CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Active Learning for Surrogate Models to Augment AI-Driven Molecular Design

Master's thesis in Computer science and engineering

CHRISTIAN JOSEFSON
CLARA NYMAN

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

Active Learning for Surrogate Models to Augment AI-Driven Molecular Design

CHRISTIAN JOSEFSON
CLARA NYMAN

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

Active Learning for Surrogate Models to Augment AI-Driven Molecular Design

CHRISTIAN JOSEFSON
CLARA NYMAN

Industrial Supervisor: Jeff Guo, AstraZeneca
Academic Supervisor: Morteza Haghir Chehreghani, Department of Computer Science and Engineering
Examiner: Devdatt Dubhashi, Department of Computer Science and Engineering

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2022

Active Learning for Surrogate Models to Augment AI-Driven Molecular Design

CHRISTIAN JOSEFSON
CLARA NYMAN
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

This project investigated whether an active learning (AL) framework can help mitigate computational costs for AI-driven molecular design, without negatively impacting accuracy. The surrogate models Random Forest (RF) and Support Vector Regression (SVR) were tested together with the acquisition functions (AF) Random, Thompson Sampling (TS), Tanimoto Similarity, Expected Improvement (EI), Probability of Improvement (PI), Upper Confidence Bound (UCB) and $\varepsilon-$Greedy. Of these, the combination RF and Random acquisition were concluded to perform the best with regards to error rate, measured as root mean square error, and time consumption, measured in runtime per epoch. SVR had slightly lower error, but took substantially longer time. Depending on the choice of AF, one run using RF took approximately 2-17.5 hours, while one run using SVR took approximately 100-175 hours. Four tuning parameters were introduced to see if they could further optimize the framework. It was discovered that a longer retrain interval and a smaller acquisition batch did not significantly impact accuracy while shortening the time consumption. To summarise, an RF model with the Random AF with a 5 epoch initial pooling, no warm-up phase, a retrain interval of 20 and an acquisition batch size of 20 was selected to mitigate computational costs while simultaneously keeping the error stable.

Keywords: active learning, bayesian optimization, de novo design, molecular design, drug discovery, surrogate model, machine learning, molecular docking

Acknowledgements

There are many people we wish to thank for helping us in finishing this Master's Thesis. In particular, we are deeply indebted to our brilliant supervisor, Jeff Guo, whose patience and expertise were invaluable to us throughout the project.

We would also like to give thanks to our other supervisors at AstraZeneca who helped guide us, Hampus Gummesson, Christian Margreitter, Samuel Genheden and Pallavi Banerjee, as well as our academic supervisor, Morteza Haghir Chehreghani. We also extend a thank you to Ola Engkvist for believing in us in the first place and giving us the chance to do this project.

Additionally, we want to express our gratitude to Harry Moore and Jon Paul Janet for helping us along and providing their much needed expertise. Also thank you to to our opponents, Caroline Bükk and Linda Hoang.

And lastly, we would like to express our gratitude to our friends and loved ones who provided inspiration and whose comments, criticisms, and suggestions have made this project so much better than it otherwise would have been. Special thanks to Daniel Andersson who kept insisting we knew what we were doing, despite all evidence to the contrary.

Christian Josefson & Clara Nyman, Gothenburg, July 2022

Contents

Glossary

AB	Acquisition Batch
AF	Acquisition Function
AL	Active Learning
ECFP	Extended Connectivity Fingerprints
EI	Expected Improvement
FTS	Fingerprint Tanimoto Similarity
GRU	Gated Recurrent Unit
LSTM	Long Short-Term Memory
ML	Machine Learning
MPO	Multi-Parameter Optimization
PI	Probability of Improvement
QED	Quantitative Estimate of Drug likness
RF	Random Forest
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RMSE	Root Mean Squared Error
SMILES	Simplified Molecular Input Line System
SVM	Support Vector Machine
SVR	Support Vector Regression
TS	Thompson Sampling
UCB	Upper Confidence Bound

1

Introduction

The fact that the development of new drugs is important to humanity is hardly hard to argue, especially now when the world has recently gone through one of the greatest health crises in modern time. However, drug design is an expensive and time consuming process.[1] This is due to many factors, but the one this project wishes to mitigate is the computational cost of molecular design by circumventing the use of one of the more computationally expensive aspects of it: calculating if a molecule can interact in the desired way with a protein.

This introductory chapter attempts to give the reader a relevant background to some of the project's key concepts such as active learning, Bayesian optimization and the tool used for the molecular generation and design, REINVENT. Then it moves on to expand on the project's purpose and goal as well as the demarcations of the project. Lastly, ethical considerations and potential societal impacts are touched upon.

1.1 Background

The set of all possible biologically relevant molecules, defined collectively as the "drug-like" chemical space, is estimated to be on the order of 10^{23} to 10^{60} number of molecules[2]. This immense size entails a difficulty in finding one's way within the space. *De novo* molecular design means to generate a new molecule from scratch, and includes traversing chemical space in the quest for a set of molecules with bespoke properties. Thus, any tool that enables one to more easily find a way within chemical space will be beneficial for *de novo* molecular design.

Predictive modelling, defined here as computational predictions of molecular properties, can be applied to aid or augment computer simulated (*in silico*) *de novo* molecular design[3], by alleviating brute force search of drug-like chemical space.

Computational oracles enable predictive modelling and are defined as functions that compute the properties of a given input molecule, typically used to identify whether the molecule has desired properties. However, oracles have no way of traversing chemical space efficiently on their own. Often, researchers narrow the search space of molecules manually in order to keep the computation time to manageable levels. This is a major challenge of molecular design: traversing the vast chemical space while simultaneously satisfying a large number of desired properties.

Multi-parameter optimization (MPO) objectives, which are common within molecular design, involve satisfying multiple desired properties. Querying oracles can be

used to steer towards molecules which satisfy the MPOs. Efficient treatment of the corresponding MPO objective enables a much greater number of molecules to be triaged and has shown promise in identifying candidate molecules that have been experimentally validated.[4]

Recently, advances in machine learning (ML) have shown potential in accelerating the rate of molecular discovery. [5] This has been done using generative models[6], genetic algorithms[7], and by learning a molecular latent space[8]. These methods guide molecular design by intelligently identifying the solution space to the chosen MPO objective. The combination of applying ML techniques and querying an oracle has drastically accelerated the design of candidate molecules and has become the status quo in industrial molecular discovery efforts.[9]

In an ideal world, calls to the oracle would be inexpensive enough for molecular property predictions to be queried on an as-needed basis. However, this is rarely the case as oracles typically involve computationally expensive chemistry methods. Examples of these are molecular docking[4], molecular mechanics[10], and molecular dynamics simulations[11], which require extensive CPU and GPU resources.

The use of surrogate models mitigate some computational costs associated with expensive oracles. They do this by predicting the oracle output in order to replace a portion of the oracle calls by instead using these predictions of the likely outcome. [12] This has repeatedly been shown to be effective in reducing computational cost in the context of drug discovery.[13, 14, 15] More details regarding surrogate models can be found in section 2.3.

Querying the oracle according to predetermined strategies has previously yielded significant reductions in amount of training required until the surrogate model achieves satisfactory accuracy[16, 17]. Taken together, these strategies constitute an area of ML called active learning. This will be further expanded upon in subsection 1.1.2.

1.1.1 Bayesian Optimization

Bayesian optimization is a strategy used to optimize objective functions, often when the function is expensive or time consuming to evaluate.[18] It does this by building a probabilistic model of the objective function and selects samples to evaluate the true objective function.[19]

The true objective function is fixed and if the system has unlimited resources, it could simply compute every single point of the objective function to know its actual shape and thus receive perfect information. In reality, however, unlimited resources do not exist, and therefore it can be beneficial to build a probability model instead to act as a surrogate. Using the oracle to calculate a subset of the range of the objective function, the surrogate model is built to approximate the rest of the objective function.

There are many different types of models; the ones used in this project are introduced in section 2.3. To help reduce the amount of samples to be computed, the dataset can be curated to improve accuracy while reducing dataset growth. One way to curate the data is active learning.

1.1.2 Active Learning

Active learning (AL) is a subfield of ML which explores different strategies to increase the accuracy of a model using less data by 'intelligently' picking data points to augment the training data. The key question of AL is how to pick these data points. AL is especially useful within fields where data is abundant but labelling is expensive, since studies have shown that one can use AL to achieve equal accuracy with fewer labelled data.[20] One example of a strategy is to, at every step, pick the of points that the model is most uncertain about. These points are added to the dataset to be trained on, and the cycle continues. [16]

The active learning loop contains four steps: Acquire, Query, Append and Train; see Figure 1.1. Given a dataset of unlabelled data and a trained surrogate model, a subset of the dataset is acquired according to a certain strategy, called an Acquisition Function (AF). This subset is fed to an oracle, which labels it (this is sometimes called querying the oracle) and then this pool of labelled data is appended to the training pool. The model is then trained on this pool of labelled data and the cycle is repeated.

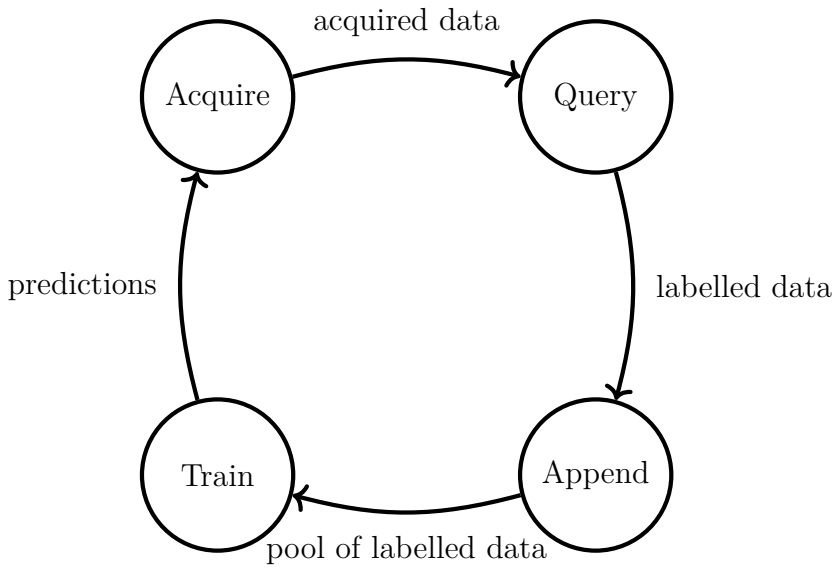

Figure 1.1: Active learning loop. The model acquires points according to a chosen strategy, it labels the acquired points and appends these labelled data to the training pool. The model is trained and the cycle is repeated.

1.1.3 Molecular Generation Tool for Small Molecules

REINVENT is a tool for *de novo* design of small molecules. It is an application that uses a generative model to create small molecules that can be used for discovering new molecules to use in drug design.

REINVENT's generative model uses a dataset from ChEMBL which is a database consisting of bioactive molecules with drug like characteristics that has been manually curated.[21] This dataset is then set to create randomised simplified molecular-input line-entry systems (SMILES) to train on.[9] SMILES is a form of line notation

for describing a molecule using ASCII strings in order to easily analyse and store them digitally.[22]

For every epoch in REINVENT, it samples a number of SMILES equal to the generational batch size (within REINVENT this is merely called the batch size but to avoid confusion with the acquisition batch size, defined later in the text, it will be labelled as the generational batch size here). The generational batch size used for this project is 128. Sometimes the number of molecules returned by REINVENT is less than the generational batch size; this can happen when REINVENT generates invalid SMILES (meaning they cannot be transformed into a molecule) which are then discarded.[6] The valid SMILES are then randomised during training.

The reason for the randomisation of the SMILES is that by randomising the compounds' multiple SMILES encodings, REINVENT "will likely learn the grammar rather than memorizing specific strings or parts of them".[6] The resulting model shows improved generalization potential when compared with previous models and produces SMILES strings with a validity above 99%.[9]

The generative model in REINVENT is built using a recurrent neural network (RNN) model built on either Long Short-Term Memory (LSTM) or Gated Recurrent Unit (GRU) layers.[23] This model generates SMILES strings sequentially using conditional probabilities. Equation 1.1 contains the probability of generating a string S, consisting of tokens $s_1, ..., s_T$.[24]

$$p(S) = p(s_1) \prod_{t=2}^{T} p\left(s_t \mid s_{t-1}, \ldots, s_1\right) \tag{1.1}$$

Take for example the SMILES string "N#N", representing dinitrogen. The first token has probability p("N") to be generated, after which the following tokens have to be conditioned on the preceding tokens. The second token has probability p("#"|"N") to be generated, and the final token has probability p("N"|"N#"). Thus the entire string is generated with probability p("N") $\cdot\, p$("#"|"N'') $\cdot\, p$("N"|"N#").

In order to generate SMILES that represent molecules with desired properties, the problem is framed as a partially observable Markov decision process, where the next character to be generated is chosen by an agent based on the current state. In reality, the agent is an RNN, which works in conjunction with another RNN termed the *prior*. The prior was trained using maximum likelihood estimation to obtain a policy for generation of molecules. [25]

In Equation 1.2, an augmented likelihood is defined as the sum of the prior likelihood and $\sigma S(A)$, where σ is a scalar and $S(A)$ is a scoring function which quantifies the desirability of a sequence. [25]

$$\log P(A)_{\mathbb{U}} = \log P(A)_{\text{prior}} + \sigma S(A) \tag{1.2}$$

From this $G(A)$ is defined, which measures the agreement between the agent likelihood $\log P(A)_{\mathbb{A}}$ and the augmented likelihood,

$$G(A) = -\left[\log P(A)_{\mathbb{U}} - \log P(A)_{\mathbb{A}}\right]^2. \tag{1.3}$$

In order to ensure an agent policy which agrees with the augmented likelihood as well as possible, Equation 1.3 should be maximized. The loss $J(\Theta)$ used is defined as

$$J(\Theta) = -G. \tag{1.4}$$

The scoring function $S(A)$ itself contains multiple components, e.g. including slow predictive models, and by version 3.0 of REINVENT also molecular docking.[26]

Molecular docking predicts favourable binding orientations between a molecule and a target based on thermodynamic contributions. This can in turn be used to predict the binding affinity between two molecules if implemented into a scoring function.[27]

The docking component of the scoring function is set up through Icolos, which is a workflow manager that is used as a wrapper for computational chemistry tools in REINVENT, including molecular docking.[26] Molecular docking is used to encourage REINVENT to generate SMILES with similar docking scores to known inhibitors. The scoring function is the costliest part of the RL loop due to it containing components that are computationally intensive.[6] This project will focus on alleviating this computational cost by building an AL framework to act as a surrogate for molecular docking.

When training generative ML models, a balance between two separate objectives called *exploration* and *exploitation* is sought after. In the case of drug-like molecules, exploration refers to generating molecules which are spread out in chemical space. This is to encourage a more encompassing spread of molecules, as well as to discourage getting stuck in local optima. Exploitation, however, is then seen as generating molecules with higher probability of being global optima, or a higher probability of satisfying the MPO well. [28] Balancing these two goals is a difficult task. If the model explores too much it will not hone in on promising molecules, and if the model exploits too much it will get stuck in sub-optimal regions.

In order to discourage excessive exploitation, REINVENT is outfitted with diversity filters. These filters can discourage the frequent generation of similar compounds by penalizing the scoring function when the model has sampled from the same chemical space too often.[6]

1.2 Purpose and Goal

The purpose of this project consists of investigating whether the machine learning technique active learning can help mitigate the computational costs of querying oracles for molecular design without negatively impacting accuracy. Our goal is to devise an active learning framework that can mitigate the computational cost by reducing the number of calls to an expensive oracle and instead use a surrogate model to predict its output. In order to do so, different strategies for improving the surrogate models are evaluated for how well they account for the sampling behavior of the REINVENT generative model. How these are connected can be seen in Figure 1.2.

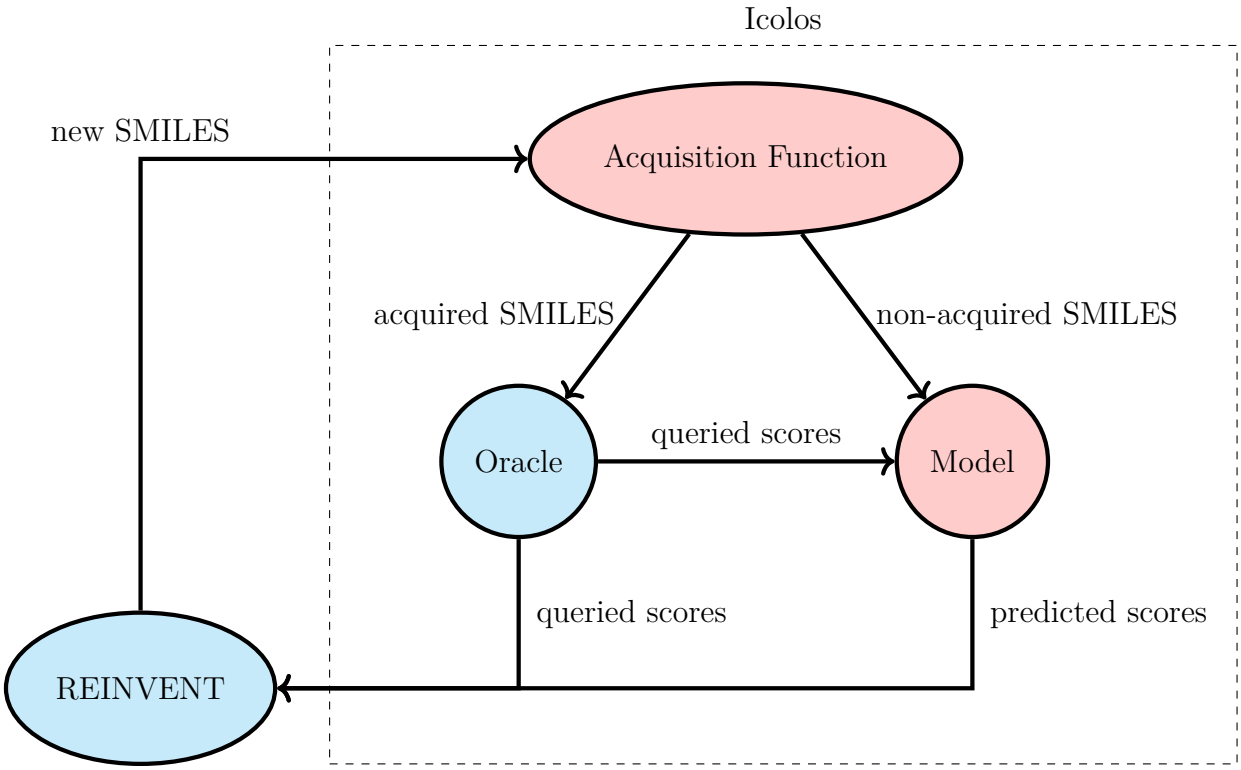

Figure 1.2: An overview of the surrounding architecture of the active learning framework. The nodes in red are the components this project will focus on.

The results of this work will be beneficial to augment AI-driven molecular design with potential to significantly increase accessibility and throughput to real-life drug discovery tasks.

1.2.1 Objective

The project has three main objectives:

- Elucidate the best practices for using active learning to train surrogate models in the context of generative models.
- Show that surrogate models can achieve sufficient performance to replace oracle calls, both with respect to time and error.
- Investigate the impact of acquisition function choice on the performance of the surrogate models.

1.3 Demarcations

The project aims to investigate best practices to train surrogate models to predict computationally expensive molecular descriptors via active learning. Application of the findings will help mitigate computational costs of using generative models for molecular design tasks.

The project does not aim to improve the generative process *per se*, but to maintain a sufficient level of accuracy. Thus, the project will make no changes to the core architectures of REINVENT and Icolos.

The project will not investigate alternative machine learning methods for molecular generation nor tune the hyperparameters of the surrogate models since the optimal hyperparameters will differ greatly depending on the predetermined desired properties of the molecules.

1.4 Ethics and Societal Impacts

Within any research, and within medical research especially, considering the end use (and potential abuse) of the result is of paramount importance. It is crucial to always have in mind where the research might lead, and to be conscious of how scientists shape and interpret the data both given and generated.

1.4.1 The Good

What happens if the best case scenario presents itself; a more cost effective way to discover new molecules for drug discovery is successfully built?

There are many answers to this question, depending on levels of optimism. Creating new drugs to take to market is for many reasons an incredibly expensive process. Taking a new drug to market from pre-clinical phases requires a minimum time frame of 10-12 years and over 2 billion dollars in resources.[1][29]

To add to that, the success rates for drugs entering Phase I clinical trials have approximately 10% chance of gaining FDA approval for the desired indication, with similar or sometimes worse rates within the EU market.[30]

However, the time and resources necessary for proper clinical trials are not inherently something one might wish to change. Rigorous trials to prove safety and efficacy of a new compound is a safeguard for the end-user and a way to uphold the standards of modern medicine. However, since the trial process is so time consuming and expensive, making the actual drug discovery process faster is of even greater importance.

If this work enables more efficient drug discovery then this can lead to better treatments and therapies for people who need it and maybe even the development of cures to illnesses that have so far eluded modern medicine. Finding new solutions to old problems is also of value for many. To find a cheaper way of treating an illness might save as many lives as finding a treatment to a previously untreatable illness.

1.4.2 The Bad

The bad outcome that runs the highest risk of coming to fruition with this kind of exploratory project is for the result to not be useable. If the generated data turns out to not be a good representation of real data, for example, then the analysis is

of no use to anyone. This is also true if it turns out that the implementation of the the models or acquisition functions are incorrect.

What would not be an entirely useless but maybe slightly disappointing outcome would be if it was discovered that there was next to no difference between our different models and acquisition functions. If such a conclusion is made, however, that will at least tell future researchers where *not* to look in terms of building a more efficient system for drug discovery.

1.4.3 The Ugly

The section name here refers to if the project is successful in its goal and has created a way to make the drug discovery process more cost efficient, but the tools created are being used with "ugly" intentions. This dilemma is often called the dual-use dilemma. The dual-use dilemma refers to the dual nature of, for example, efficient drug discovery. It can bring immense good to many people by discovering drugs that can help to mitigate, alleviate and possibly even cure illness. However, it can also be used to discover molecules that can do massive harm in unspeakably efficient ways. [31]

In 2022, The Swiss Federal Institute for Nuclear, Biological and Chemical Protection set up a conference to identify developments in chemistry, biology and enabling technologies that may have implications for the Chemical and Biological Weapons Conventions.[32] At this conference a company used its own artificial intelligence network, built for toxicity prediction, to explore how AI could be used to design toxic molecules. What began as a thought exercise ultimately evolved into a computational proof of concept for making biochemical weapons.[33]

Within 6 hours of running their newly optimised network they generated over 40 000 compounds within their own threshold for human toxicity. In addition, many known chemical warfare agents were identified through visual confirmation with structures in public chemistry databases. Many new molecules were also designed which were predicted to be more toxic than publicly known chemical warfare agents.[33]

This is not a problem this project is in any way equipped to solve or even handle. However, it is important to keep these perspectives in mind when making exploratory research and be conscious of the nefarious scenarios the research might lead to as well as the good ones.

2
Method

This chapter goes through both the theory and implementation of the project and details the data used in the retrospective analytical runs. Further, it goes through the acquisition functions and surrogate models used in the comparative study. Pseudo code snippets are shown to give the reader insight into the programming structure, when appropriate.

2.1 Overview of Project

The goal of this project was to create an active learning framework and evaluate if this could be used to replace a portion of the expensive oracle calls used in molecular docking. The active learning framework consists of two parts: a model to train and an acquisition function to acquire points to add to the training data. As can be seen in 2.1, REINVENT generates new SMILES. These are fed through the acquisition function which acquires a subset of the SMILES that it thinks are most important to train on. These are then sent to the oracle, the acquired SMILES's docking scores are calculated and then sent to the model.

The next step is the model then uses these acquired SMILES where the docking score is known to train on and from this data it tries to predict what the docking scores of the rest of the non-acquired SMILES should be. Then both the queried scores and the predicted ones are fed back into REINVENT and the cycle should start again.

In a real use-case the queried and predicted scores would be fed back into REINVENT where its own generative model would use this data to better generate the new batch of SMILES. However, the experiments for this project were made using a retrospective pregenerated dataset, so the output does not affect REINVENT's molecule generation for this study.

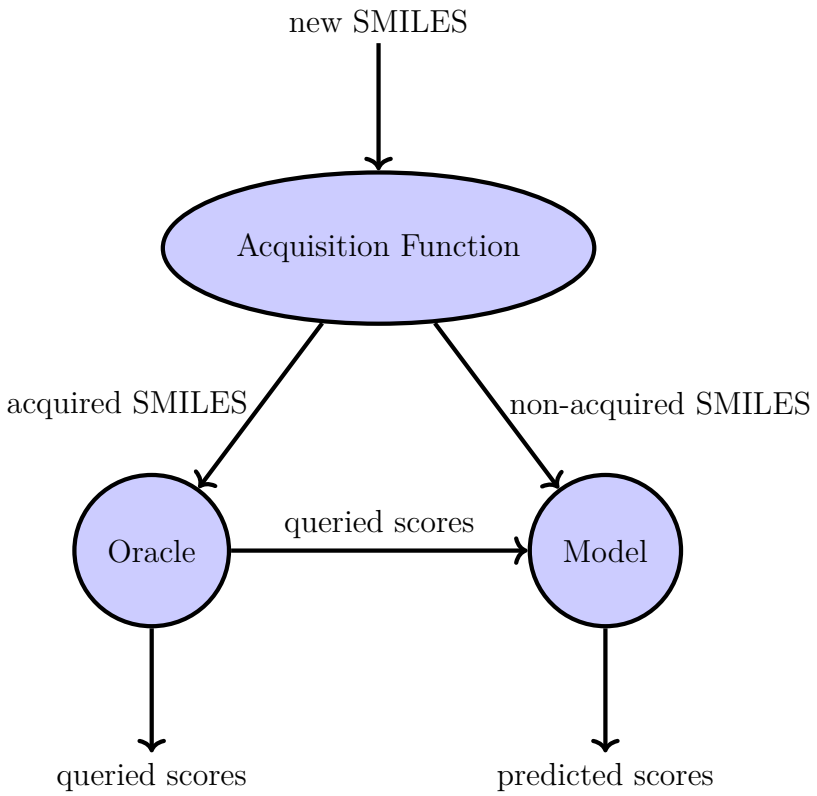

Figure 2.1: Overview of the implemented active learning with the surrogate model. REINVENT generates new SMILES, the AF takes in and samples the SMILES. It acquires points based on a specific strategy and queries the oracle for the score of these SMILES. The surrogate model then acquires new sampled SMILES every epoch based on the selection of the acquisition function and uses these points to predict the score of the non-acquired SMILES. The labelled scores and of those predicted by the model are then fed back into REINVENT.

In this project, two models and eight acquisition functions were evaluated based on their accuracy and time consumption. The focus is mainly on time consumption as the goal is not to optimize the framework but to make it faster. As long as the error rate is within acceptable bounds, time consumption is the more important metric. The main code loop used in this project can be seen in Algorithm 1.

First, initial pooling is done: for the first n epochs, where n is the parameter *initial pooling*, data is read from an input file (the input is molecules in the SMILES format), and every point is labelled and added to the labelled pool. The model is then fit on the pool of acquired data.

For every subsequent epoch, a fraction of the input data is collected and a model prediction is made for the rest. The data points are acquired based on the chosen acquisition function and then labelled by querying the oracle. This labelled data is then added to the pool of labelled data, so for every epoch the pool of labelled data grows. Then, if the amount of epochs since last training is equal to the *retrain* parameter, the model is trained on the data; the end of the loop iteration has then been reached.

Algorithm 1: Main loop

```
for epoch in range(1 to n): # n is initial pooling
    data ← collect data # read from file or directly from REINVENT
    extend pool with data
fit model on pool
for epoch in range(n to final epoch): # n is initial pooling
    data ← collect data # read from file or directly from REINVENT
    pred ← model predictions for data
    acquire datapoints from data according to AF
    query oracle to label acquired data
    extend pool with added labelled data
    if epoch number mod m = 0: # m is the retrain parameter
        fit model on pool
    calculate errors between labelled and predicted scores
```

2.2 Creation of Dataset

While running the surrogate model during a prospective REINVENT run would give a more accurate representation of the behavior of our framework, doing so during development would amount to introducing expensive computational overhead at a time where the positive effects of doing so are uncertain. Instead a retrospective analysis was done, the results from which can be found in chapter 3.

To create the data, REINVENT was run using the parameters seen in Table 2.1. The scoring function was made using a weighted geometric average containing docking, quantitative estimate of drug likeness [34], molecular weight and number of hydrogen-bond donors. For docking, the target protein was COX2, using a protein database structure of 1CX2. Molecular weight is enforced to be in the range 200 to 550, while the number of hydrogen-bond donors is enforced to the range 0 to 7. These are then transformed into the [0,1] range using a reverse sigmoid transformation. The diversity filter used was IdenticalMurckoScaffold, with a bucket size of 25.

The reason behind using these other oracles within the scoring function besides molecular docking, the one of interest, is to ensure that the retrospective analysis more resembles an actual use case. A danger with only using docking would be that the REINVENT agent might generate molecules which dock well but are not biologically relevant. In other words, the other objectives help ensure biological relevance when generating molecules.

Parameter	Value
Epochs	1000
σ	128
Learning rate	0.0001
Generational batch size	128
Diversity filter	IdenticalMurckoScaffold, bucket size 25
Scoring function	Custom product with Icolos (docking), QED, molecular weight and number of hydrogen-bond donors.

Table 2.1: Parameters used in REINVENT to generate the datasets used. The parameter σ is a coefficient that controls the impact of the scoring compared to the prior likelihood (see [25] for more details). The scoring function is a custom product containing docking, QED (quantitative estimate of drug likness, see [34]), molecular weight (enforced to within 200-550) and number of hydrogen-bond donors (enforced to within 0-7). The diversity filter used was IdenticalMurckoScaffold with a bucket size of 25.

An additional part of the scoring function is the score transformation. The score transformation used when creating the three retrospective datasets entails that errors have different impacts not only depending on their size but also where the scores in question are located.

Score transformations are typically used to ensure that the scores are more uniform in order to not weight different scoring functions unduly, when working with MPOs. They can also weight different parts of the input space for the objectives. See for example Figure 2.2, which visualizes the score transformation used here.

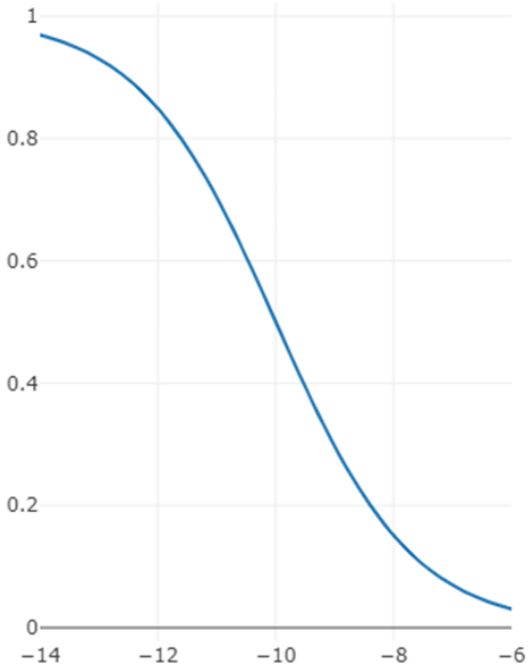

Figure 2.2: The score transformation used when creating the datasets used, a reverse sigmoid. Due to the steepness in the middle, an error close to -10 would entail a larger difference in score, i.e. it penalizes errors close to the midpoint.

Note that for this transformation, errors close to the mid point are penalized heavier than errors closer to the edges. As such, the definition of what an acceptable error size is not a question that can be answered definitively, as it depends on the location of the scores, both predicted and ground truth.

The dataset was created in batches of 128 molecules encoded in the form of SMILES, although sometimes some were excluded as a result of the diversity filter, and their corresponding docking scores. The docking scores are negative numbers where a lower score is better (i.e. the further away from zero a score is, the better). If a molecule did not dock at all, it was assigned a docking score of 0. In pursuit of reproducibility, three separate runs were done to create three replicate datasets.

2.3 Surrogate Models

Two models which have been shown to work well for similar problems are Random Forest (RF) and Support Vector Regression (SVR).[12, 35]

There are a selection of different hyperparameters to tune for each model, however this will be outside the scope of this investigation as this project is not primarily looking to build the most accurate model to predict docking scores. It is rather aiming to train a sufficiently accurate model, that can be used for any protein receptor, in which the optimal hyperparameters would likely vary. This leads to a combinatorial explosion of possible hyperparameters and since the aim is to introduce a

general method of using active learning for prospective generative model runs this is not desirable.

2.3.1 Random Forest

An RF model is an ensemble method utilizing decision trees. A decision tree is a linear partitioning of input space; for a given point, the tree repeatedly checks whether the point is lesser or greater than different lines. This process is often visualized as a binary tree, where at each branch such a comparison is made. The leaves correspond to different decisions, typically either classification or regression.[36]

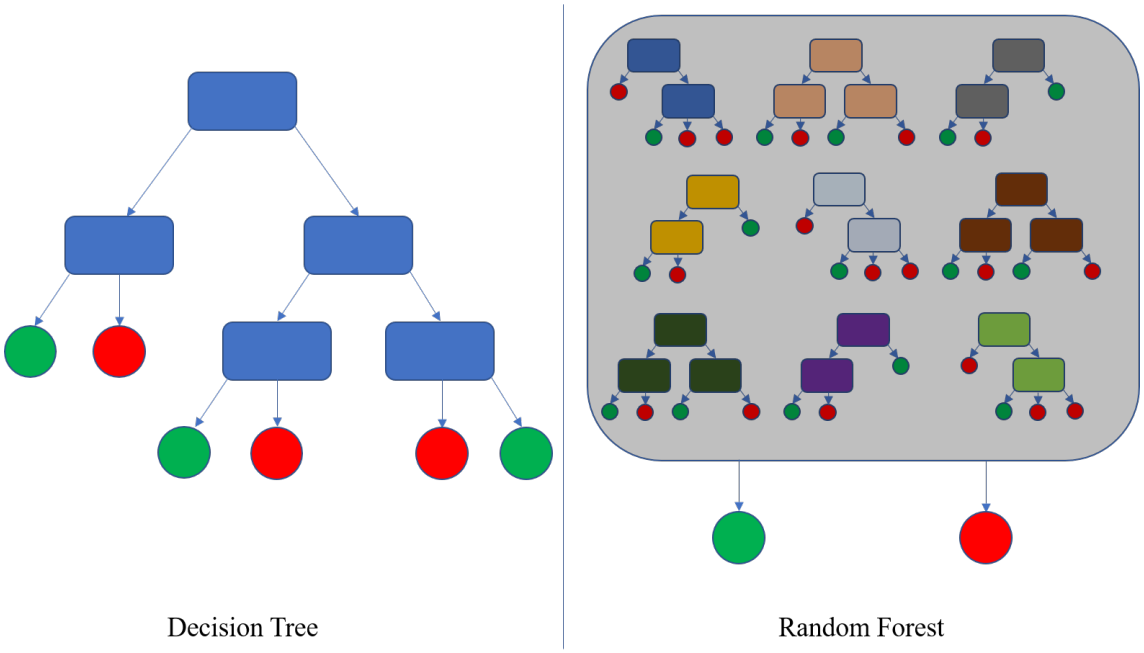

Figure 2.3: Image visualising the difference between an ordinary decision tree an RF model.[37]

A random forest is an ensemble of decision trees with randomly selected features to be considered, a visualisation of this can be seen in figure 2.3. Mathematically, given a dataset containing p features, $m < p$ random features are selected to be considered at every branch, and the input space is partitioned into two subspaces using hyperplanes parallel to an axis. The trees then make a decision by committee. For the case of regression, which will be used here, the final decision of the random forest is the mean of the decisions made by the trees. This has the added benefit of preventing overfitting as it is an ensemble method because of the random partitioning of features.[36]

The forest as a collection of trees is used to calculate the standard deviation for a score. What this means is that every separate tree is used to predict scores for every molecule and then the different scores for a molecule is used to generate its standard deviation. Thus, if a forest containing 100 trees is used, every molecule has 100 different predicted scores which are used to calculate the standard deviation.

2.3.2 Support Vector Regression

A support-vector machine (SVM) is a machine learning model traditionally used in classification tasks that tries to classify data by finding hyperplanes that divides the data into classes. It then classifies the new point depending on whether it lies on the positive or negative side of the hyperplane depending on the classes to predict. Support Vector Regression (SVR) uses the same tactic as the SVM, but for regression problems.[38]

In most linear regression models, the objective is to minimize the sum of squared errors. However, there are problems where the interest does not lie in reducing the error as much as possible but rather to stay within a certain range. This is true for the problem this project wishes to solve as the goal is to minimize computational cost while staying within acceptable error rates.

Here is where SVR can be of use, as it uses a decision boundary to define the acceptable error and find an appropriate hyperplane to fit the data. As opposed to trying to minimize the sum of squared errors, the objective function of SVR is to minimize the l2-norm of the coefficient vector, $\mathbf{w}$.[39]

In SVR the absolute error is set to less than or equal to a specific margin called the maximum error, ε. The objective function and its constraints are constructed thusly:

$$\begin{aligned} \text{Minimize:} \quad & \frac{1}{2}||\mathbf{w}||^2 \\ \text{Constraints:} \quad & |y_i - w_i x_i| \leq \varepsilon, \end{aligned}$$

where x_i and y_i represent input data and score, respectively. A visualization of an SVR can be seen in Figure 2.4.

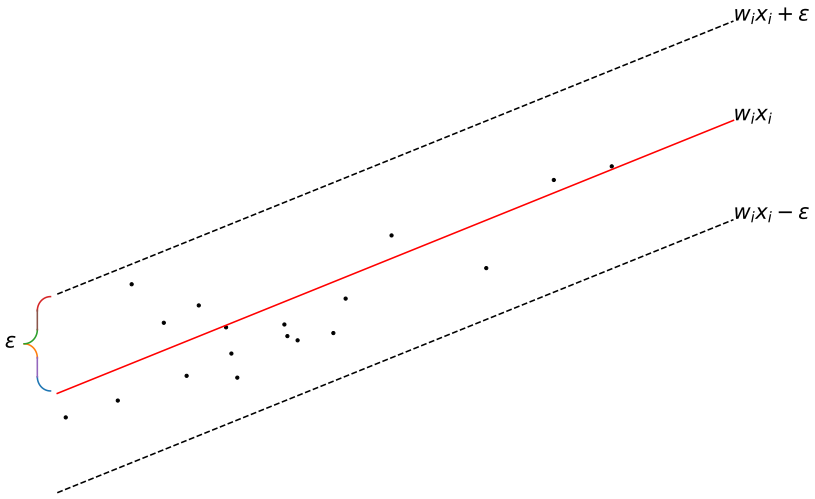

Figure 2.4: A simple example showing the decision boundaries defined by the maximum error, ε.

Calculating standard variation for SVR is problematic. SVR is a deterministic method, and thus ensemble methods do not work. Several of the AFs are thus not possible to implement; these AFs will not be included when running the experiments with SVR.

2.4 Acquisition Functions

A key step of bayesian optimization is which acquisition function to use. The role of this acquisition function is to decide which data point to query, i.e. which data point is to be labelled next. Depending on the strategy, the AF can be more or less exploratory or exploitative. For example, the Greedy acquisition function which samples only the best scoring points is very exploitative. Random, on the other hand, which samples points randomly, is very exploratory.[16]

Worth nothing here is that the EI, PI and TS acquisition functions assumes that the standard deviation derived from the surrogate model follows a normal distribution. If the data follows a different distribution other functions are needed.

The standard deviation is needed for a measure of uncertainty for some acquisition functions. This is done in different ways depending on the model being used. Notably, SVR does not lend itself well to calculating standard deviation and as such the acquisition functions that require it are not used for this model.

For a mathematical overview of the functions used for each AF, see table 2.2. In

essence, for a list of molecules x, a corresponding score list $y(x)$ is calculated according to the second column. The top scores in $y(x)$ are then selected to be acquired.

Acquisition function	Score
Random	$y(x) \sim \text{uniform}(0,1)$
$\varepsilon-$Greedy	$y(x) \begin{cases} \sim \text{uniform}(0,1) & \text{with probability } \varepsilon \\ = \mu(x) & \text{otherwise.} \end{cases}$
Tanimoto	$y(x) = -\frac{1}{n}\sum_{i=1}^{n} T(x, x_i)$
PI	$y(x) = \Phi(\zeta(x))$
EI	$y(x) = \gamma(x)\Phi\left(\zeta(x)\right) + \sigma(x)\phi\left(\zeta(x)\right)$
TS	$y(x) \sim \text{normal}\left(\mu(x), \sigma^2(x)\right)$
UCB	$y(x) = \mu(x) + \beta\sigma(x)$

Table 2.2: Acquisition functions and their corresponding ways of calculating the scores. For a given list of molecules, the score y is found using the above, and the top results are acquired by the acquisition function. The functions ϕ and Φ are the probability density function and the cumulative distribution function for the standard normal distribution. Here, $\mu(x)$ is the mean of x, i.e. the prediction given by the surrogate model, $\sigma(x)$ is the standard deviation and y^* is the current optimum found. For brevity, we define $\gamma(x) = \mu - y^* + \xi$ and $\zeta(x) = \frac{\gamma(x)}{\sigma(x)}$. The parameters used in these experiments were $\beta = 2$ and $\xi = 0.01$. For Tanimoto, x_i are the molecules already in the pool, and $T(x, x_i)$ is the Tanimoto Similarity between the molecules x and x_i.

2.4.1 Random

The random AF consists of picking points at random. The implementation is predictably simple, merely randomly shuffling the points and acquiring the first points equal to the acquisition batch.

The random AF is used as a baseline and to see if the AFs have a positive impact compared to a random selection.

2.4.2 Greedy and ε-Greedy

The greedy strategy uses the surrogate model to predict a score for all new molecules and picks the best performing ones; in our case it picks the points which are predicted to dock best. Greedy is a highly exploitative algorithm, focusing entirely on what the model already knows.[16]

To acquire the best scoring points, the data is predicted and sorted by the docking score. Then the n best scoring points are sampled, where n is the acquisition batch

size.

ε-Greedy is very similar to Greedy, but with the addition of a probability, ε, that random acquisition is used instead. This is one way to encourage exploration in an otherwise exploitation-heavy AF.[40]

2.4.3 Probability of Improvement

The first acquisition function designed for Bayesian optimization was Probability of Improvement (PI).[16] This function is sometimes called the "P-algorithm" and was pioneered in the 60s by H.J Kushner [41]. The function itself can be seen in table 2.2. A factor to note with PI is that it is very exploitative since points that high probability of being even the tiniest bit better will be prioritised over points with a lower probability of a much larger gain.[42]

To mitigate this, a trade-off parameter value ξ is added so that the improvement must be at least larger than ξ to be sampled. This is in order to encourage more exploration and the higher the ξ the more the function will explore. This project uses a $\xi = 0.01$, based on promising results from previous studies[12][42].

2.4.4 Expected Improvement

The Expected Improvement (EI) acquisition function uses not only the probability of the improvement but takes the magnitude of the improvement into account as well.[42, 43] With EI, the goal is to minimize the expected deviation from f^* when choosing a new point to sample. This can be seen mathematically in table 2.2. The expectation of the improvement function is to strike a better balance between *exploitation* and *exploration* than PI does. Meaning, it is built to be more exploratory.

2.4.5 Thompson Sampling

Thompson sampling has previously been shown to have strong empirical performance and been applied to several different domains with great success, for example recommendation systems and revenue management. [44]

Practically, to utilize Thompson sampling in this particular domain, every potential molecule is assigned a normal distribution which is then sampled from. If y_i is the score of molecule i, μ_i is the molecules mean prediction and σ_i is its standard deviation, it is assumed that $y_i \sim \text{normal}(\mu_i, \sigma_i^2)$, and such y_i are sampled. The molecules with highest sampled y_i are then acquired. The function for this is also shown in table 2.2.

2.4.6 Upper Confidence Bound

Upper Confidence Bound (UCB) selects the points based on uncertainty and is formulated as a linear combination between the surrogate models mean prediction and standard deviation; UCB is sometimes referred to as an optimistic strategy. [28]

The UCB balances exploration and exploitation through the hyperparameter β, as illustrated in table 2.2. If β is increased, the points selected will be of an exploratory nature, and if it is decreased the function will shift more towards an exploitative nature. The implementation for this project will be using a fixed value of $\beta = 2$, since that has shown promising results in previous studies [12] but others sample β from a gamma distribution [28] to optimize the *exploration/exploitation* trade-off.

2.4.7 Fingerprint Tanimoto Similarity

Fingerprint Tanimoto Similarity selects the points based on possessing low Tanimoto similarities of the molecular fingerprints to the current training data.[45] We will for convenience refer to the Fingerprint Tanimoto Similarity acquisition function as the Tanimoto acquisition function henceforth.

Molecular fingerprints are a way of uniquely encoding the structure of a molecule. This encoding can then be used to determine the similarity of molecules in order to for example find a match for a query substructure. The most common type of fingerprint for small molecules are Morgan fingerprints, which is a type of Extended Connectivity Fingerprints (ECFP).[46] This is also the ones used for this project.

In short, given a list of potential molecules to acquire and a list of molecules that has already been acquired, the Tanimoto AF does pairwise comparison between every potential molecule and every already acquired molecule. The similarities are averaged per molecule in the list of molecules to acquire, resulting in a scalar per potential molecule which measures the average Tanimoto similarity of that molecule compared to the pool. The lowest similarities are then acquired, to encourage selection of molecules which are dissimilar to the pooled data.

2.5 Parameters

In order to speed up the computation or increase accuracy, four tuning parameters were implemented. The first, *warm-up*, handles how many epochs are submitted entirely to the oracle function until the surrogate model is activated. The reasoning behind this parameter stems from the fact that in the beginning of a REINVENT run, the molecules typically do not satisfy the scoring function, e.g. do not dock well.[6] This leads to REINVENT likely exploring more during these initial epochs. In order to avoid a negative impact of this phase on the surrogate model, we hypothesise that warm-up can be used to skip past it and to start acquiring at a time where molecules are more likely to be relevant to the desired objective.

The second, *retrain*, handles how many epochs to go between each update of the surrogate model; a retraining parameter of 1 equates to training the surrogate model every epoch. As training the surrogate model is time-intensive, a larger interval between each retrain will lessen the overall time of a run. However, retraining cannot be skipped entirely, and so a set of intermediate values are evaluated.

Additionally, a third parameter was introduced to generate a sufficient initial training set so that an effective surrogate model can be trained, especially during the earlier epochs. This parameter controls the number of epochs where *initial pooling*

is done; this simply means that every data point is acquired instead of a smaller fraction of them. As some models require large amounts of data, the theory is that this parameter will decrease the initial error.

The fourth parameter, *acquisition batch* (AB), controls how many points are to be acquired at each step. A larger AB leads to a greater pool of data for the surrogate model to train on, which leads to a greater predictive performance. However, due to said growth of the pool, the training takes longer.

3

Results

This section covers the findings of the comparative retrospective study. Firstly, an experiment was run comparing the models with their respective available acquisition functions in order to find the best use case. A model and acquisition function was then selected and with these followed parameter tuning experiments with differing values of *initial pooling*, *retrain* interval, *acquisition batch* size, and *warm-up* phase.

In these experiments, the model predicted docking score for the molecules generated during the current epoch, and the predicted docking scores were compared to the ground truth which is the docking score derived from the oracle. The metrics used were root mean square error (RMSE) and time consumption per epoch.

For ease of comparison, the docking scores are negative numbers, in the case of the specific docking target used in these experiments, commonly in the range -5 to -10, where a lower score is better, and molecules that did not dock were given a docking score of 0.

In order to ensure reproducibility, all of the runs were done in triplicate; there were three separate datasets created, and every run was done separately on each of them. The intention was to reduce interference from stochasticity and, when time was investigated, to reduce the impact of practicalities which are not of interest, e.g. impact from other processes running on the same computer. The results are averaged. Any shading in a figure shows the maximum and minimum values, which is taken to represent the variation in the runs.

A selection has been made as to which results are shown, especially when investigating which model and AF to use. For graphs showing more raw (and complete) data, see Appendix A.

3.1 Selection of Model and Acquisition Function

The models to be evaluated are RF and SVR. Since only the performance of the models were of interest in this part of the study, other parameters were the same between both models. This was done with the default parameters seen in table 3.1.

Default Parameters for Model Comparison	
Acquisition Batch Size	40
Warm-up	200
Initial Pooling	5
Retrain	5
Number of Epochs	1000

Table 3.1: Default parameters for each surrogate model.

However, some factors could not be kept identical between models. Due to differences in implementation mentioned in subsection 2.3.2, SVR is incompatible with the AFs using standard deviation. Due to this, SVR was evaluated with four acquisition functions: Random, Greedy, ε-Greedy and Tanimoto Similarity. RF was implemented using eight acquisition functions: Random, Greedy, ε-Greedy, Tanimoto, EI, PI, UCB and TS.

In Figure 3.1 and 3.2, rolling averages of RMSE for RF and SVR, respectively, are shown. The combination of model and AF that achieves the minimum rolling average RMSE is SVR and Tanimoto Similarity, with the minimum rolling average RMSE 1.55 kcal/mol. Note that in both models the Random AF is among the best performing, with comparable error to the Tanimoto Similarity for SVR. The Random AF is thus picked for both models to enable a model-centric comparison; see Figure 3.3, which shows the rolling average RMSE for the Random AF between the two models. SVR has smaller errors overall, but the difference between the models is smaller than the internal variation.

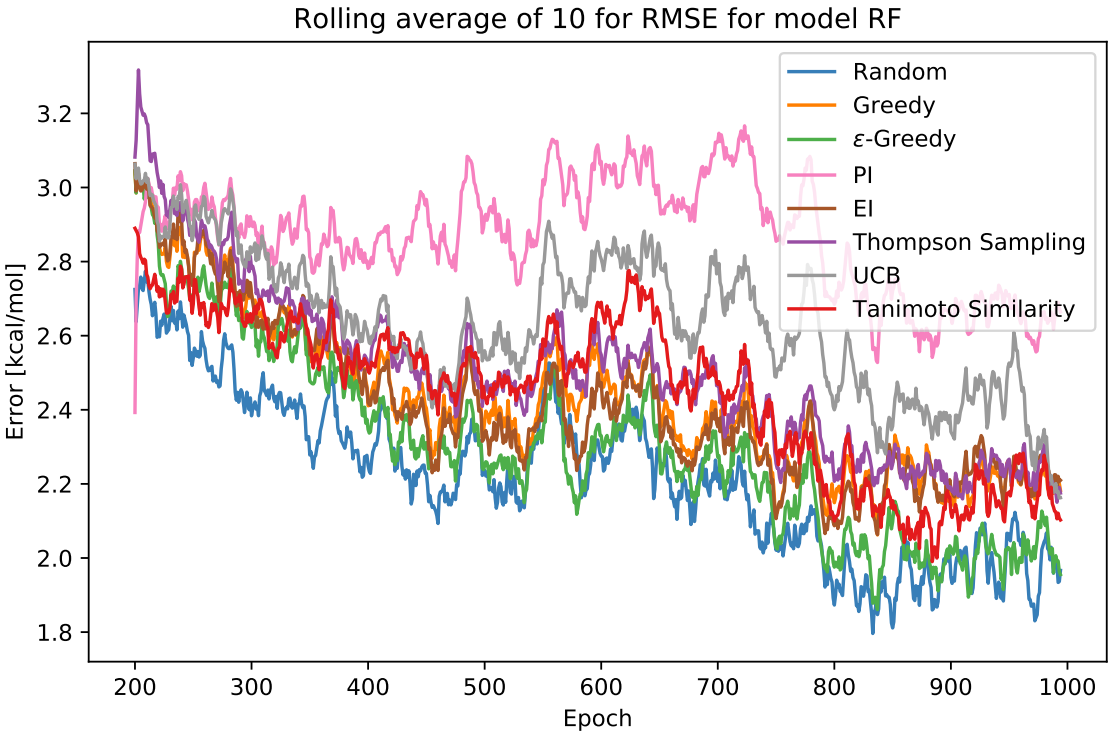

Figure 3.1: Rolling average of RMSE for RF and all AFs. There is a cluster of AFs in the bottom, consisting of Random, Greedy, ε-Greedy, EI, Thompson Sampling and Tanimoto Similarity.

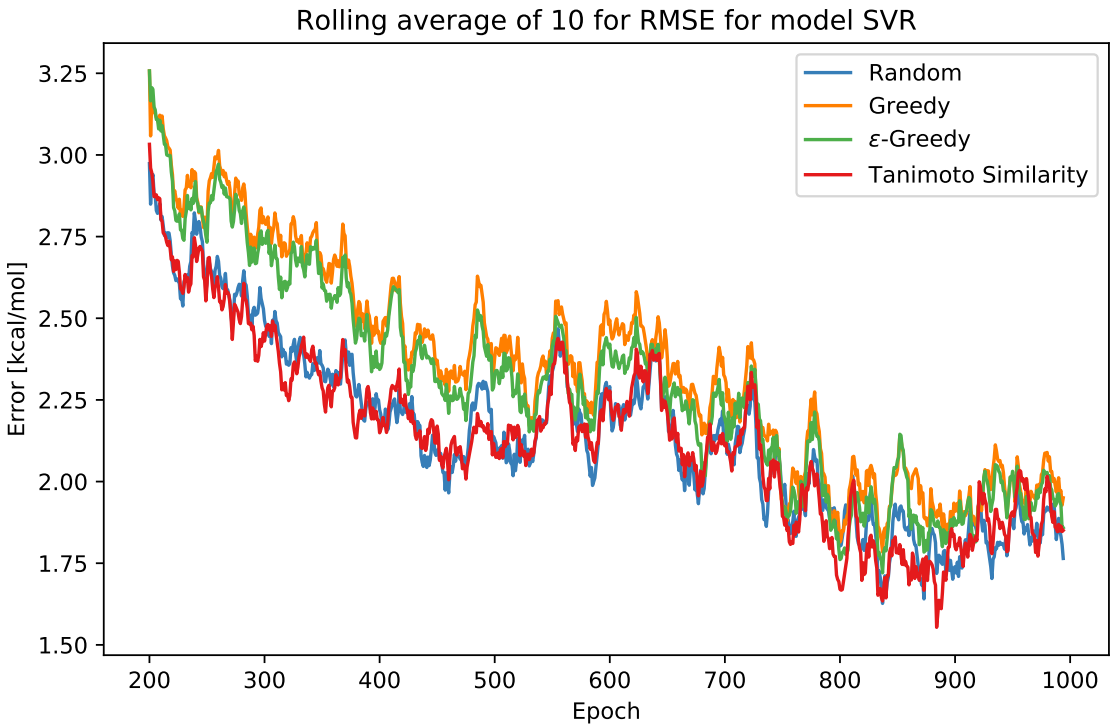

Figure 3.2: Rolling average of RMSE for SVR and all AFs. The four AFs cluster entirely in the second half, while a small distinction occurs between two pairs of AFs in the first 300 epochs. The pair with lower error is Random and Tanimoto Similarity, while the pair with higher error is Greedy and ε-Greedy.

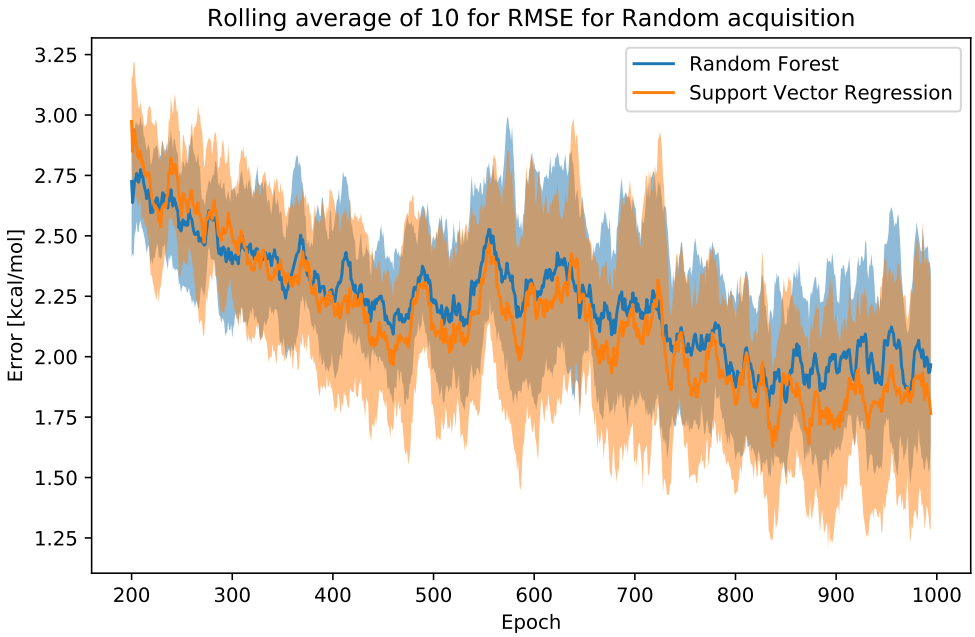

Figure 3.3: Comparison of the rolling average of the RMSE for RF and SVR using the Random AF. The errors are similar between the models; there is more internal variation than there is variation between them.

However, the difference in amount of time the surrogate model training take help distinguish between them. Figure 3.4 shows this difference. More specifically it is the difference in cumulative runtime, i.e. how much longer an SVR run took compared to an RF run. In fact, it is a comparison between the quickest run (which was using the Random AF) for SVR and the slowest run (which was using the UCB AF) for RF. At best, SVR takes 120 hours longer than RF. Figure 3.5 shows the amount of time taken by RF. As such, the extra amount of time taken by SVR is a factor 10 larger than the overall time taken by RF.

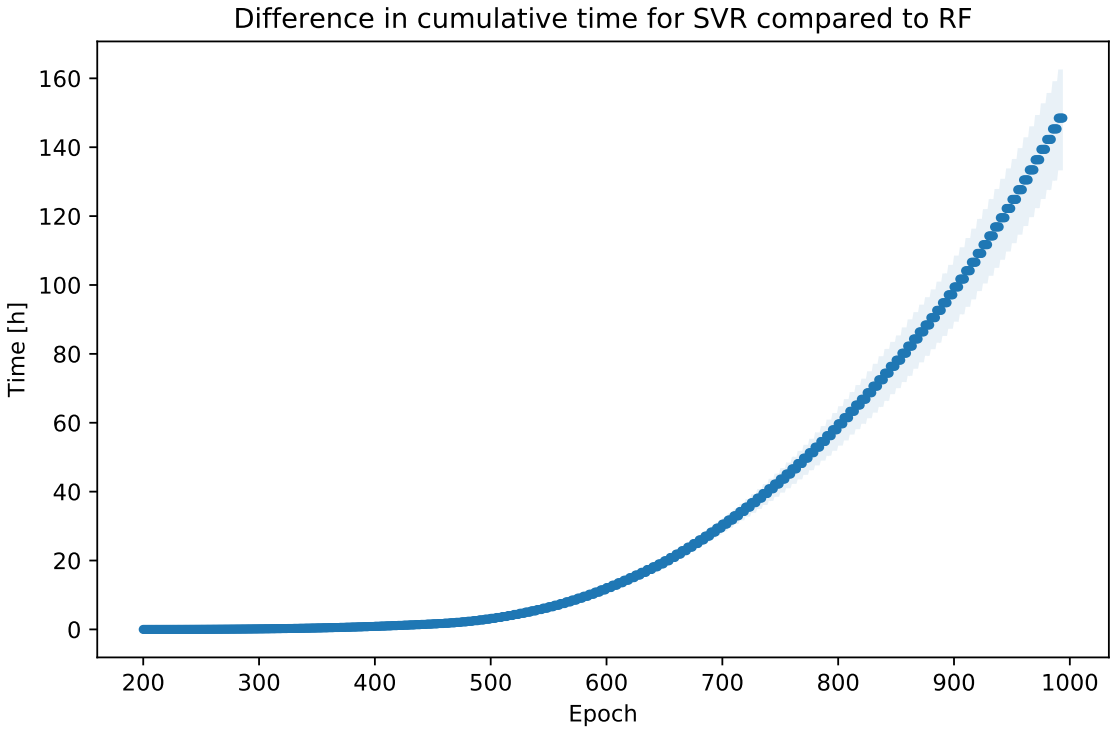

Figure 3.4: The difference in cumulative run times for SVR and RF. The specific runs chosen are the ones with the least difference, i.e. the quickest SVR run and the slowest RF run. These were using the Random AF for SVR and the UCB AF for RF. The minimum and maximum over three replicates is shown as the shaded area around the curve.

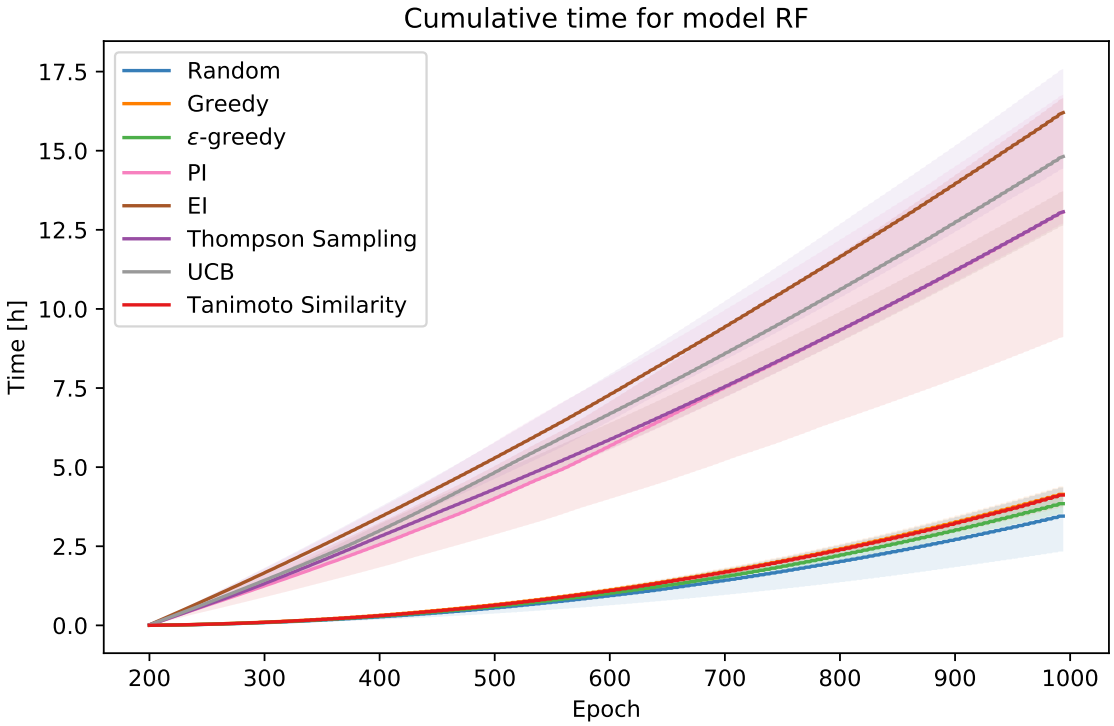

Figure 3.5: Cumulative run times for the different AFs used with RF. Notice the two groups of AFs. The quicker group contains Random, Greedy, ε-greedy and Tanimoto Similarity. The slower group contains PI, EI, Thompson Sampling and UCB. The salient difference between these groups is that the slower group contains the AFs that require the calculation of standard deviation.

As such, the decision was to continue using the RF model for any further experiments.

In order to prevent a combinatorial explosion in amount of experiments when further investigating the differences in errors and run times between different parameter combinations, four AFs were chosen based on the results in Figure 3.1. The final decision, based on the results shown in Figures 3.1 and 3.5, was to use Random, ε-greedy, Thompson Sampling and Tanimoto Similarity.

3.2 Selection of Tuning Parameters

In this section the chosen model and acquisition function is evaluated for the parameters mentioned in section 2.5. The plots show the difference, Δ, between the RMSE for each respective comparison parameter value, e_c, when compared to the default value, e_d, and for the difference between time used per epoch for the comparison and the default value, t_c and t_d.

$$e_c - e_d = \Delta_e$$
$$t_c - t_d = \Delta_t$$

For the RMSE plots, a negative value means that default value, e_d, is worse than the comparison value, e_c, since it means that the e_c is smaller than the default value and therefore has a smaller RMSE error. A positive value for the RMSE plots means that e_d performs better than the compared value. A value close to zero means that there is close to no difference. Similarly, the difference between t_d and t_c is visualized with a negative value being a shorter time and a positive value being a longer time.

The values shown for the time per epoch are the averages over the entire run, so an average difference in error of 1 means that on average, the comparison value e_c is one higher than default value e_d for the same epoch, and a difference in time of 1 means that on average, the comparison run took one second longer per epoch.

For further visualisations of all parameter tuning runs, see Appendix B.

3.2.1 Initial Pooling

The *initial pooling* parameter was introduced to generate a sufficient initial training set so that an effective surrogate model can be trained, especially during the earlier epochs. This parameter controls the number of epochs where initial pooling is done; this simply means that every data point is acquired instead of a smaller fraction of them. As some models require large amounts of data, the hope was that a higher *initial pooling* parameter will decrease the initial error without adversely affecting the time.

3.2.1.1 RMSE

For the Random and Tanimoto AF there seems to be an initial decreased error for the first 300 epochs. After this initial decrease, the difference between the *initial pooling* values evens out. However, looking at the magnitude of the error difference, it is apparent that the error decrease for both the Random and Tanimoto AF is not big enough to justify labelling it an improvement.

For the ε-Greedy and the TS AF, the initial error does not decrease at all for neither *initial pooling* 10 or 15.

3.2.1.2 Time per Epoch

For the cumulative time study there are varying results across the AFs. For the Random AF, a shorter initial pooling seems to correlate with a lesser time consumption.

For ε-Greedy, the *initial pooling* of 10 seems to be faster than the default *initial pooling* of 5 but the trend does not hold with the 15 *initial pooling* being slower. To reiterate, a 10 epoch *initial pooling* is faster than 5 epoch *initial pooling* but a 15 *initial pooling* is slower than both.

For the TS AF both the *initial pooling* of 10 and 15 epochs is faster than the default parameter of 5. The 15 epoch *initial pooling* does however flatten out while the difference for 10 *initial pooling* keeps growing.

For the Tanimoto AF the *initial pooling* of 15 is slower but the *initial pooling* of 10 is once again faster than the default value of 5 epoch *initial pooling*.

To summarise, some counter intuitive results since the expected behavior would be that for a larger initial pool the time per epoch would grow. More on this in chapter 4.

3.2.2 Warm-up Phase

The warm-up parameter handles how many epochs the oracle will be queried for all the points before the surrogate model is activated. The reasoning behind this parameter is that REINVENT will initially be more exploratory and therefore not find many molecules that dock well, so it might be more helpful to start acquiring points at a time where the molecules generated by REINVENT are more relevant to the objective.

3.2.2.1 RMSE

The default parameter of 200 warm-up consistently performs better across all AFs. The magnitude, however, differs between AFs with TS having the largest difference and Random having the smallest.

3.2.2.2 Time per Epoch

The time per epoch consistently shows that having a *warm-up* of 200 epochs is faster compared to a *warm-up* of 0, due to the fact that the 200 warm-up contains fewer datapoints to train on. There are however differences in intensity between AFs with the ε-Greedy AF having the largest average difference and TS AF having the smallest.

One aspect of the warm-up parameter that is not reflected in the results below is that although a warm-up of 0 leads to longer average time per epoch, there is also a gain in time from not querying the oracle for every molecule. As such, the full run times have been estimated according to Equation 3.1, resulting in Table 3.2.

$$\begin{aligned} T_{\text{est}} &= t_{\text{oracle}} \cdot n_{\text{warm-up}} + t_{\text{oracle}} \cdot (n_{\text{full}} - n_{\text{warm-up}}) \cdot \frac{n_{\text{AB}}}{n_{\text{GB}}} + T_{\text{AL}} \\ &\approx 182 \cdot n_{\text{warm-up}} + 182 \cdot (1000 - n_{\text{warm-up}}) \cdot \frac{1}{3} + T_{\text{AL}} \end{aligned} \tag{3.1}$$

Here, n_{GB} is the amount of molecules generated, n_{AB} is the number of acquired moleculs (the acquisition batch) and t_{oracle} is the average time per epoch for the oracle. In the creation of this data, $t_{\text{oracle}} = 182$ s. In reality, the fraction between acquired molecules and generated molecules varies depending on how many SMILES strings are deemed invalid. It is approximated with 1/3 ($\approx 40/128$), as typically only a few molecules are deemed invalid. T_{AL} is the time that the AL framework took, i.e. the times reported by the program itself.

For convenience, the estimated full run times of the AL framework are also listed in the table, and the times have been converted to hours.

	Warm-up	Random	ε-greedy	TS	Tanimoto
Only AL (T_{AL})	0	5.59	6.34	17.11	6.17
	200	3.45	3.85	13.07	4.12
Estimated (T_{est})	0	22.45	23.20	33.96	23.02
	200	27.04	27.44	36.66	27.71

Table 3.2: Estimated full run times of the warm-up experiments. "Only AL" refers to the time the programs were run, while "Estimated" refers to the estimate described in Equation 3.1. The times have been converted to hours.

3.2.3 Retrain Interval

The retrain parameter handles the interval of each update of the surrogate model; a retraining parameter of 1 equates to training the surrogate model every epoch. As training the surrogate model is time-intensive, a higher value of retrain will likely lessen the overall time of a run.

3.2.3.1 RMSE

The difference for both 10, 15 and 20 retrain intervals fluctuate around 0, showing very little effect on the RMSE initially or over more epochs across all AFs. For the Tanimoto AF the difference is consistently positive, meaning the default parameter of 5 attains a lower RMSE. The difference, however, is very small and not likely to have any effect.

3.2.3.2 Time per Epoch

The time consumption is consistently lower for longer intervals of retrain across all AFs except for the TS AF where the behavior deviates from the trend of the other AFs. The retrain interval of 20 seems to be the slowest and the retrain interval of 15 the fastest, as can be seen in Figure 3.6.

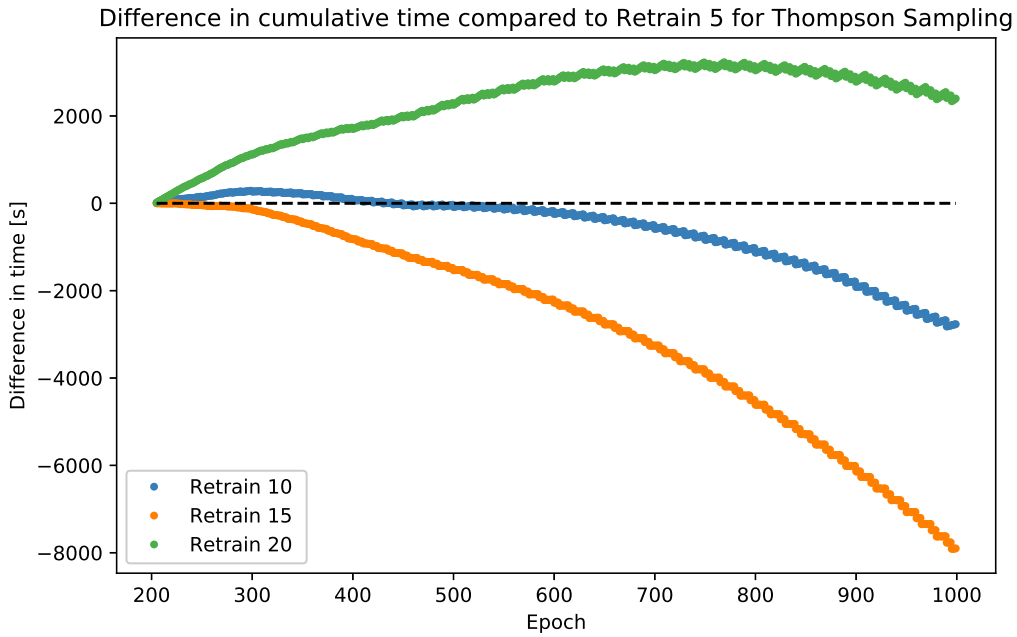

Figure 3.6: Comparison parameters for *retrain* of 10, 15 and 20 compared with the default parameter of 5 for the TS AF. Here one can see that for a larger interval between *retrains*, the model actually gets slower.

3.2.4 Acquisition Batch Size

The AB controls how many points are to be acquired each epoch. A larger AB leads to a greater pool of data for the surrogate model to train on, which should lead to a more accurate predictive performance. However, a larger acquisition batch also means more oracle calls, the avoidance of which is the entire purpose of this project.

3.2.4.1 RMSE

An AB of 20 consistently performs worse than the default AB of 40. A larger AB of 60 produces a smaller average error in both the Random and the Tanimoto AF, but the difference is very small and it performs worse than the default for both the ε-Greedy and the TS AF.

3.2.4.2 Time per Epoch

A larger AB of 60 consistently takes longer to run across all AFs and the smaller AF of 20 is faster for all AFs except for the TS AF where it was faster than an AB of 40.

3.2.5 Summary of Results

Table 3.3 shows the averages of each comparison made against the default parameter for RMSE and for time per epoch. To reiterate, a negative value means that

comparison value, e_c, performs better than the default value, e_d, since it means that the e_c is smaller than the default value and therefore has a smaller RMSE error. Similarly, the difference between t_d and t_c is visualized with a negative value being a shorter time for the comparison value and a positive value being a longer time for the comparison value.

parameter e_c	**RMSE Average Difference per AF**			
	Random	ε-greedy	TS	Tanimoto
initial pooling 10	-0.000	-0.001	-0.002	-0.004
15	0.002	0.015	0.018	-0.004
Warm-up 0	0.048	0.114	0.167	0.088
retrain 10	0.001	0.005	-0.002	0.004
15	-0.004	0.005	0.010	0.008
20	0.017	0.005	0.024	0.012
AB 20	0.033	0.030	0.153	0.094
60	-0.022	0.043	0.008	-0.069
parameter t_c	**Time Average Difference per AF**			
	Random	ε-greedy	TS	Tanimoto
initial pooling 10	1.816	-1.672	-5.962	-1.366
15	1.787	1.741	-0.490	-0.014
Warm-up 0	8.471	9.926	6.306	8.008
retrain 10	-6.572	-8.255	-3.481	-9.621
15	-8.975	-10.166	-9.939	-12.218
20	-10.757	-12.886	3.017	-12.835
AB 20	-6.376	-7.595	-3.050	-8.976
60	10.961	10.210	9.266	7.327

Table 3.3: Table showing the average difference for all parameters across all AFs for the full 800 or 1000 epochs, depending on the tuning parameter. The errors are in kcal/mol, and the times are in seconds.

For the RMSE comparison, the average difference across all AFs for all parameters is very small, with the largest difference being 0.167 kcal/mol for a warm-up of 0 instead of 200 for the TS AF, meaning that for a warm-up of 200 there is a decrease of the RMSE by an average of 0.167 kcal/mol every epoch.

For the time per epoch comparison, there are larger differences across the tuning parameters and AFs. The *retrain* parameter has the largest impact on time per epoch, with a larger *retrain* interval correlating with a shorter time consumption. There is an outlier for the TS AF for the *retrain* interval of 20, which takes longer when the trend shows it should be faster. This will be discussed in chapter 4.

Run	Average time per epoch per AF			
	Random	ε-greedy	TS	Tanimoto
Standard	15.6	17.4	59.2	18.7
AB 20	9.3	9.8	56.1	9.7
60	26.6	27.6	68.4	26.0
Warm-up 0	20.2	22.9	61.9	22.3
Initial pooling 10	17.5	15.8	53.3	17.4
15	17.6	19.4	58.9	18.9
Retrain 10	9.1	9.2	55.7	9.1
15	6.7	7.2	49.2	6.4
20	4.9	4.5	62.2	5.8

Table 3.4: Average times per epoch of the different runs. The times are given in seconds.

4

Conclusion

This project concludes that out of the evaluated combinations of surrogate model and AF, the RF surrogate model paired with the Random AF performed the best with regards to time per epoch and acceptable error. It reached a minimum RMSE of 1.796294 within 1000 epochs and took approximately 3.5 hours to finish. There were combinations which performed better if only RMSE is regarded, but with a substantial time increase involved, e.g. SVR with Tanimoto which had a minimum RMSE of 1.553680 but a run time of approximately 152 hours.

It is worth noting that these experiments were performed using retrospective data and that any potential errors of the surrogate model do not impact the REINVENT agent. A prospective run might develop different behavior, good or bad, especially since there is currently no way to see how the active learning framework affects REINVENT's reinforcement learning. The recommendation is therefore to also use other AFs than the Random AF when performing additional tests, as the difference between AFs for these experiments were at times negligible.

For the parameter tuning experiments, only small differences in RMSE performance could be gleaned and no difference large enough to warrant a change from default parameters. However, the time difference was more noticeable, especially for the *retrain* interval. With only an increase of 0.033 in RMSE when comparing a *retrain* value of 5 to 20 with a decrease of 10.757 seconds per epoch for the Random AF, it is worthwhile to consider a longer *retrain* interval. The TS AF is an outlier for the *retrain* interval parameter, wherein a *retrain* interval of 20 takes longer than an interval of 5. The reason for this behavior is unclear.

A smaller AB would be preferable since it would reduce oracle calls and could shorten computation time while not unduly affecting the RMSE. There is a time decrease of between 6.376 seconds per epoch for the Random AF to up to 8.976 seconds per epoch for the Tanimoto AF while only affecting the RMSE by 0.033 to 0.092, respectively. Worth noting here is that the TS AF is yet again an outlier, where the RMSE is affected more dramatically and the decrease in time per epoch is not as substantial.

The rationale behind implementing the *warm-up* parameter was to account for the initially volatile behavior of the REINVENT agent. However, there is little difference in RMSE between no *warm-up* and a *warm-up* of 200, and thus the time should be the deciding factor.

Changing from a *warm-up* of 200 to a *warm-up* of 0 leads to an increase of 2.5 hours, on average. These results should not be compared entirely in a vacuum, however,

as this is merely an increase in how long the AL framework is active. A *warm-up* of 200 entails that the oracle is run for all molecules during those initial epochs, and as the oracle is costlier than the surrogate model, a larger *warm-up* leads to a net increase in time. On average, decreasing the *warm-up* from 200 to 0 leads to an decreasing of approximately 4 hours for the entire run.

Summarily, this project recommends the active learning framework parameters seen in the table below. Here the recommendation is the Random AF due to its slightly superior performance in this study, but as stated above any future research should consider testing multiple AFs, as the behavior of the AF might be different for other use-cases.

The AL Framework

Model	RF
AF	Random
Initial pooling	5
Warm-up	0
Retrain	20
AB	20

4.1 Discussion

Although there are definitive distinctions between different AFs and surrogate models, especially with regards to time, the errors are large enough that improvements are likely required to effectively utilize the framework in realistic contexts. However, the true behavior of the AL framework cannot be known from only retrospective analysis as there is no feedback from the framework back into REINVENT as there would be in a prospective run. The parameter combination mentioned above are then to be seen as a suggestion for initial exploration, not as a definitive answer.

4.1.1 The Random Acquisition Function

The random acquisition function consistently performed the best over both models and all tuning parameter runs. This might seem counter intuitive when the point of active learning is to intelligently pick points to acquire to augment the training data. To pick points randomly is the opposite of intelligent selection, so why does it perform so well?

One major aspect worth noting is the choice of metrics investigated during the project, i.e RMSE and time per epoch. Choosing these metrics resulted in a focus

on efficiency and accuracy in predicting the score of the molecule, but there is no focus on the docking score itself.

This matters because the exploitative AFs, like Greedy for example, do not optimize for accuracy but for a higher docking score. In other words, the more exploitative acquisition functions try to use existing knowledge to reach a balance between accuracy and the molecular docking score, while purely exploratory AFs only take accuracy into account. However, since the score is not a metric considered, then there is no visible payoff for exploitation at all in the analysis.

In short, there is no way for any exploitation strategy to gain any advantage when only looking at RMSE and time per epoch since an exploratory AF will always be more diverse and a more accurate general model.

4.1.2 The effect of Parameter tuning

The difference in RMSE performance across all AFs and all parameter values is as previously mentioned quite small. There are likely many reasons for this. For the *initial pooling* parameter the difference is among the smallest, and there are two reasons why this might be the case. It might have to do with the REINVENT policy update. The agent might not "move" across chemical space fast enough for an *initial pooling* difference of 10 or 15 to make much of a difference since what REINVENT is sampling very similar SMILES in either case. Additionally, if the *initial pooling* is 5 this means that the active learning will be activated at the 6th epoch and the 0-5 epochs have been pooled. For an *initial pooling* of 10, the active learning will be activated on the 11th epoch, meaning the 0-10 epochs have been pooled. Note the 50% overlap in the respective pools. This is likely a major culprit of the similar performance and hints at the need to use a larger *initial pooling* to get a proper performance boost. However, due to the limited data points and the increase in computational time, this is likely not a practical solution for this project.

The AB parameter is also not producing a dramatic change in the RMSE. It is, however, slowing down the run time since if the framework acquires more points per epoch it will take longer to train. Additionally, a larger acquisition batch means more oracle calls, which is not desirable since the point of this project is to do away with as many oracle calls as possible. While acquiring more points per epoch leads to a more exploratory behavior which has shown to be advantageous for the framework (seeing as the Random AF performs the best out of the AFs) our research indicates that an AB of 20 is enough to receive a sufficiently exploratory behavior. There is little benefit to acquiring more points than 20 since it adds substantially to the computational cost.

There is a similar behavior for the *retrain* interval. A surprisingly stable RMSE can be seen even for longer *retrain* intervals. This is why the decision was made to change from the default parameter of 5 *retrain* to an interval of 20. For further research, it might be worth looking into even longer *retrain* intervals to see if the RMSE can retain its stability while further reducing the time.

4.1.3 Outlier Analysis

There are some notable outliers for the parameter tuning experiments worth discussing. For example, the TS AF has some erratic behavior for the *retrain* experiments. The time per epoch measurements show that the longer interval is actually slower, which does not make intuitive sense since training the system more should take more time.

The runs are all done in triplicate so as to mitigate outliers due to computational fluctuations. However, when plotting these out in Figures 4.1, 4.2 and 4.3, looking at the replicate runs individually we do see the outlier behavior in nearly all replicates, when compared with the default run of a *retrain* of 5, also done in 3 replicates. The only run that does not show this behavior is when comparing default replicate 1 with comparison replicate 2, where the *retrain* interval of 20 is faster than a *retrain* interval value of 5, at least after about 350 epochs.

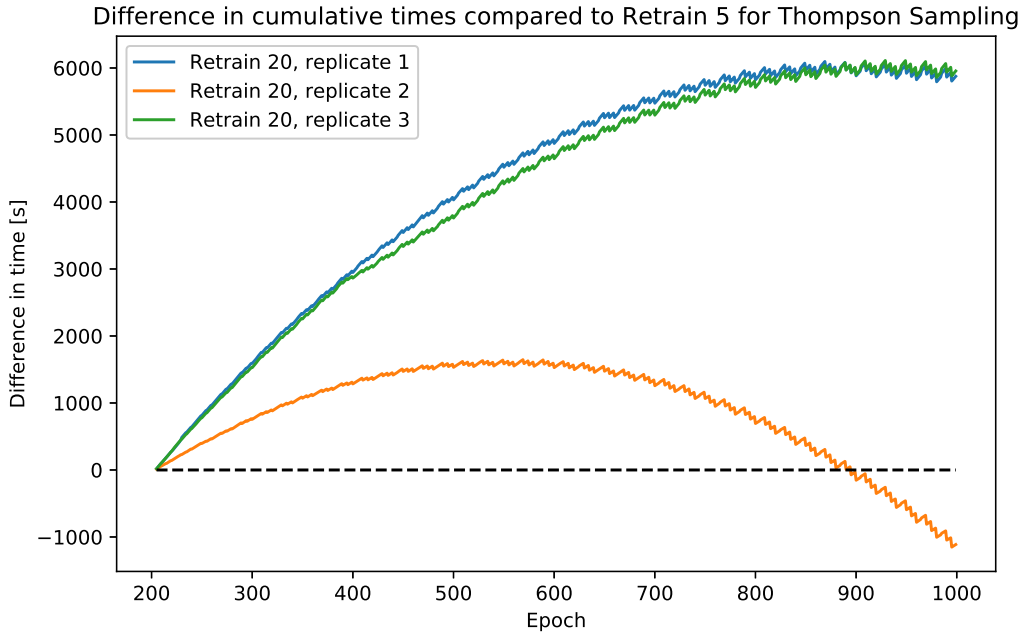

Figure 4.1: Comparison of the difference in cumulative time between a *retrain* interval of 20 and a *retrain* interval of 5 for, plotted for all 3 replicates of the comparison value compared with replicate 1 for the default value.

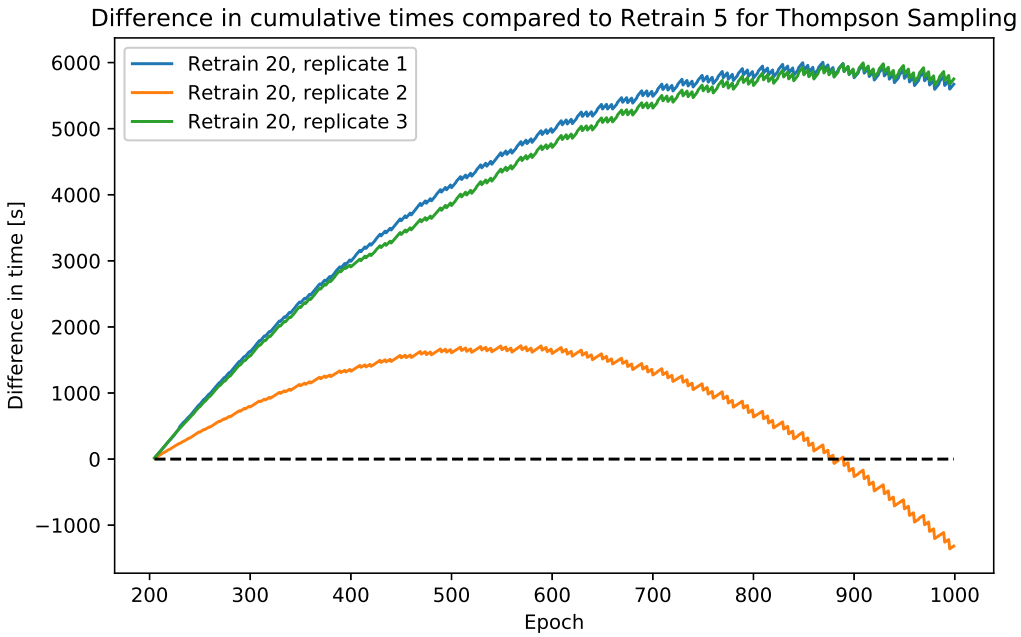

Figure 4.2: Comparison of the difference in cumulative time between a *retrain* interval of 20 and a *retrain* interval of 5 for, plotted for all 3 replicates of the comparison value compared with replicate 2 for the default value.

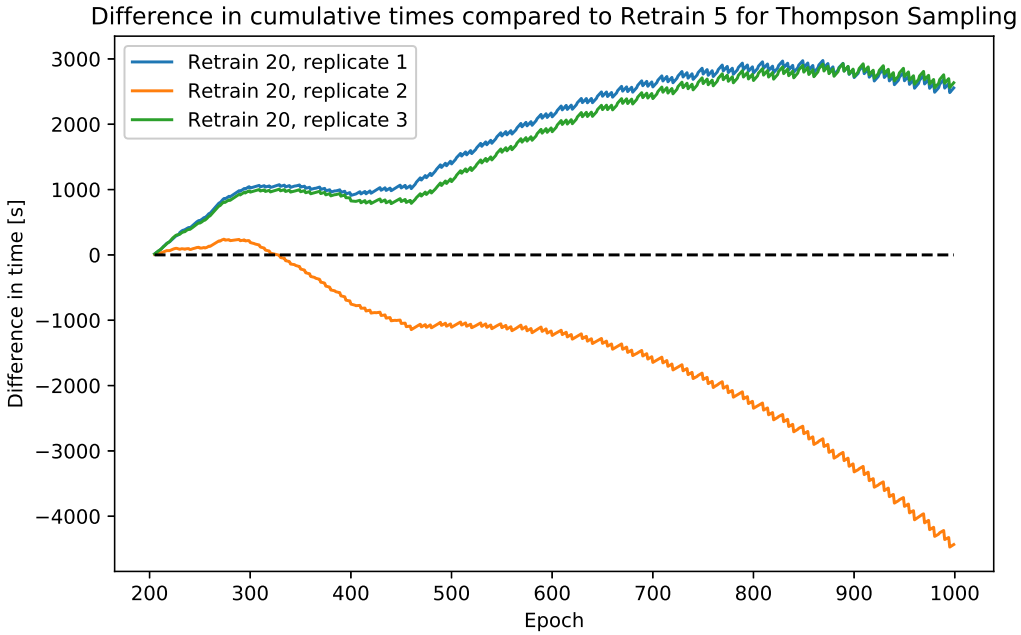

Figure 4.3: Comparison of the difference in cumulative time between a *retrain* interval of 20 and a *retrain* interval of 5 for, plotted for all 3 replicates of the comparison value compared with replicate 3 for the default value.

These runs were made within similar time frames so while the pervasive behavior suggests that the behavior seen for the TS AF in regards to *retrain* interval is less likely to be due to computational fluctuations, it is still a possible culprit. More research is needed to properly identify the cause.

4.1.4 Future Research

The general context in which this study was made was to investigate whether active learning could be used with a molecular design tool such as REINVENT to hasten its training and enable more effective usage of such *de novo* molecular design tools. A prospective run would use the results from the surrogate model as feedback for the molecular design tool, impacting which molecules are generated. The errors are to be seen as a measure of how well the surrogate model accounts for the sampling behavior of the particular molecular design tool used, REINVENT. If the discrepancy between said sampling behavior and the results from our surrogate model is large, utilizing the model in a prospective context can lead to less useful molecular design tools. Caution should thus be used when considering to replace an oracle such as docking using this framework. It can however be used as a springboard for further analysis.

Additionally, there are aspects of this framework's behavior that can only be illustrated using a prospective run, meaning when the framework fully integrates with REINVENT and feeds the scores predicted by the model back into REINVENT. Only then is it possible to know the overall effect of the AL framework on the molecular design. This should be the first step in any further analysis of this framework.

Moreover, other metrics than RMSE and time per epoch should be analysed, especially if prospective runs are made. As mentioned, the current analysis does not take into account the quality of the molecules, only the accuracy of the surrogate model. In theory, a more exploitative acquisition function could have a lower RMSE but could still be a better choice since it might be better at predicting the scores of molecules with a higher docking score. Meaning, it would be bad at predicting the score of the molecules that would be discarded either way.

Finally, if the *exploitation/exploration* balance can be solved, and the framework shows promising results in prospective runs, further parameter tuning to optimize the framework could be of use. Testing even lower acquisition batches and longer retrain intervals would be of interest in the name of further optimizing this framework.

Bibliography

[1] Joseph A DiMasi. "Pharmaceutical R&D performance by firm size: Approval success rates and economic returns". In: *American journal of therapeutics* 21.1 (2014), pp. 26–34.

[2] Jean-Louis Reymond. "The chemical space project". In: *Accounts of Chemical Research* 48.3 (2015), pp. 722–730.

[3] Pavel G Polishchuk, Timur I Madzhidov, and Alexandre Varnek. "Estimation of the size of drug-like chemical space based on GDB-17 data". In: *Journal of computer-aided molecular design* 27.8 (2013), pp. 675–679.

[4] Luca Pinzi and Giulio Rastelli. "Molecular docking: shifting paradigms in drug discovery". In: *International journal of molecular sciences* 20.18 (2019), p. 4331.

[5] José Jiménez-Luna et al. "Artificial intelligence in drug discovery: Recent advances and future perspectives". In: *Expert Opinion on Drug Discovery* (2021), pp. 1–11.

[6] Thomas Blaschke et al. "REINVENT 2.0: an AI tool for de novo drug design". In: *Journal of Chemical Information and Modeling* 60.12 (2020), pp. 5918–5922.

[7] AkshatKumar Nigam, Robert Pollice, and Alan Aspuru-Guzik. "JANUS: Parallel Tempered Genetic Algorithm Guided by Deep Neural Networks for Inverse Molecular Design". In: *arXiv preprint arXiv:2106.04011* (2021).

[8] Rafael Gómez-Bombarelli et al. "Automatic chemical design using a data-driven continuous representation of molecules". In: *ACS central science* 4.2 (2018), pp. 268–276.

[9] Josep Arús-Pous et al. "Randomized SMILES strings improve the quality of molecular generative models". In: *Journal of cheminformatics* 11.1 (2019), pp. 1–13.

[10] Veronica Salmaso and Stefano Moro. "Bridging molecular docking to molecular dynamics in exploring ligand-protein recognition process: an overview". In: *Frontiers in pharmacology* 9 (2018), p. 923.

[11] Zoe Cournia et al. "Rigorous free energy simulations in virtual screening". In: *Journal of Chemical Information and Modeling* 60.9 (2020), pp. 4153–4169.

[12] David E Graff, Eugene I Shakhnovich, and Connor W Coley. "Accelerating high-throughput virtual screening through molecular pool-based active learning". In: *Chemical Science* (2021).

[13] Francesco Gentile et al. “Deep docking: a deep learning platform for augmentation of structure based drug discovery”. In: *ACS central science* 6.6 (2020), pp. 939–949.

[14] Ying Yang et al. “Efficient exploration of chemical space with docking and deep learning”. In: *Journal of Chemical Theory and Computation* 17.11 (2021), pp. 7106–7119.

[15] Wei Ma et al. “Deep Learning Model of Dock by Dock Process Significantly Accelerate the Process of Docking-based Virtual Screening”. In: *arXiv preprint arXiv:2110.10918* (2021).

[16] Burr Settles. *Active Learning Literature Survey*. Computer Sciences Technical Report 1648. University of Wisconsin–Madison, 2009.

[17] Xueying Zhan et al. “A Comparative Survey: Benchmarking for Pool-based Active Learning”. In: *Proceedings of the Thirtieth Joint Conference on Artificial Intelligence (IJCAI-21)* (2021).

[18] Peter I Frazier. “A tutorial on Bayesian optimization”. In: *arXiv preprint arXiv:1807.02811* (2018).

[19] Jonas Mockus. *Bayesian approach to global optimization: theory and applications*. Vol. 37. Springer Science & Business Media, 2012.

[20] Yanyao Shen et al. “Deep active learning for named entity recognition”. In: *arXiv preprint arXiv:1707.05928* (2017).

[21] EMBL-EBI. *ChEMBL Database*. 2022. URL: `https://www.ebi.ac.uk/chembl/` (visited on 05/05/2022).

[22] David Weininger. “SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules”. In: *Journal of chemical information and computer sciences* 28.1 (1988), pp. 31–36.

[23] Thomas Blaschke et al. *Supporting Information REINVENT 2.0–an AI tool for de novo drug design*. URL: `https://pubs.acs.org/doi/10.1021/acs.jcim.0c00915?goto=supporting-info`.

[24] Marwin HS Segler et al. “Generating focused molecule libraries for drug discovery with recurrent neural networks”. In: *ACS central science* 4.1 (2018), pp. 120–131.

[25] Marcus Olivecrona et al. “Molecular de-novo design through deep reinforcement learning”. In: *Journal of cheminformatics* 9.1 (2017), pp. 1–14.

[26] J Harry Moore et al. “Icolos: A workflow manager for structure based post-processing of de novo generated small molecules”. In: (2022).

[27] Garrett Morris and Marguerita Lim-Wilby. “Molecular Docking”. In: *Methods in molecular biology (Clifton, N.J.)* 443 (Feb. 2008), pp. 365–82. DOI: `10.1007/978-1-59745-177-2_19`.

[28] Julian Berk et al. “Randomised gaussian process upper confidence bound for bayesian optimisation”. In: *arXiv preprint arXiv:2006.04296* (2020).

[29] Natalie K Boyd, Chengwen Teng, and Christopher R Frei. “Brief overview of approaches and challenges in new antibiotic development: A focus on drug repurposing”. In: *Frontiers in Cellular and Infection Microbiology* 11 (2021), p. 442.

[30] Asher Mullard. “Parsing clinical success rates”. In: *Nature Reviews Drug Discovery* 15.7 (2016), pp. 447–448.

[31] Seumas Miller and Michael J Selgelid. "Ethical and philosophical consideration of the dual-use dilemma in the biological sciences". In: *Science and engineering ethics* 13.4 (2007), pp. 523–580.

[32] SPEIZ Laboratory. *Spiez CONVERGENCE*. 2022. URL: `https://www.spiezlab.admin.ch/en/home/meta/refconvergence.html` (visited on 05/09/2022).

[33] Fabio Urbina et al. "Dual use of artificial-intelligence-powered drug discovery". In: *Nature Machine Intelligence* 4.3 (2022), pp. 189–191.

[34] G Richard Bickerton et al. "Quantifying the chemical beauty of drugs". In: *Nature chemistry* 4.2 (2012), pp. 90–98.

[35] S Kavitha, S Varuna, and R Ramya. "A comparative analysis on linear regression and support vector regression". In: *2016 online international conference on green engineering and technologies (IC-GET)*. IEEE. 2016, pp. 1–5.

[36] Trevor Hastie et al. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer, 2009.

[37] Jeremy Beauchamp. *Decision Tree vs. Random Forest*. 2020. URL: `https://commons.wikimedia.org/wiki/File:Decision_Tree_vs._Random_Forest.png` (visited on 05/17/2022). This file is licensed under the Creative Commons Attribution-Share Alike 4.0 International license.

[38] Alex J Smola and Bernhard Schölkopf. "A tutorial on support vector regression". In: *Statistics and computing* 14.3 (2004), pp. 199–222.

[39] Mariette Awad and Rahul Khanna. "Support vector regression". In: *Efficient learning machines*. Springer, 2015, pp. 67–80.

[40] George De Ath et al. "ε-shotgun: ε-greedy batch bayesian optimisation". In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 2020, pp. 787–795.

[41] Harold J Kushner. "A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise". In: (1964).

[42] Eric Brochu, Vlad M Cora, and Nando De Freitas. "A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning". In: *arXiv preprint arXiv:1012.2599* (2010).

[43] Donald R Jones, Matthias Schonlau, and William J Welch. "Efficient global optimization of expensive black-box functions". In: *Journal of Global optimization* 13.4 (1998), pp. 455–492.

[44] Daniel J Russo et al. "A tutorial on thompson sampling". In: *Foundations and Trends® in Machine Learning* 11.1 (2018), pp. 1–96.

[45] Peter Willett, John M Barnard, and Geoffrey M Downs. "Chemical similarity searching". In: *Journal of chemical information and computer sciences* 38.6 (1998), pp. 983–996.

[46] Harry L Morgan. "The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service." In: *Journal of chemical documentation* 5.2 (1965), pp. 107–113.

A

Selection of Model and Acquisition Function

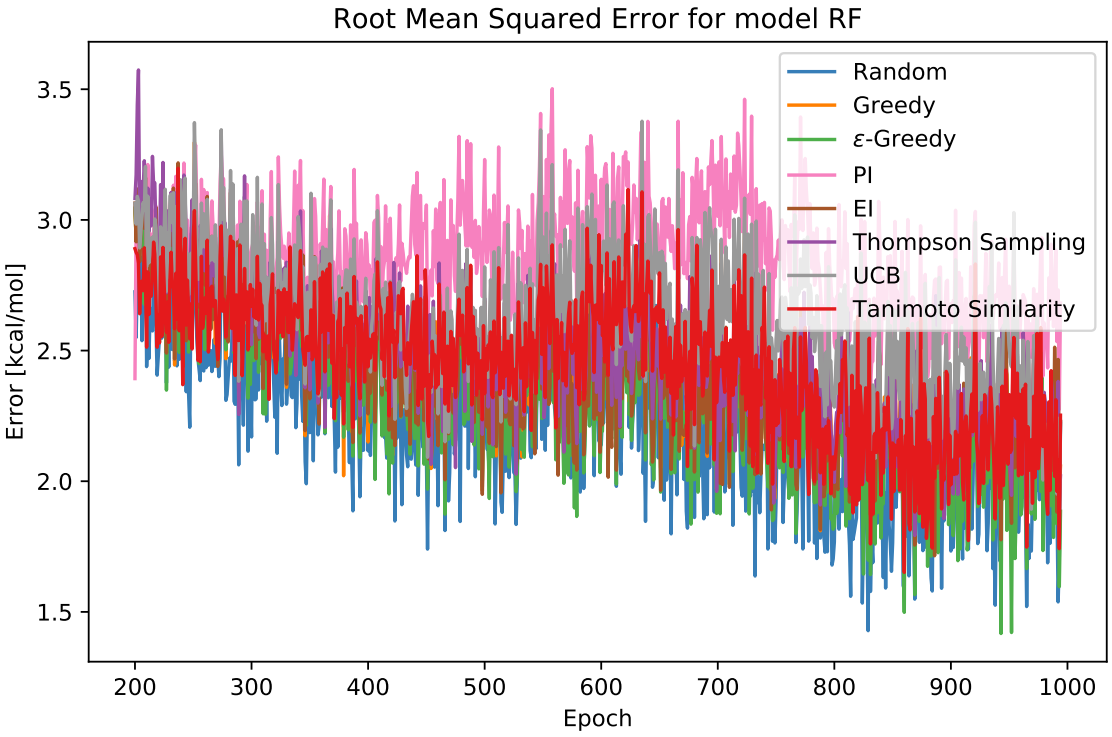

Figure A.1: The RMSEs for all the acquisition functions using the Random Forest model.

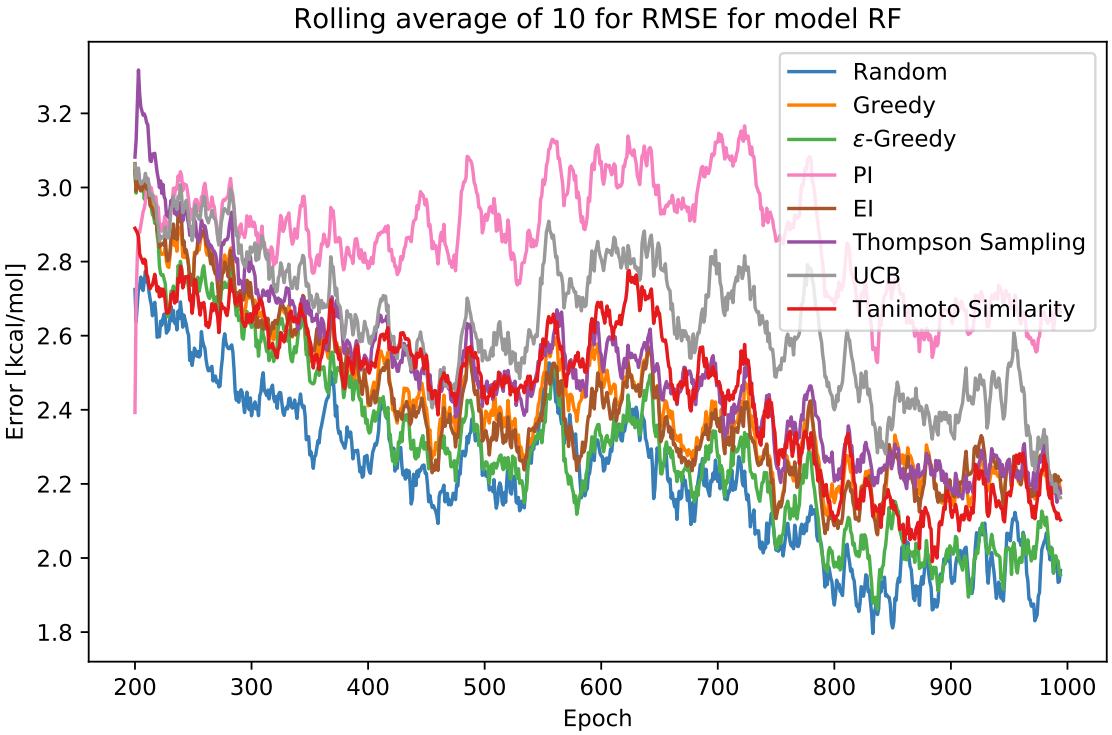

Figure A.2: The rolling average of the RMSEs using the Random Forest model. All acquisition functions are present. The window used for the rolling average is 10, with a minimum window of 1.

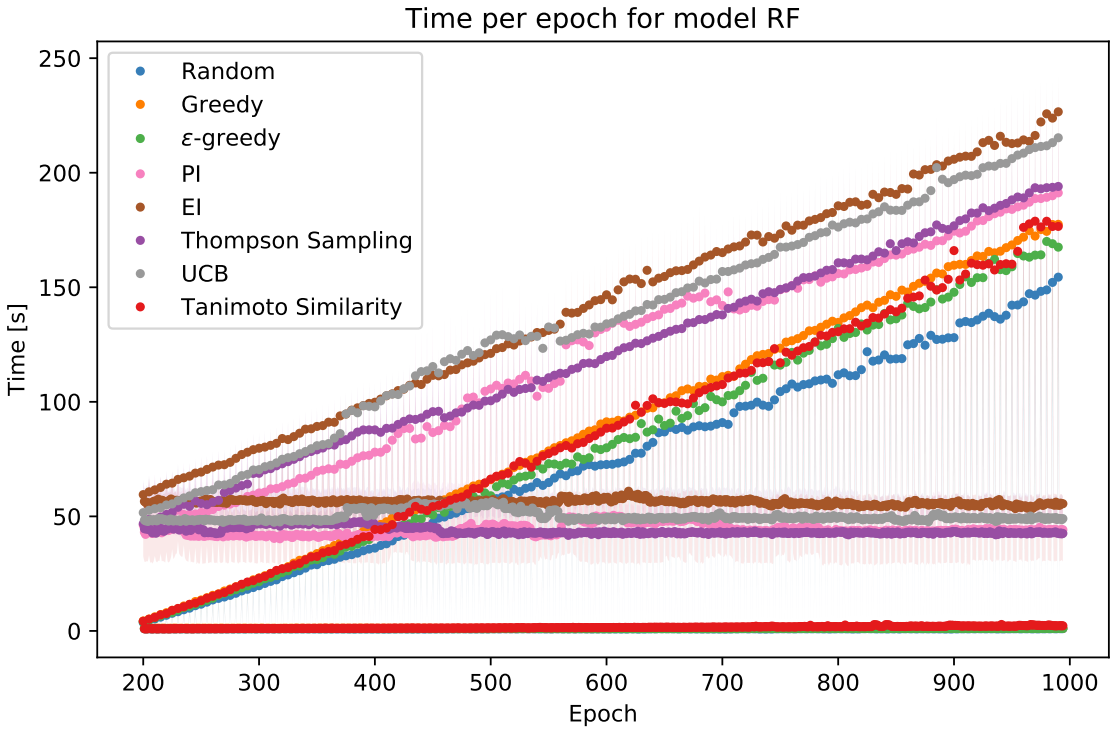

Figure A.3: The times taken by every acquisition function using the Random Forest model. Across all the acquisition functions, there are two lines shown: one constant, which shows the time taken during epochs where retraining does not occur, and one growing (typically linear), which represents the time taken during epochs where retraining occurs.

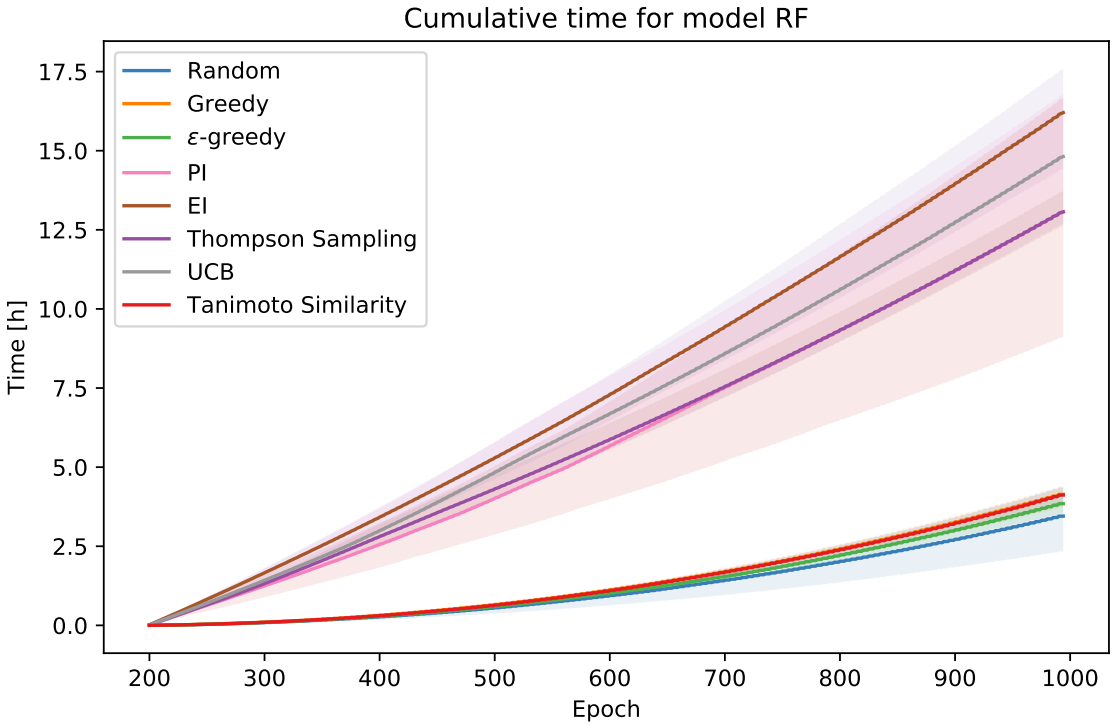

Figure A.4: The cumulative time taken by the Random Forest model. There are two groups present in the figure. The group that took less time, i.e. the lower one, contains the acquisition functions that do not require calculation of the standard deviation, while the one that took more time, i.e. the higher one, contains those that do require standard deviation.

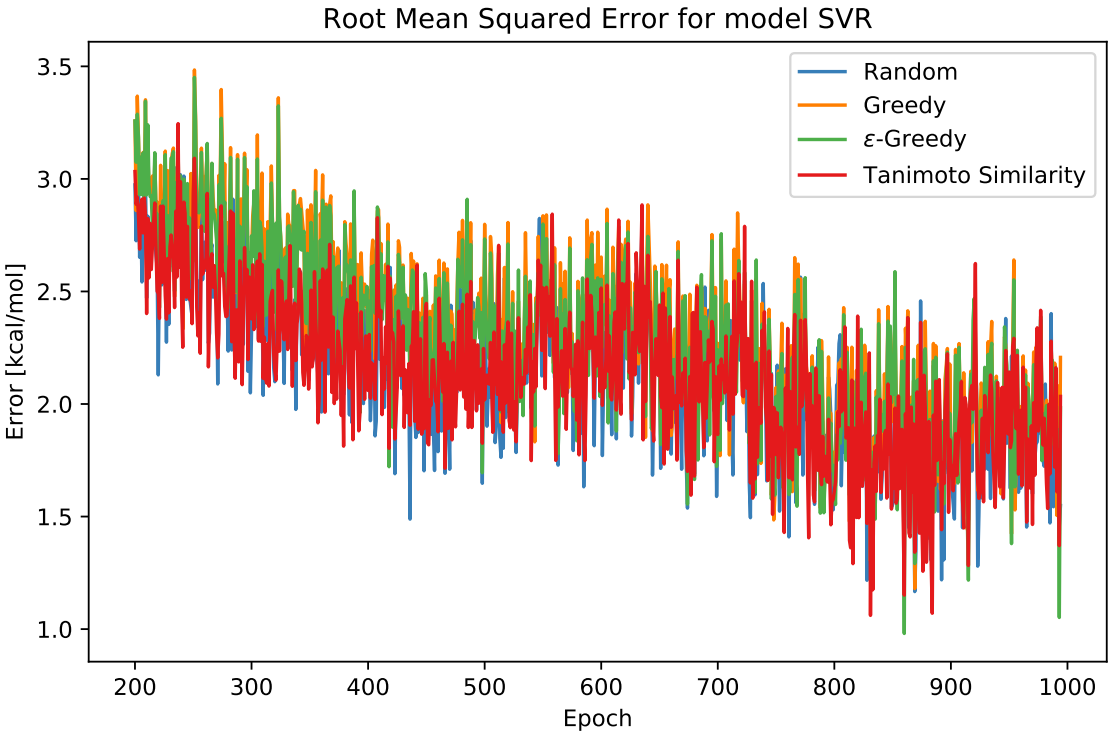

Figure A.5: The RMSEs for all the acquisition functions using the Support Vector Regression model. Note that there are fewer implemented than for the Random Forest model. The acquisition functions missing are the ones requiring standard deviation, as that was not implemented for the Support Vector Regression model.

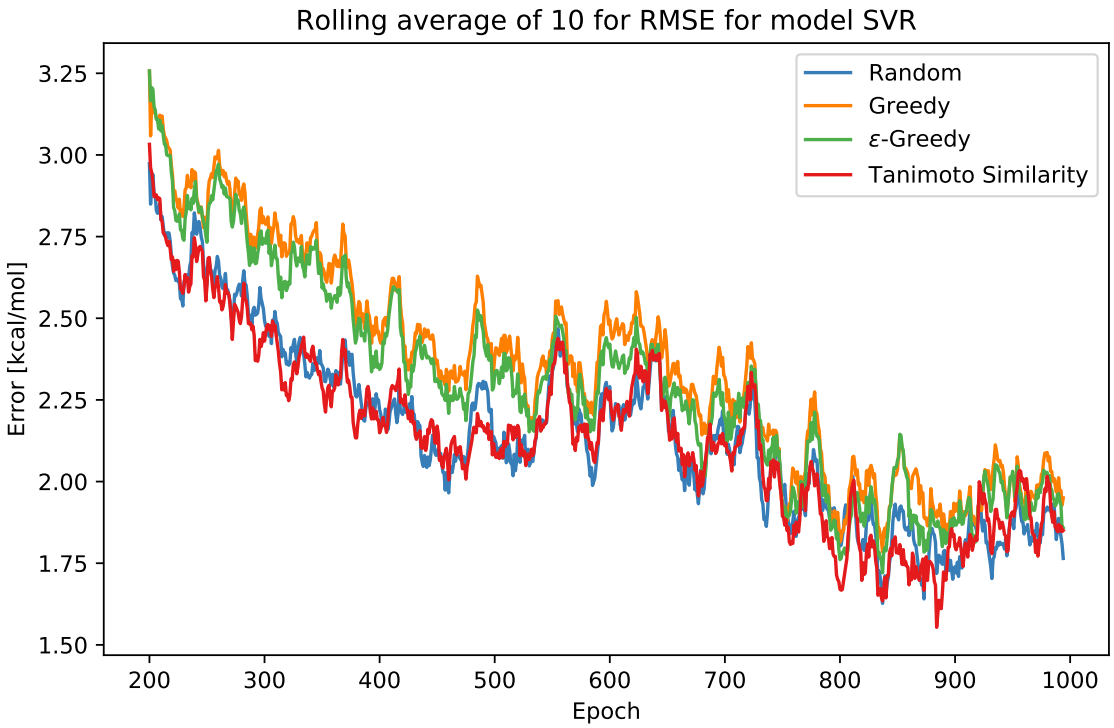

Figure A.6: The rolling average of the RMSEs using the Support Vector Regression model. All implemented acquisitions, i.e. the ones not requiring standard deviation, are present. The window used for the rolling average is 10, with a minimum window of 1.

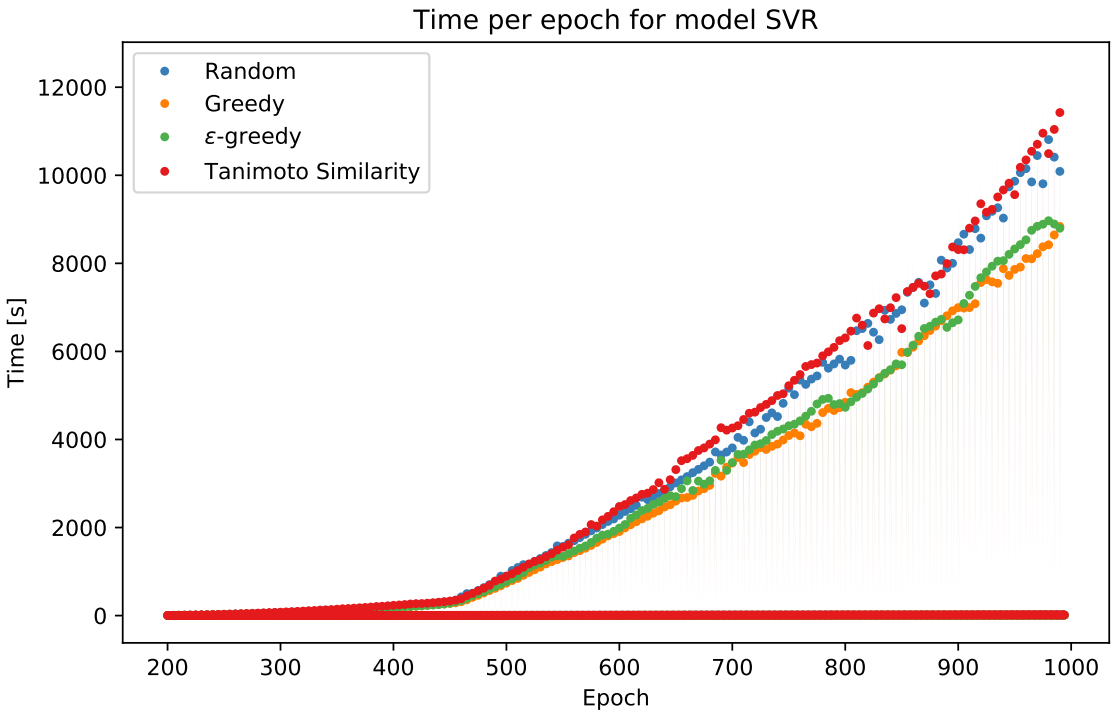

Figure A.7: The times taken by every implemented acquisition function using the Support Vector Regression model. There are two lines present for every acquisition function, one constant (close to zero) and one growing. The constant line represents the epochs where training does not occur, while the growing lines represent epochs where the model is retrained.

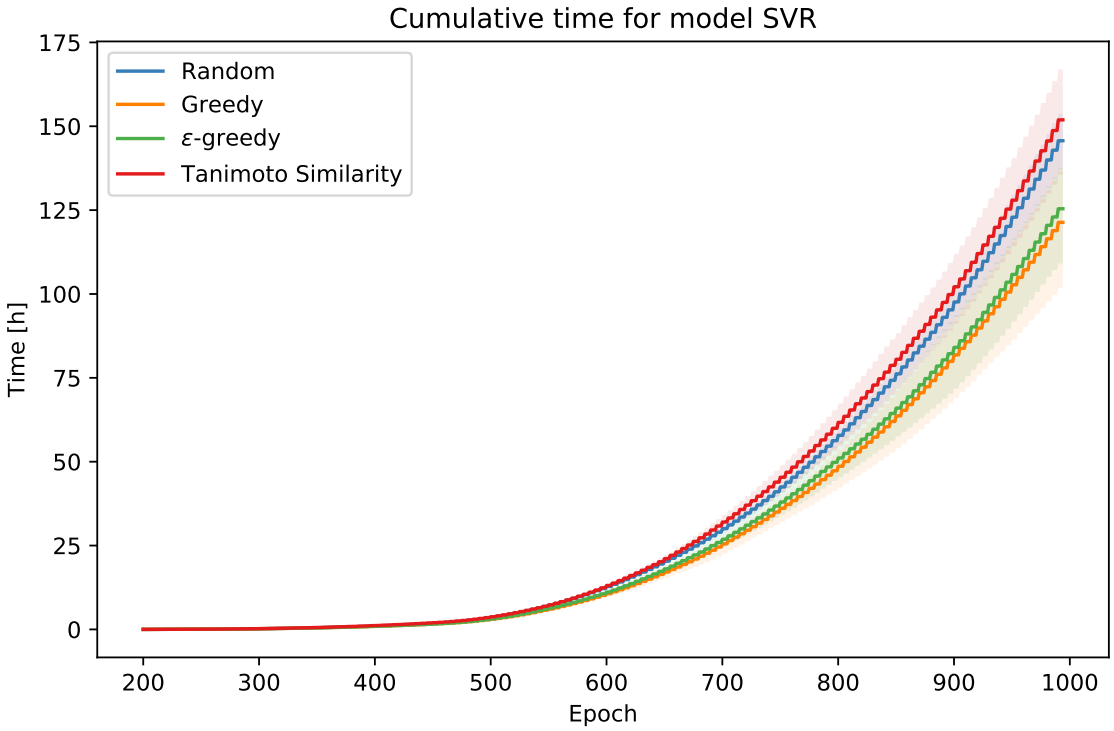

Figure A.8: The cumulative time taken by the Support Vector Regression model.

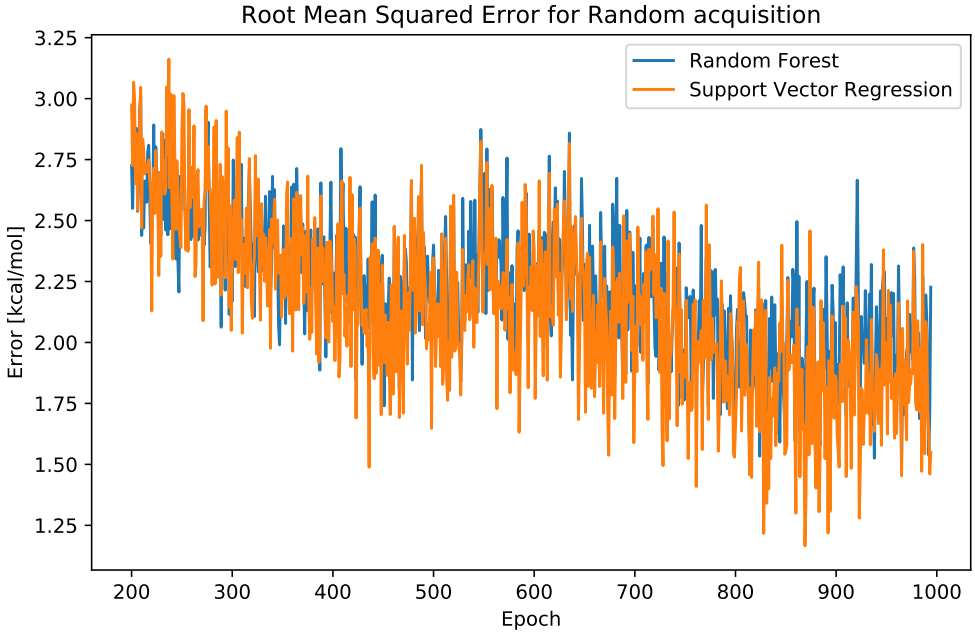

Figure A.9: Comparison of the RMSEs between the Random Forest model and the Support Vector Regression model for the Random acquisition function.

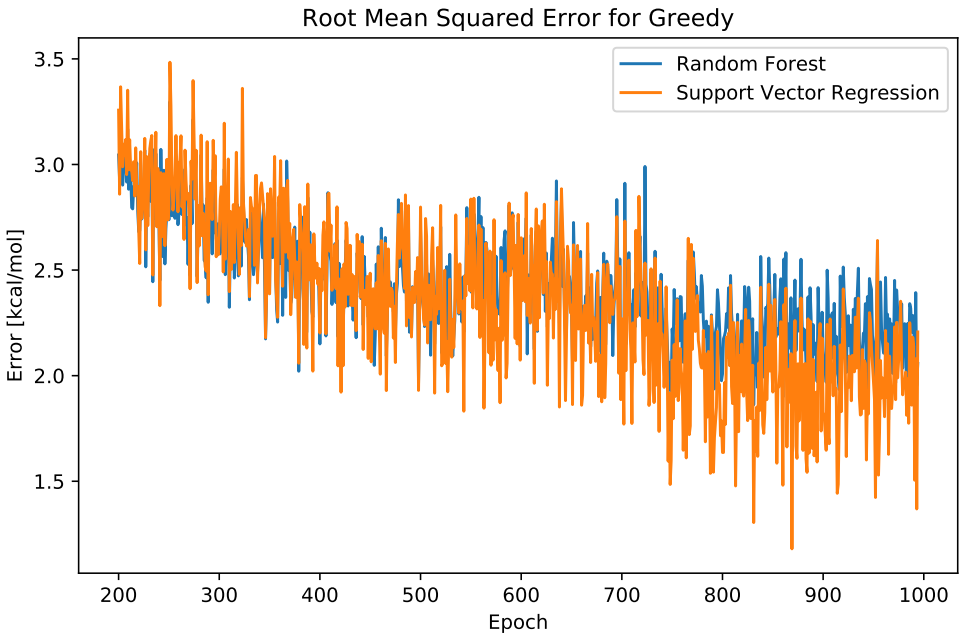

Figure A.10: Comparison of the RMSEs between the Random Forest model and the Support Vector Regression model for the Greedy acquisition function.

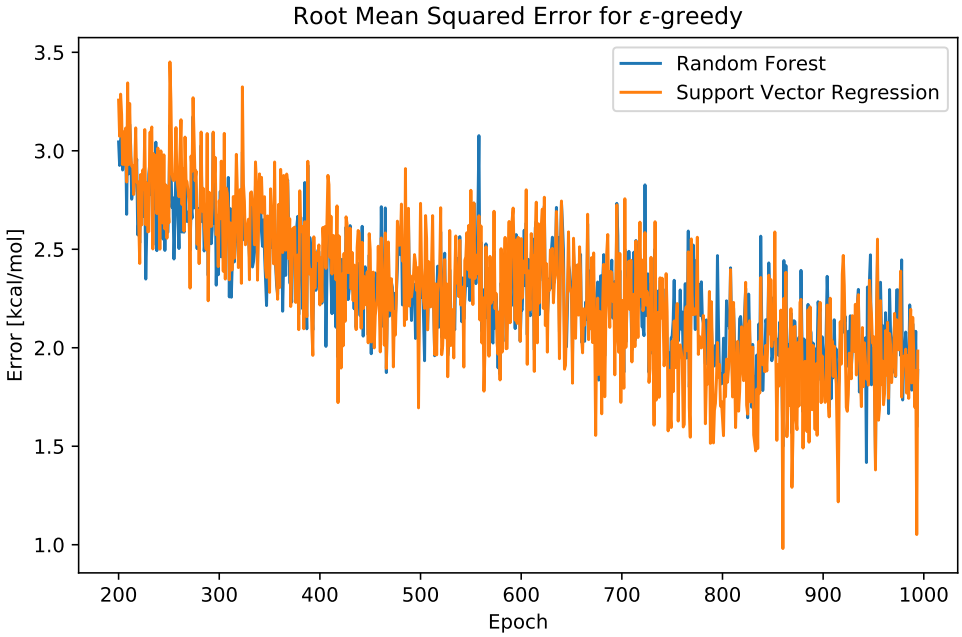

Figure A.11: Comparison of the RMSEs between the Random Forest model and the Support Vector Regression model for the ε-Greedy acquisition function.

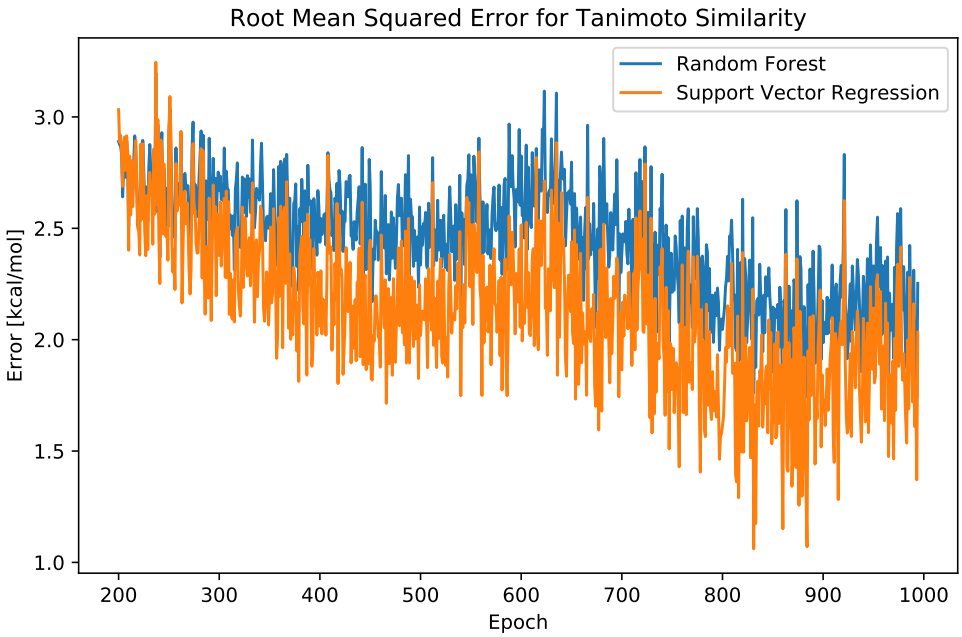

Figure A.12: Comparison of the RMSEs between the Random Forest model and the Support Vector Regression model for the Tanimoto Similarity acquisition function.

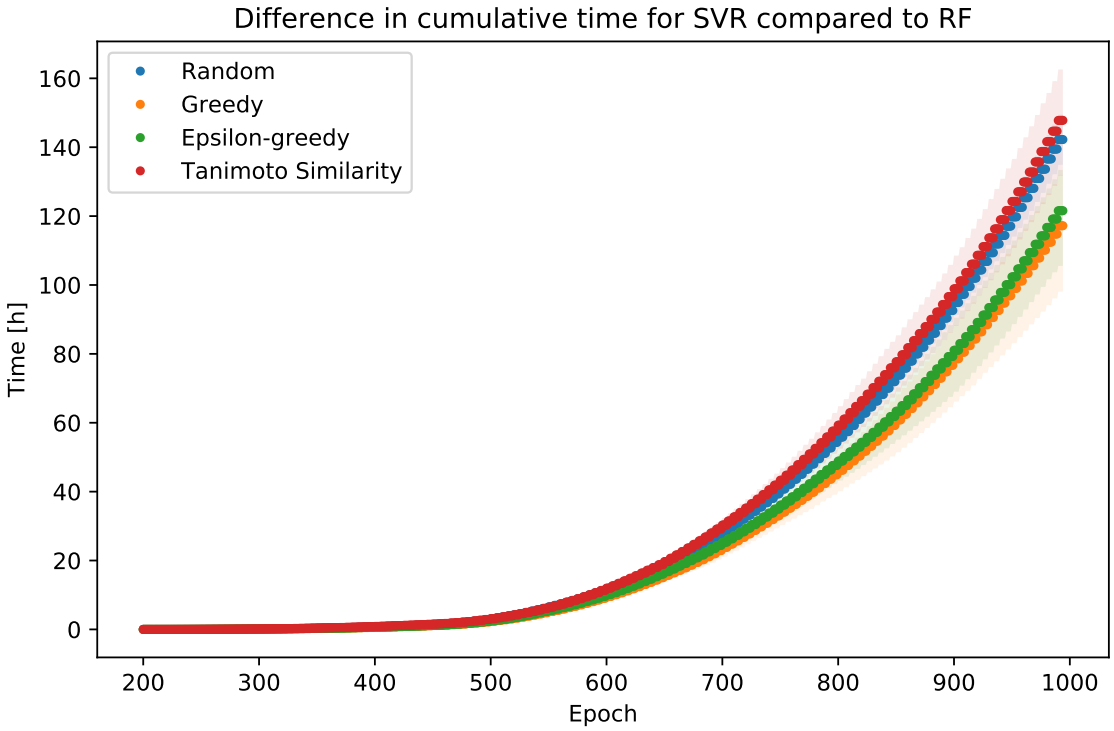

Figure A.13: The difference in cumulative runtime between the Support Vector Regression model and the Random Forest model. The curves measure how much longer Support Vector Regression took than Random Forest.

B

Parameter Tuning

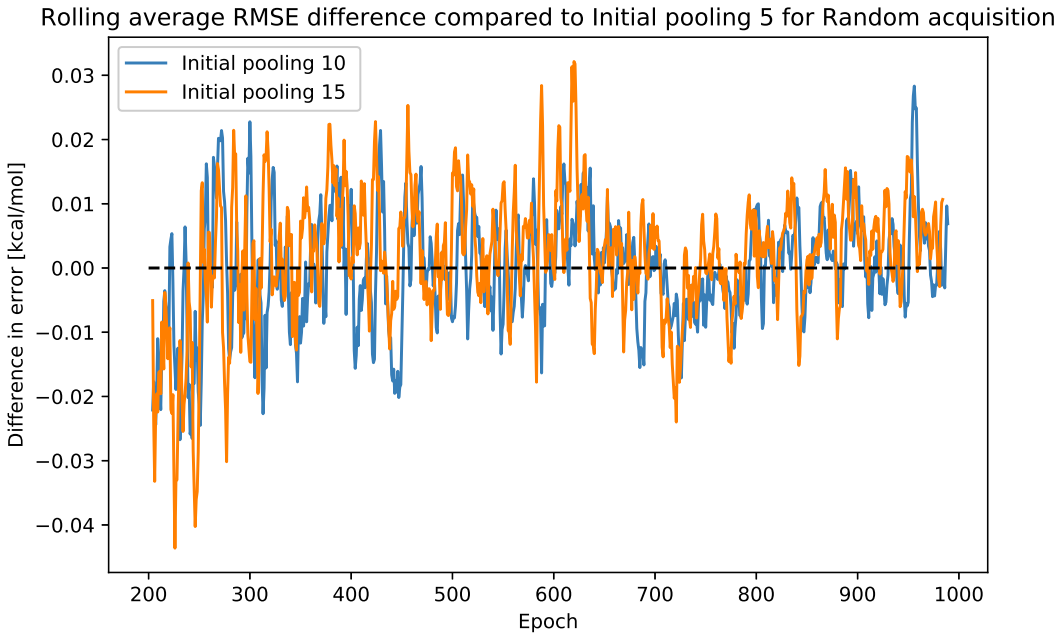

Figure B.1: Comparison parameters for *initial pooling* of 10 and 15 compared with the default value of 5 for the Random AF. There is an initial decrease in error for the first 100 epochs, after that the difference for both 10 and 15 fluctuate around 0, showing very little effect on the RMSE initially or over more epochs.

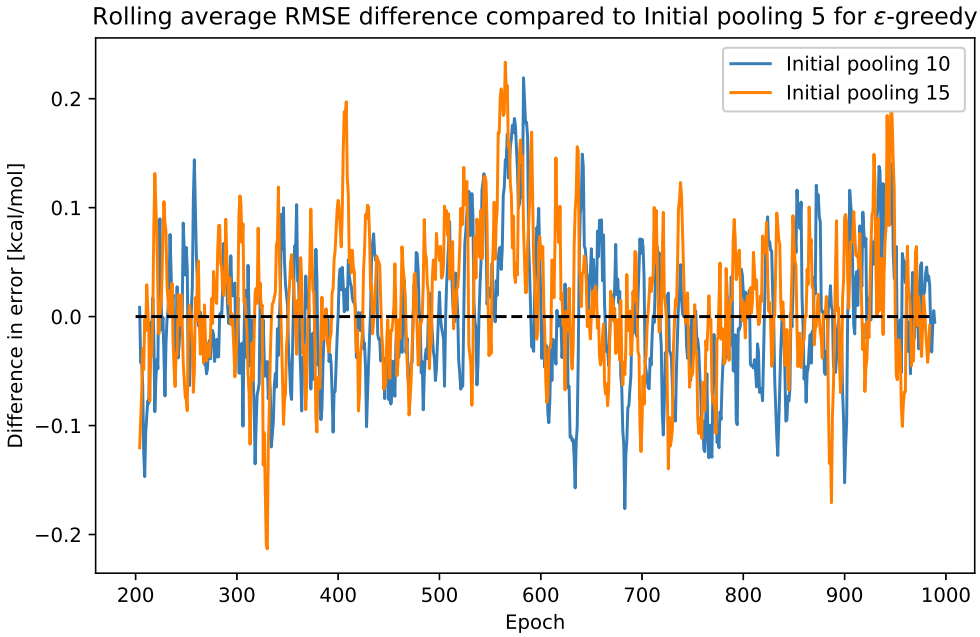

Figure B.2: Comparison parameters for *initial pooling* of 10 and 15 compared with the default value of 5 for the ε-Greedy AF. The difference for both 10 and 15 fluctuate around 0, showing very little effect on the RMSE initially or over more epochs.

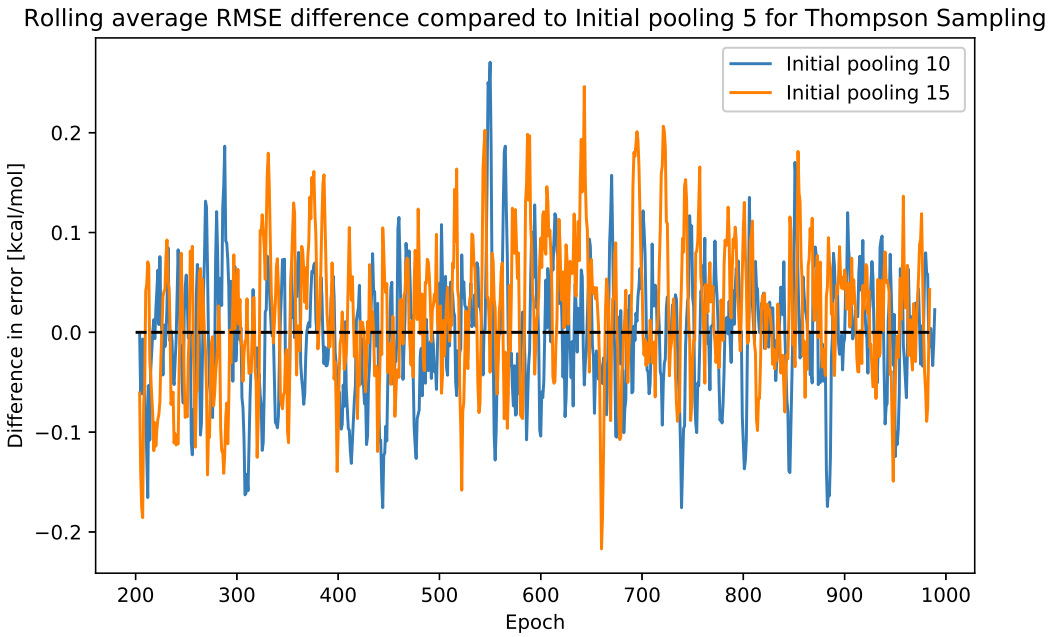

Figure B.3: Comparison parameters for *initial pooling* of 10 and 15 compared with the default value of 5 for the TS AF. The difference for both 10 and 15 fluctuate around 0, showing very little effect on the RMSE initially or over more epochs.

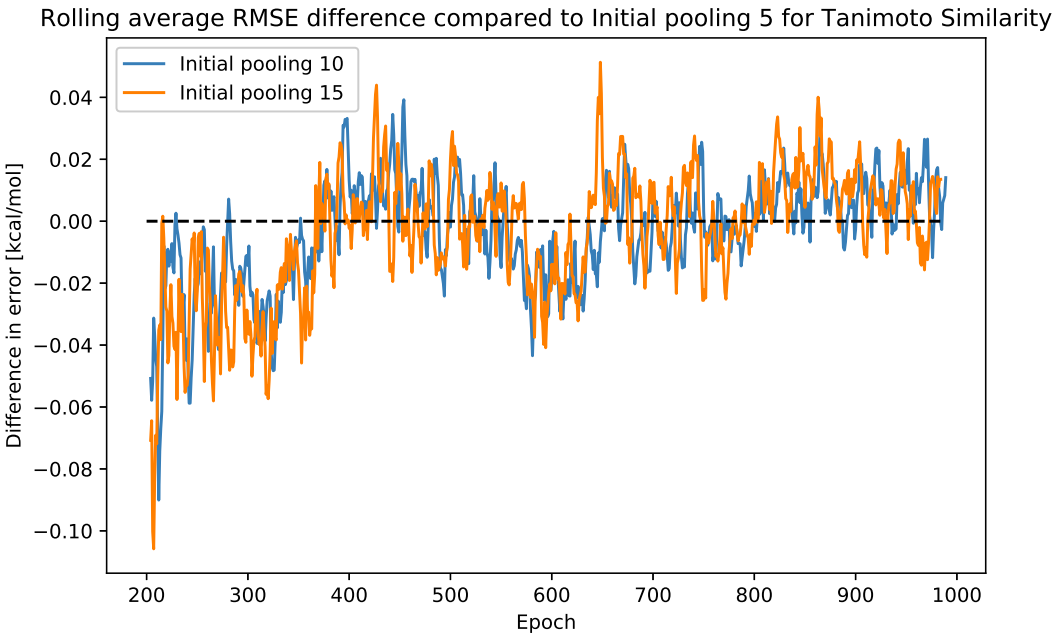

Figure B.4: Comparison parameters for *initial pooling* of 10 and 15 compared with the default value of 5 for the Tanimoto AF. There is an initial decrease in error for the first 100 epochs, after that the difference for both 10 and 15 fluctuate around 0, showing very little effect on the RMSE initially or over more epochs.

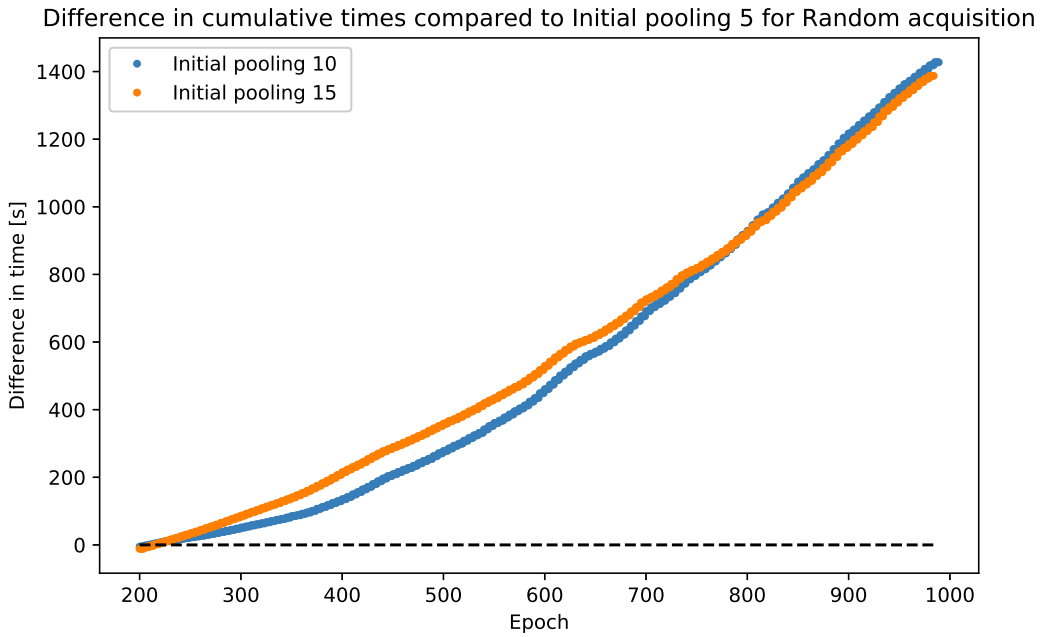

Figure B.5: Comparison parameters for *initial pooling* of 10 and 15 compared with the default value of 5 for the Random AF. Both an initial pooling of 10 and 15 takes longer than the default parameter of 5 initial pooling.

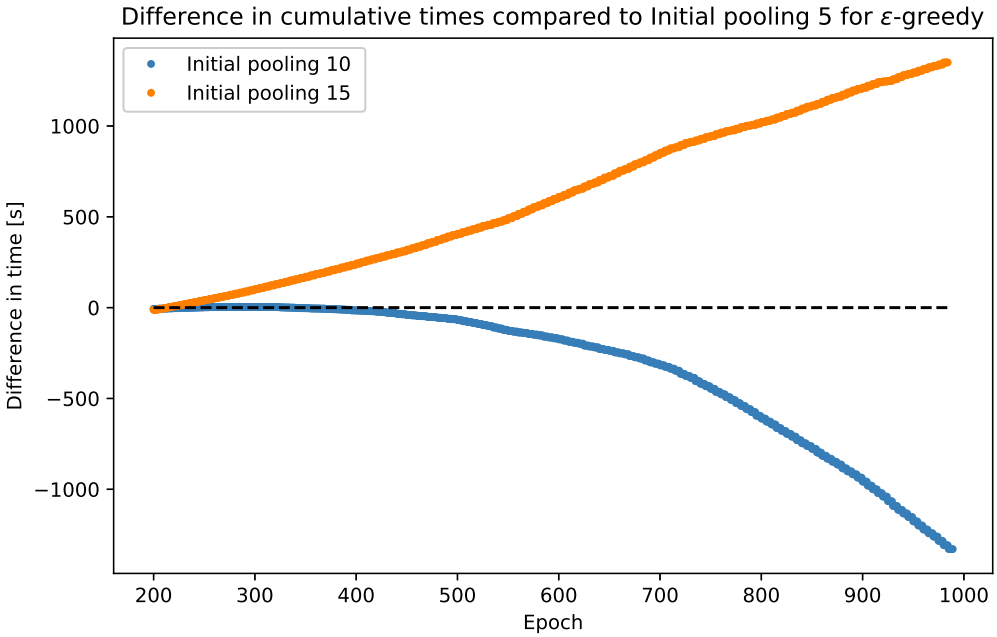

Figure B.6: Comparison parameters for *initial pooling* of 10 and 15 compared with the default value of 5 for the ε-Greedy AF. Here we see that an *initial pooling* of 10 is in fact faster than an *initial pooling* of 5.

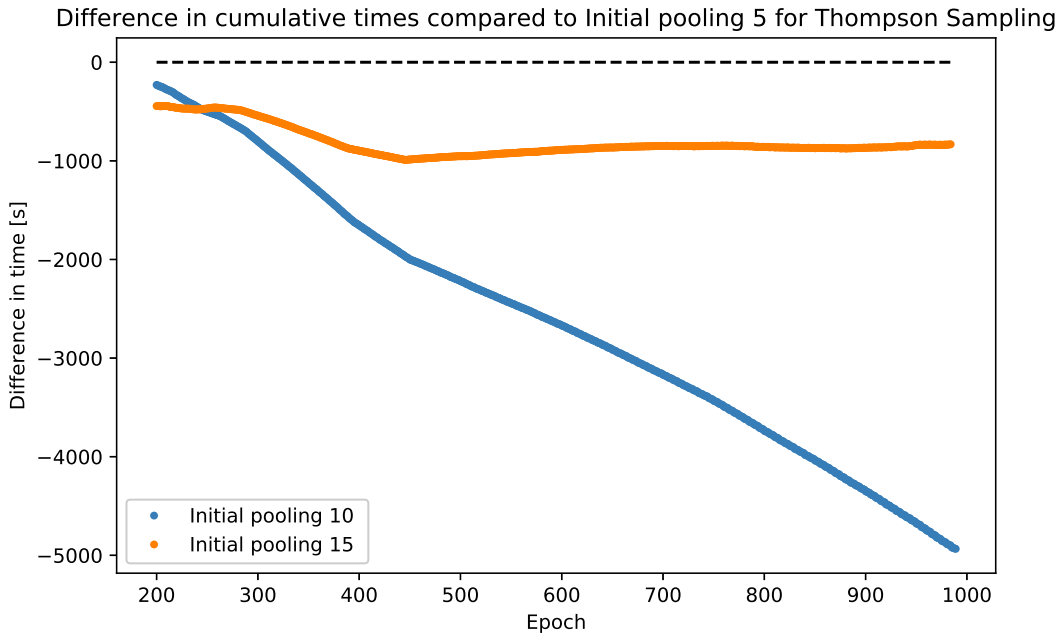

Figure B.7: Comparison parameters for *initial pooling* of 10 and 15 compared with the default value of 5 for the TS AF. Here both the *initial pooling* of 10 and 15 are faster, with the most dramatic difference in *initial pooling* 10.

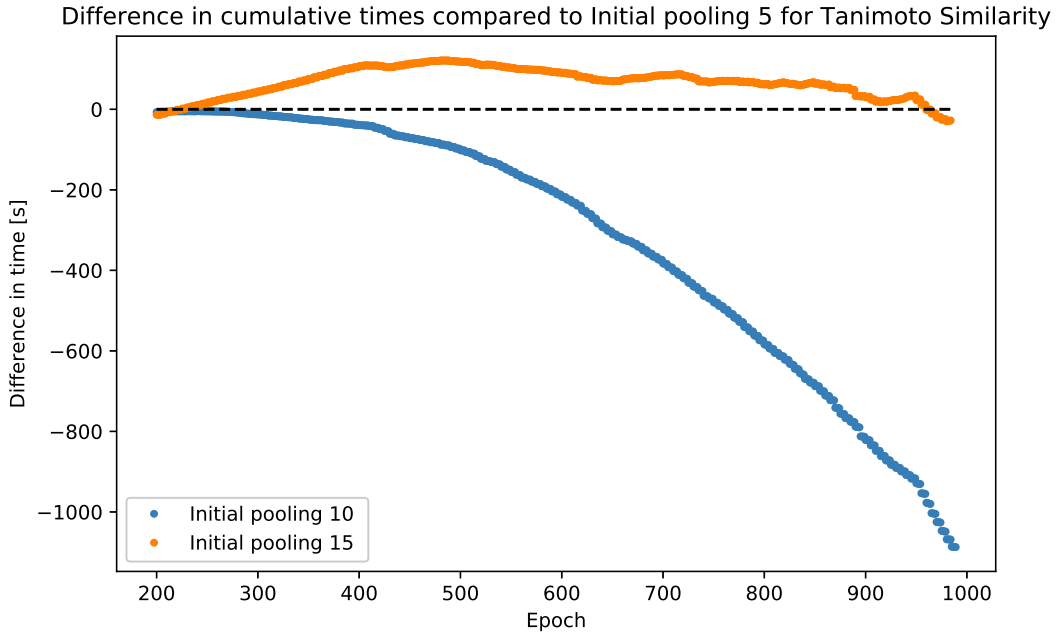

Figure B.8: Comparison parameters for *initial pooling* of 10 and 15 compared with the default value of 5 for the Tanimoto AF. Here we see that an *initial pooling* of 15 is slightly worse, but an *initial pooling* of 10 is faster.

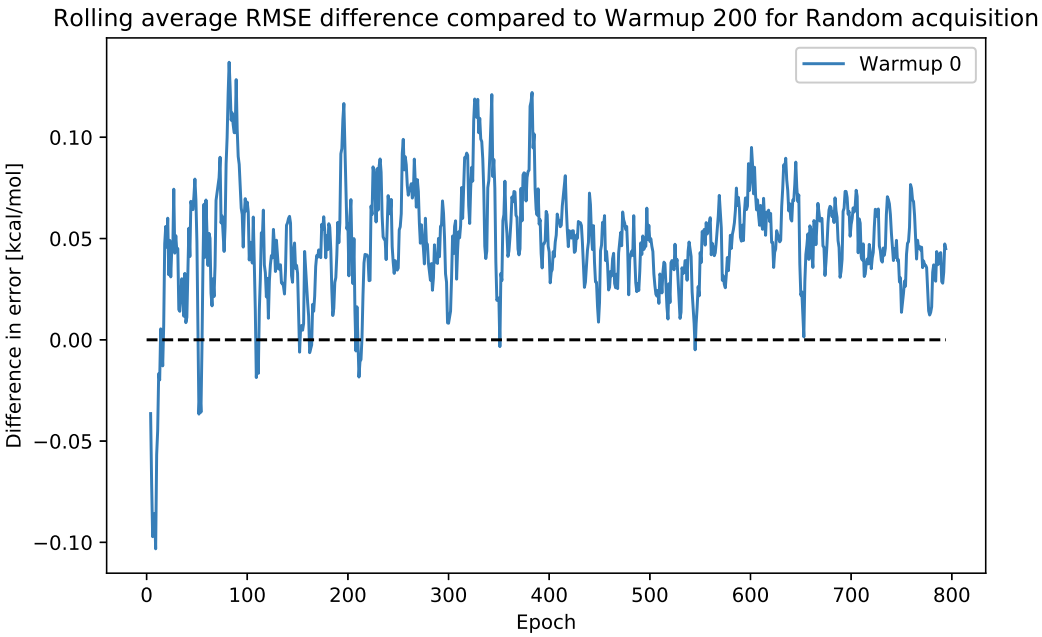

Figure B.9: Comparison parameters for *warm-up* of 0 compared with the default value of 200 for the Random AF. Here most values are positive, meaning the comparison warm-up value of 0 has a larger RMSE error than the default value of 200 for the Random AF. Meaning, the default value of 200 warm-up performs better than 0 warm-up.

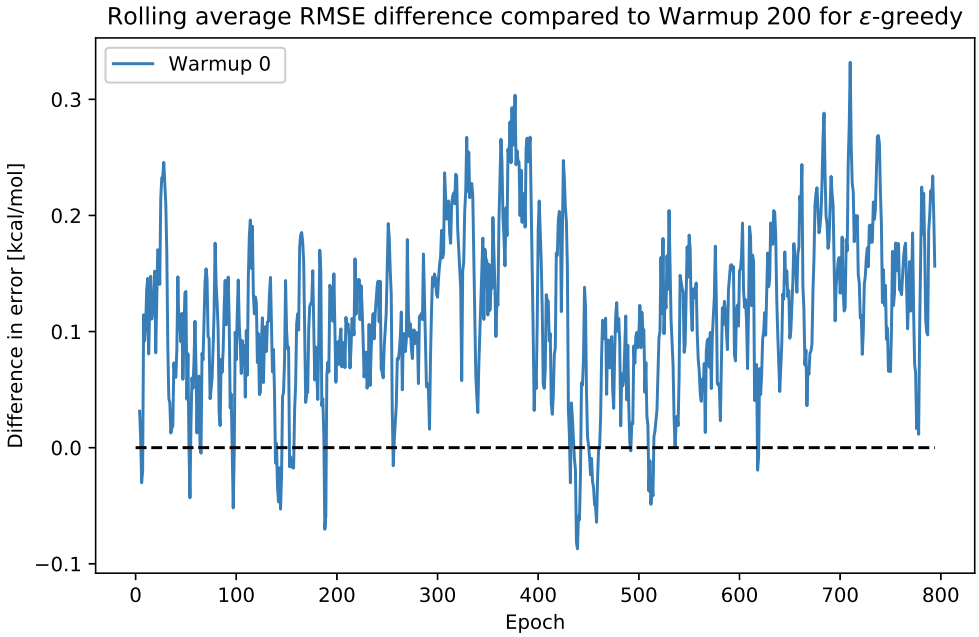

Figure B.10: Comparison parameters for *warm-up* of 0 compared with the default value of 200 for the ε-Greedy AF. Here most values are positive, meaning the comparison warm-up value of 0 has a larger RMSE error than the default value of 200 for the Random AF. In other words, the default value of 200 warm-up performs better than 0 warm-up.

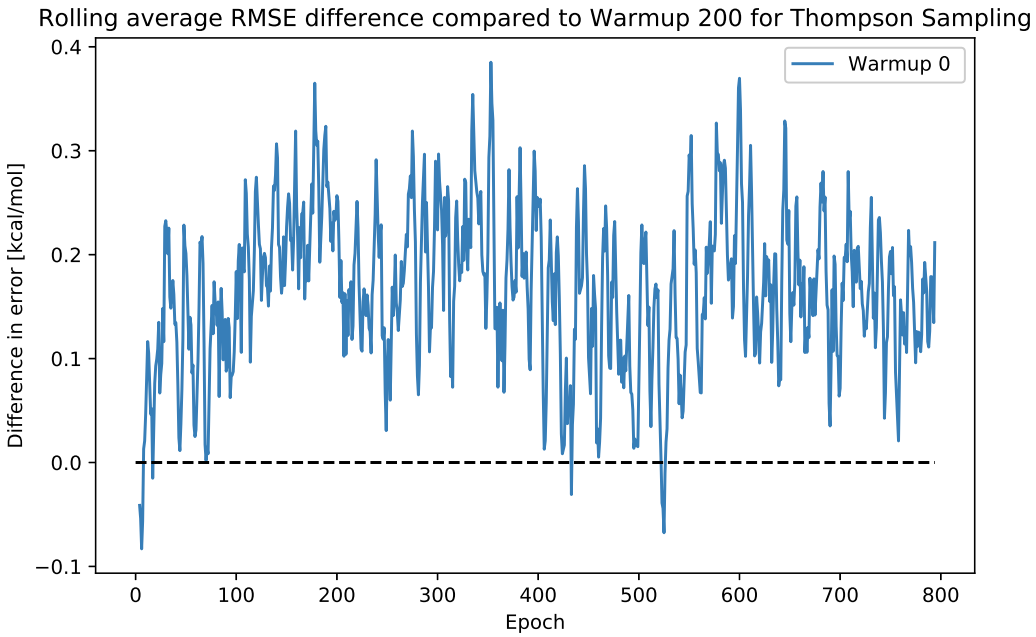

Figure B.11: Comparison parameters for *warm-up* of 0 compared with the default value of 200 for the TS AF. Here most values are positive, meaning the comparison warm-up value of 0 has a larger RMSE error than the default value of 200 for the Random AF. In other words, the default value of 200 warm-up performs better than 0 warm-up.

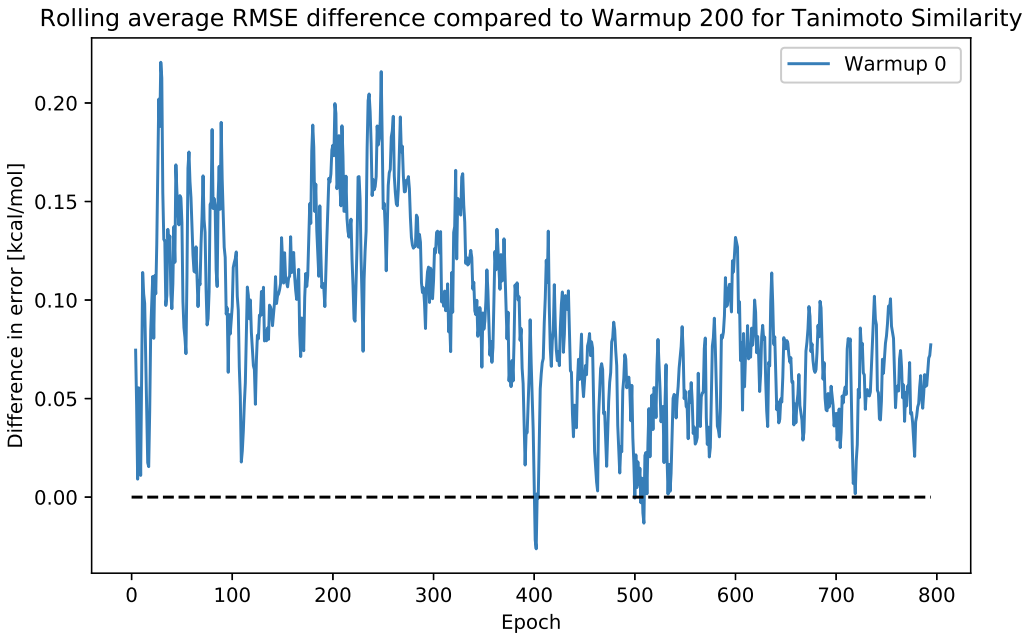

Figure B.12: Comparison parameters for *warm-up* of 0 compared with the default value of 200 for the Tanimoto AF. Here most values are positive, meaning the comparison warm-up value of 0 has a larger RMSE error than the default value of 200 for the Random AF. In other words, the default value of 200 warm-up performs better than 0 warm-up.

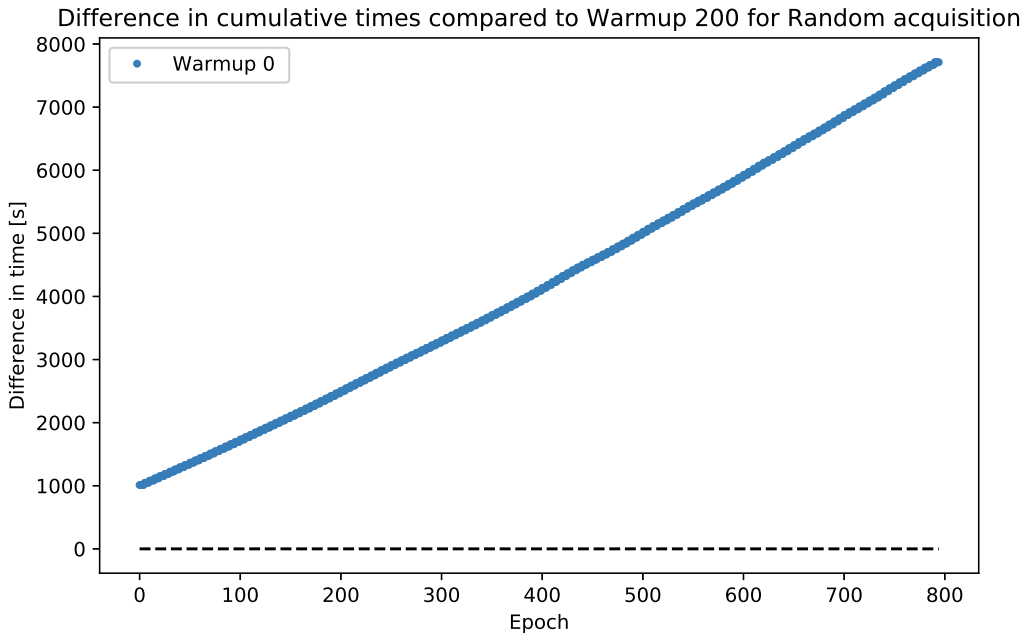

Figure B.13: Comparison parameters for *warm-up* of 0 compared with the default value of 200 for the Random AF. The comparison value of 0 *warm-up* is slower than the default *warm-up* of 200.

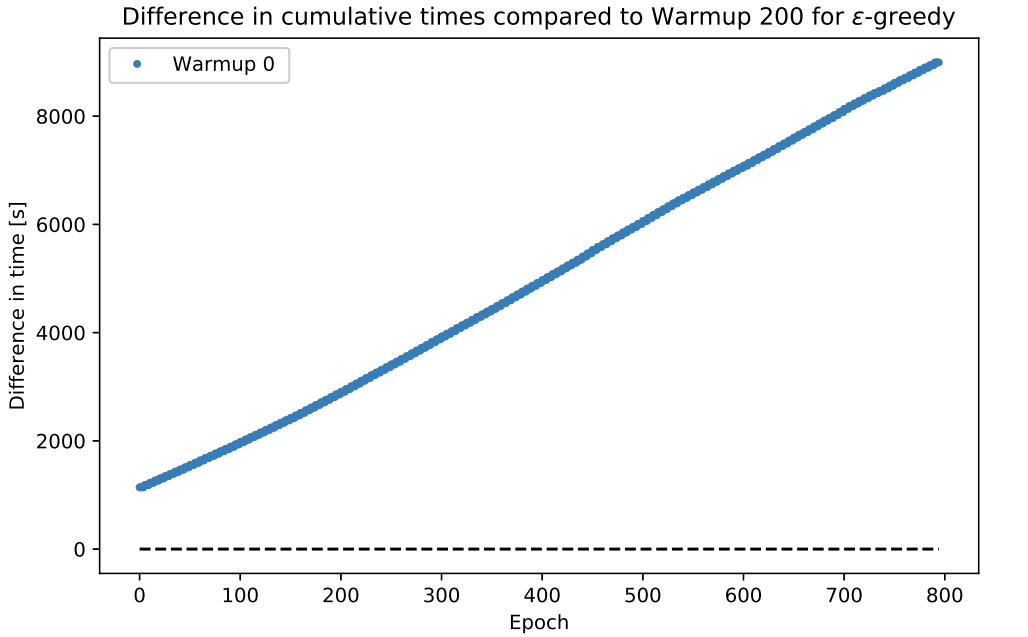

Figure B.14: Comparison parameters for *warm-up* of 0 compared with the default value of 200 for the ε-Greedy AF. The comparison value of 0 *warm-up* is slower than the default *warm-up* of 200.

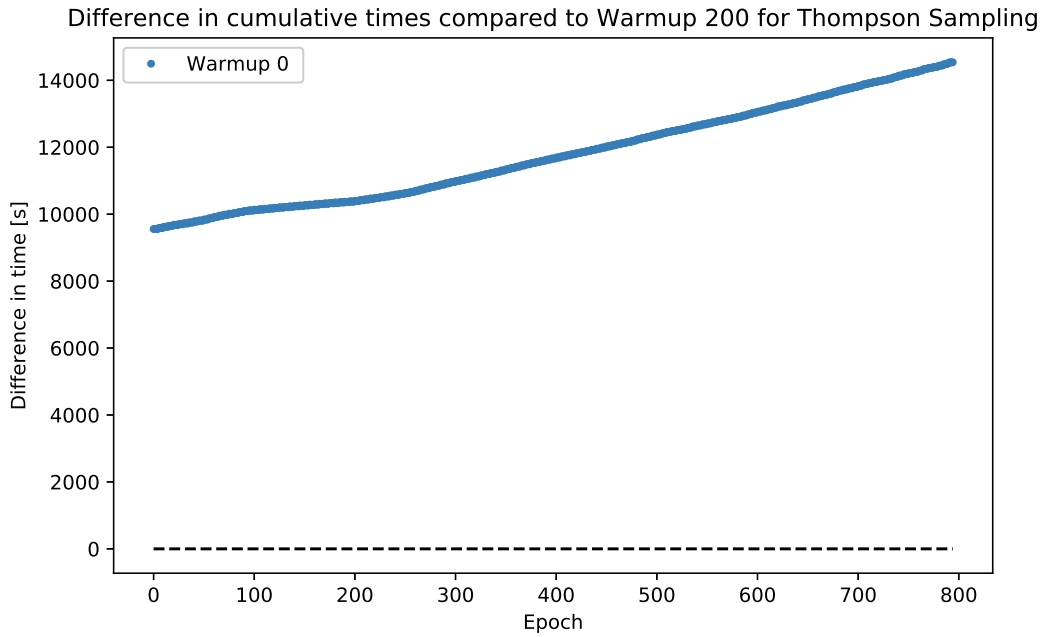

Figure B.15: Comparison parameters for *warm-up* of 0 compared with the default value of 200 for the TS AF. The comparison value of 0 *warm-up* is slower than the default *warm-up* of 200.

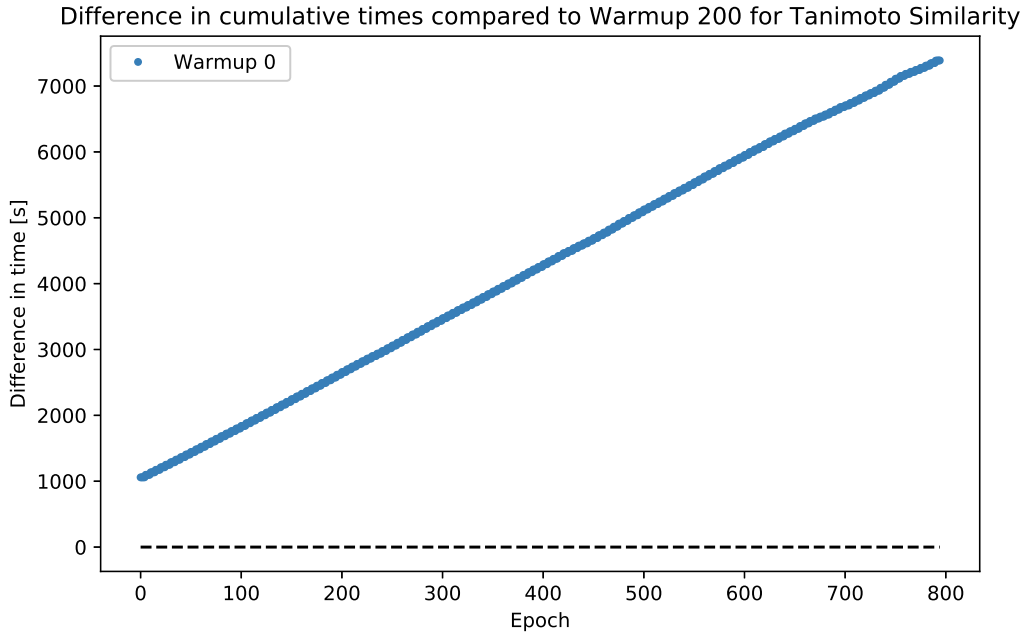

Figure B.16: Comparison parameters for *warm-up* of 0 compared with the default value of 200 for the Tanimoto AF. The comparison value of 0 *warm-up* is slower than the default *warm-up* of 200.

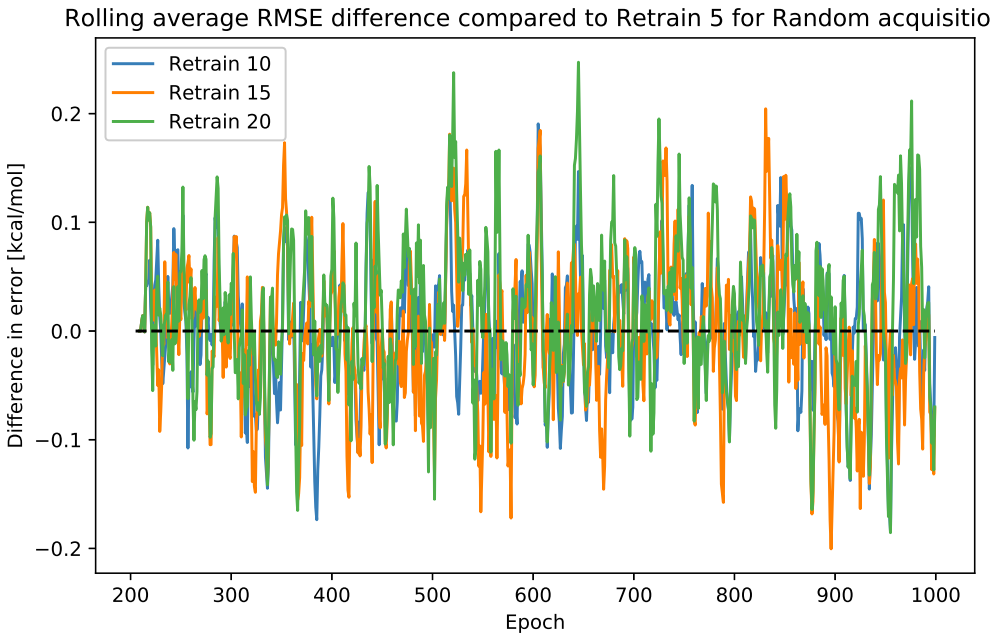

Figure B.17: Comparison parameters for *retrain* of 10, 15 and 20 compared with the default parameter of 5 for the Random AF. The difference for both 10, 15 and 20 fluctuate around 0, showing very little effect on the RMSE initially or over more epochs.

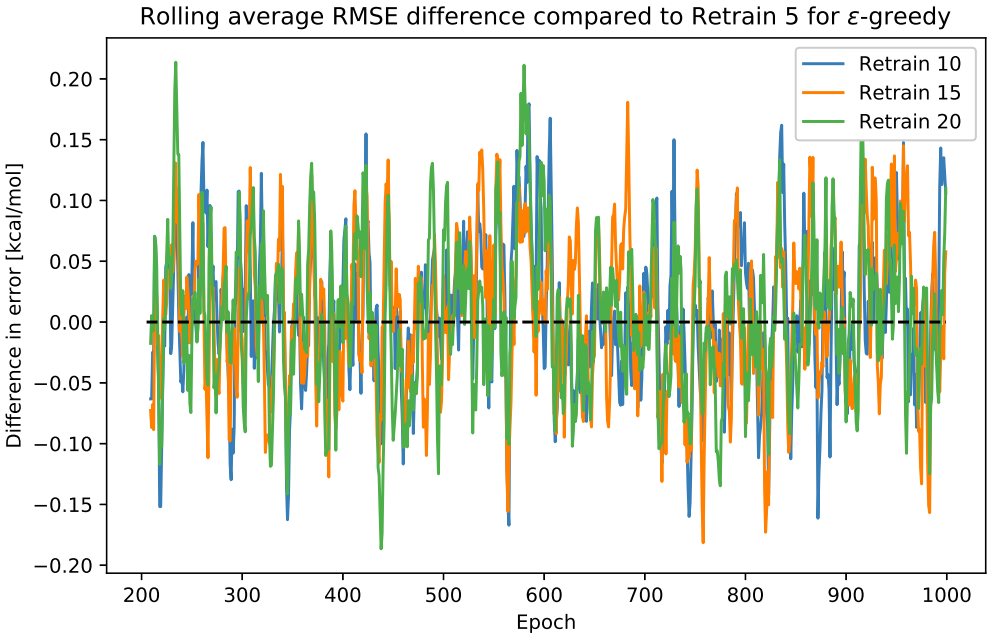

Figure B.18: Comparison parameters for *retrain* of 10, 15 and 20 compared with the default parameter of 5 for the ε-greedy AF. The difference for both 10, 15 and 20 fluctuate around 0, showing very little effect on the RMSE initially or over more epochs.

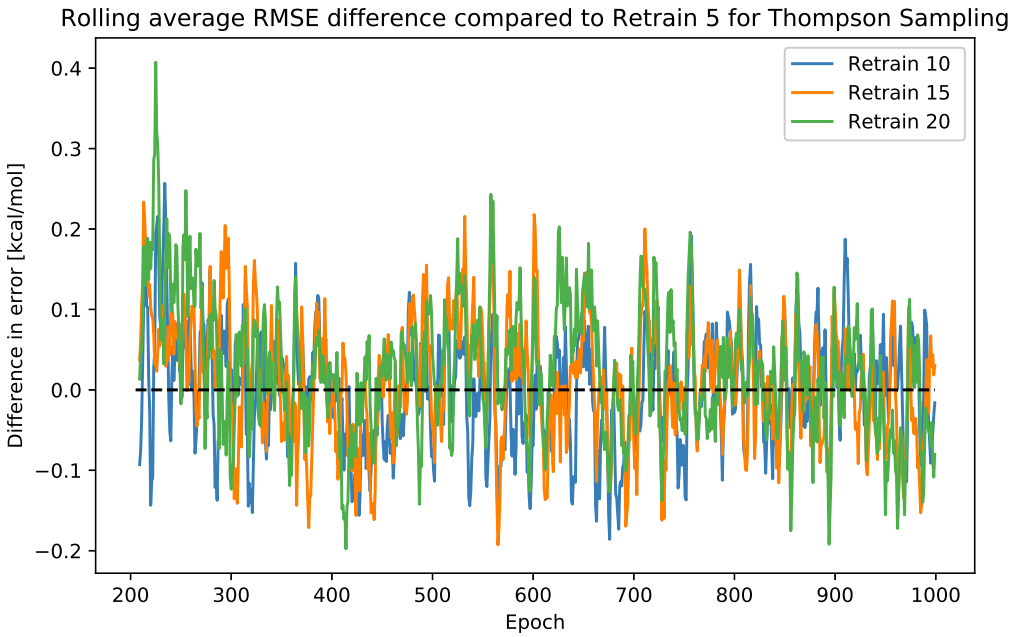

Figure B.19: Comparison parameters for *retrain* of 10, 15 and 20 compared with the default parameter of 5 for the TS AF. The difference for both 10, 15 and 20 fluctuate around 0, showing very little effect on the RMSE initially or over more epochs.

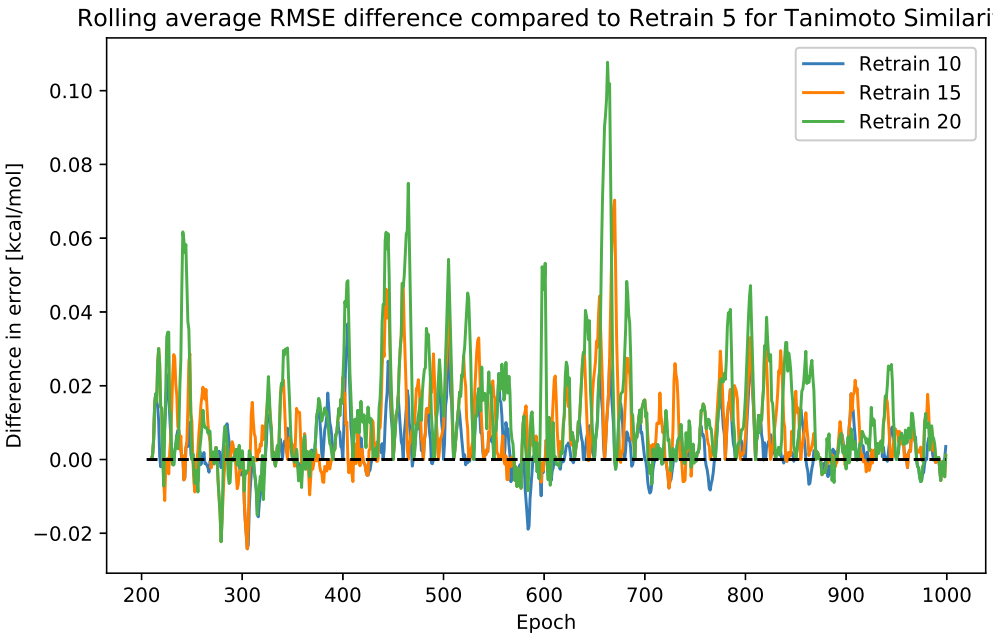

Figure B.20: Comparison parameters for *retrain* of 10, 15 and 20 compared with the default parameter of 5 for the Tanimoto AF. Here, the difference is consistently positive, meaning the default parameter of attains a lower RMSE. The difference, however, is quite small.

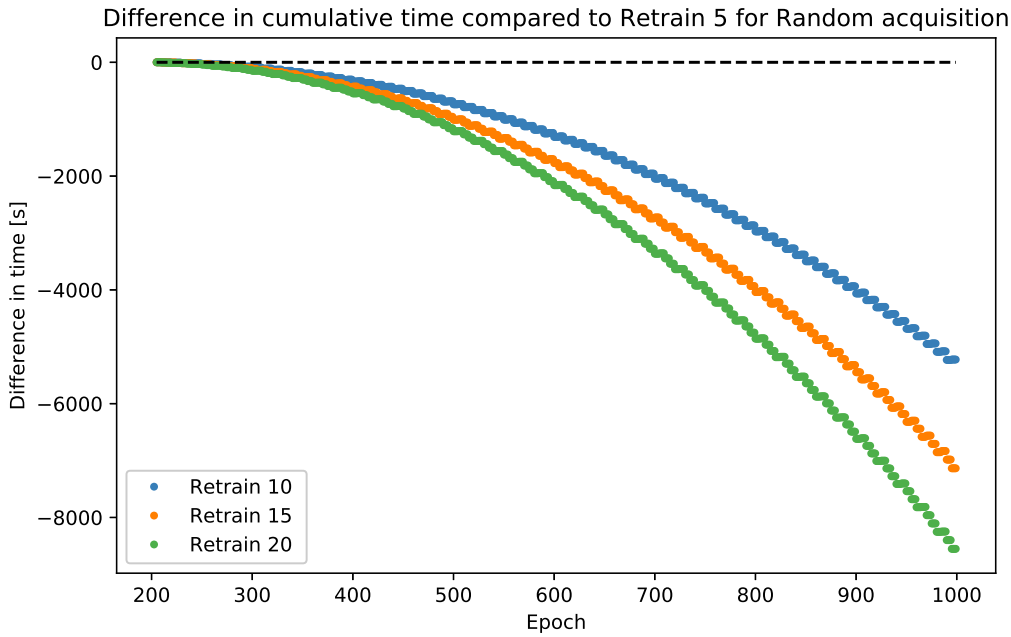

Figure B.21: Comparison parameters for *retrain* of 10, 15 and 20 compared with the default parameter of 5 for the Random AF. For a larger interval of *retrain* the the model gets faster.

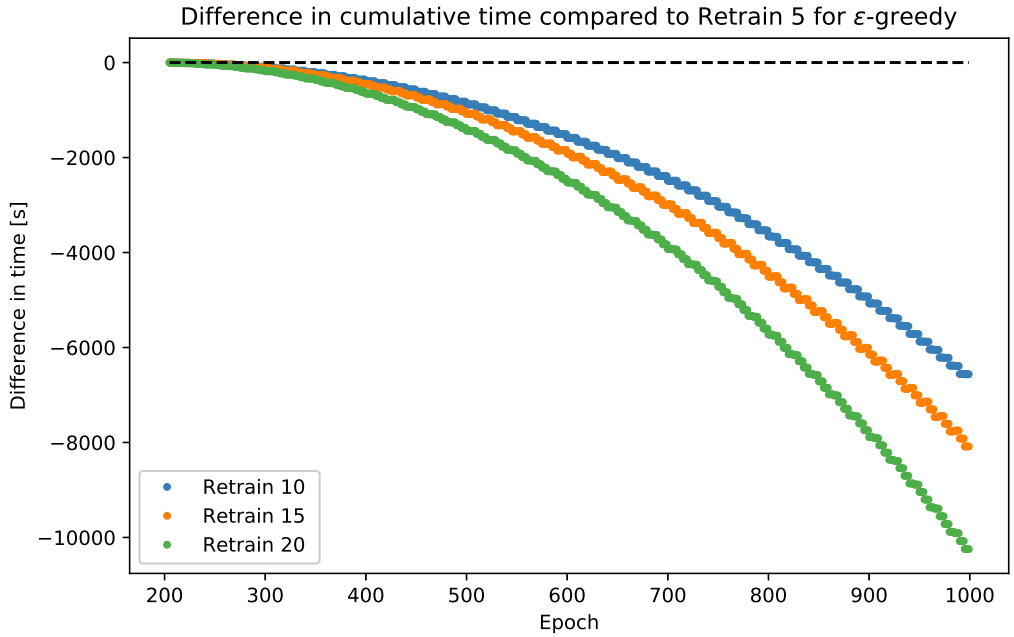

Figure B.22: Comparison parameters for *retrain* of 10, 15 and 20 compared with the default parameter of 5 for the ε-Greedy AF. For a larger interval of *retrain* the the model gets faster.

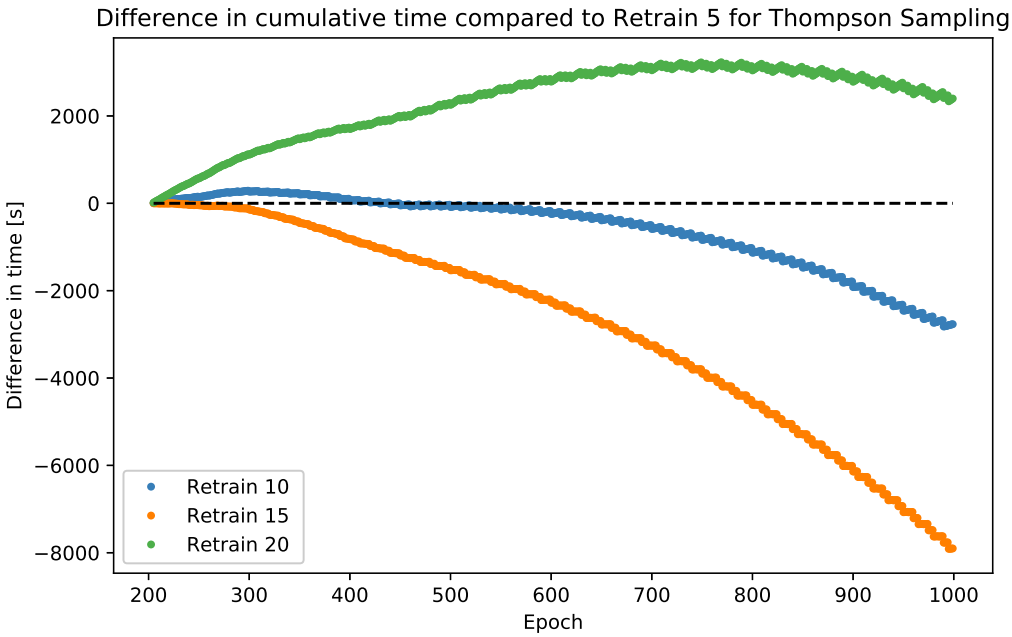

Figure B.23: Comparison parameters for *retrain* of 10, 15 and 20 compared with the default parameter of 5 for the TS AF. Here one can see that for a larger interval between *retrains*, the model actually gets slower.

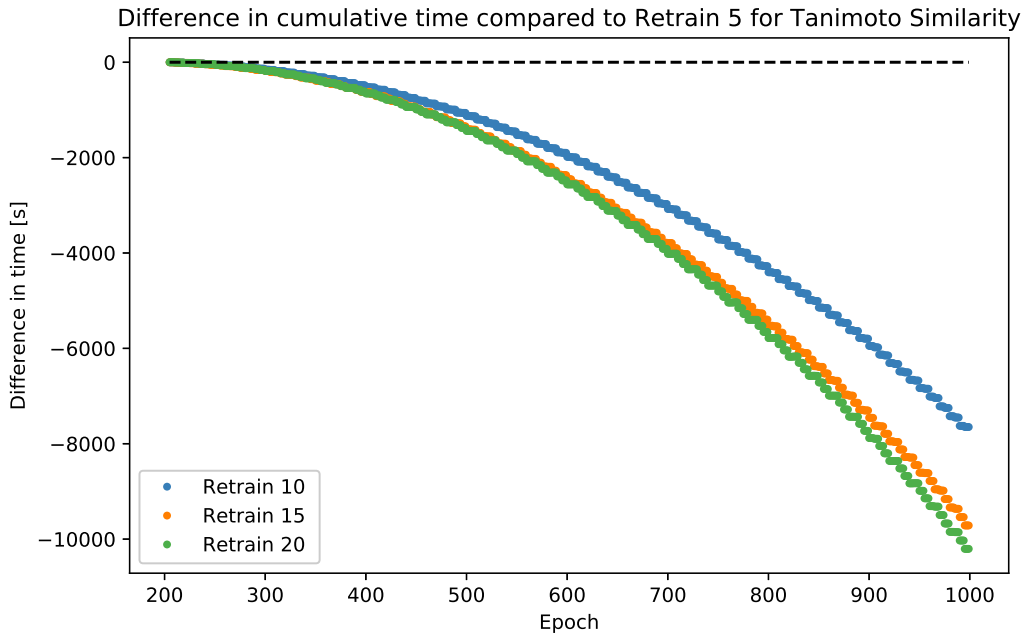

Figure B.24: Comparison parameters for *retrain* of 10, 15 and 20 compared with the default parameter of 5 for the Tanimoto AF. For a larger interval of *retrain* the the model gets faster, there is however very little difference between *retrain* 15 and 20.

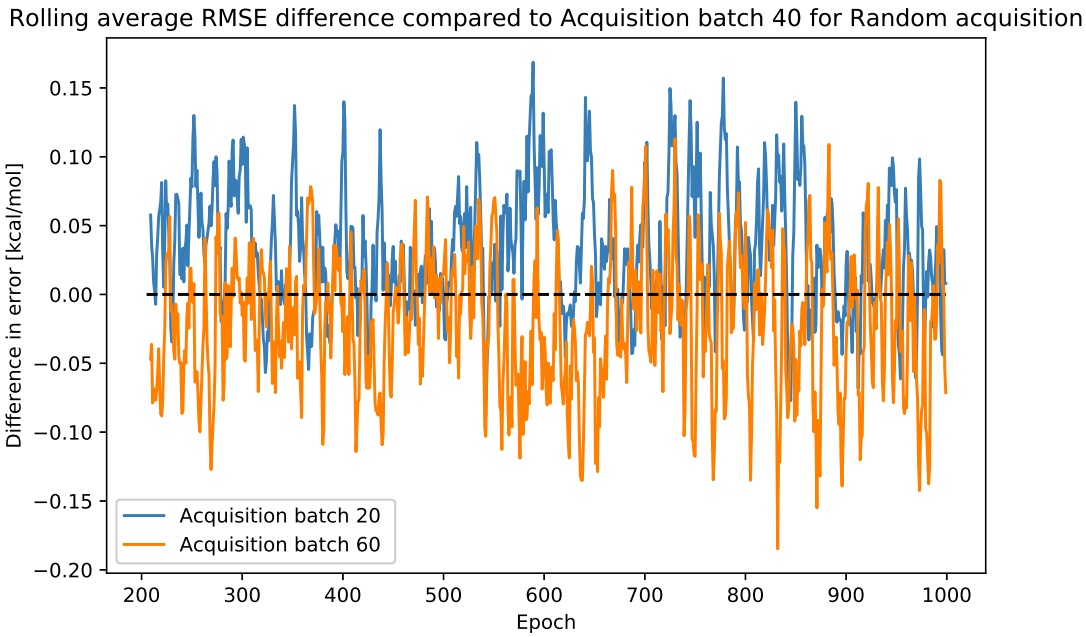

Figure B.25: Comparison parameters for *Acquisition Batch Size* of 20 and 60 compared with the default parameter of 40 for the Random AF. Both an AB of 20 and 60 show little difference, but the 60 AB does perform slightly better.

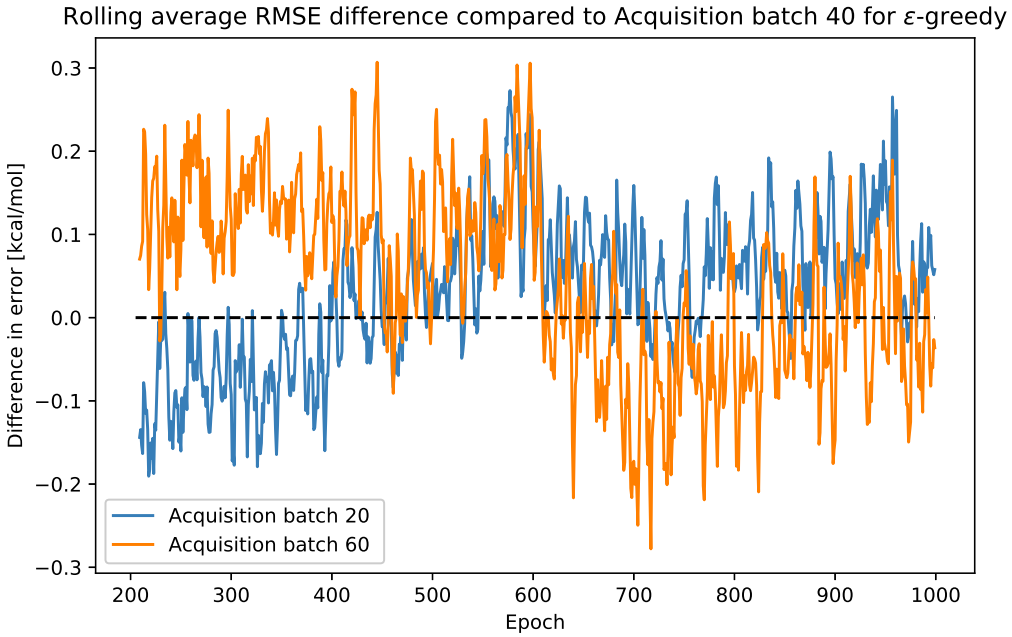

Figure B.26: Comparison parameters for *Acquisition Batch Size* of 20 and 60 compared with the default parameter of 40 for the ε-Greedy AF. The 60 AB is initially worse than both the default AB, 40, and the 20 AB, but the difference planes out by later epochs. Inversely, the 20 AB has a lower RMSE initially but grows over later epochs.

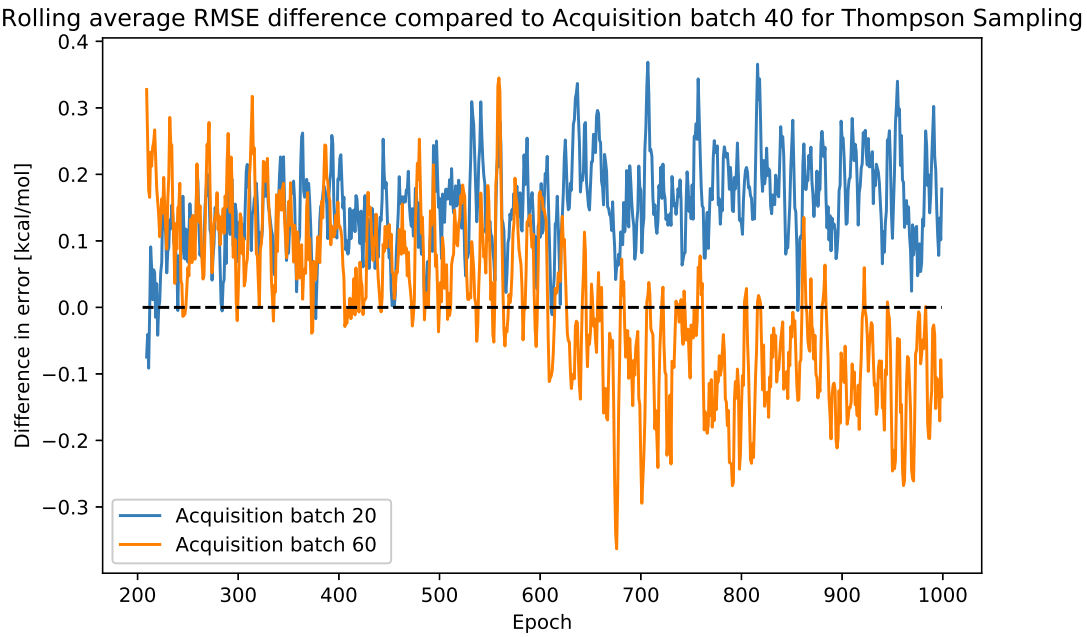

Figure B.27: Comparison parameters for *Acquisition Batch Size* of 20 and 60 compared with the default parameter of 40 for the TS AF. The 20 AB is consistently worse than the default 40 AB. The 60 AB is initially worse but performs better after about 600 epochs. The average difference is however close to 0.

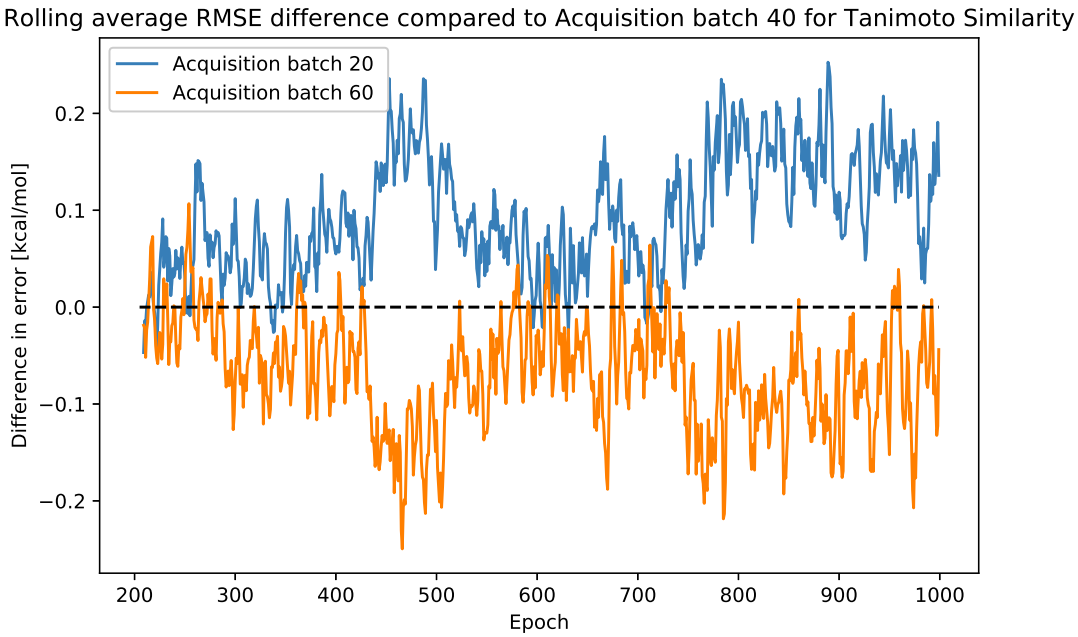

Figure B.28: Comparison parameters for *Acquisition Batch Size* of 20 and 60 compared with the default parameter of 40 for the Tanimoto AF. The 20 AB is consistently worse than the default 40 AB while the 60 AB is consistently better, however the magnitude of the difference is small.

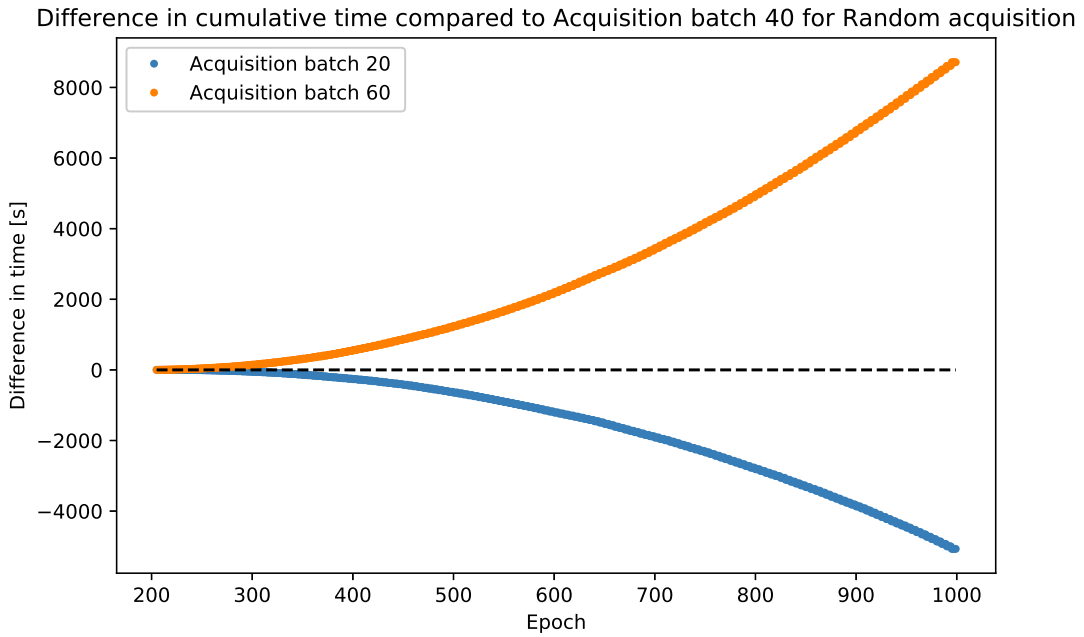

Figure B.29: Comparison parameters for *Acquisition Batch Size* of 20 and 60 compared with the default parameter of 40 for the Random AF. A larger acquisition batch seems to correlate with a slower run.

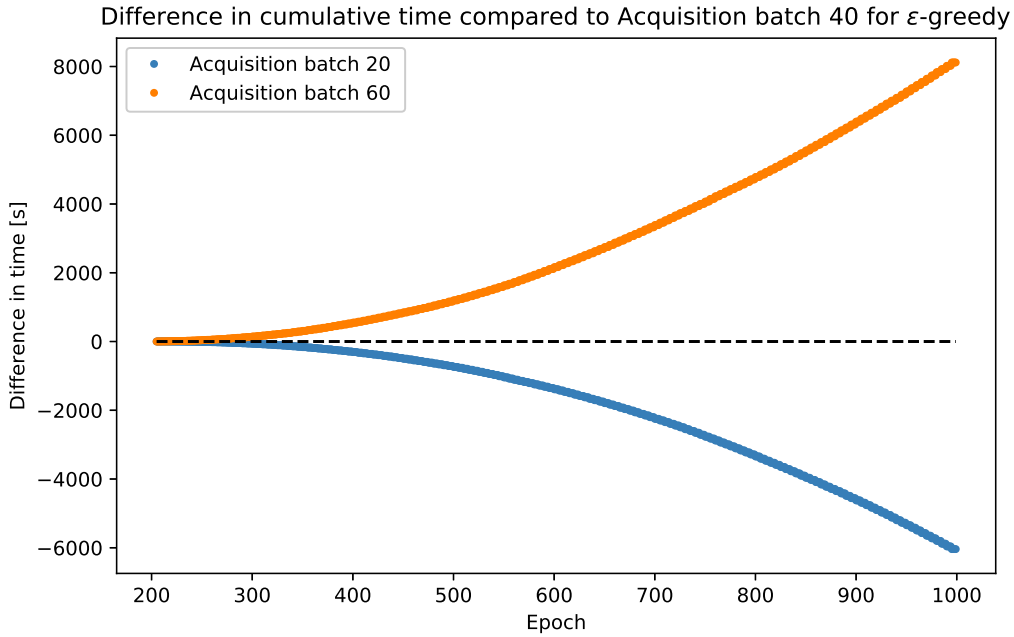

Figure B.30: Comparison parameters for *Acquisition Batch Size* of 20 and 60 compared with the default parameter of 40 for the ε-Greedy AF. A larger acquisition batch seems to correlate with a slower run.

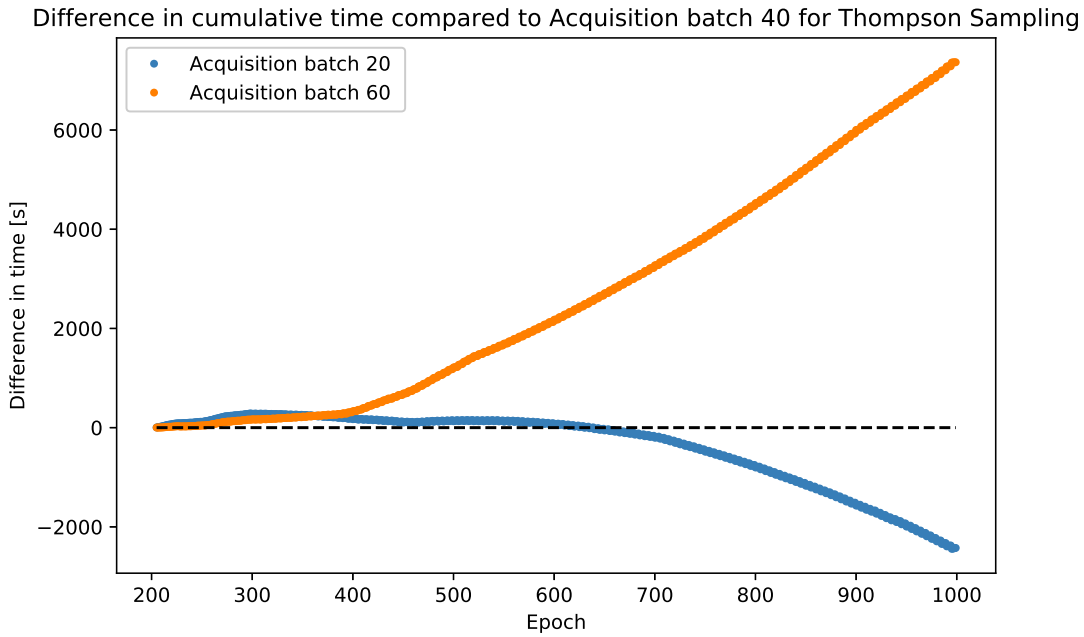

Figure B.31: Comparison parameters for *Acquisition Batch Size* of 20 and 60 compared with the default parameter of 40 for the TS AF. The 60 AB is slower but the AB of 20 is almost equal to the default AB until epoch 600.

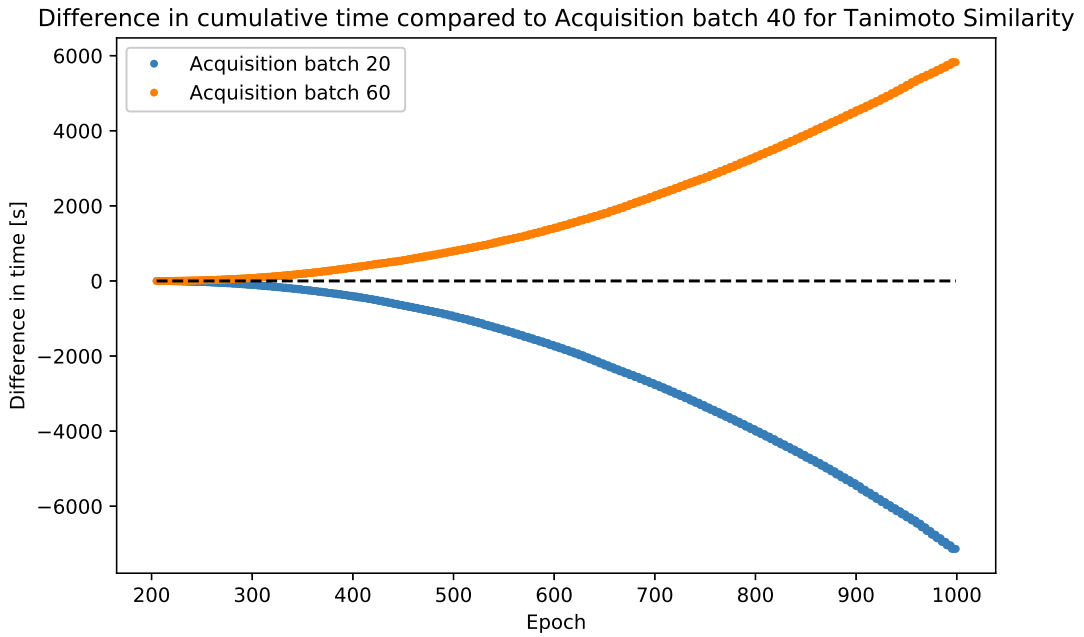

Figure B.32: Comparison parameters for *Acquisition Batch Size* of 20 and 60 compared with the default parameter of 40 for the Tanimoto AF. A larger acquisition batch seems to correlate with a slower run.