Evaluating the Effectiveness of Metamodeling in Emulating Quantitative Models

Michael Rausch[1] and William H. Sanders[2]

[1] University of Illinois at Urbana-Champaign, Urbana IL, USA
mjrausc2@illinois.edu
[2] Carnegie Mellon University, Pittsburgh PA, USA
sanders@cmu.edu

Abstract. It is often prohibitively time-consuming to do sensitivity analysis, uncertainty quantification, and optimization with complex state-based quantitative models because each model execution or solution takes so long to complete, and many such executions are necessary to complete the analysis. One way to approach this problem is to use metamodels that emulate the behavior of the base model but run much faster. These metamodels may be automatically constructed using machine learning techniques, and then the relevant analysis may be conducted on the fast-running metamodel in place of the slow-running model. In this work, we evaluate the effectiveness of several different types of metamodels in emulating seven publicly available PRISM and Möbius models. In our evaluation, we found that the metamodels are reasonably accurate and are several thousand times faster than the corresponding models they emulate. Furthermore, we find that stacking-based metamodels are significantly more accurate than state-of-the-practice metamodels. We show that metamodeling is a powerful and practical tool for modelers interested in understanding the behavior of their models, because it makes feasible analysis techniques that would otherwise take too long to run on the original models.

Keywords: metamodels · surrogate models · emulators · security models · reliability models · sensitivity analysis · uncertainty quantification · optimization

1 Introduction

Many realistic state-based quantitative models have a large number of input variables. The precise values of the input variables are often unknown in practice. Alternatively, some of the input variable values may be under the direct control of the modeler, and the modeler must choose values that will maximize the system's performance or utility. Sensitivity analysis (SA) and uncertainty quantification (UQ) can help a modeler understand and manage the resulting uncertainties, while optimization can help a modeler determine which combination of input variable values may be best to achieve a particular goal. SA,

UQ, and optimization techniques are all of vital importance to modelers. However, applying SA, UQ, or optimization techniques directly to the model can be impractical for models with long execution times, since the techniques usually mandate that the model be solved for many different combinations of values for the input parameters.

Alternatively, a modeler can perform SA, UQ, or optimization indirectly through the use of a metamodel. A metamodel is a model of a model. The metamodel (usually imperfectly) emulates the behavior of the model (which we shall refer to as the *base model* in this paper). The metamodel generally runs significantly faster than the base model, trading a great gain in speed for slightly less accuracy. It is unnecessary to construct metamodels manually; metamodels can be constructed automatically using machine learning techniques. Once a metamodel has been built, SA, UQ, or optimization techniques can be applied to it as a "stand-in" for the base model and completed in a reasonable amount of time. Using metamodels, modelers can indirectly perform SA, UQ, and optimization on models that run so slowly that it would be impractical to evaluate them directly.

In recent work, a stacking-based metamodel architecture was applied to the analysis of a botnet simulation model [9] and an Advanced Metering Infrastructure (AMI) simulation model [10]. The resulting metamodels ran hundreds to thousands of times faster than the original base models and were more accurate than traditional metamodels in emulating their behavior. While the results were impressive, it was unknown whether the approach would generalize and produce similarly good results for other models. If the metamodeling techniques were shown to work on a wider variety of test cases, modelers would gain greater confidence in the efficacy of metamodels for quantitative modeling.

In this work, we use seven previously published models (six PRISM models and one Möbius model) as test cases to evaluate the speed and accuracy of metamodels built via machine learning. We evaluate state-of-the-practice metamodel architectures, the recently proposed stacking-based metamodel architecture, and several variants. We show that the metamodels we built run thousands of times faster than the corresponding base models, and all are reasonably accurate. In addition, based on the results of our experiments, we give advice on which metamodel architectures to use for those who wish to apply the techniques to their own models. Our work gives confidence that metamodels can be a useful tool for performing SA, UQ, and optimization on models with many uncertain input parameters that would normally take too long to execute.

The rest of this paper is organized as follows. In Section 2, we describe the seven models we use as test cases. In Section 3, we briefly explain the metamodeling approach in general, the current state-of-the-practice, the recently developed stacking-based approach, and variants to the stacking-based approach. In Section 4, we show the speed and accuracy of the metamodels on the test case models. We discuss key insights, use cases, and limitations in Section 5. We briefly review related work in Section 6, and conclude in Section 7. The appendix contains details of the initial conditions of our test case models.

2 Test Cases

We evaluate seven different test cases. We used the botnet model that was previously used as a test case in a metamodeling investigation [9], plus six published PRISM [4] models that, to the best of our knowledge, have never been used as subjects of metamodeling. We examined the publicly available PRISM case studies[1]. Since we are especially interested in the use of metamodeling for sensitivity analysis, uncertainty quantification, and optimization, which are commonly used when a model has many uncertain input variables, we selected case studies that had many model input variables. We also tried to select models that were non-trivial. We chose to use published models, rather than create synthetic models, to help evaluate the real-world effectiveness of the metamodeling techniques. We are also pleased that these models cover a variety of domains: cybersecurity, reliability, chemistry, and biology. We shall briefly describe each test case in turn.

- **Botnet** The Botnet model is a Möbius model that is described in detail in [12] and was used as the sole test case in [9]. The model can be used to study the growth of a botnet over the course of a week given certain conditions, such as the probability that an uninfected computer will become infected, and the rate of removal of bots from the net. The values of eleven input variables are uncertain in this model. The input variables, and the values that they assume in our evaluation, can be found in the Appendix in Table 7.
- **Circadian** The Circadian Clock model[2], based on the abstract model found in [1] and [15], is a CTMC PRISM model. Given rates for transcription, translation, binding, release, and degradation, for activator and repressor genes and mRNA the model calculates the amount of activator protein at a given time. There are 17 input variables in this model. The input variables, and the range of values that they assume in our evaluation, can be found in the Appendix in Table 8.
- **Cluster** The Workstation Cluster model[3] is a PRISM model taken from [3]. It can be used to calculate the quality of service (QoS) of a workstation cluster arranged in a star topology. The components of the cluster fail and are repaired at specific rates. The values of thirteen input variables are uncertain in this model. The majority of these uncertain input variables are various failure and repair rates. The input variables, and the range of values that they assume in our evaluation, can be found in the Appendix in Table 9.
- **Cyclin** The Cyclin model[4] is a CTMC PRISM model based on a formal specification from [5]. It models cell cycle control in eukaryotes, given a specific quantity of various molecules and various base reaction rates. This model contains 14 input variables. The input variables, and the range of

[1] Available here: https://www.prismmodelchecker.org/casestudies/
[2] Model code here: https://www.prismmodelchecker.org/tutorial/circadian.php
[3] Model code here: https://www.prismmodelchecker.org/casestudies/cluster.php
[4] Model code here: https://www.prismmodelchecker.org/casestudies/cyclin.php

values that they assume in our evaluation, can be found in the Appendix in Table 10.

- **Embedded** The Embedded System model[5] is a CTMC PRISM model of an embedded control system based on a model description from [7]. The embedded system consists of three sensors, a sensor input processor, a main processor, an output processor, two actuators, and a bus that connects the processors. The components have a probability of failing, either permanently or in a transient manner. Some failures can be repaired by a system reboot. This model can be used to calculate reliability and availability metrics. There are six input variables in this model. The input variables, and the range of values that they assume in our evaluation, can be found in the Appendix in Table 11.
- **Kanban** The Kanban Manufacturing System model[6] is a CTMC PRISM model based on a model found in [2]. The model can be used to estimate the throughput of a manufacturing system. There are fourteen input variables in this model. The input variables, and the range of values that they assume in our evaluation, can be found in the Appendix in Table 12.
- **Molecules** The Simple Molecular Reactions model[7] is a CTMC PRISM model. Given a particular amount of Na, Cl, and K, various reaction rates, and length of time, it can calculate the expected percentage of Na/K molecules. There are nine input variables in this model. The input variables, and the range of values that they assume in our evaluation, can be found in the Appendix in Table 13.

3 Approach

A *metamodel* or *emulator* or model surrogate is a model of a model. Recall that we refer to the model that the metamodel emulates as the *base model*. The metamodel will attempt to produce the same output as the base model, given the same vector of input variable values. In general, a metamodel will not emulate the base model perfectly, but will run much faster than the base model. It is possible to construct metamodels using machine learning (ML) by following this general process:

1. Collect training and testing data by generating a number of different inputs, running the base model with those inputs, and recording each input vector and the corresponding modeling result.
2. Train a machine learning regressor to emulate the base model using the training data.
3. Test the trained regressor with the testing data. This is done by running each test input vector through the metamodel and observing the difference between the metamodel output and the previously-recorded base model output.

[5] Model code here: https://www.prismmodelchecker.org/casestudies/embedded.php
[6] Model code here: https://www.prismmodelchecker.org/casestudies/kanban.php
[7] Model code here: https://www.prismmodelchecker.org/casestudies/molecules.php

In this work, we generated all training and testing data inputs uniformly at random for ease of analysis and coding, though it is possible to use more sophisticated sampling methods (such as Latin Hypercube Sampling or Sobol Sequence Sampling) to collect the training data, which may produce slightly more accurate metamodels [9].

The choice of the kind of regressor to train is important. For example, someone may create a metamodel by training a random forest, multilayer perceptron, Gaussian Process regressor, or something more exotic. Some types of regressors may be substantially more accurate than others, and it is difficult to know *a priori* which will perform well. In much of the related work, the author chose regressors based on intuition, familiarity and comfort with a particular technique, or certain properties of the data. In other papers, the authors evaluate several different regressor types, and chose the best-performing regressor (the *Best of Many* approach). In recent work [9] [10], a stacking-based metamodel approach was shown to significantly outperform the Best of Many approach. However, it was only shown to work on two test cases, and no variants of the metamodel architecture were explored.

In the remainder of the section, we shall briefly review the stacking-based metamodel approach for SA and UQ introduced in [9], and then describe the variants of the architecture that we developed and evaluate in this paper.

3.1 Stacking Review and Variants

In machine learning, an ensemble is a collection of learning algorithms that are used together with the goal of creating a stronger overall learner than any of the constituent learning algorithms alone. One of the simplest ensemble techniques one could imagine is to create a voting committee of heterogeneous regressors (for example, one multilayer perceptron, one random forest, one KNN regressor, etc.), each trained separately on the training data. The output of the committee would be the average of the regressor predictions. However, better performance could be achieved if the predictions of more accurate regressors were given more weight in the vote than inaccurate regressors. In particular, it may be that one regressor may perform relatively well in one region of the input space, and relatively poorly in another region of the input space. The appropriate weighting is not obvious, but could be learned using a separate regressor or committee of regressors. This is the core idea of stacking. With stacking, the predictions of the regressors in the first committee are appended to the training data, and then a second committee of regressors is trained with the augmented training datasets [16]. The stacking approach was successfully used to win Kaggle ML competitions [11].

To build a stacked metamodel, one would first train multiple regressors with the training dataset, to form the first level committee. Next, every regressor in the first level committee would be run with every input vector from the training dataset to obtain the corresponding regressor prediction for that input vector, and then the regressor prediction would be appended to the input vector as another feature. At the end of the process, the modeler will have an augmented

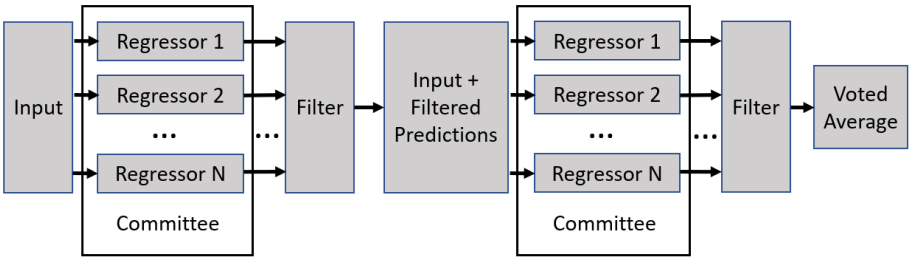

Fig. 1. Overview of the metamodel architecture.

training dataset that contains all of the original input vectors, but each input vector will include additional features: the predictions of the first level regressors. The augmented dataset would then be used train another set of regressors to form the second level committee. The regressors in the second level will have the benefit of additional information in their training dataset to help them make better predictions: the "recommendations" of the first level regressors.

See Figure 1 for a diagram of the flow of execution in the metamodel architecture. Once the metamodel has been built, it may be executed. To execute the metamodel, an input vector is given to each regressor in the first level committee, and each regressor calculates its prediction. We have experimentally found that it is useful to filter predictions from inaccurate regressors (later in this section we discuss details of the filter), but predictions from accurate regressors are appended to the input vector. This augmented input vector is then given to each of the regressors in the second level committee, which in turn make their predictions. The predictions are filtered again, and the predictions that make it through the filter are averaged to calculate the final metamodel prediction.

In general, one may choose any set of regressors as members of the two committees. However, one must of course choose a specific set of regressors. This is an important choice, since the overall accuracy of the metamodel heavily depends on which regressors are members of the two committees. Only one kind of committee was evaluated in [9] and [10]. We go further than those works by evaluating five different committees with a variety of properties.

Similarly, appropriate filters can help improve the performance of the metamodel. In general, one may use the predictions of each regressor in the committee, regardless of the accuracy of the regressor. However, we have experimentally found that it can be useful to filter predictions from inaccurate regressors. Only one kind of filter was evaluated in [9] and [10], however, filtering can be accomplished in a variety of ways. We build upon previous work by evaluating variants on two different types of filters.

We shall describe the different committee and filter types we evaluate in this work.

Committees We evaluate five different committees in this work. We hypothesize that larger committees would outperform smaller committees, and committees with more heterogeneity would outperform those with less, and wish to test the hypothesis. To that end, we evaluate committees of different sizes, and with varying degrees of heterogeneity in membership. We evaluate a large committee with many heterogeneous regressors (Committee 0), a couple of small committees with relatively homogeneous members (Committees 1 and 2, which only contain different kinds of random forests and multilayer perceptrons, respectively, as members), a medium-sized committee with heterogeneous members (Committee 3), and a small committee with heterogeneous members (Committee 4). What follows is a description of each committee. A summary of the properties of the committees can be found in Table 1.

0. Committee 0: The regressors in this committee are the same as the regressors used in the committees in [9]. It is also the largest committee we evaluate, with 25 members. The members are 1 random forest regressor, 8 different multilayer perceptrons (each with a different combination of solvers and activation functions), 4 different Gradient Boosting Regressors (each with a different loss function), a RidgeCV regressor, 10 different KNN regressors (each vary by the number of neighbors and whether the neighbor votes are weighted uniformly or by distance), a Gaussian Process regressor, and a stochastic gradient descent regressor. Hyperparameter tuning is less of an issue in this large committee compared to the smaller committees, because one can simply include many variants of the same "base" algorithm with different hyperparameters without being particularly concerned with finding the "best" set of hyperparameters. In effect, the stacking algorithm will indirectly perform the work of finding the "best" set of hyperparameters in the given set for the modeler in the course of its normal operation.
1. Committee 1: All of the regressors in this committee are variants of the random forest architecture with different hyperparameters. There are 4 regressors in this committee, and each varies by the number of trees in the forest (either 10 or 100) and by the criterion used (either *mean squared error* or *mean absolute error*).
2. Committee 2: All of the regressors in this committee are variants of the multilayer perceptron architecture with different hyperparameters. There are 6 regressors in this committee, and each has a unique combination of solver (either *adam*, *sgd*, or *lbfgs*) and activation function (either *logistic* or *tanh*).
3. Committee 3: This committee is our representative example of a committee with a moderate number of regressors. There are six regressors in this committee: one random forest, one multilayer perceptron, one support vector machine, one Gaussian process regressor, one KNN regressor, one Gradient Boosting regressor, and one stochastic gradient descent regressor.
4. Committee 4: This committee is our representative example of a small committee of just three regressors, each having a different architecture. This committee contains a random forest, a multilayer perceptron, and an Gaussian process regressor.

Table 1. Properties of committees.

Committee Index	Regressor Mix (Heterogeneous or Homogeneous)	Size of Committee (# members)
0	Heterogeneous	Largest (25)
1	Homogeneous	Small (4)
2	Homogeneous	Moderate (6)
3	Heterogeneous	Moderate (6)
4	Heterogeneous	Small (3)

Filters We evaluate two different types of filters in this work: *Top k* and *Within n Percent*. The *Top k* filter only allows the predictions from the k most accurate regressors to pass through. In this work we consider $k = \{1, 2, 3\}$. The *Within n Percent* filter will allow predictions from all regressors through as long as the regressor's error is no worse than n percent worse than the best performing regressor. In this work we consider $n = \{10\%, 25\%, 50\%, 100\%\}$. With this filter, it is not known *a priori* how many regressors will be filtered - theoretically, all of the regressor predictions could be filtered (except the predictions from the best regressor), or all of them could pass through the filter. As a control, we also construct metamodels with no filters. The *Within 25 Percent* filter was used in the architecture of the metamodel found in [9].

4 Evaluation of Accuracy and Speed

In this evaluation, we first determine the most effective combination of committee and filter to use for the metamodel, and rank the committees and filter types by their effectiveness. Then, we calculate the accuracy of the metamodel on each of the test cases. We also investigate by how much the accuracy of the metamodels increase with more training data. Finally, we evaluate speed of the metamodel compared to the base models, as well as the time it takes to train the metamodels.

Before we begin the evaluation, it is important to explain how the accuracy of the metamodels is calculated. For each of the test case base models we created a dataset of one thousand randomly generated input vectors and executed the base model with those vectors to obtain the corresponding outputs to create a "ground truth" test dataset. The test dataset was distinct from the training dataset. The trained metamodel is executed with all of the input vectors from the test dataset, and the metamodel prediction for each input vector is recorded. We then calculate the absolute error for each vector by taking the absolute value of the difference between the metamodel's prediction and the ground truth model output. We sum all of the absolute errors and then divide by the number of input vectors in the test dataset to obtain the mean absolute error. Finally, to

Table 2. Most accurate committee/filter combination for each test case. *Note: Description of committee by given index found in Section 3.*

Base Model	**Committee Index**	**Filter Index**
Botnet	0	*Within n Percent*, n=50%
Circadian	1	No filter
Cluster	0	*Within n Percent*, n=25%
Cyclin	1	*Within n Percent*, n=25%
Embedded	0	*Within n Percent*, n=10%
Kanban	0	*Within n Percent*, n=50%
Molecules	0	*Within n Percent*, n=10%

facilitate comparison between the different test cases (whose outputs have very different ranges of values), we normalize the errors by dividing the mean absolute error by the range of the base model's outputs in the test dataset.

We wrote the code to build the metamodels in Python with the aid of the scikit-learn package [8].

4.1 Accuracy Given Different Committee Compositions and Filters

As we described in Section 3, we evaluate five different committee types and eight different filters, so there are a total of $8 \times 5 = 40$ different combinations of committee and filter. The most accurate stacked metamodel committee/filter combination for each test case we consider is given in Table 2. We see that Committee 0, the largest and most diverse of the committees, is the best performing committee for five of the seven test cases, and that Committee 1, which is composed of four different random forest regressors, is the best performing committee for the remaining two test cases. The best performing filters were all variants of the *Within n Percent* filter, with the exception of the Circadian test case, where not having a filter outperformed all of the other filters.

It would be useful to know which committee/filter combination works best in general. Table 2 shows that the best committee/filter combination for one base model will not be the best combination for another base model. One may of course try all forty combinations and select the best for their particular model. It does not take long to train a metamodel, so it is feasible to try many different variants. However, it is illuminating to know which committee and filter combinations work well across many different kinds of models. To determine this, we ranked each committee/filter combination for each of our seven test cases from 1 to 40, from most accurate to least. We then calculated the *average rank* for each committee/filter combination by summing all of the individual ranks and then dividing by the number of test cases. The committee/filter combination with the lowest average rank would be the most accurate metamodel across the test cases.

When we performed that calculation, we found that Committee 0 (the largest and most heterogeneous committee) paired with the *Within n Percent, n=10%* filter was the most accurate combination across all of the test case models.

It would also be illuminative to rank the committees (each paired with the filter that maximizes its performance) and filters (each paired with the committee that maximizes its performance) across the test models, respectively. We shall describe the process to determine the rank of a committee. A similar process can be used to rank a filter. To rank the committees, we first calculate the average rank of each combination of the 40 combinations of committee/filter as described in the paragraph above, and then sort this list from the smallest value to the largest. We then start at the top of this list and if we see an entry with a committee we have seen before, we delete that entry. When we reach the end of the list we will have a ranked list of committees. By ranking in this way, we compare each committee when paired with the filter that maximizes its performance.

The ranking of the committees and filters can be found in Table 3. From the table it is clear that the stacking approach benefits from having committees with many heterogeneous members. Committees with more homogeneity (e.g. Committees 1 and 2) and small committees (e.g. Committee 4) do not perform as well in general (though, as we saw previously in Table 2, they may perform better than the alternatives on specific models). The results validate the intuition that an ensemble with many diverse members will outperform an smaller ensemble with fewer, more homogeneous members.

Table 3 also provides striking results for the filter. Clearly, the *Within n Percent* filter dramatically outperformed the *Top k* filter. In fact, the *Top k* filter was worse than having no filter at all. It is interesting that the more restrictive *Within n Percent* filters outperformed the less restrictive filters of the same type, while the less restrictive *Top k* filters outperformed the more restrictive. We hypothesize that the *Top k* filters were too restrictive to allow for the diversity necessary for a well-performing ensemble.

4.2 Metamodel Accuracy: Naive vs. Best of Many vs. Stacked

Up to this point, we have only evaluated the accuracy of different variants of the stacked metamodel relative to one another. It is also important to know (a) the accuracy of the stacked metamodel compared to other metamodels, and (b) the absolute accuracy of the model.

First, it is important to know whether sophisticated ML techniques perform better than very naive methods. To help make this comparison, we created a *Naive metamodel*, which consists of a regressor that predicts that the output will be the average of the outputs in the training data, regardless of the input. More sophisticated metamodels must show that they are substantially better than this naive metamodel to demonstrate utility.

Table 4 reports the errors of the Naive, Best of Many, and Stacked metamodels. The errors we report are the mean absolute error normalized by the range of observed test values. We normalize the errors to make it possible to

Table 3. Committees and filters ranked by accuracy. Description of committee by given index found in Section 3.

Rank	Committee Index	Committee Description
1	0	Large
2	3	Medium
3	1	4 RF
4	4	MLP, RF, GPR
5	2	6 MLP

Rank	Filter Description
1	*Within n Percent*, n=10%
2	*Within n Percent*, n=25%
3	*Within n Percent*, n=50%
4	No filter
5	*Within n Percent*, n=100%
6	*Top k*, k=3
7	*Top k*, k=2
8	*Top k*, k=1

compare the accuracy of the metamodels across test cases. We chose to use the stacked metamodel architecture that was most suited for each individual test case (e.g. the metamodel variants listed in Table 2), rather than use the same committee/filter combination for every test case. All metamodels were trained with a datasets that contained 1000 random inputs each.

By examining the table we see that the Best of Many and Stacked metamodels always substantially outperform the Naive metamodel, as we had hoped. Further, for five of the seven test cases, the stacked metamodel was more than 5% more accurate than the Best of Many metamodel, and it was never less accurate. The stacked metamodels were, on average, 8.2% more accurate than the base models. These numbers demonstrate the effectiveness of the stacked approach compared to the current state-of-the-practice Best of Many approach, and validates its general applicability.

It is obvious that if a metamodel is not accurate enough it will not be a practical tool for modelers. However, it is difficult to know how accurate a metamodel must be to be a good emulator. One of the primary challenges in determining an acceptable accuracy threshold is that a modeler may use metamodels for different reasons, and different use cases may require more accuracy than others. We believe that the stacked metamodels are likely accurate enough to be useful for a variety of applications given the errors reported in Table 4, perhaps with the exception of the Circadian test case. We leave to future work an investigation to determine the acceptable accuracy threshold for a variety of common applications of metamodels.

4.3 Accuracy Given Different Training Sample Dataset Sizes

It can be time consuming to collect the training data because the base model runs slowly. To be time efficient it would be best to collect as little training data as possible while maintaining reasonable metamodel accuracy. For this reason

Table 4. Average metamodel prediction error. The error is the mean absolute error normalized by the range of test values.

Metamodel Type	Naive Metamodel Error	Best of Many Metamodel Error	Stacked Metamodel Error	Error Reduction Stacked vs. Best of Many
Botnet	0.00161	0.00111	0.00100	10.4%
Circadian	0.08461	0.05658	0.05648	0.2%
Cluster	0.03322	0.01181	0.01141	3.4%
Cyclin	0.14129	0.02369	0.02111	10.9%
Embedded	0.03726	0.00459	0.00432	5.9%
Kanban	0.17832	0.02798	0.02508	10.3%
Molecules	0.06610	0.03899	0.03262	16.3%

Table 5. Mean absolute error normalized by range of stacked metamodel trained with training datasets of different sizes.

Metamodel Type	250 Training Samples	500 Training Samples	750 Training Samples	1000 Training Samples
Botnet	0.03817	0.03227	0.01816	0.01626
Circadian	0.07011	0.064872	0.06065	0.05707
Cluster	0.02151	0.01631	0.01562	0.01349
Cyclin	0.05285	0.04281	0.03048	0.02606
Embedded	0.02445	0.01547	0.01271	0.01126
Kanban	0.03956	0.03222	0.02956	0.02555
Molecules	0.05672	0.05688	0.05072	0.04750

we investigated the impact of the size of the training set on the accuracy of the metamodel. We trained stacked metamodels with Committee 0 (the largest and most heterogeneous committee) and the *Within n percent, n=10%* filter with datasets that contained 250, 500, 750, and 1000 input vectors, respectively, and evaluated their accuracy. The results of our investigation are contained in Table 5. As expected, the metamodels trained with more training data are more accurate than those that were trained with less. However, it is encouraging to see that even those metamodels that were trained with just 250 training samples are often reasonably accurate compared to the metamodels trained with 1000 training samples.

Table 6. Base model vs. metamodel execution speed comparison and metamodel training time (all in seconds).

Name	**Base Model Execution (seconds)**	**Metamodel Execution (seconds)**	**Metamodel Training (seconds)**
Botnet	1115.5	0.3	137.4
Circadian	18291.6	0.5	204.7
Cluster	2034.9	0.6	66.4
Cyclin	2668.2	0.5	76.9
Embedded	684.7	0.2	60.6
Kanban	5737.6	0.3	64.5
Molecules	3714.5	1.1	67.1

4.4 Speed Comparison

We performed these speed experiments on an Ubuntu VM running on a laptop with an Intel i7-7500U processor and 8 GB of RAM. For each test case, both the base model and the metamodel were run with the same 200 randomly generated inputs, and the time it took to execute with those inputs was recorded. In addition, we recorded the time it took to train the metamodel for each case. We used the same committee/filter combination for each metamodel in these experiments to make it easier to compare across test cases. Committee 0 paired with the *Within n Percent, n=10%* filter was the combination we chose, for two reasons. First, we had previously determined that that combination was, on average, the most accurate across all the test cases. Second, Committee 0 is the largest committee, so it will likely take longer to train compared to the other committees, so it can be considered a "worst-case" training time. More members will take longer to train, since each member of the committee is trained separately in sequence. None of the code used in these experiments was parallelized, though regressor training could be easily parallelized.

Table 6 shows that the metamodel runs faster than the base model by several orders of magnitude, thousands of times faster. On average it took an hour and twenty minutes to run the base model with the dataset containing 200 inputs, while it took the metamodels on average half a second to run the same dataset. The metamodels are almost ten thousand times faster than their corresponding base model on average. The table also shows that training the metamodel with the training dataset can be done reasonably quickly.

5 Discussion and Recommendations

Our analysis can help modelers who are interested in using metamodeling for their own models. At a high level, it appears that reasonably accurate metamod-

els can be created for a variety of different kinds of models using a variety of different machine learning algorithms. Even relatively slow running complicated ensemble methods run thousands of times faster than the base model, making feasible analyses that would otherwise be unfeasible if performed directly on the base model. We recommend that modelers consider the use of metamodels to help with analyses involving slow-running models that have many uncertain input variables.

A stacking-based ensemble approach appears to produce more accurate metamodels than approaches that use a single regressor, even if that regressor is the best among many candidate regressors. Our analysis of the seven test cases shows that the best stacking metamodels were never worse than those metamodels produced by using the Best of Many approach, and were often significantly better. The accuracy of the metamodel was impacted by the size and heterogeneity of the constituent committees: more accurate metamodels had larger and more heterogeneous committees, while less accurate metamodels had smaller and more homogeneous committees. Judicious use of filters can increase the accuracy of the metamodel, with relatively restrictive versions of the *Within n Percent* filter being the best we evaluated. Our evaluation shows that it does not take long to train a metamodel if one already has the training dataset, so it is possible to try a number of different metamodel variants to see which would work best for that particular dataset. We recommend that modelers using metamodeling should (1) use stacking rather than the more common Best of Many approach, (2) use committees that are large and heterogeneous in their stacked metamodels, (3) use a filter to remove predictions of under performing regressors, and (4) take advantage of how quickly metamodels can be trained by trying a number of different metamodel variants to find one that works particularly well for the dataset.

We found that the accuracy of the metamodel depends in part on the size of the training dataset: the larger the training dataset, the more accurate the metamodel. However, the stacked metamodels were reasonably accurate even with little training data. This is encouraging, because our original motivation for using metamodels was as a replacement for models that run slowly. Collecting training data can be the most time consuming stage of the ML-based metamodeling, so it is encouraging that metamodels do not require onerously large training datasets to be accurate. We recommend that modelers obtain as much training data as is practical, but to not be discouraged if only a small training dataset can be obtained.

A modeler should be aware of the limitation of the approach. If the model being considered runs too slowly it may not be feasible to collect enough training data to build a reasonably accurate metamodel. However, if the model being considered runs quickly, it would be better to do the analysis (whether SA, UQ, optimization, or something else) directly on the model instead of doing the analysis indirectly with the use of a metamodel, because metamodels usually don't perfectly emulate the base model and can introduce errors into the analysis. For this reason, in some cases it may be beneficial to do a hybrid approach

in which a broad metamodel-based analysis is paired with (and perhaps even guides) a more limited and narrowly focused analysis on the base model.

6 Related Work

To the best of our knowledge, no other published research exists that evaluates the effectiveness (accuracy and speed) of metamodeling on multiple state-based quantitative models. We also know of no other work that evaluates as many different kinds of metamodels.

As mentioned previously, the stacking metamodel approach applied to quantitative state-based models was first demonstrated in [9] and [10], though only one test case was used in each of those papers, and only one committee/filter combination was considered. We, however, demonstrate the effectiveness of our approach with numerous test cases and a variety of committee/filter combinations. Many other kinds of metamodel approaches were used before stacking was introduced as a means to build a metamodel; consult [6] and [14] for a broad overview of the topic of metamodeling in general. Eisenhower et al. performed a metamodel-based analysis of a building energy model with hundreds of parameters, though the metamodel was a simple support vector machine with a Gaussian kernel, not an ensemble. Our work demonstrates that an ensemble can significantly outperform a single regressor. The approach shown in [13] uses multiple metamodels, but not as an ensemble, and [17] uses an ensemble but with a recursive arithmetic average method to combine the predictions of multiple regressors, while we use stacking. Stacking has been shown to be effective in winning machine learning competitions [11].

7 Conclusion

We have evaluated the effectiveness of metamodeling on seven real-world published models. We showed that the metamodels we created can be quite accurate in emulating the behavior of the base model, and run almost ten thousand times faster on average. Since the metamodels are so fast relative to the base model, analyses that could take too long to complete on the base model directly (such as sensitivity analysis, uncertainty quantification, and optimization) can be done indirectly with the aid of the metamodel instead in a reasonable amount of time. Metamodels can be constructed automatically with the aid of machine learning, minimizing the manual efforts of the modeler. We evaluated a variety of metamodels, and found that stacked metamodels performed better than the more commonly used Best of Many metamodels. Of the stacked metamodels, those that had larger and more diverse committees were more accurate than those that had smaller and more homogeneous committees. Appropriate filters, particularly the *Within n Percent* filters increase metamodel accuracy. We believe that ML metamodels, like those shown in this work, are an underused and relatively-unknown tool in the modeler's toolbox. We hope that our descriptions

and evaluations will encourage their future use in exploring slow-running state-based quantitative models with many uncertain input variables. Future work should evaluate models with dozens or hundreds of input variables to evaluate the scalability of the approach.

Acknowledgements

The authors would like to thank Jenny Applequist and the reviewers for their feedback on the paper. This material is based upon work supported by the Maryland Procurement Office under Contract No. H98230-18-D-0007. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the Maryland Procurement Office.

References

1. Barkai, N., Leibler, S.: Biological rhythms: Circadian clocks limited by noise. Nature **403**, 267–268 (2000)
2. Ciardo, G., Tilgner, M.: On the use of Kronecker operators for the solution of generalized stocastic Petri nets. ICASE Report 96-35, Institute for Computer Applications in Science and Engineering (1996)
3. Haverkort, B., Hermanns, H., Katoen, J.P.: On the use of model checking techniques for dependability evaluation. In: Proc. 19th IEEE Symposium on Reliable Distributed Systems (SRDS'00). pp. 228–237. Erlangen, Germany (October 2000)
4. Kwiatkowska, M., Norman, G., Parker, D.: Prism: Probabilistic symbolic model checker. In: International Conference on Modelling Techniques and Tools for Computer Performance Evaluation. pp. 200–204. Springer (2002)
5. Lecca, P., Priami, C.: Cell cycle control in eukaryotes: A BioSpi model. In: Proc. Workshop on Concurrent Models in Molecular Biology (BioConcur'03). Electronic Notes in Theoretical Computer Science (2003)
6. Liu, H., Ong, Y.S., Cai, J.: A survey of adaptive sampling for global metamodeling in support of simulation-based complex engineering design. Structural and Multidisciplinary Optimization **57**(1), 393–416 (2018)
7. Muppala, J., Ciardo, G., Trivedi, K.: Stochastic reward nets for reliability prediction. Communications in Reliability, Maintainability and Serviceability **1**(2), 9–20 (July 1994)
8. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: Machine learning in Python. Journal of Machine Learning Research **12**, 2825–2830 (2011)
9. Rausch, M., Sanders, W.H.: Sensitivity analysis and uncertainty quantification of state-based discrete-event simulation models through a stacked ensemble metamodel. In: Proceedings of the International Conference on Quantitative Evaluation of Systems. vol. 12289, pp. 276–293. Springer (2020)
10. Rausch, M., Sanders, W.H.: Stacked metamodels for sensitivity analysis and uncertainty quantification of ami models. In: Proceedings of the 2020 IEEE International Conference on Communications, Control, and Computing Technologies for Smart Grids (SmartGridComm). pp. 1–7 (2020)

11. Risdal, M.: Stacking made easy: An introduction to StackNet by competitions grandmaster Marios Michailidis (KazAnova). http://blog.kaggle.com/2017/06/15/stacking-made-easy-an-introduction-to-stacknet-by-competitions-grandmaster-marios-michailidis-kazanova/, accessed: 2019-12-13
12. Ruitenbeek, E.V., Sanders, W.H.: Modeling peer-to-peer botnets. In: Proc. 2008 Fifth International Conference on Quantitative Evaluation of Systems. pp. 307–316 (Sep 2008)
13. Tenne, Y.: An optimization algorithm employing multiple metamodels and optimizers. International Journal of Automation and Computing **10**(3), 227–241 (2013)
14. Viana, F., Gogu, C., Haftka, R.: Making the most out of surrogate models: Tricks of the trade. Proceedings of the ASME Design Engineering Technical Conference **1**, 587–598 (Jan 2010)
15. Vilar, J., Kueh, H.Y., Barkai, N., Leibler, S.: Mechanisms of noise-resistance in genetic oscillators. Proc. National Academy of Sciences of the United States of America **99**(9), 5988–5992 (2002)
16. Wolpert, D.H.: Stacked generalization. Neural Networks **5**(2), 241–259 (1992)
17. Zhou, X.J., Ma, Y.Z., Li, X.F.: Ensemble of surrogates with recursive arithmetic average. Structural and Multidisciplinary Optimization **44**(5), 651–671 (2011)

A Appendix: Values of Input Variables for Test Case Models

In this appendix we list the input variables for each of the seven models we use as test cases in our evaluation, along with the corresponding domain. When the domain was not specified by the original authors, we estimated it, informed by the original paper and subject matter expertise.

Table 7. List of inputs used in the Botnet test case.

Variable Name	**Domain**
ProbConnectToPeers	[0.25, 1]
ProbPropagationBot	[0.05, 0.15]
ProbInstallInitialInfection	[0.05, 0.15]
Prob2ndInjctnSuccessful	[0.25, 1]
RateConnectBotToPeers	[0.0166, 6]
RateOfAttack	[0.0166, 6]
RateSecondaryInjection	[0.0166, 6]
RateBotSleeps	[0.05, 0.15]
RateBotWakens	[0.0005, 0.0015]
RateActiveBotRemoved	[0.05, 0.15]
RateInactiveBotRemoved	[0.00005, 0.00015]

Table 8. List of inputs used in the Circadian test case.

Variable Name	**Domain**
T	[0, 200]
transc_da	[45, 55]
transc_da_a	[450, 550]
transc_dr	[0.009, 0.011]
transc_dr_a	[45, 55]
transl_a	[45, 55]
transl_r	[4.5, 5.5]
bind_a	[0.9, 1.1]
bind_r	[0.9, 1.1]
deactivate	[1.8, 2.2]
rel_a	[45, 55]
rel_r	[90, 110]
deg_a	[0.9, 1.1]
deg_c	[0.9, 1.1]
deg_r	[0.18, 0.22]
deg_ma	[9, 11]
deg_mr	[0.45, 0.55]

Table 9. List of inputs used in the Cluster test case.

Variable Name	**Domain**
ws_fail	[0.0002, 0.02]
switch_fail	[0.000025, 0.0025]
line_fail	[0.00002, 0.002]
startLeft	[5, 15]
startRight	[5, 15]
startToLeft	[5, 15]
startToRight	[5, 15]
startLine	[5, 15]
repairLeft	[1, 3]
repairRight	[1, 3]
repairToLeft	[0.125, 0.375]
repairToRight	[0.125, 0.375]
repairLine	[0.0625, 0.1875]

Table 10. List of inputs used in the Cyclin test case.

Variable Name	**Domain**
N	[2, 4]
t	[0, 60]
k	[0, 4]
R1	[0.0045, 0.055]
R2	[0.0009, 0.0011]
R3	[0.0027, 0.0033]
R4	[0.45, 0.55]
R5	[0.27, 0.33]
R6	[0.0045, 0.0055]
R7	[0.0081, 0.0099]
R8	[0.0081, 0.0099]
R9	[0.009, 0.011]
R10	[0.0153, 0.0187]
R11	[0.018, 0.022]

Table 11. List of inputs used in the Embedded test case.

Variable Name	**Domain**
lambda_p	[1/(2*365*24*60*60), 1/(30*24*60*60)]
lambda_s	[1/(90*24*60*60), 1/(7*24*60*60)]
lambda_a	[1/(4*30*24*60*60), 1/(7*24*60*60)]
tau	[1/90, 1/30]
delta_f	[1/(2*24*60), 1/(8*60*60)]
delta_r	[1/(5*60), 1]

Table 12. List of inputs used in the Kanban test case.

Variable Name	**Domain**
t	[1, 5]
in1	[0.5, 1.5]
out4	[0.45, 1.35]
synch123	[0.2, 0.6]
synch234	[0.25, 0.75]
back	[0.15, 0.45]
redo1	[0.18, 0.54]
redo2	[0.21, 0.63]
redo3	[0.195, 0.585]
redo4	[0.165, 0.495]
ok1	[0.42, 1.26]
ok2	[0.46, 1.38]
ok3	[0.455, 1.365]
ok4	[0.385, 1.155]

Table 13. List of inputs used in the Molecules test case.

Variable Name	**Domain**
T	[0, 0.003]
i	[0, 10]
N1	[10, 100]
N2	[10, 100]
N3	[10, 100]
e1rate	[90, 110]
e2rate	[9, 11]
e3rate	[27, 33]
e4rate	[18, 22]