Stacked Metamodels for Sensitivity Analysis and Uncertainty Quantification of AMI Models

Michael Rausch
Department of Computer Science
University of Illinois at Urbana-Champaign
mjrausc2@illinois.edu

William H. Sanders
Department of Computer Engineering
Carnegie Mellon University
sanders@cmu.edu

***Abstract*—Models can help architects design effective and secure advanced metering infrastructure (AMI) deployments. Because of the complex interactions among the numerous smart meters, smart home devices, customers, the utility, and potential adversaries, the models are often complex and have long execution times. In addition, the models often contain a large number of uncertain input variables.**

Modelers seek to understand the impact of uncertain input variables on the model through the use of sensitivity analysis (SA) and uncertainty quantification (UQ). However, long-running models are not amenable to such techniques, since they require that the model be run many times. One approach to help overcome this challenge is to build a metamodel (a model of the model) that accurately emulates the original model but is much faster.

In this paper, we explain an approach we developed to do fast and thorough SA and UQ using a specially designed metamodel of stacked regressors that can be used to analyze AMI models. We demonstrated the approach by applying it to a complex AMI security model. We show that our metamodel is substantially faster than the base AMI model, more accurate than other existing metamodel approaches, and amenable to SA and UQ.

I. Introduction

Advanced metering infrastructure (AMI) deployments can give a utility increased oversight and control of the electric grid by adding cyber capabilities to smart meters. Unfortunately, such deployments often have a larger attack surface than traditional deployments and can be easier to attack remotely. The effects of a successful attack on the electric grid could be devastating to society. Therefore, AMI deployments must be carefully designed with a focus on security and resilience.

Modeling is an absolutely critical activity in the design process. Modeling helps experts in different domains (e.g. power, security, business) to collaborate and create a shared vision of the design. Models make assumptions explicit, codify knowledge, and help architects make informed choices among different proposed designs. Though fundamental to the design process, AMI models are threatened by two significant practical issues: long run times caused by the size and complexity of the model, and uncertain input variables.

AMI models are often large and take a long time to solve because they must take into account the complexity of the grid (which frequently contains relatively novel cyber components) and the interplay between the grid and the various human parties that interact with it, including the operators at the utility, the customers, and potential adversaries. Aspects of the behavior of the adversaries, customers, and any novel grid components may be impossible to forecast precisely, but must be modeled, which introduces uncertainty. The uncertainty can manifest in uncertain input variables, whose values are not precisely known.

Traditionally, sensitivity analysis (SA) and uncertainty quantification (UQ) techniques have been used by modelers to understand the uncertainty in the model inputs and its effect on the output. These techniques rely on running (or solving) the model many times with different combinations of input values. However, as previously mentioned, AMI models often take such a long time to run that traditional SA and UQ techniques cannot be applied directly to the model.

Metamodels offer a way to overcome that issue. A metamodel is a model that emulates the behavior of the base model (a model of a model). The metamodel trades some accuracy for increased speed, which makes it possible to use SA and UQ techniques that would be impractically slow on the base model. To be useful, the metamodel (a) must be easy to construct, preferably automatically; (b) must run significantly faster than the base model; and (c) must emulate the base model as accurately as possible.

In this work we show an approach we developed to perform fast and thorough SA and UQ through the use of a specially built metamodel composed of stacked regressors, and demonstrate through the use of a test case how it can be applied to large AMI models. We show that the metamodel is significantly faster than the base model, more accurate than other metamodeling techniques, and amenable to SA and UQ. The approach shown in this work can help system architects to (a) validate models, (b) extract more useful information from models by exploring the design space faster and more thoroughly, and (c) make more informed design decisions.

The remainder of this work is organized as follows. We shall briefly review the related work in Section II. Then we shall describe in Section III the AMI model we use as a test case, summarize in Section IV the metamodeling approach, and explain in Section V how we applied the approach to the test case model. We shall then present in Section VI the results we achieved by applying the methods to the test case model. Finally, we conclude in Section VII with a discussion of the limitations, use cases, and future directions of the approach.

II. Background and Related Work

We believe that we are the first to use machine learning techniques to automatically construct fast and accurate metamodels of AMI models for sensitivity analysis and uncertainty quantification. We utilize the stacked regressor metamodel approach described in [1] to build our metamodels. We build the metamodels to emulate the AMI security model found in [2], [3] so that we may indirectly conduct SA and UQ.

Relatively simple metamodels have previously been used in the energy domain. In [4], a Gaussian process regressor metamodel was used to perform load forecasting in a power system, while [5] uses linear emulators for parameter estimation and performance prediction of building energy models, and [6] discusses use of a Support Vector Machine (SVM) metamodel to optimize a building energy model with more than 1000 parameters. Those efforts differ from ours in (a) their choices of metamodel, and (b) the purposes for which the metamodels are used. Most other researchers choose to use a single regressor for the metamodel, while we demonstrate the superiority of an ensemble technique in this work. For surveys of metamodel-based approaches outside the energy domain, see [7], [8].

Sensitivity analysis is often employed in the study of models of smart grids and AMI. Examples include [9] [10] [11] [12]. In that work, sensitivity analysis was applied directly to the model or system, while in our work we indirectly perform the sensitivity analysis through the use of a metamodel. Consult [13] for a recent survey of academic work on measurement uncertainty in the energy domain, and [14] for a general review of global sensitivity analysis techniques.

III. The ADVISE AMI IDS Test Case

The test case we use to demonstrate the approach is built around a model constructed using the ADVISE formalism [15] to compare the effectiveness of different intrusion detection systems (IDSes) in an advanced metering infrastructure (AMI) deployment. The ADVISE model is composed of an adversary profile and an Attack Execution Graph (AEG) (see Figure 1), which contains a number of attack steps that the adversary may chain together to perform a full attack on the system, and the system variables that serve as preconditions and postconditions of the attack step. The model's output metric is the monetary damage done to the utility as a result of the adversary's attack, given the presence of a particular IDS in the system. A system architect can use the model to help him or her make an informed choice among the available IDS options. The description of the model was first published in [2], while [3] provides a more in-depth look at the model. Please see those publications for full details.

We consider 18 model input variables in our analysis. The variables are listed in Table I. The first input variable determines the adversary type: the adversary may be a disgruntled insider employee, a customer who attempts to steal power, a stealthy and well-funded nation-state, or a publicity-seeking terrorist organization. The second input variable indicates the IDS present in the AMI: either no IDS is present, or a

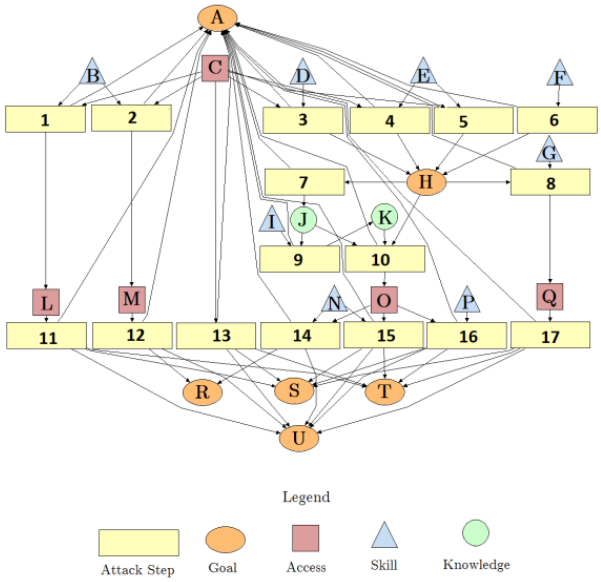

Fig. 1. Attack Execution Graph of the ADVISE AMI IDS test case model.

TABLE I
List of inputs used in the ADVISE AMI IDS case study.

Variable Name	Domain
Adversary	{Insider, Cstmr., Nat. St., Terrorist}
IDS	{None, Central, Dedicated, Embedded}
IDS Multiplier	[0.0005, 0.5]
Payoff Scalers (5 total)	[0.2, 2]
Cost Scalers (10 total)	[0.2, 2]

centralized, dedicated, or embedded IDS is present. An IDS can lessen the probability that an attacker will successfully complete a particular attack step in the model, or increase the cost of attempting that attack step. The third input variable determines by how much the probability of successfully completing the attack step is reduced by the IDS. The value is not known precisely, so we consider a range of values in our evaluation. The values of the next five input variables scale the payoff of each of the adversary's five goals. We do not know precisely how much an adversary values achieving a particular goal, so in our evaluation we establish a baseline value for each goal, and each of these values may be scaled by its respective payoff scaler. Similarly, we use ten cost scalers in the evaluation to scale the baseline estimated cost of attempting an attack step. The baseline values and value ranges for all the input variables were drawn from [3].

In this analysis, we are primarily interested in answering two questions. First, we would like to use sensitivity analysis to rank the contribution of each input variable to the output uncertainty. Armed with that knowledge, a modeler can focus his or her limited time and resources on reducing the uncertainty of the most impactful input variables. Second, we would like to know the degree to which the system architect's choice of IDS is sensitive to the uncertainty in the IDS effectiveness multiplier, the particular adversary goal payoff estimates, and

the attack step cost estimates in the model. SA and UQ will help us answer those questions.

IV. Approach

The AMI model takes a relatively long time to execute once, and SA and UQ require many model executions. Metamodeling is a promising approach for quickly and indirectly conducting both SA and UQ. In the context of this work, a *metamodel* (also known elsewhere as a *model surrogate* or *emulator*) is a model of a model. We call the model that the metamodel emulates the *base model*. The metamodel is designed, given a particular input, to attempt to produce an output that is the same as the output that the base model would have produced had it been given the same input. In practice, it is rare that a metamodel would be able to perfectly emulate the base model. Usually a metamodel produces an output that differs as little as possible from the output that would be produced by the base model given the same input. An *input* is defined to be a vector consisting of one value for every input variable in the model. A metamodel can be constructed by hand, but we use machine learning (ML) techniques to do it automatically, as ML is usually faster, easier, and more generalizable.

In general, ML approaches to metamodeling have four main stages: (a) data acquisition for training and testing, (b) model construction, (c) evaluation of the metamodel's accuracy and performance, and (d) use of the metamodel in place of the base model (in our case, for sensitivity analysis and uncertainty quantification). We shall devote the rest of this section to exploring each stage in turn.

A. Acquisition of Training and Testing Data

ML approaches require training data to learn, and testing data to determine accuracy. We gather data for training and testing by running the base model with many different inputs and recording the resulting outputs. There exist a number of strategies for sampling the input space, and each has different strengths. Options include Latin hypercube sampling (LHS) [16] [17], sampling based on Sobol sequences [18], adaptive sampling [8], and random sampling. Though [1] suggests that training data collected through Sobol sequence sampling may produce slightly more accurate metamodels than random sampling or LHS sampling, we chose to use uniform random sampling in this work for its simplicity, well-understood properties, and ease of use. In future work, we will compare the effectiveness of various sampling strategies.

B. Building the Metamodel

A metamodel may be built using the training data through the use of machine learning. Different ML approaches for metamodel construction exist. In this work, we compare two sophisticated approaches with each other and with a simpler approach for testing. The three metamodel types we consider are the *Naive* metamodel, the *Best of Many* metamodel, and the *Stacked* metamodel. We shall briefly describe each in turn.

Naive Metamodel: The first metamodeling approach is extremely simple and is used solely to benchmark the other approaches. The Naive metamodel approach takes the training data and calculates the average of the outputs. The metamodel then uses this average as its output, regardless of the input it receives. The accuracy of the Naive metamodel sets a low threshold that all of the other metamodels should convincingly surpass to be considered viable.

Best of Many Metamodel: We call the next metamodeling approach that we consider the *Best of Many* approach. It is motivated by (a) the existence of many different types of regressors, (b) the fact that many regressors have hyperparameters that must be tuned, and (c) the difficulty of knowing which regressor and hyperparameter settings would produce the most accurate result given a particular training set. The Best of Many approach involves taking a number of different types of regressors (with different hyperparameter settings), training each with the training data, and then selecting the most accurate regressor as the metamodel. The Best of Many approach is the current state-of-practice found in most of the related work. We strive to surpass its performance with our Stacked ensemble approach. We use twenty-five regressors as the committee, and select the most accurate. The regressors include:

- 1 random forest (RF) regressor,
- 7 different multilayer perceptrons (MLPs) (each with a different combination of solver and activation function),
- 4 different gradient boosting regressors, each with a different loss function,
- 1 Ridge regressor,
- 10 different k-nearest neighbors (KNN) regressors, with the number of neighbors drawn from the set {1,2,4,8,16} and the weighting drawn from the set {uniform, distance},
- 1 Gaussian process (kriging) regressor, and
- 1 stochastic gradient descent regressor.

Stacked Metamodel: The third type of metamodel construction process we consider is the *Stacked* metamodel. We shall briefly summarize the approach here; for details, please consult [1]. The Stacked metamodel approach is motivated by the intuition that a committee of regressors working together may perform better than the strongest single regressor from the committee working alone (the Best of Many approach described above), since the regressors will have different strengths and weaknesses. As an example, Regressor *A* may be more accurate in one region of the input space, and Regressor *B* may be more accurate in another region.

Taking the average of the committee outputs as the final output of the committee is perhaps the simplest and most straightforward way to use multiple regressors together. However, if a simple average is taken, poorly performing ensemble members will have the same "vote" or contribution to the final output as the highly performing ensemble members. The approach could be strengthened if the contributions of the highly performing members were weighted more highly

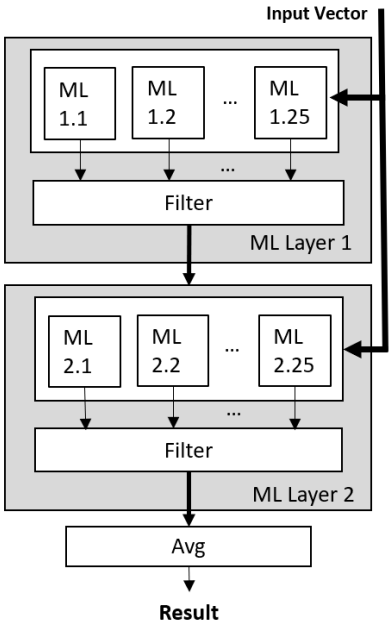

Fig. 2. Illustration of the Stacked metamodel approach.

than those of the lower-performing members. A natural way to accomplish such weighting is to use ML to learn the appropriate weighting.

Stacking is a ML technique that is particularly well-suited for the task, and has helped competitors build ensembles that win ML competitions [19]. Stacking involves obtaining the predictions from a committee of trained regressors, adding them to the original training data, and using the augmented dataset to train another regressor or committee of regressors. The outputs/predictions from the first-level committee helps inform the second-level regressor or committee.

Following [1], we use the same 25 regressors that form the Best of Many committee described earlier as both the first-level committee and the second-level committee. We filter especially poorly performing regressors from each committee before forwarding its results to the next level. We average the results of the second-level committee to obtain the final metamodel output. The Stacked metamodel approach we employ is illustrated in Figure 2.

C. Evaluating Metamodel Accuracy

Once the metamodel has been trained, we can use the test data collected in the first stage to evaluate the accuracy of a metamodel. We use the test inputs to obtain the predictions of the metamodel, and compare the predicted values with the actual outputs produced by the base model. The absolute value of the difference between the metamodel output and the base model output is the *error* of the metamodel. The average absolute error helps the modeler determine whether the metamodel is accurate enough to serve as an effective model surrogate.

D. Using the Metamodel to Perform Sensitivity Analysis

Sensitivity analysis and uncertainty quantification techniques may be applied directly to the metamodel in the usual way once the modeler has made the determination that the metamodel is an acceptable surrogate for the base model. Numerous methods for performing sensitivity analysis exist [14]. Prominent examples include the Morris method [20] [21], the Feature Importance method [14] [22], and Sobol sensitivity analysis [23]. In this work, we chose to use the Sobol SA method, as it is a commonly used and powerful method for performing global sensitivity analysis. We employ a Monte Carlo approach for uncertainty quantification adapted from [24].

V. Application: One-Hot Encoding vs. Splitting

The Stacked metamodel approach described in [1] implicitly assumes that all the input variables will have quantitative values. However, the AMI model in our test case has two qualitative (categorical) inputs: the IDS type and the adversary type. We had to carefully consider how to use the qualitative inputs, which required us to adapt the approach. We chose to evaluate two methods.

The first, the *one-hot encoding* approach, was inspired by a technique commonly used in ML applications for representing categorical data. With one-hot encoding, the categorical input is removed, and a new binary variable for each category value is added. Thus, using the test case model as an example, the input variable representing the IDS type (which can take one of four values: *None*, *Centralized*, *Dedicated*, or *Embedded*) is removed and replaced with four new binary input variables, one for each value the old input could take. Only one of the four newly-introduced binary input variables will be true at any one time. The metamodel is trained once the dataset has been processed by the one-hot encoding technique.

The second approach, which we call the *split* approach, is to split the training dataset such that there is a separate partition for each possible combination of qualitative values, and then to train a metamodel for each separate partition. With the split approach, the training data from the AMI test case model we consider would be split into 16 partitions, because there are two qualitative inputs that can each take one of four values $IDS \times ADV$, where $IDS = \{None, Centralized, Dedicated, Embedded\}$ and $ADV = \{Customer, Insider, NationState, Terrorist\}$. Then a new metamodel would be trained for each dataset, for a total of 16 different metamodels. During testing or use, inputs would be similarly divided and processed by the corresponding metamodel.

The two approaches have different strengths. The one-hot encoding stacked approach uses all of the available data to train the metamodel, while the split-stacked approach can use only a fraction of the data (about one-sixteenth of the data for the AMI test case model) to train any one of the split metamodels. The ability to utilize all of the data is a strength for the one-hot encoding stacked approach, especially

in situations like ours in which it is difficult and time-consuming to obtain additional training data. On the other hand, training a separate metamodel for each data partition may simplify the regression problem. Some machine learning techniques may perform better if trained on a smaller dataset that contains no categorical data. It is not clear *a priori* which of the two approaches will produce the most accurate results, so we compare them in Section VI.

VI. Analysis

To be effective, a metamodel must (a) accurately emulate the behavior of the base model, (b) be significantly faster than the base model, and (c) be amenable to SA and UQ techniques. In this section, we will show that the metamodel we have constructed has these properties. We used the scikit-learn Python package [25] to build the regressors, and the SALib Python package to perform Sobol sensitivity analysis [26]. All experiments were performed on a computer with an Intel i7-5829K processor and 32 GB of RAM.

A. Accuracy

Recall that each metamodel is trying to predict, as closely as possible, what the base model would output given the same input. The base model, in turn, gives a forecast of the monetary damage a particular adversary would inflict on an AMI protected by a particular kind of IDS. The *average error* in the metamodel represents the average difference between the metamodel's prediction and the base model's actual output. We had a test set consisting of 1000 randomly generated inputs (in the range given in Table I) and the corresponding outputs from the base model. The smallest value among the outputs was 0, and the largest was 11,608,170. Table II shows the average error of the different metamodels we trained. The three metamodel types we trained were the Naive metamodel, the Best of Many metamodel, and the Stacked metamodel. We had two versions each of the Best of Many and Stacked metamodels: one trained with the dataset processed by one-hot encoding, and one trained with the split dataset, as described in Section V. We trained three metamodels of each type with datasets containing 250, 1000, and 4000 samples, respectively.

As expected, the Naive metamodels perform poorly (all of the other, more sophisticated metamodels are substantially more accurate) and demonstrate little improvement with additional training samples. While the Stacked metamodels trained with 250 samples perform about the same as the Best of Many metamodels trained with 250 samples, the other Stacked metamodels are more accurate than their corresponding Best of Many metamodels. That result suggests that the ensembles composed of stacked regressors are no worse, and, if given enough training samples, are substantially better than the best of the collection of regressors by itself, demonstrating the utility of the stacking/filtering approach given in [1]. Finally, the splitting approach yielded substantially more accurate metamodels than one-hot encoding for our AMI model. All of the split stacked metamodels had an average error that was less than 1% of the observed range of outputs (previously found to be 11,608,170), which should be sufficiently accurate for our sensitivity analysis and uncertainty quantification.

B. Speed

We rounded all reported times to the nearest second. We ran the base model one thousand times with random inputs, which took 6733 seconds (a little under 2 hours). We also ran the split stacked metamodel one thousand times with the same random inputs, and that took a total of 38 seconds. The metamodel is thus more than 100 times faster than the base model. Training of the split stacked metamodel with a training dataset consisting of 4000 random samples took 14 minutes and 31 seconds. The code was not parallelized, but both the metamodel training and the execution could easily be parallelized for additional gains in speed. For example, each regressor at a particular level can be trained independently of the others at the same level, and each regressor at a particular level can be executed to obtain its prediction independently of the others at the same level.

C. Sensitivity Analysis

We conducted a Sobol sensitivity analysis in an effort to determine how much the uncertainty in each input variable contributes to the uncertainty of the output. We performed the sensitivity analysis on both the base model and the split stacked metamodel trained with the dataset containing 4000 random samples. The Sobol sensitivity analysis used 48000 samples. It took over 90 hours to complete on the base model, and approximately 30 minutes to complete on the trained metamodel.

The results of the sensitivity analysis may be found in Table III. The table shows the total order index calculated by the Sobol sensitivity analysis for each input variable. That index measures the contribution of the uncertainty of the input variable on the output. We also show the ranking of the importance of each input variable from most sensitive to least as measured by the total order index. We show the total order index and ranking for both the metamodel and the base model. We also show the *rank error*: the absolute value of the difference in SA ranking between the base model and the metamodel. Only one input variable (*CostScalar6*) has a rank error of more than 2. On average the metamodel's ranking differed from the base model's ranking by only 1.25 places. The metamodel also correctly determined the three most sensitive and three least sensitive input variables.

The results show that the metamodel is a very good surrogate for the base model, as (a) it is much faster, and (b) a sensitivity analysis applied to the metamodel produces results that are very similar to the results obtained by a sensitivity analysis applied directly to the base model.

A modeler may use the rankings provided by the SA to focus his or her attention on reducing the uncertainty of the most sensitive input variables (Cost Scalars 1, 4, and 9) while spending less time on the least sensitive input variables (the IDSMultiplier, and Cost Scalars 2 and 3).

TABLE II
AVERAGE PREDICTION ERROR GIVEN TRAINING DATA AND REGRESSOR TYPE.

Training Data Num. Samples	Naive Metamodel	One-Hot Encoding Best of Many Metamodel	Split Best of Many	One-Hot Encoding Stacked Metamodel	Split Stacked Metamodel
250	1,594,770	243,531	71,798	249,163	69577
1,000	1,369,770	187,585	60,614	113,093	46,364
4,000	1,379,670	108,962	60,121	79,006	45,675

Input Name	Metamodel Sobol SA Method		Base Model Sobol SA Method		Rank Error
	Total Ord. Ind.	Rank	Total Ord. Ind.	Rank	
CostScalar1	1.069	1	1.221	1	0
CostScalar4	1.055	2	1.189	2	0
CostScalar9	1.047	3	1.180	3	0
PayoffScalar5	0.949	5	1.174	4	1
CostScalar5	0.937	6	1.163	5	1
PayoffScalar1	0.913	7	1.149	6	1
PayoffScalar3	0.910	8	1.140	7	1
CostScalar7	0.874	10	1.102	8	2
PayoffScalar2	0.848	11	1.083	9	2
CostScalar10	0.890	9	1.075	10	1
PayoffScalar4	0.841	12	1.036	11	1
CostScalar8	0.838	13	1.033	12	1
CostScalar6	1.030	4	1.0294	13	9
IDSMultiplier	0.672	14	0.6933	14	0
CostScalar2	0.563	15	0.609	15	0
CostScalar3	0.242	16	0.332	16	0

TABLE III
AMI SENSITIVITY ANALYSIS.

D. Uncertainty Quantification

Recall that we are interested in determining whether the choice of IDS is sensitive to the uncertainty in the model input variables. To help answer this question, we ran 80,000 random samples through the metamodel (20,000 for each IDS option) to thoroughly explore the input space. The results of our uncertainty quantification analysis are summarized in Table IV, which shows the average, 95th percentile, and 99th percentile damage calculated by the metamodel for each IDS option. The results confirm that any IDS provides substantially better protection than none, even when the uncertainty in the input variables is factored in. The modeler can use these results to help select the correct IDS for an AMI, given the modeler's risk tolerance and the costs of acquisition and maintenance.

VII. CONCLUSION

In this work, we demonstrated a metamodeling-based approach for performing fast and thorough SA and UQ on an AMI model. We evaluated two ways to adapt the metamodel approach [1] to handle a mix of qualitative and quantitative inputs and found that splitting produced more accurate metamodels than one-hot encoding did. We also confirmed that the stacked metamodel approach is significantly more accurate than the Best-of-Many approach. Most importantly, we found that a sensitivity analysis conducted on the trained metamodel yielded results similar to those of a sensitivity analysis conducted on the base model, but was more than one hundred times faster. Further, we found that thorough uncertainty quantification can be quickly conducted on the metamodel to help a modeler make design choices.

IDS Type	Average	95%	99%
None	$3,076,859	$11,608,170	$11,608,169
Centralized	$447,718	$3,036,137	$3,040,767
Distributed	$255,162	$1,019,546	$1,019,546
Embedded	$325,785	$1,019,546	$1,019,546

TABLE IV
ESTIMATED MONETARY DAMAGE GIVEN IDS.

A. Limitations

The metamodel-based approach described in this paper works best when certain conditions are met. If the base model runs very quickly, it is preferable to apply the sensitivity analysis and uncertainty quantification techniques to the model directly, since metamodels rarely perfectly emulate the base model. On the other hand, if the base model runs very slowly, it may be impossible to obtain enough training and testing samples to construct and evaluate the metamodel. If that is the case, the modeler should consider either building the metamodel by hand or modifying the base model so that it runs more quickly. It would be prudent for a modeler to consider such runtime issues before attempting to apply our technique to their own models. Another limitation is that if the structure of the base model changes, the metamodel will need to be retrained with new data collected from the changed base model. Finally, our approach only addresses uncertainty in the input variables, not uncertainty involving the model's structure.

B. Uses

We believe that the stacked metamodel approach outlined in this paper can help modelers quickly and thoroughly explore, evaluate, and validate their models. The ability to view and study complex, high-fidelity, and long-running AMI models using this approach should help system architects make better decisions during the design stage, which will in turn lead to

more secure, safer, cheaper, and more effective AMI deployments. We see this approach primarily as a tool that can be used by system architects during the design phase.

However, the speed of the metamodel approach opens intriguing possibilities for its use as a security aid during the operational lifetime of the AMI deployment. Traditional complex high-fidelity AMI security models often run too slowly to be much use in responding to security incidents as they happen. Metamodels, on the other hand, could be quickly queried to help intrusion-prevention systems or human operators make informed security decisions in near real-time. Information from sensors throughout the network could be fed into the metamodel as inputs. Using that constantly updated source of information, the metamodel could predict the adversary's next steps and recommend appropriate responses.

C. Future Directions

The most time-consuming part of the metamodel-based stacking approach is collection of the data needed to build and test the regressors. In the future, we would like to explore whether adaptive sampling approaches (e.g., those presented in [8]) could help in creating better training datasets with fewer samples than are required by the non-adaptive sampling approaches explored in [1]. Adaptive sampling techniques could ultimately save the modeler time and increase accuracy by focusing exploration of the input space on high-impact, poorly explored regions. We would also like to investigate how changing the size and heterogeneity of the committee impact the accuracy of the metamodel. Finally, we would like to parallelize our code to realize even greater speed gains in both metamodel training and use.

References

[1] M. Rausch and W. H. Sanders, "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*. Springer, 2020, to appear.

[2] M. Rausch, B. Feddersen, K. Keefe, and W. H. Sanders, "A comparison of different intrusion detection approaches in an advanced metering infrastructure network using ADVISE," in *Quantitative Evaluation of Systems: 13th International Conference, QEST 2016, Quebec City, QC, Canada, August 23-25, 2016, Proceedings*, G. Agha and B. Van Houdt, Eds. Cham: Springer International Publishing, 2016, pp. 279–294.

[3] M. Rausch, "Determining cost-effective intrusion detection approaches for an advanced metering infrastructure deployment using ADVISE," Master's thesis, University of Illinois at Urbana-Champaign, 2016.

[4] G. Xie, "Robust and data-efficient metamodel-based approaches for online analysis of time-dependent systems," Ph.D. dissertation, Virginia Tech, 2020.

[5] Q. Li, G. Augenbroe, and J. Brown, "Assessment of linear emulators in lightweight Bayesian calibration of dynamic building energy models for parameter estimation and performance prediction," *Energy and Buildings*, vol. 124, pp. 194–202, 2016.

[6] B. Eisenhower, Z. O'Neill, S. Narayanan, V. A. Fonoberov, and I. Mezi, "A methodology for meta-model based optimization in building energy models," *Energy and Buildings*, vol. 47, pp. 292 – 301, 2012.

[7] F. Viana, C. Gogu, and R. Haftka, "Making the most out of surrogate models: Tricks of the trade," *Proceedings of the ASME Design Engineering Technical Conference*, vol. 1, pp. 587–598, 2010.

[8] H. Liu, Y.-S. Ong, and J. Cai, "A survey of adaptive sampling for global metamodeling in support of simulation-based complex engineering design," *Structural and Multidisciplinary Optimization*, vol. 57, no. 1, pp. 393–416, 2018.

[9] A. A. Cárdenas, R. Berthier, R. B. Bobba, J. H. Huh, J. G. Jetcheva, D. Grochocki, and W. H. Sanders, "A framework for evaluating intrusion detection architectures in advanced metering infrastructures," *IEEE Transactions on Smart Grid*, vol. 5, no. 2, pp. 906–915, 2014.

[10] D.-H. Choi and L. Xie, "A framework for sensitivity analysis of data errors on home energy management system," *Energy*, vol. 117, pp. 166–175, 2016.

[11] S. Bhattacharjee, A. Thakur, S. Silvestri, and S. K. Das, "Statistical security incident forensics against data falsification in smart grid advanced metering infrastructure," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, 2017, pp. 35–45.

[12] G. Barai and K. Raahemifar, "Optimization of distributed communication architectures in advanced metering infrastructure of smart grid," in *Proc. 2014 IEEE 27th Canadian Conference on Electrical and Computer Engineering (CCECE)*, 2014, pp. 1–6.

[13] H. Carstens, X. Xia, and S. Yadavalli, "Measurement uncertainty in energy monitoring: Present state of the art," *Renewable and Sustainable Energy Reviews*, vol. 82, pp. 2791–2805, 2018.

[14] B. Iooss and P. Lemaître, *A Review on Global Sensitivity Analysis Methods*. Boston, MA: Springer US, 2015.

[15] E. LeMay, "Adversary-driven state-based system security evaluation," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2011.

[16] M. D. McKay, R. J. Beckman, and W. J. Conover, "Comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 21, no. 2, pp. 239–245, 1979.

[17] R. L. Iman, J. C. Helton, and J. E. Campbell, "An approach to sensitivity analysis of computer models: Part I–Introduction, input variable selection and preliminary variable assessment," *Journal of Quality Technology*, vol. 13, no. 3, pp. 174–183, 1981.

[18] I. Sobol, "On the distribution of points in a cube and the approximate evaluation of integrals," *USSR Computational Mathematics and Mathematical Physics*, vol. 7, no. 4, pp. 86–112, 1967.

[19] M. Risdal, "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.

[20] M. D. Morris, "Factorial sampling plans for preliminary computational experiments," *Technometrics*, vol. 33, no. 2, pp. 161–174, 1991.

[21] "An effective screening design for sensitivity analysis of large models," *Environmental Modelling and Software*, vol. 22, no. 10, pp. 1509 – 1518, 2007.

[22] A. Razmjoo, P. Xanthopoulos, and Q. P. Zheng, "Online feature importance ranking based on sensitivity analysis," *Expert Systems with Applications*, vol. 85, pp. 397–406, 2017.

[23] I. Sobol, "Global sensitivity indices for nonlinear mathematical models and their Monte Carlo estimates," *Mathematics and Computers in Simulation*, vol. 55, no. 1, pp. 271–280, 2001.

[24] A. Cunha, R. Nasser, R. Sampaio, H. Lopes, and K. Breitman, "Uncertainty quantification through the Monte Carlo method in a cloud computing setting," *Computer Physics Communications*, vol. 185, no. 5, pp. 1355–1363, 2014.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[26] J. Herman and W. Usher, "SALib: An open-source Python library for sensitivity analysis," *The Journal of Open Source Software*, vol. 2, no. 9, 2017. [Online]. Available: https://doi.org/10.21105/joss.00097