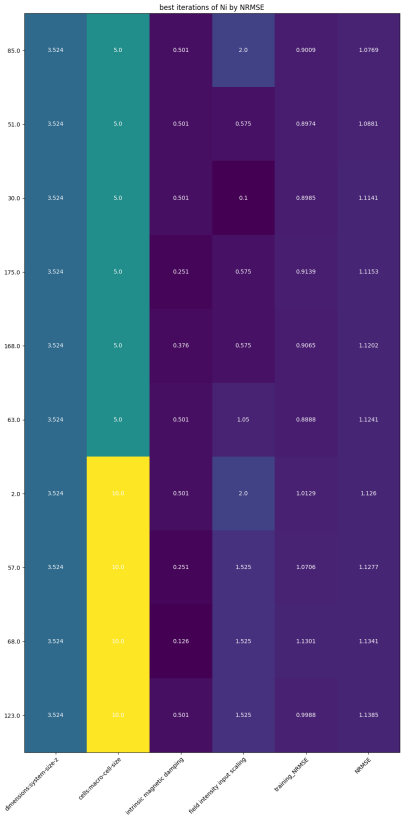
(c) S1: Ni



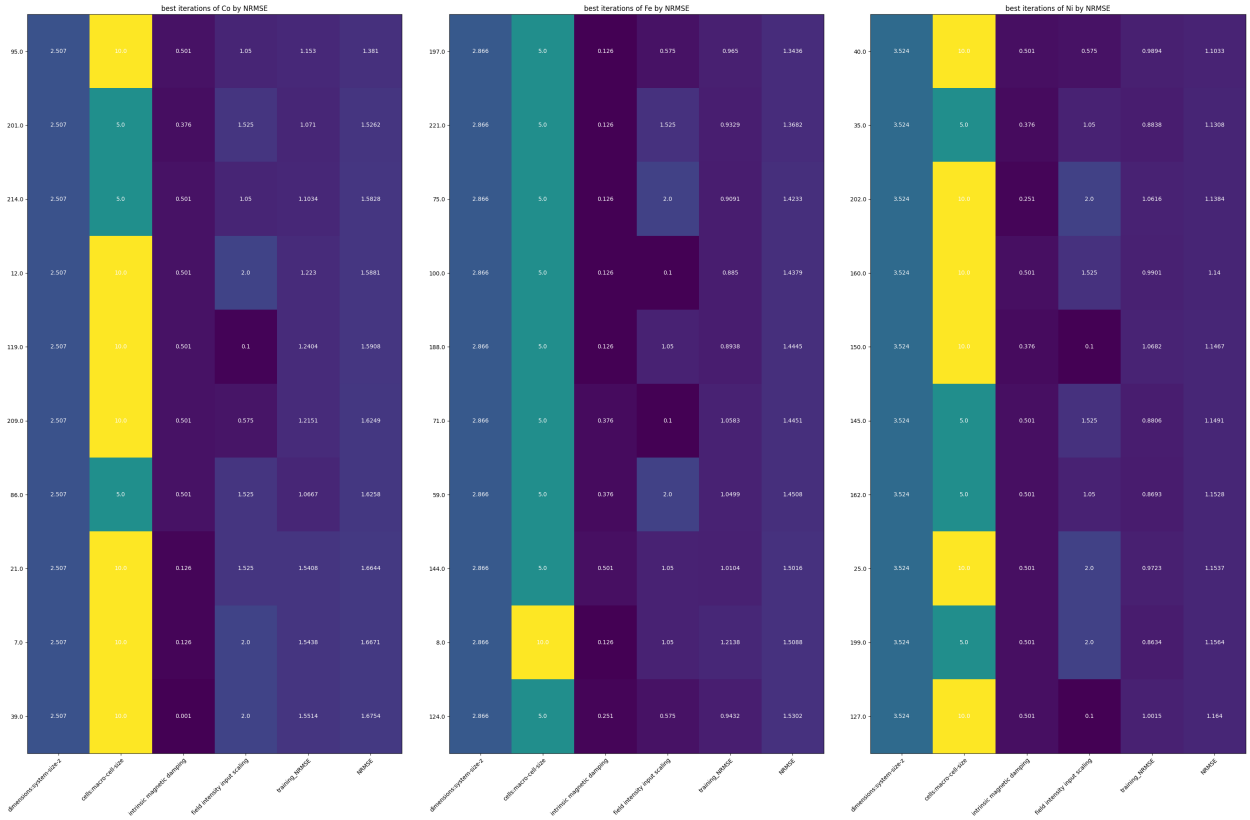
(a) S2: Co



(b) S2: Fe



(c) S2: Ni



(a) S3: Co (b) S3: Fe (c) S3: Ni

D Third-Party Software

This section is a simple enumeration of the main software packages and Python libraries required to develop and thereafter use the system I developed. It is worth noting that several if not all of these software and libraries have many dependencies. which I have chosen to omit. All project software is run in Ubuntu, known for its performance optimisation and efficiency, comes with Python pre-installed and is the intended for running Vampire.

Software	Version	Function	run method
Ubuntu [60]	22.04.4	OS	OS
Python [61]	3.10.12	Random search algorithm	IDE
Pycharm [62]	2024.1.1	Python IDE	Executable
Vampire [54]	5.0.0	Atomistic simulation	Console
Povray [63]	3.6.1	Simulation plotting	Console
OpenSSH [64]	8.9p1	Server communication	Console
OpenSSL [65]	3.0.2	Server communication	OpenSSH Invocation
OpenMPI [66]	4.1.5	Vampire parallelisation	Console
Bash [67]	5.1.16	Shell	Console

Table 4: Main software packages.

I have only included the non-standard libraries that aren't automatically included with the

Python distribution. All packages were installed with either pip [68] or the Pycharm package manager.

Library	Version
`numpy` [69]	1.26.4
`pillow` [70]	10.3.0
`matplotlib` [71]	3.8.3
`tqdm` [72]	4.66.2

Table 5: Essential Python libraries.

E Acquiring Vampire

The original distribution of Vampire can be compiled from source via makefile after cloning from the Vampire repository, with the following shell commands:

```
git clone git://github.com/richard-evans/vampire.git
cd vampire
make all
```

The most recent version at the time of writing is version 6.0.0. To acquire Dale's modified distribution, which builds upon version 5.0.0, can be acquired and installed with similar steps:

```
git clone git://github.com/MaterialMan/Spinspired
cd 'Support files'/vampire/'Source files'/
unzip 'build files.zip'
cd .. ; cd .. ; cd .. ; cd 'build files'
make all
```

F Plotting Data

F.1 Exploration Plots

After the search ends all of the iteration files are saved to `VAMPIRE_TEST_RESULTS/` in their own folder, named after the iteration during which they were obtained. The function `recoup_data` from `RecoupData.py` parses over and through all of these folders and extracts the simulation parameters and results from the `accuracy_scores.txt` file contained within each folder. It then collates all of this information and writes to a file called `best_iterations.txt`, which acts as a record of all search data. It skips over iterations which have failed and allows for a threshold to be set, only saving runs which produced a lower NRMSE. The script `CreatePlots.py` houses four functions for plotting the data contained within the best iterations file. The first, `create_plot_data`, extracts the data and applies some preprocessing before invoking the three plotter functions:

- `plot_XY`: performs a cartesian product to create all possible parameter combinations and then plots them as an XY trace.

- `plot_table`: used to assess the effects of all parameters that make up a combination on the NRMSE and training NRMSE. These plots are in effect heatmaps with iteration number on the y-axis and parameters on the x-axis, arranged from top to bottom in ascending order of NRMSE.

- `material_comparison`: plots a comparison of the NRMSE and training NRMSE between different materials as a combined bar graph, with two bars per material.

The last type of plotted data are box plots which are generated directly after the `TrialBest` class from `TrialBest.py` has tested each best run for each material 40 times. The plotting of search data could have been automated as part of the search algorithm, but I chose to keep the search, data extraction and plotting as separate processes to have a greater degree of control and customisation.

F.2 The VDC

The VDC binary is a Vampire tool for converting output data into plottable formats, such as GNUplot, Rasmol and Povray, of which I used the latter for this project. Converting data is very simple as it only requires an invocation with a flag denoting the desired format:

```
VDC --povray
```

This will clutter the current directory with files for plotting alongside a `spins.pov` file, from which the main plot parameters can be set, such as camera and light source positioning, colours etc. To plot scenes using povray you must specify which frames within the timeseries to plot, as well as the desired dimensions:

```
povray Height=1000 Width=1600 +KFI0 +KFF499 +A0.3 spins.pov +P
```

This command would plot 500 scenes of dimensions 1600×1000. The `A0.3` flag is the anti-aliasing threshold, whereas `+P` provides a low resolution preview, so you can see what is currently being rendered.