Sensitivity Analysis and Uncertainty Quantification of State-Based Discrete-Event Simulation Models through a Stacked Ensemble of Metamodels

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. Realistic state-based discrete-event simulation models are often quite complex. The complexity frequently manifests in models that (a) contain a large number of input variables whose values are difficult to determine precisely, and (b) take a relatively long time to solve.
Traditionally, models that have a large number of input variables whose values are not well-known are understood through the use of sensitivity analysis (SA) and uncertainty quantification (UQ). However, it can be prohibitively time consuming to perform SA and UQ.
In this work, we present a novel approach we developed for performing fast and thorough SA and UQ on a metamodel composed of a stacked ensemble of regressors that emulates the behavior of the base model. We demonstrate the approach using a previously published botnet model as a test case, showing that the metamodel approach is several orders of magnitude faster than the base model, more accurate than existing approaches, and amenable to SA and UQ.

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

1 Introduction

Many state-based discrete-event simulation models of real-world systems are complex, large, and contain uncertain input parameters. It is challenging to make realistic quantitative models smaller and simpler (and thus faster to execute) because the world is large and complex. It is also very difficult to remove uncertainty in the model input values. Obtaining precise, certain input values in many domains may be prohibitively expensive or even impossible. Special approaches must be developed and used to make effective use of such models, given the issues of long run times and uncertain input values.

The traditional way of handling uncertain input parameter values is to perform (a) sensitivity analysis (SA) to determine the most sensitive inputs, and

(b) uncertainty quantification (UQ) to determine how the uncertainty in the inputs propagates to uncertainty in the model output. Both SA and UQ typically require that models be solved many times, with the input variable values being varied each time. If the calculation of the model's metrics could be done quickly, comprehensive SA and UQ could be accomplished with a reasonable computation and time budget. If the model input values were known with certainty, it would be unnecessary to execute the model multiple times to perform SA and UQ, so the time and computation required to obtain a single model solution would be less of a concern. However, the twin issues of long solution times and uncertain model input values in complex state-based models present a significant challenge to modelers.

We propose the use of *metamodels* (also known as *emulators* or *surrogate models*) to address the two issues. Metamodels are models of the original base model that attempt to approximate the relationship between the base model's inputs and outputs, and can generally be executed much more quickly than the base model. With an acceptably accurate metamodel, fast and comprehensive sensitivity analysis and uncertainty quantification can be performed on the metamodel in place of the original base model. While metamodels can be constructed by hand, normally they are automatically constructed using machine learning techniques (e.g. Gaussian process regressors, multilayer perceptrons, and random forests).

A chief concern with metamodeling is the choice of an appropriate machine learning technique, as each has its own strengths and weaknesses. While most related work arbitrarily chooses a particular machine learning technique, or evaluates a small handful of different techniques and chooses the strongest, in this work we use an ensemble of heterogeneous regressors in an effort to benefit from the strengths of each approach while mitigating the weaknesses. We structure the ensemble using a custom stacking approach. Stacking is a cutting-edge technique used by the winners of some recent machine learning competitions [15], but we adapt it for use on state-based discrete-event simulation models. We use a sophisticated botnet model [16] as a test case and have performed sensitivity analysis and uncertainty quantification on the model, using the metamodeling approach.

To the best of our knowledge, we are the first to propose and demonstrate a metamodeling-based approach to the analysis of complex quantitative security models, and the first to use a stacking-based approach to perform SA and UQ on real-world quantitative models. We show that our stacked metamodels are several orders of magnitude faster than the original models, are more accurate than traditional metamodels, and are amenable to SA and UQ that could not be performed on the original base model within a reasonable time budget. We have demonstrated the approach with the aid of a pre-existing published peer-to-peer botnet security model [16], which we use as a test case.

The rest of this paper is organized as follows. In Section 2, we explain the process by which we use stacking to construct an accurate metamodel. In Section 3, we describe the sensitivity analysis and uncertainty quantification techniques

we employ to analyze the metamodel. In Section 4, we briefly explain the botnet model we used as a test case to demonstrate and evaluate the approach. We show the results of our analysis of the botnet model in Section 5. We discuss limitations, use cases, and future directions for the approach in Section 6, review related work in Section 7, and conclude in Section 8.

2 Approach

In our context, a *metamodel*, also known as a *model surrogate* or *emulator*, is a model of a model (which we will refer to in this work as the *base model*) that attempts, given a particular vector of input variables (which we shall call an *input*), to produce an output that matches as closely as possible the output that the base model would produce given the same input, for all inputs. In this work we consider quantitative outputs (metrics), though the approach could be extended to consider qualitative outputs (i.e., by using classifiers rather than regressors). Metamodels can rarely achieve perfect accuracy in emulating the base model, but they often run much faster than the base models. The long time needed to run the base model, together with the need to run the base model many times to conduct sensitivity analysis, uncertainty quantification, and optimization, provides the motivation to find fast metamodels that can emulate the base model with acceptable accuracy.

While high-quality metamodels can be constructed manually by an expert familiar with the base model, it is often easier and faster to build the metamodel automatically. At a high level, the metamodeling process is conducted in three stages:

1. Data for training and testing are acquired by generating a number of different model inputs and running the base model with those inputs to observe the resulting outputs.
2. The training data are used by a machine learning algorithm to train a metamodel.
3. The test data are used to assess the quality of the trained metamodel.

To begin, in the first stage, data for training and testing must be acquired. Time and computation constraints restrict the maximum number of inputs that can be run on the base model. We can imagine that an n-dimensional input vector describes a point in the n-dimensional input space. The metamodel will benefit from high-quality training data that gives the most complete view of the input space possible, given the limited number of samples. For example, if all the training inputs are clustered closely together in the input space, the trained metamodel may be accurate only in that limited region of the input space. The input space should be explored as efficiently as possible. The most important decision at this first stage is the choice of strategy for input selection. We consider three input selection strategies in this paper: random sampling, Latin hypercube sampling, and Sobol sequence sampling.

In the second stage, the training data gathered in the previous stage are used to train a metamodel: a model of the original base model that attempts to produce the same output the base model would produce if it were given the same input. As mentioned previously, we consider only quantitative model output in this work, so the metamodels will be regressors. One can choose from a variety of machine learning regressors, including, e.g., kriging (Gaussian process regressors), random forest regressors, support vector machine regressors, and k-nearest neighbors (KNN) regressors. The most important decision at this stage is the selection of the machine learning technique that will be able to produce the most accurate metamodel given the training data collected in the first stage. Instead of selecting one regressor, we use the predictions from multiple heterogeneous regressors through stacking.

In the third stage, the test data are used to evaluate the accuracy of the trained metamodel. Given each input in the test data, the metamodel will produce an output, and the metamodel's output is compared to the base model's output. The absolute value of the difference between the two outputs quantifies the error of the metamodel. If test inputs are generated randomly and independently, one can use the standard statistical methods to determine the average error and associated confidence interval.

The remainder of this section will be devoted to explaining the procedures for obtaining the training data and constructing the metamodel.

2.1 Sampling

Acquiring appropriate data for training and testing is vitally important for constructing metamodels and evaluating how well they emulate the base model. The general idea is to choose an input (in other words, a vector that contains a specific value for each input variable), execute the base model with that input, observe the resulting model output or metric, and then record the input and output by adding them to the list of previously observed input-output pairs. This process is repeated multiple times until some stopping criterion is reached, such as the exhaustion of the allocated time budgeted for acquiring training data. The choice of input at each iterative stage is very important. If all the inputs in the training data are "close" to one another in the input space, the metamodel trained on that data may not be able to accurately emulate the base model in other regions. In this paper, we consider three ways of exploring the input space: random sampling, Latin hypercube sampling, and Sobol sequences. The differences between the three methods are illustrated in the two-dimensional case in Figures 1, 2, and 3.

Random Sampling Random sampling is the simplest sampling strategy that we consider. Random sampling is conducted by selecting a value for each input variable from the range of possible values at random, independently of any prior sample. Random sampling has a chance of exploring any region in the input space, unlike deterministic sampling, but random sampling can have the unfortunate side effect that samples may cluster in regions of the input space that

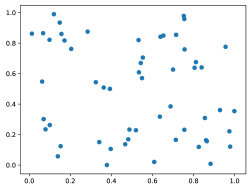

Fig. 1. 55 Random samples.

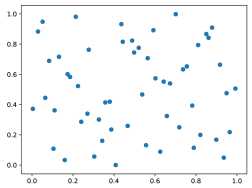

Fig. 2. 55 Latin hypercube samples.

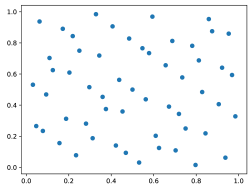

Fig. 3. 55 Sobol samples drawn from a uniform distribution.

have already been well explored while ignoring regions that have not been explored at all. The clustering effect can easily be observed in the two dimensional case shown in Figure 1.

Latin Hybercube The Latin hypercube sampling (LHS) [8] [5] strategy attempts to explore the space more evenly than random sampling. It is inspired by the Latin square puzzle, which consists of an n by n array of n unique symbols such that a particular symbol will appear exactly once in every row and every column. [3] In the two-dimensional case, Latin hypercube sampling generates N samples by dividing the range in the x and y dimensions into N equally likely intervals, forming a grid of N rows and N columns, and choosing a sample such that each row and each column is sampled exactly once.[4] Similarly, in the case of a finite number of inputs, LHS generates N samples by dividing each input variable into N equally probable intervals and choosing exactly one sample from each interval. LHS provides stronger guarantees of coverage than random sampling, because each interval is sampled once, while a particular interval may not be sampled at all with random sampling. However, LHS may not evenly cover the space very well. Indeed, there exist pathological cases in which Latin hypercube sampling leaves large regions of the input space totally unexplored. [5] An illustration of 55 samples chosen via Latin Hypercube sampling is shown in Figure 2. Notice that the sampling approach produces some pairs that are close to one another in the input space.

Sobol Sequences The Sobol method, in contrast to random sampling and LHS, generates low-discrepancy sequences. Informally, a low-discrepancy sequence will sample the input region relatively evenly. For a formal definition of discrepancy and technical descriptions of the Sobol generation procedure, please see [17].

[3] A solved Sudoku puzzle is an example of a Latin square.

[4] This is similar to the classic "8 rooks" problem in chess.

[5] For example, taking a sample at every cell along the diagonal is a valid Latin hypercube sample sequence in two dimensions.

An illustration in the 2-dimensional case can be seen in Figure 3. Notice how evenly the Sobol method samples the region of interest compared to the random sampling method and LHS.

2.2 Metamodeling

A machine learning model can be trained to emulate the base model, producing outputs that are as close as possible to the base model outputs, given the same input. If the base model output is qualitative, a machine learning classifier can be trained as the metamodel. If, on the other hand, the base model output is a quantitative metric, the metamodel will be a machine learning regressor. In this work, we only consider base models that produce quantitative metrics, but the techniques we utilize could be naturally extended to study base models that produce qualitative results. A wide variety of regressors exist, each with its own strengths. Unfortunately, in general, it is not possible to know a priori which particular machine learning technique will produce the most accurate metamodel for a given black box base model.

We consider a number of regressors in our analysis. Each regressor has its own strengths, weaknesses, and assumptions about the underlying data. We consider seven major types of regressors: random forest (RF) regressors, multilayer perceptrons (MLP), gradient-boosting machines (GBM), the RidgeCV regressor, k-nearest neighbors (KNN) regressors, Gaussian process (kriging) regressors, and stochastic gradient descent (SGD) regressors. Many of the regressors have hyperparameters that change their behavior. For example, the number of neighbors used in the k-nearest neighbors regressor can be any positive integer; different solvers and activation functions can be used by the multilayer perceptron; and the loss function used by the gradient-boosting machine can be changed. In this work we consider twenty-five different regressors of the seven types mentioned above: one random forest, seven different multilayer perceptrons, four different gradient-boosting machines, one Ridge regressor, ten different k-nearest neighbor regressors, one Gaussian process regressor, and one stochastic gradient descent regressor.

One could simply choose a particular regressor to serve as the metamodel (based on intuition or subject matter expertise), or one could train several different candidate regressors, compare their levels of accuracy, and select the one with the best performance to serve as the metamodel. However, the best regressor alone may not perform as well as several regressors working together. The simplest way to use multiple regressors together is to establish a voting committee of regressors, where the regressor predictions are averaged to produce a combined prediction. A drawback of that approach is that poorly performing, inaccurate regressors have the same vote as the most accurate regressor. It would be beneficial to weight the votes, such that the more accurate regressors have more say in the final prediction than the less accurate regressors. Stacking is one way to accomplish such weighting.

Stacking is a machine learning technique that accomplishes the weighting by training another regressor (or committee of regressors) on the original training

data plus the predictions that the original regressors made [21]. We shall refer to the regressors trained solely on the training data as *layer-1 regressors*, and the regressors that were trained on the training data combined with the predictions from the layer-1 regressors as *layer-2 regressors*. We do not know a priori which regressor will perform the best in the second layer, so we train all twenty-five regressors as layer-2 regressors. We then take the average prediction of the layer-2 regressors as the final metamodel prediction.

In practice, we find that some of the layer-1 regressors produce predictions that are so inaccurate that they significantly degrade the performance of some of the layer-2 regressors. Therefore, we filter the layer-2 training data so they include only the predictions from the most accurate, best-performing layer-1 regressors. Similarly, some of the layer-2 regressors are so inaccurate that their predictions would significantly affect the quality of the final vote, so we filter the predictions of those layer-2 regressors as well. The filtering threshold is set to remove the predictions of any regressor whose average prediction error is more than 25% worse than that of the most accurate regressor at that layer. We found that this filtering strategy was effective in the evaluation of our case study model, but the threshold may have to be adjusted for other models, or a completely different strategy could be used (e.g., using the best k regressors out of the set of n, where $k < n$). We plan to evaluate other filtering strategies in future work.

An illustration of the approach can be found in Figure 4. Once trained, the stacked ensemble of regressors can form an accurate metamodel of the base model. Input analysis (such as sensitivity analysis and uncertainty quantification) techniques can then be applied to the metamodel in place of the base model.

3 Analysis Techniques

Uncertainty quantification and sensitivity analysis can help a modeler gain a deeper understanding of the model's input variables. Uncertainty quantification determines the likelihood of different outcomes given the uncertainty in the values of the input variables, and can be conducted in a straightforward manner using a Monte Carlo method [2]. Unfortunately, sensitivity analysis techniques are more complicated, and the different techniques may give differing answers. Sensitivity analysis attempts to determine the degree to which an input variable influences the output. A number of different techniques can be used to perform sensitivity analysis. We briefly describe three that we use in this work: Sobol sensitivity analysis, the Morris method, and the feature importance method. Sensitivity analysis can be used in a variety of ways. For example, a modeler may be able to dedicate only a limited amount of time and resources to reducing the uncertainty in particular model inputs (e.g., by performing experiments or seeking the opinions of experts). If that is the case, the modeler would like to know the input variables whose values have the greatest effect on the model output, so the uncertainty-reduction efforts can be focused on those variables.

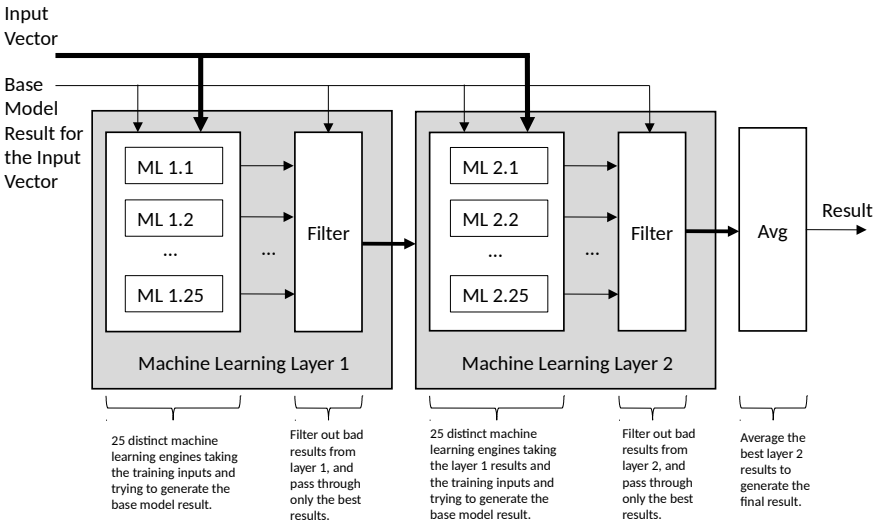

Fig. 4. Overview of the metamodel construction approach.

Sobol Sensitivity Analysis: Sobol sensitivity analysis [18] is a method for performing global sensitivity analysis. The method calculates the total order index for each value, which measures its total contribution to the variance in the output. The indices can be used to rank the input variables to determine the most and least sensitive inputs.

Morris Method: The Morris method is used to perform global sensitivity analysis; for details, consult [9] [1]. The method varies one input variable at a time. First, it selects a point in the input space and runs the model with that input to determine the resulting output, which is used as a baseline. Next, each one of the input variables is assigned a new value, one at a time, while the others are held at the original baseline value, and the model is run to determine the magnitude of the difference between the resulting output and the baseline output. Once every input variable has been modified in turn, a completely new input is chosen (i.e., all the values for the input variables are changed at once), and the process repeats several times. The Morris method can use the collected data to perform a global sensitivity analysis, calculating a value μ^* for each input variable that quantifies its impact on the model output. By comparing the μ^* values, one can order the input variables based on their impact on the model output.

Feature Importance Method: The feature importance method [6] [14] can be used as a way to rank the influence each input has on the model output. We shall describe the method at a high level. First, a metamodel is trained on the training data, and its baseline accuracy is recorded. Then, all the values of one input variable in the training data are perturbed through addition of noise.

The metamodel is retrained with the modified training data, and the model's accuracy is compared with the accuracy of the baseline metamodel. If the input value has a large impact on the output value, we would expect a relatively large decrease in the accuracy of the new metamodel. If the input value has no impact on the output value, we would expect no appreciable drop in accuracy. The values of each input variable are perturbed in turn in the same way. The technique can be used to rank the input variables from most impactful to least impactful.

4 Test Case

Our test case considers a stochastic model of the growth of botnets in different conditions. A full description of the model may be found in [16]. A botnet is a collection of computers that have been compromised and hijacked by a malicious actor. A botnet can grow or shrink in size. The model gives an estimate of the size of the botnet at the end of one week, given a number of assumptions, including the rate at which bots are removed from the botnet by defenders. We imagine that defenders may have a choice among several different methods for removing the bots from the botnet. Methods that remove the bots faster may cost more or have deleterious side effects on the system's performance. In this imagined scenario, the defender would like to know the slowest rate at which the bots may be removed while ensuring that the botnet does not grow above a certain fixed size. The defenders must take into account their lack of knowledge of the precise values of all the input variables. In addition, the defenders would like to know which input variables have the largest effects on the model output, so that they may obtain the most accurate value estimates possible for the sensitive input variables.

We will give a brief overview of the pertinent details. The eleven input variables used in the model are listed in Table 1. In [16], all the input variables were assigned baseline values based on suggestions from subject matter experts. In our analysis, we assume that all the inputs are uncertain, and the uncertainty range is $\pm 50\%$ of the baseline values given in the original paper, with the following exceptions: $ProbConnectToPeers$ and $Prob2ndInjctnSuccessful$ are probabilities and thus may not be greater than 1, so we assign $[0.25, 1]$ as a reasonable range of uncertainty; and we increased the range of uncertainty for $RateConnectBotToPeers$, $RateOfAttack$, and $RateSecondaryInjection$ so that the rates fell between once every 10 seconds to once every hour, which we believed to be realistic assumptions.

5 Evaluation

We used Python to write our sampling, metamodeling, and analysis scripts. The construction of metamodels was accomplished with the aid of the scikit-learn Python package [10], and the SALib package was used to perform Sobol and Morris method sensitivity analysis [4]. All the experiments were run on a machine with an Intel i7-5829K processor and 32 GB of RAM.

Table 1. 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]

5.1 Speed Comparison

All reported times were rounded to the nearest tenth of a second. The base model was run one thousand times with random inputs, and it took 7,246.1 seconds (over 2 hours) to obtain the corresponding one thousand model outputs. That works out to just over 7 seconds per input, with a standard deviation of 24.8 seconds. The longest and shortest times it took to calculate an individual output were 510.6 and 0.8 seconds, respectively. The metamodel was also run one thousand times with random inputs, and it took a total of 1.9 seconds to obtain the corresponding one thousand model outputs. It follows that the metamodel can run several thousand times faster than the base model.

We found that the time needed to train the metamodel was correlated with the size of the dataset. Training of the stacked metamodel took 5 minutes and 19.1 seconds with the dataset that contained 4,000 random inputs, 1 minute 36.0 seconds with the dataset that contained 1,000 random inputs, and 32.0 seconds with the dataset that contained 250 random inputs. The time it takes to collect the training data is significantly greater than the time needed to train the metamodel. Recall that in our approach, only a limited number of samples can be collected and used to train the metamodel, because it takes such a long time to execute the base model. The training datasets will therefore be relatively small, leading to relatively fast training times.

5.2 Metamodel Accuracy

Having established the speed with which the metamodel can be executed, we turn to an evaluation of the metamodel's accuracy. Recall that the metamodel attempts to produce an estimate of the final size of the botnet after one week. The estimation error is the absolute value of the difference between the metamodel's

estimate and the base model's estimate, given a particular input. The errors reported are the average absolute difference between the metamodel's predictions and the corresponding base model's outputs across all the data in the test set. 2,500 randomly generated inputs (and associated base model outputs) comprised the test set. The minimum and maximum botnet sizes recorded in the test set were 0 and 37,143, respectively.

First, we consider the accuracy of the metamodel given the three different input sampling strategies and three different training dataset sizes. The results can be seen in Figure 5. Unsurprisingly, we found that the metamodel error decreased when the training set contained more data. We also found that the Sobol sequence was the best-performing sampling strategy as the number of samples grew: as the number of samples grows, the effect of the low-discrepancy attribute of the Sobol sampling sequence becomes more obvious. Encouragingly, it appears that the metamodel can perform well even with a relatively low number of training samples: the metamodel trained with 250 random samples, the least accurate of the nine shown in Figure 5, had an average estimation error that was less than 1% of the range found in the test data.

Next, we show that using our stacking approach is better than simply using the predictions from the best-performing of the twenty-five regressors we consider (which we call the *Best of Many* metamodel), and that the predictions of both methods perform better than a naive metamodel. Our naive metamodel calculates the average value of the outputs in the training dataset, and gives that average as its prediction regardless of the model input. Therefore, any well-performing regressor should produce more accurate predictions than the naive metamodel. We compare the errors of the naive metamodel, the most accurate single regressor (the Best of Many metamodel), and the stacked metamodel, given different training data. The results can be seen in Table 2. We see that the Best of Many metamodel has about half the average prediction error as the naive metamodel. In the worst case, the metamodel composed of stacked regressors has a 10% average error reduction compared to the best single regressor, and in the best case, it achieves a 32% average error reduction. This analysis shows that regressor stacking can lead to a significantly more accurate metamodel for our cybersecurity model compared to simply using a single regressor that is the best among many candidate regressors.

5.3 Uncertainty Quantification: Determination of Optimal Removal Rate

Assume that a defender has the ability to remove nodes from the botnet at a specific rate, but a faster rate costs the defender more than a slower rate. That may be the case when the defender can implement more effective but more expensive countermeasures to respond to the attack. The defender may wish to know how quickly the botnet can be expected to grow given different removal rates, and uncertainty in the other input parameters.

We conducted an experiment in which we used our stacked metamodel trained on the dataset consisting of four thousand Sobol samples. For each experiment,

Table 2. Average metamodel prediction error (lower is better).

Training Data	Naive Metamodel Error	Best of Many Metamodel Error	Stacked Metamodel Error	Error Reduction Stacked vs. Best of Many
Random250	691	338	281	17%
Random1000	589	285	230	20%
Random4000	973	245	192	22%
LHS250	646	330	260	21%
LHS1000	696	296	222	25%
LHS4000	848	243	219	10%
Sobol250	761	371	282	24%
Sobol1000	825	321	230	28%
Sobol4000	704	234	161	32%

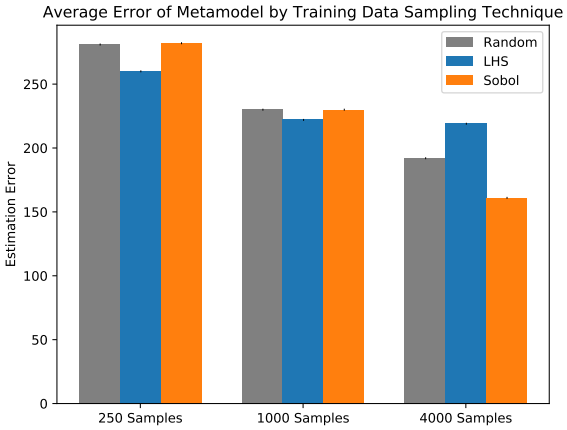

Fig. 5. Comparison of regressors trained on data obtained from random sampling, Latin hypercube sampling, and Sobol sequence sampling, respectively.

we fixed the value of the *RateActiveBotRemoved* variable. We then generated ten thousand Sobol samples for the other input variables in the ranges given by Table 1, and ran the regressor with those inputs to observe the predicted botnet size given the conditions described by the input variables. For each value of *RateActiveBotRemoved* we tested, we found the average botnet size, the 95^{th} percentile, the 99^{th} percentile, and the largest recorded botnet. That information can help a defender determine the slowest permissible removal rate given uncertainty in the input parameters. The results of our analysis can be found in Table 3.

Table 3. Estimated botnet size given removal rate.

Removal Rate	Average	95%	99%	Largest
0.05	170	614	2436	9277
0.04	264	984	3861	11568
0.03	493	1970	6330	30004
0.02	1000	4229	10033	31364
0.01	1661	7206	27029	35593
0.001	2163	8594	27573	48152
0.0001	2207	8650	27965	48152

5.4 Sensitivity Analysis

We performed a sensitivity analysis to determine the degree to which model inputs impact the value of the output. As discussed in Section 2, traditional SA techniques often cannot be applied directly to the base model because of the long model run times. To overcome that issue, we used several different SA techniques and compared the results to determine the highly sensitive and highly insensitive model inputs. We conducted sensitivity analysis through the Morris method, the feature importance method, and the Sobol method.

The results of the analysis can be found in Table 4. Each of the three methods has two columns; the left column gives the score calculated by the method (M_μ^* for the Morris method, increase in metamodel error rate for the feature importance method, and the total order indices for the Sobol method).[6] Finally, in the rightmost two columns, we show the average ranking across the three methods for each input variable and the standard deviation. We will comment on the results of each SA method in turn, and then discuss how they may be interpreted when taken together.

First, the Morris method returns a μ^* value for each parameter, and since higher values indicate higher sensitivities, we can rank the input parameters relative to one another by using this value. The Morris method indicates that the model output is most sensitive to the $RateOfAttack$ input variable, and least sensitive to the $RateBotSleeps$ variable.

Second, the feature importance method determines how much the average error of the regressor's prediction increases (or conversely, how much its accuracy decreases) when a particular input variable's value is distorted with noise. The regressor is least accurate when the $RateOfAttack$ input variable's value is corrupted with noise, indicating that variable's importance. The negative error calculated for the $RateSecondaryInjection$ and $RateConnectBotToPeers$ input variables indicates that the regressor does not make much use of these values when performing the regression.

[6] The μ^* values and the feature importance errors were rounded to the nearest integer, and all other values in the table were rounded to the nearest hundredth.

Input Name	Morris		F. I.		Sobol		Combined	
	μ^*	Rank	Error	Rank	Total Ord. Ind.	Rank	Avg. Rank	Std. Dev.
ROfAttack	3620	1	355	1	3.54	1	1	0
RActiveBRemoved	2648	3	303	3	3.53	2	2.67	0.58
PPropagationB	2492	4	307	2	1.25	5	3.67	1.53
PInstall1stInfection	2675	2	74	7	1.20	6	5	2.65
RBWakens	1311	8	120	4	1.69	4	5.33	2.31
PConnectToPeers	2041	6	13	9	1.83	3	6	3
RInactiveBRemoved	2373	5	77	6	1.09	10	7	2.65
RBSleeps	0	11	103	5	1.13	7	7.67	3.06
PSecondaryInjection	1979	7	74	8	1.09	9	8	1
RSecondaryInjection	1113	9	-20	11	1.13	8	9.33	1.53
RConnectBToPeers	1073	10	-19	10	1.06	11	10.33	0.58

Table 4. Inputs variables ranked from most to least sensitive by taking the average of the rankings provided by the Morris, feature importance, and Sobol methods.

Third, the Sobol method calculates total-order indices for the input variables, with higher values indicating more sensitivity. We can see that the Sobol method is in agreement with the other two methods in finding that the model output is most sensitive to the value of the *RateOfAttack* variable. The output is also quite sensitive to the *RateActiveBotRemoved* variable.

Finally, when they are taken together, it can be seen that there is broad agreement among the three methods in their ranking of the inputs. In Table 4, we show the combined average rank for each input, which we calculated by summing the input variable's sensitivity ranks as determined by the three SA methods and dividing by the number of SA methods. We also calculated the standard deviation of the three rankings; a low standard deviation shows that the methods are in agreement about the sensitivity rank of an input variable, while a high standard deviation shows disagreement. For example, each method ranked the *RateOfAttack* variable as the most sensitive, so it is given an average rank of $(1 + 1 + 1)/3 = 1$ and a standard deviation of 0, which shows perfect agreement. On the other hand, the three methods disagreed the most on the relative sensitivity ranking of the *RateBotSleeps* variable. The Morris method ranked it as the least sensitive, while the feature importance method ranked it as the fifth most sensitive, and the Sobol method ranked it as the seventh most sensitive, for a combined average rank of $(11 + 5 + 7)/3 = 7.67$, and a standard deviation of 3.06. In the absence of ground truth from the base model, it is encouraging to find that multiple SA methods largely agree on the rankings.

6 Discussion

Limitations We believe that the metamodeling approach outlined in this work can aid in the evaluation and validation of realistic, complex, long-running computer security models. However, there are limitations to the approach. First, the approach generally sacrifices some accuracy for speed, since usually the metamodel cannot perfectly emulate the behavior of the base model. If the base model can be run very quickly, or if the modeler can afford to wait for the base model to run, the speedup achieved through the approach described in this work may not justify the decrease in accuracy. In practice, we believe that there are many realistic models that cannot be solved quickly, and that it would be prohibitively expensive to obtain enough computational resources to sufficiently decrease model execution times. On the other hand, the model may run so slowly that it is infeasible to obtain enough training samples to train a sufficiently accurate metamodel. If enough training samples cannot be obtained, we recommend modifying the base model so it runs more quickly, or building a metamodel by hand. A further limitation is that the metamodels may be accurate for most inputs, but inaccurate in a region that the modeler considers particularly interesting. In that case, we recommend that the modeler use adaptive sampling to drive the sampling towards the interesting region [7]. Finally, changing the base model invalidates the training data and the metamodel constructed using the training data, so the process would need to be repeated for every change to the base model.

Use Cases We believe that our approach can help architects design better, more secure systems by helping them understand how a system might perform given different conditions and assumptions. The approach can be applied (1) to simulation and analytical models, as this paper demonstrates; (2) to long-running emulation experiments, e.g., experiments that use virtual machines and virtual switches to emulate a real network with real hosts cheaply and realistically, and (3) to prototypes of a system.

In the cybersecurity context, many realistic security models run too slowly to be used in an online fashion, so they can be used only at design time. However, a metamodel may be able to give helpful, near real-time guidance to human defenders responding to an ongoing cyber attack. An advanced intrusion prevention system (IPS) may also be able to use information from the metamodel in an online fashion (along with normal monitoring data, such as alerts from intrusion detection systems (IDS)), to thwart attacks more effectively.

Future Work We are in the process of applying the metamodeling-based analysis approach to other published security models, particularly, to an advanced metering infrastructure (AMI) cybersecurity model [13] [11] and a password cybersecurity model [12], to see whether the results generalize. We intend to explore whether more accurate metamodels can be constructed if adaptive sampling is used to obtain the training data [7]. We also intend to further explore different ways to structure the ensemble stack, for example, by using more than two layers.

7 Related Work

To the best of our knowledge, no prior work exists that uses metamodels in place of complex cybersecurity models. In addition, we know of no published work on the application of model stacking to state-based discrete-event simulation models. For a thorough introduction to the topic of metamodeling, see the excellent survey papers on the topic [7] [20]. Xiao, Zuo, and Zhou propose an adaptive sampling approach that can be used to construct metamodels for reliability analysis, and demonstrate its use on four small reliability model problems [22]. Eisenhower et al. demonstrated a methodology that uses a support vector machine with a Gaussian kernel as a metamodel to perform an optimization for a building energy model with over 1,000 parameters [3]. Tenne proposes an approach that uses multiple metamodels and optimizers [19], but in a one-at-a-time fashion; the metamodel predictions are never fused or combined, whereas in our work the predictions of multiple regressors are taken together to form a better overall prediction. Zhou et al. like us, propose to use an ensemble of metamodels together, rather than train several metamodels and pick the one that performs best [23]. However, their work uses a recursive arithmetic average method to combine the predictions from multiple regressors, while we use stacking. Many papers have been published on sensitivity analysis in general. An overview of methods for global sensitivity analysis (including metamodeling-based approaches) can be found in [6].

8 Conclusion

In this work, we introduced a metamodeling-based approach for performing sensitivity analysis and uncertainty quantification on complex real world models. We demonstrated the approach on a published cybersecurity model. Our stacked metamodel is orders of magnitude faster than the base model, and more accurate than common existing approaches to metamodeling. The metamodeling-based approach outlined in this paper can be used by system architects to evaluate and validate realistic long-running models with a number of input variables whose values are not known with certainty.

Acknowledgements

The authors would like to thank Jenny Applequist, Lowell Rausch, 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. Campolongo, F., Cariboni, J., Saltelli, A.: An effective screening design for sensitivity analysis of large models. Environmental Modelling and Software **22**(10), 1509–1518 (2007)
2. Cunha, A., Nasser, R., Sampaio, R., Lopes, H., Breitman, K.: Uncertainty quantification through the Monte Carlo method in a cloud computing setting. Computer Physics Communications **185**(5), 1355–1363 (2014)
3. Eisenhower, B., O'Neill, Z., Narayanan, S., Fonoberov, V.A., Mezi, I.: A methodology for meta-model based optimization in building energy models. Energy and Buildings **47**, 292–301 (2012)
4. Herman, J., Usher, W.: SALib: An open-source Python library for sensitivity analysis. The Journal of Open Source Software **2**(9) (January 2017), https://doi.org/10.21105/joss.00097
5. Iman, R.L., Helton, J.C., Campbell, J.E.: An approach to sensitivity analysis of computer models: Part i – introduction, input variable selection and preliminary variable assessment. Journal of Quality Technology **13**(3), 174–183 (1981)
6. Iooss, B., Lemaître, P.: A review on global sensitivity analysis methods. In: Dellino, G., Meloni, C. (eds.) Uncertainty Management in Simulation-Optimization of Complex Systems: Algorithms and Applications, pp. 101–122. Springer US, Boston, MA (2015)
7. 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)
8. McKay, M.D., Beckman, R.J., Conover, W.J.: Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics **21**(2), 239–245 (1979)
9. Morris, M.D.: Factorial sampling plans for preliminary computational experiments. Technometrics **33**(2), 161–174 (1991)
10. 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)
11. Rausch, M.: Determining Cost-Effective Intrusion Detection Approaches for an Advanced Metering Infrastructure Deployment Using ADVISE. Master's thesis, University of Illinois at Urbana-Champaign (2016)
12. Rausch, M., Fawaz, A., Keefe, K., Sanders, W.H.: Modeling humans: A general agent model for the evaluation of security. In: Proc. International Conference on Quantitative Evaluation of Systems. pp. 373–388. Springer (2018)
13. Rausch, M., Feddersen, B., Keefe, K., Sanders, W.H.: A comparison of different intrusion detection approaches in an advanced metering infrastructure network using ADVISE. In: Agha, G., Van Houdt, B. (eds.) Quantitative Evaluation of Systems: 13th International Conference, QEST 2016, Quebec City, QC, Canada, August 23-25, 2016, Proceedings, pp. 279–294. Springer International Publishing, Cham (2016)
14. Razmjoo, A., Xanthopoulos, P., Zheng, Q.P.: Online feature importance ranking based on sensitivity analysis. Expert Systems with Applications **85**, 397–406 (2017)
15. 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
16. 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)
17. Sobol, I.: On the distribution of points in a cube and the approximate evaluation of integrals. USSR Computational Mathematics and Mathematical Physics **7**(4), 86–112 (1967)
18. Sobol, I.: Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates. Mathematics and Computers in Simulation **55**(1), 271–280 (2001)
19. Tenne, Y.: An optimization algorithm employing multiple metamodels and optimizers. International Journal of Automation and Computing **10**(3), 227–241 (2013)
20. 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)
21. Wolpert, D.H.: Stacked generalization. Neural Networks **5**(2), 241–259 (1992)
22. Xiao, N.C., Zuo, M.J., Zhou, C.: A new adaptive sequential sampling method to construct surrogate models for efficient reliability analysis. Reliability Engineering & System Safety **169**, 330–338 (2018)
23. 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)