
QUANTIFYING AND ATTRIBUTING SUBMODEL UNCERTAINTY IN STOCHASTIC SIMULATION MODELS AND DIGITAL TWINS

Mohammadmahdi Ghasemloo

Department of Industrial and Systems Engineering
Texas A&M University
College Station, TX, USA
mohammad_ghasemloo@tamu.edu

David J. Eckman

Department of Industrial and Systems Engineering
Texas A&M University
College Station, TX, USA
eckman@tamu.edu

Yaxian Li

Intuit AI
Mountain View, CA, USA
yaxian_li@intuit.com

ABSTRACT

Stochastic simulation is widely used to study complex systems composed of various interconnected subprocesses, such as input processes, routing and control logic, optimization routines, and data-driven decision modules. In practice, these subprocesses may be inherently unknown or too computationally intensive to directly embed in the simulation model. Replacing these elements with estimated or learned approximations introduces a form of epistemic uncertainty that we refer to as *submodel uncertainty*. This paper investigates how submodel uncertainty affects the estimation of system performance metrics. We develop a framework for quantifying submodel uncertainty in stochastic simulation models and extend the framework to digital-twin settings, where simulation experiments are repeatedly conducted with the model initialized from observed system states. Building on approaches from input uncertainty analysis, we leverage bootstrapping and Bayesian model averaging to construct quantile-based confidence or credible intervals for key performance indicators. We propose a tree-based method that decomposes total output variability and attributes uncertainty to individual submodels in the form of importance scores. The proposed framework is model-agnostic and accommodates both parametric and nonparametric submodels under frequentist and Bayesian modeling paradigms. A synthetic numerical experiment and a more realistic digital-twin simulation of a contact center illustrate the importance of understanding how and how much individual submodels contribute to overall uncertainty.

Keywords Uncertainty Quantification, Digital Twin, Stochastic Simulation, AI-driven Simulation

1 Introduction

Many stochastic systems studied in operations research and management science can be viewed as collections of interconnected subprocesses that each represent a specific aspect of system behavior such as demand generation, service dynamics, routing logic, or optimization-based decision making. When such systems are studied using stochastic simulation, the modeler must determine how to model each subprocess. In many cases, the true subprocess cannot be accessed or fully understood and instead can only be observed in the real world, such as when observing customer arrivals in a service system. In other cases, the true subprocess may be accessible, such as when it represents some operational decision made according to some policy or sophisticated solution method, e.g., optimization, but directly embedding the subprocess within the simulation may be impractical due to concerns about privacy, latency, computational cost, or data accessibility. For these reasons, the modeler may, out of necessity or choice, replace certain subprocesses with approximate representations, such as generative models, simplified decision rules, heuristics, or optimization proxies trained on historical data. These approximations, which we henceforth refer to as *submodels*, may take several forms.

In the case of modeling stochastic inputs, such as customer demand or service times, probability distributions are fitted to real-world data. In other cases, a submodel may be a function mapping subsystem inputs to outputs trained on input-output data from that subprocess using supervised learning techniques. In both cases, submodel selection can involve trade-offs between accuracy, interpretability, and computational cost.

The use of submodels within a stochastic simulation model introduces errors that propagate through the model to its outputs. Understanding how these errors propagate is critical for quantifying uncertainty in estimators of key performance indicators (KPIs) and directing efforts to reduce uncertainty. While prior work in the simulation literature has extensively studied *input uncertainty* (IU) [4, 19, 23]—the epistemic uncertainty due to estimating stochastic input models from limited data—less attention has been given to uncertainty originating from other types of submodels that influence the internal dynamics of the simulated system, including the decision logic, event ordering, and system state. The errors introduced by both types of submodels can influence simulation behavior in complex and path-dependent ways. Along these lines, [11] adopt methods from IU to study uncertainty arising from embedding machine learning (ML) surrogates of decision-support systems (DSSs) within simulation models.

This paper builds on the ideas of [11] to introduce a unifying framework that encompasses many potential sources of epistemic uncertainty in stochastic simulation models. We collectively refer to this more general form of epistemic uncertainty as *submodel uncertainty*. This paper presents general-purpose methods for systematically quantifying submodel uncertainty to support and enhance operational decision making. More specifically, we employ bootstrapping and Bayesian model averaging (BMA) to generate plausible submodels that drive a designed simulation experiment. We leverage design of experiments (DOE), specifically stacked Latin hypercube (LH) designs, to more efficiently explore the space of submodel instances when studying systems with multiple submodels. The experiment results are then used to construct quantile-based confidence or credible intervals (CIs) that account for both aleatoric and epistemic uncertainty. We propose a tree-based method that provides importance scores quantifying how the overall uncertainty can be decomposed into aleatoric and epistemic terms and how the epistemic uncertainty can be further attributed to individual submodels. These importance scores enable practitioners to identify the most influential submodels and thereby prioritize further efforts to reduce overall uncertainty by, for example, acquiring additional training data or refining their modeling specifications. The framework’s strength is its generality, as it can accommodate both parametric and non-parametric submodels under either the frequentist or Bayesian modeling paradigms. Moreover, whereas some existing approaches for analyzing IU assume that the input models are independent of each other [15], we make no such assumption about the submodels.

Submodel uncertainty is also highly relevant in digital-twin settings, wherein a simulation model is enhanced by integrating real-time or periodically observed system state data to maintain a synchronized virtual representation of the physical system [25, 16, 34]. Digital twins are used for monitoring, forecasting, and evaluating alternative operational strategies in real time. Recent work has emphasized the central role of uncertainty quantification (UQ) for reliable prediction and decision support. For example, [37] develop a non-intrusive sensitivity-metrics toolbox for integrating black-box digital twins into the design process in the presence of uncertain data and evolving performance indicators.

In digital-twin settings, simulation models are repeatedly initialized from observed system states and commonly rely on submodels—such as forecasting tools, routing models, or optimization proxies—to emulate future system behavior. For more discussion on the increasing use of ML submodels within digital twins, see [35]. We extend our framework to the digital-twin setting by accommodating cases in which the system state evolves over time and the submodels may be time dependent. In particular, we apply our tree-based method on each state and aggregate the importance scores across states to quantify and compare the relative contributions of different submodels to uncertainty over time. We also propose a method for estimating a form of state-average bias when data on historical system states and KPIs are available. In summary, our approach provides a unified and scalable framework for uncertainty quantification and decomposition that supports reliable decision making in offline simulation and online digital-twin applications.

The remainder of the paper is organized as follows. Section 2 formally defines submodels in simulated systems and categorizes them based on their functional forms. Section 3 mathematically introduces submodel uncertainty. Sections 4 and 5 describe resampling-based methods for UQ and tree-based methods that attribute uncertainty to individual submodels along with a numerical example. Submodel uncertainty is extended to the digital-twin setting in Section 6 and demonstrated through a contact-center example. Concluding remarks and future research directions are given in Section 7.

2 Submodels in Simulated Systems

A stochastic simulation model can be viewed as a mapping g that takes as input a collection of user-specified variables \mathbf{X} and random primitives ξ and returns a response \mathbf{Y} , i.e.,

$$\mathbf{Y} = g(\mathbf{X}, \xi). \quad (1)$$

Simulation models typically consist of multiple interconnected subprocesses, each of which emulates a specific aspect of system behavior. Various practical limitations often make it infeasible to incorporate the real-world subprocess directly in the simulation model. First, the true subprocess may not be fully accessible. For example, input processes governing customer arrivals to a service system are inherently unavailable. Second, even when the true subprocess is accessible, data intensity and latency constraints often render it infeasible to make repeated calls to the true subprocess from within the simulation model. Third, in many real-world settings, decision modules, such as those that support routing, dispatch, and allocation decisions, are shared across real-time applications. Directly invoking these modules within a simulation experiment may therefore interfere with live operations, introduce contention for computational resources, or expose sensitive information. These issues can generally be resolved by approximating or replacing the real-world subprocesses with submodels. For a more detailed discussion of the motivations for replacing DSSs with ML surrogate models, see [11].

The submodels we investigate in this paper can generally be expressed in the form of (1). We categorize submodels into three classes depending on whether their mapping g features only \mathbf{X} , only ξ , or both \mathbf{X} and ξ .

2.1 Deterministic Submodels

We call submodels for which $\mathbf{Y} = g(\mathbf{X})$ *deterministic submodels*, because their output is a deterministic function of the input variables. Deterministic submodels do not introduce any aleatoric uncertainty to the simulation. Roughly speaking, deterministic submodels can be thought of as rules-based logic in the simulation that maps the system state \mathbf{X} to a deterministic decision or action \mathbf{Y} . A deterministic submodel g is trained on data consisting of paired observations (\mathbf{X}, \mathbf{Y}) and then embedded within the simulation model. The mapping g may correspond to an optimization proxy [7] that replaces an optimization solver and is invoked within the simulation whenever an instance of the corresponding optimization problem needs to be solved. In this case, \mathbf{X} denotes the set of parameters describing the problem instance and \mathbf{Y} is the resulting optimal solution, which is then used to advance the simulation. Alternatively, g may approximate the actions of a reinforcement learning (RL) [32] or otherwise intelligent agent, where \mathbf{X} is the state of the system and \mathbf{Y} is the associated action. Inverse reinforcement learning (IRL) [2] can be used to learn a reward function from historical state-action observations and subsequently train a policy. The submodel g might then be the table in tabular learning or be parameterized by a deep neural network [36] that determines the optimal policy. The DSSs studied in [11] are examples of deterministic submodels.

2.2 Unconditional Stochastic Submodels

We refer to submodels that inject aleatoric uncertainty into the simulation model and are not influenced by external factors as *unconditional stochastic submodels*. These submodels are of the form $\mathbf{Y} = g(\xi)$, where ξ represents random primitives, \mathbf{Y} is a random quantity, and g can be parametric or nonparametric. Training data for this class of submodels consist solely of observations of \mathbf{Y} . Well-known examples of unconditional stochastic submodels arise in random variate generation, where the function g could be an inverse cumulative distribution function that transforms a pseudo-random number $\xi \sim \mathcal{U}(0, 1)$ into a random variate \mathbf{Y} or could represent an acceptance-rejection algorithm given access to a sequence of pseudo-random numbers [9]. In more modern generative modeling approaches, such as neural input modeling (NIM) [6], the mapping g may be parameterized by a neural network, for example, within a variational autoencoder (VAE), while the latent variable ξ is typically assumed to follow a standard normal distribution. Alternatively, g may be represented by neural architectures that incorporate recurrent components, such as long short-term memory (LSTM) layers, to capture temporal dependencies in the input process [6].

2.3 Conditional Stochastic Submodels

We refer to submodels of the general form $\mathbf{Y} = g(\mathbf{X}, \xi)$ as *conditional stochastic submodels*. As an example, g may be a generative metamodel (surrogate) of a stochastic simulation model embedded within a larger simulation framework. In this case, \mathbf{X} may include the system state used to execute the submodel, while ξ is generated from the stochastic input models that govern the simulation submodel's aleatoric uncertainty. Building upon a previous example of deterministic submodels, g may instead represent a *stochastic* optimization proxy [1] that replaces a stochastic optimization solver within the simulation model. Here, \mathbf{X} is again the set of parameters describing the problem instance and ξ captures either inherent stochasticity in the optimization algorithm or that emanating from a stochastic submodel (e.g., a simulation submodel) invoked by the solver in the course of its run. Random variate generators that depend on contextual information \mathbf{X} , such as conditional variational autoencoders (CVAEs) [6], also fall into this category.

2.4 Supply Chain Example

We now provide an illustrative example of a supply chain simulation model featuring the three types of submodels. Consider a multi-echelon supply chain in which products must be dynamically allocated from factories to distribution centers and retail locations based on incoming demand information and the current state of inventories and transportation resources. Decisions made at each epoch influence future inventory levels, production schedules, and transportation availability across the network. A stochastic simulation model of this system may contain several submodels. First, customer demand arrivals and their associated order quantities can be represented through unconditional stochastic submodels that generate demand realizations and serve as inputs to the simulation model [30]. Second, allocation decisions regarding where to send each product across the network may be carried out by repeatedly invoking smaller embedded simulation models that evaluate mean completion times [31]. Or, if such simulation submodels are too computationally expensive to embed, they could be replaced with cheaper conditional stochastic submodels, such as CVAEs, that generate synthetic outputs of the simulation submodel. Finally, within each factory, production scheduling and sequencing decisions may be made by RL agents trained to optimize operational performance over time. When the true RL policy is inaccessible or too computationally expensive to deploy online, its behavior can be approximated by a deterministic submodel that learns the reward function using IRL.

3 Submodel Uncertainty

Let $\mathbf{S}^c = (\mathbf{S}_1^c, \mathbf{S}_2^c, \dots, \mathbf{S}_L^c)$ represent a collection of L true subprocesses that govern the real-world system. When hypothetically driving a stochastic simulation model with the true subprocesses, the (random) output on a given replication $j = 1, 2, \dots$ can be expressed as

$$Y_j(\mathbf{S}^c) = \mu(\mathbf{S}^c) + \varepsilon_j(\mathbf{S}^c),$$

where $\mu(\mathbf{S}^c) \equiv \mathbb{E}[Y_j(\mathbf{S}^c)]$ represents the expected output, and $\varepsilon_j(\mathbf{S}^c)$ is a mean-zero random variable capturing the aleatoric uncertainty in the output. In this paper, we study the estimation of a single expected simulation output, hence $Y_j(\mathbf{S}^c)$ is a scalar.

A standard approach to estimating $\mu(\mathbf{S}^c)$ is to obtain estimated submodels $\hat{\mathbf{S}}$ from real-world data, run n independent replications using $\hat{\mathbf{S}}$, and compute

$$\bar{Y}(\hat{\mathbf{S}}) = \frac{1}{n} \sum_{j=1}^n Y_j(\hat{\mathbf{S}}).$$

A confidence interval for $\mu(\mathbf{S}^c)$ can then be constructed as $\bar{Y}(\hat{\mathbf{S}}) \pm t_{\alpha/2, n-1} \cdot s/\sqrt{n}$, where s is the sample standard deviation of $Y_1(\hat{\mathbf{S}}), Y_2(\hat{\mathbf{S}}), \dots, Y_n(\hat{\mathbf{S}})$, and $t_{\alpha/2, n-1}$ is the $1 - \alpha/2$ quantile of a Student's t -distribution with $n - 1$ degrees of freedom. This confidence interval, however, overlooks the epistemic uncertainty from estimating \mathbf{S}^c using $\hat{\mathbf{S}}$ and therefore may fail to cover $\mu(\mathbf{S}^c)$ with the prescribed confidence. This undercoverage can be explained by the observation that replacing \mathbf{S}^c with $\hat{\mathbf{S}}$ can affect both the variance and bias of the estimator $\bar{Y}(\hat{\mathbf{S}})$.

Understanding the variance of $\bar{Y}(\hat{\mathbf{S}})$ is crucial for building an asymptotically valid confidence interval or credible interval for $\mu(\mathbf{S}^c)$. Applying the law of total variance, the variance of $\bar{Y}(\hat{\mathbf{S}})$ —where $\hat{\mathbf{S}}$ is treated as random—can be decomposed as

$$\text{Var}(\bar{Y}(\hat{\mathbf{S}})) = \text{Var}(\mathbb{E}[\bar{Y}(\hat{\mathbf{S}}) | \hat{\mathbf{S}}]) + \mathbb{E}[\text{Var}(\bar{Y}(\hat{\mathbf{S}}) | \hat{\mathbf{S}})] \equiv \sigma_{\text{SU}}^2 + \sigma_{\text{MC}}^2.$$

This decomposition separates the aleatoric uncertainty (i.e., the simulation noise) σ_{MC}^2 , from the epistemic uncertainty introduced by the use of submodels, σ_{SU}^2 . This additive decomposition of the total uncertainty into epistemic and aleatoric uncertainty has been used extensively in the IU literature [4]. We further attribute σ_{SU}^2 across submodels in Section 5.

The presence of estimated submodels also affects the bias of the estimator:

$$\text{Bias} = \mathbb{E}[\mu(\hat{\mathbf{S}})] - \mu(\mathbf{S}^c),$$

where the expectation in the first term is taken over both $\hat{\mathbf{S}}$ and replications of the simulation model. Although bias estimation is not the primary focus of this paper, it nonetheless can play an important role in how well the resulting CIs cover $\mu(\mathbf{S}^c)$. Bias has received less attention in the IU literature, partly because it is difficult to estimate and is expected to vanish faster than the variance as the size of the training data increases asymptotically [21]. In Section 6, we propose a method to estimate a related form of bias when outputs from the real-world system are available.

Our framework for studying the effects of submodel uncertainty on KPI estimation entails obtaining random instances of each submodel and using them to drive the simulation model in a designed experiment. The next two sections elaborate on how to generate these instances, design the experiment, and analyze the results.

4 (Re)Sampling Submodels

In this section, we discuss ways to sample from the space of submodels when they are formulated under the frequentist and Bayesian statistical philosophies.

4.1 Frequentist Submodels and Bootstrapping

In the frequentist paradigm, a submodel is assumed to have a fixed but unknown structure or parameterization. Parametric submodels are typically constructed by first selecting a functional or distributional form based on domain knowledge and then estimating parameters from data. Uncertainty arises from limited data and submodel misspecification and is commonly assessed through resampling-based techniques such as bootstrapping [10]. Bootstrapping is a non-parametric resampling technique that involves resampling observations *with replacement* to create multiple bootstrapped datasets and has been extensively applied to quantify IU [4]. A straightforward way to leverage bootstrapping for studying submodel uncertainty is to resample the training data with replacement and generate plausible instances of the submodel by successively fitting it to the bootstrapped datasets [33]. Although bootstrapping does not impose strong distributional assumptions on the data, it can be challenging to apply in certain contexts, e.g., time series data, which could arise, for example, if the submodel were a forecasting tool. Specialized techniques, such as moving block bootstrap [18] or stationary bootstrap [24], have been developed to address this case, but they can be harder to implement. Bootstrapping also tends to multiply the required computational effort by the number of bootstrap instances. In certain IU settings, bootstrapping is relatively inexpensive because it entails fitting probability distributions to data, which is typically fast. In contrast, when bootstrapping is used to quantify, for instance, a deterministic or conditional stochastic submodel’s uncertainty, the computational burden can be higher because a supervised learning model needs to be trained on each bootstrapped dataset. Complex ML models, such as deep neural networks, often require substantial training time, especially when the data are high dimensional or the model architecture is deep. When using bootstrapping for IU, the dominant cost is often the simulation runs themselves, whereas with submodel uncertainty the cost of repeatedly training submodels may become comparable or even dominant.

4.2 Bayesian Submodels and Bayesian Model Averaging

Bayesian submodels explicitly consider uncertainty by treating submodel parameters as random variables. Prior distributions encode existing knowledge or beliefs and are updated using observed data to produce posterior distributions. Bayesian model averaging offers a principled framework for accounting for uncertainty in the form of a posterior distribution—obtained via Bayes’ rule—which inherently captures the remaining uncertainty about the model after observing the data [8]. This probabilistic formulation yields a collection of plausible submodels when sampling from the posterior.

Bayesian models, e.g., GPs or BNNs, are equipped with posterior distributions on their predictions as functions of inputs for GPs [26] or parameter weights for BNNs [22]. Instead of resampling and retraining the submodel multiple times, as required in bootstrapping, Bayesian model averaging requires simply sampling from the posterior distribution of the model parameters. For example, when working with BNNs, generating new submodels requires only sampling from the posterior distribution rather than performing a full retraining process. On the other hand, sampling from GPs becomes increasingly expensive as the number of evaluation points grows, which in our setting corresponds to the number of times the submodel is invoked within a simulation replication.

5 Uncertainty Quantification

Let $\{\hat{S}_l^{(b)} : b = 1, 2, \dots, B \text{ and } l = 1, 2, \dots, L\}$ represent B instances of each submodel generated via either bootstrapping or BMA. We assume a common B across submodels, but this is not strictly necessary. It can be prohibitively expensive to simulate all B^L possible combinations (henceforth called configurations) of these instances, even for moderate values of B and L . To address this challenge, we turn to design of experiments (DOE) to choose a smaller design of configurations that covers the space of submodel instances. In this paper, we employ Latin Hypercube (LH) designs [29], which ensure that every level of every factor (in our context, every sampled instance of every submodel) appears exactly once in a design. More specifically, a LH design is constructed by sampling *without replacement* from the set of B instances of each submodel to form configurations until all instances have been exhausted. This process can be repeated to create additional LH designs, which can then be stacked to form a larger design [28]. The total number of configurations in such a stacked design is $B' = BS$, where S is the number of stacks. The simulation model is then run n times at each configuration.

The choices of B , S , and n affect the computational cost of the experiment and the accuracy of our estimation of the distribution of $\bar{Y}(\hat{\mathbf{S}})$. Increasing B enhances the diversity of the sampled submodel instances, increasing S improves the coverage of the cross-product of those instances, and increasing n reduces the impact of simulation noise by averaging over more replications. Note that the number of submodels that must be fitted or retrained depends on B , not S .

Our framework can also accommodate time-dependent submodels, e.g., models of time non-homogeneous arrivals. If we can assume that the behavior of the submodel is unchanging within certain periods, then the submodel can be further divided into distinct submodels—one for each period—and our approach applied as described.

We next describe how one can construct a CI for $\mu(\mathbf{S}^c)$ and use regression trees to decompose and attribute overall uncertainty to individual submodels.

5.1 Confidence and Credible Intervals and Decomposing Uncertainty

Suppose the sample means $\{\bar{Y}^{(1)}, \bar{Y}^{(2)}, \dots, \bar{Y}^{(B')}\}$ are obtained from the designed experiment, where

$$\bar{Y}^{(b')} = \frac{1}{n} \sum_{j=1}^n Y_j(\hat{\mathbf{S}}^{(b')}) \quad \text{for } b' = 1, 2, \dots, B'.$$

An approximate $100(1 - \alpha)\%$ quantile-based confidence or credible interval for $\mu(\mathbf{S}^c)$ is given by

$$[Q_{\alpha/2}(\bar{Y}), Q_{1-\alpha/2}(\bar{Y})], \quad (2)$$

where $Q_p(\bar{Y})$ denotes the empirical p -quantile of the sample means. This CI incorporates both submodel and aleatoric uncertainty. In the frequentist setting, the confidence interval is asymptotically valid as the size of the training datasets grows large, assuming that the true subprocesses belong to the chosen classes of submodels. Note that the CI in (2) does not distinguish the submodels' individual contributions to overall uncertainty. Consequently, decision makers looking to reduce uncertainty—such as by increasing the number of simulation replications, collecting more real-world data, or enhancing the submodels through hyperparameter tuning or switching to a more flexible architecture—could benefit from techniques that attribute uncertainty to specific submodels.

We present a method for quantifying the contribution of different sources to overall uncertainty. Our approach leverages *variable importance* metrics from regression trees [17], which assign a value to each feature based on the total reduction in the total sum of squares (TSS) resulting from splits on that feature. Our approach strongly resembles analysis of variance (ANOVA); however, whereas ANOVA separately accounts for interaction effects, tree-based importance scores absorb interactions into the main effects. Interaction contributions are typically difficult to interpret and may not correspond to actionable modeling or data-collection decisions. For example, even if a strong interaction is detected between two submodels, it may not be clear whether additional data should be collected for one submodel, the other, or both simultaneously. Because our approach forces the interaction effects on certain submodels, we do not assume that submodels are independent, which we believe is unlikely to hold in practice. Furthermore, our approach can preserve any inherent or assumed dependence across submodels by bootstrapping the training data or sampling from the joint posterior distribution accordingly.

We first consider a regression tree trained on the dataset produced by the designed experiment, where the L submodels are treated as categorical features with levels corresponding to the B sampled submodel instances, e.g., $\hat{\mathbf{S}}_\ell^{(1)}, \hat{\mathbf{S}}_\ell^{(2)}, \dots, \hat{\mathbf{S}}_\ell^{(B)}$ for $\ell = 1, 2, \dots, L$. The tree is grown using recursive binary splitting, iteratively minimizing the residual sum of squares (RSS), until each configuration is its own terminal node, at which point the RSS corresponds to the portion of the TSS that is not explained by the submodels. The TSS is defined as

$$\text{TSS} = \sum_{b'=1}^{B'} \sum_{j=1}^n \left(Y_j^{(b')} - \bar{Y} \right)^2, \quad \text{where} \quad \bar{Y} = \frac{1}{nB'} \sum_{b'=1}^{B'} \sum_{j=1}^n Y_j^{(b')},$$

and the RSS is given by:

$$\text{RSS} = \sum_{b'=1}^{B'} \sum_{j=1}^n \left(Y_j^{(b')} - \hat{Y}_j^{(b')} \right)^2,$$

where $\hat{Y}_j^{(b')}$ is the predicted value of the KPI for configuration b' from the fully grown regression tree, which in our case is $\bar{Y}^{(b')}$.

To further understand the role of model complexity in this framework, we consider the concept of *degrees of freedom* (DoF), which represents the effective number of parameters used by a model to fit data. In regression settings, DoF is

often interpreted as a measure of a model’s flexibility or complexity and plays a critical role in error decomposition [38]. For regression trees, a common approximation for DoF is the number of terminal (leaf) nodes [13]. Furthermore, the contribution of individual features to model complexity is the number of times each feature is used for splitting, where each split contributes approximately one additional DoF [20]. To attribute uncertainty to each source, we compute the reduction in TSS attributable to each submodel and normalize it by the number of times the corresponding submodel appears in the tree splits. Finally, we divide the RSS by its DoF, $nB' - B' + 1$, to obtain the simulation error’s contribution to the overall variance.

Regression trees, are known to exhibit high variance, as small perturbations in the training data can lead to substantially different tree structures. To mitigate this variability, we adopt a bootstrap aggregation (bagging) strategy [5]. Specifically, we first generate bootstrap training datasets by resampling observations with replacement. For each bootstrap sample, a regression tree is trained and feature importance scores are computed as described above. The resulting importance scores are then averaged across bootstrap instances, in a manner analogous to the aggregation mechanism used in random forests. This aggregation can also be performed using a weighted average, where the weights are derived either from the distribution of the observed system states or from the relative magnitude of the total TSS associated with each tree. This procedure yields a set of importance estimates with reduced variance relative to those obtained from a single regression tree, without needing to train additional submodels or run more simulation replications.

5.2 Numerical Example

In our first experiment, we study the effect of submodel uncertainty on the output of a synthetic simulation model. We consider two independent stochastic input models $X_1 \sim \mathcal{N}(1, 0.5^2)$ and $X_2 \sim \mathcal{N}(1, 2^2)$ and two deterministic transformation functions $p(x) = x^7 + x^4 + 3x^3 + \sin(x) + 4$ and $q(x) = x^3 + x^2 + 4x$. The output of the simulation model is defined as $Y = p(X_1) + q(X_2)$ and our quantity of interest is the expected value $\mathbb{E}[Y]$, which can be derived analytically and equals 49.13.

We examine three cases that differ in terms of the sources of epistemic uncertainty that are accounted for.

1. **No epistemic uncertainty:** The simulation is run using only the estimated submodels.
2. **Input uncertainty only:** UQ is performed for the input models associated with X_1 and X_2 , while only the estimated submodels for p and q are used. This setting represents the existing practice of analyzing IU.
3. **Submodel uncertainty:** UQ is performed for all submodels, including input models.

For each case, we conduct 100 macro-replications. Within each macro-replication, a training dataset of size 50 is generated for each submodel by running 50 replications of the simulation model with the true submodels, i.e., we obtain 50 observations of $\{X_1, p(X_1), X_2, q(X_2), Y\}$. We then bootstrap this dataset to generate $B = 100$ training datasets of size 50. For the input models X_1 and X_2 , observations are resampled independently, whereas for the response models p and q , the corresponding input-output pairs are resampled jointly. For each bootstrapped dataset, the necessary submodels are estimated and the simulation model is then run for $n = 50$ replications to estimate $\mathbb{E}[Y]$. On each macro-replication, we construct 90% empirical confidence intervals for $\mathbb{E}[Y]$ via Equation (2) and calculate their coverages and average widths across macro-replications. The 100 confidence intervals for each case are shown in Figure 1 and their summary statistics are reported in Table 1. As seen in Table 1, the coverage and average width of the intervals increase as more sources of uncertainty are considered.

Table 1: Coverage and average confidence interval width under different uncertainty scenarios.

Scenario	Coverage (%)	Avg. CI Width
No epistemic uncertainty	61.0	20.45 ± 0.14
Input uncertainty only	82.0	36.14 ± 0.35
Submodel uncertainty	87.0	38.60 ± 0.37

Because the true data-generating mechanisms are known, we further analyze all $2^4 = 16$ combinations of true and estimated submodels. For each configuration, we generate 500 independent training datasets of size 50. When a submodel is estimated in a given configuration, it is fitted using the corresponding training dataset and then embedded in the simulation model and the simulation is run for $n = 500$ replications under that configuration. Figure 2 shows the estimated variance and bias of the resulting estimator for each combination. Figures 3a and 3b show response grid plots (RGPs) [3] where the responses are the variance and the bias, respectively. The plots show that the variance of the estimator decreases when fewer submodels are utilized, and the bias increases when $p(\cdot)$ is estimated. The ANOVA results indicate that X_1 (p-value = 0.023) and p (p-value ≈ 0) have a statistically significant effect on the output

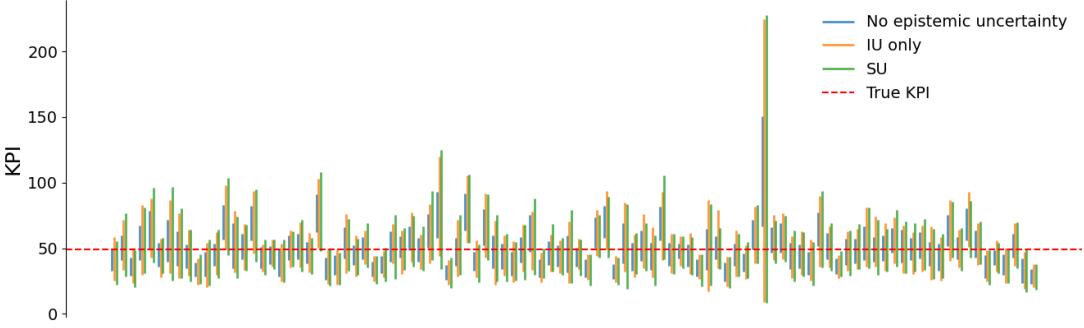


Figure 1: Confidence intervals over 100 macro-replications when accounting for no epistemic uncertainty, only input uncertainty, or submodel uncertainty (SU).

variance, whereas q (p-value = 0.04) exhibits comparatively weaker evidence of significance and X_2 (p-value = 0.12) is not statistically significant.

Figure 4 shows the decomposition of the variance across sources of uncertainty when using the bagging approach on one macro replication. The results align with those of the RGP plot and the ANOVA in that X_1 and p are identified as contributing the most to the uncertainty. Overall, this suggests that one should prioritize obtaining more data for modeling X_1 and training p in order to reduce the epistemic uncertainty.

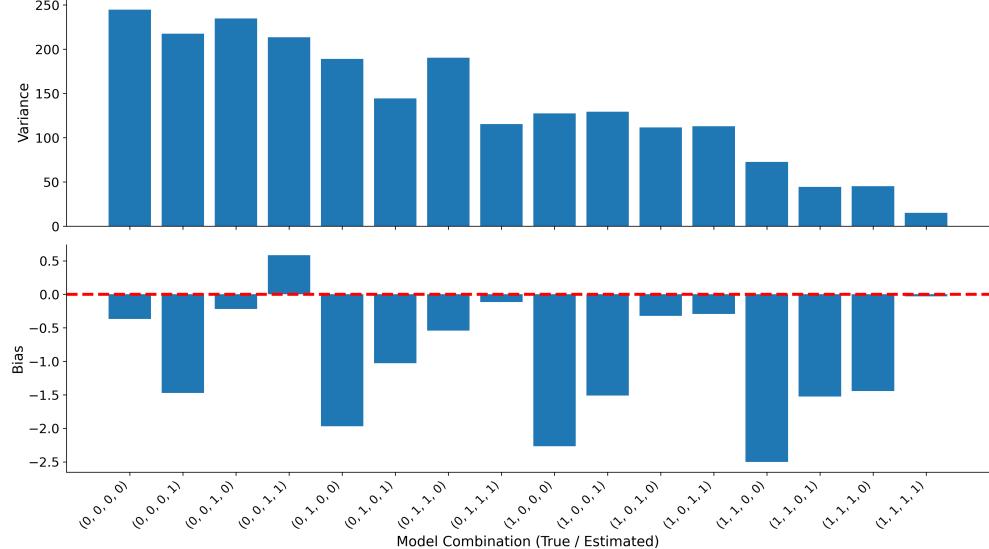


Figure 2: Bias and variance across combinations. The elements in each tuple correspond to X_1 , X_2 , p , and q , respectively; 1 indicates that the true model is used, and 0 indicates that the estimated model is used.

We also compare the variance of the feature importance scores obtained from the regression tree and the bagging approach. The experiment was conducted over 100 macro-replications. In each macro-replication, $B' = 512$ configurations ($B = 8, S = 64$) were evaluated using training datasets of size 25 for each submodel. The bagging approach used 50 regression trees. Table 2 reports the variance reduction factors (VRFs) across macro-replications and shows that bagging leads to considerable reduction in the variance of the reported importance scores.

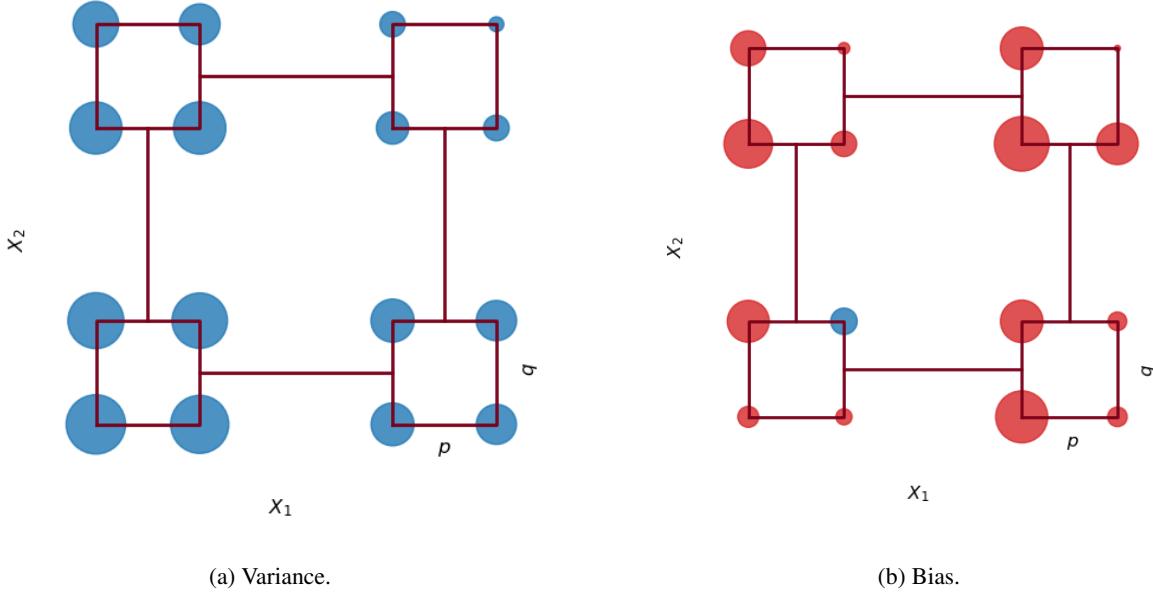


Figure 3: Response grid plots for a 2^4 full factorial design of the four submodels (true vs estimated). The radius of each circle indicates the relative magnitude. Blue (red) circles show positive (negative) values.

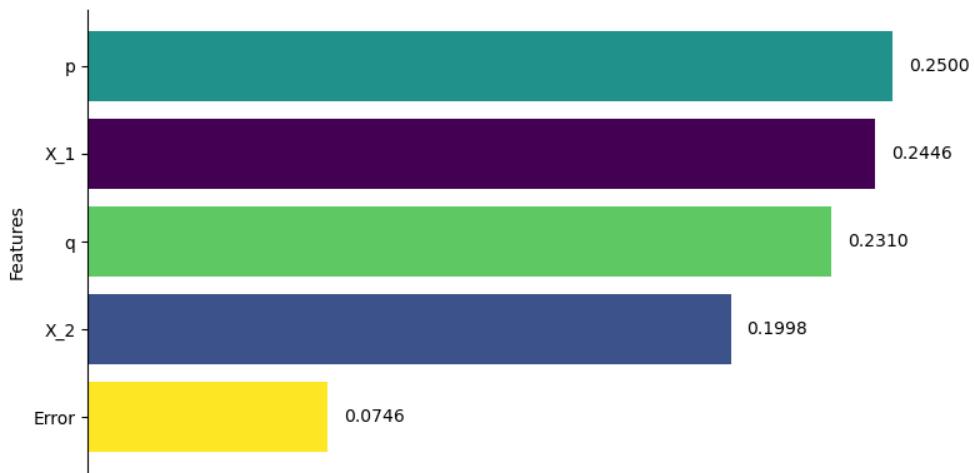


Figure 4: Feature importance plot attributing uncertainty to each of the estimated submodels and aleatoric uncertainty.

Table 2: Variance reduction factors (VRF) for individual submodels.

Submodel	X_1	X_2	p	q
VRF	5.4	4.7	3.6	5.6

6 Submodel Uncertainty in Digital-Twin Settings

A digital twin serves as a virtual counterpart of a physical system and enables forward-looking performance evaluation, robustness analysis, and what-if experimentation under alternative operating scenarios [12]. A simulation model can be extended into a digital twin by continuously ingesting real-time system information and updating the state, input models, and submodel parameters accordingly. In the context of digital twins, a simulation model can be repeatedly initialized using observed system state information and used to forecast KPIs over some future time horizon and support monitoring, planning, and decision making. To extend our UQ framework to digital-twin settings, we first address how the state of the physical system is observed in an online manner. In this setting, our objective is not only to quantify the overall uncertainty in the future KPI given the current state, but also to understand how individual submodels contribute to this uncertainty.

6.1 Uncertainty Quantification in Digital Twins

In a digital twin, the state of the physical system is observed at the beginning of an epoch, at which time the simulation model is hot started to predict the KPI at the end of the epoch. Let \mathbf{X}_t denote the (observed) state of the system at the start of epoch t , and let \mathbf{S}_t^c denote the true subprocesses governing the system during epoch t . Given the state and true subprocesses, the simulation output at the end of epoch t on a given replication $j = 1, 2, \dots$ can be expressed as

$$Y_j(\mathbf{X}_t, \mathbf{S}_t^c) \equiv \mu(\mathbf{X}_t, \mathbf{S}_t^c) + \epsilon_j(\mathbf{X}_t, \mathbf{S}_t^c).$$

In contrast to the offline simulation setting, the state of the system is now an explicit input to the simulation model that can vary across configurations. As a result, we slightly modify the DOE framework introduced in Section 5. Recall that in the LH designs we assume a common number of resampled instances for each submodel: B . In the digital-twin setting, however, the number of states of interest—whether observed in the past or produced from a future forecast—need not equal B . For each state of interest, we propose forming B' design points via a LH design, as described in the previous section, and form a *per-state* dataset as depicted in Table 3. For computational reasons, the value of B' may be chosen to be smaller than that in the offline simulation setting. Creating a separate design for each state is motivated by our desire to eliminate the impact of the variability of the system state on the variability of the KPI. We believe that in a digital-twin setting, a decision maker is interested in understanding how much each estimated submodel contributes to the overall uncertainty *across different states*. This is partly because decisions that are informed by the resulting importance scores—such as from which subprocesses more data should be collected—will typically not depend on any one particular state. We therefore propose applying our tree-based method on each state’s design and reporting the average importance scores of each submodel across states as indicators of their *aggregate* contributions to epistemic uncertainty. A weighted average can alternatively be used, where the average sum of squares (TSS) in each bagged tree is used as the weight, assigning more importance to states with higher variance.

6.2 Bias Estimation in Digital Twins

Recent research has examined the (mis)alignment between digital twins and their physical counterparts [27, 14], where simulation outputs are periodically compared against observed data using statistical tests. We propose a method for quantifying a related form of bias that uses the same data as in the aforementioned design and does not require additional simulation runs. Suppose that the real system is observed at states $\{\mathbf{x}_{t_1}, \mathbf{x}_{t_2}, \dots, \mathbf{x}_{t_N}\}$ with corresponding observed KPIs $\{y(\mathbf{x}_{t_1}, \mathbf{S}_{t_1}^c), y(\mathbf{x}_{t_2}, \mathbf{S}_{t_2}^c), \dots, y(\mathbf{x}_{t_N}, \mathbf{S}_{t_N}^c)\}$. We define the *state-average bias* to be

$$\eta_N = \frac{1}{N} \sum_{i=1}^N \left(\mathbb{E}[Y(\mathbf{x}_{t_i}, \hat{\mathbf{S}}_{t_i})] - \mu(\mathbf{x}_{t_i}, \mathbf{S}_{t_i}^c) \right).$$

The state-average bias measures the average discrepancy between the simulated and observed KPIs across the set of observed system states. This bias quantifies the systematic misalignment between the digital twin and the physical system that could persist over time if uncorrected. For each configuration in Table 3, $b' = 1, 2, \dots, B'$, we can calculate a corresponding point estimate of this bias across states:

$$\hat{\eta}_N^{(b')} = \frac{1}{N} \sum_{i=1}^N \left(\frac{1}{n} \sum_{j=1}^n Y_j(\mathbf{x}_{t_i}, \hat{\mathbf{S}}_{t_i}^{(b')}) - y(\mathbf{x}_{t_i}, \mathbf{S}_{t_i}^c) \right).$$

An approximate $100(1 - \alpha)\%$ CI for the state-average bias can then be constructed using the empirical quantiles of the bootstrapped (or resampled) estimates $\hat{\eta}_N^{(1)}, \hat{\eta}_N^{(2)}, \dots, \hat{\eta}_N^{(B')}$ as

$$[Q_{\alpha/2}(\hat{\eta}_N), Q_{1-\alpha/2}(\hat{\eta}_N)],$$

where $Q_p(\hat{\eta}_N)$ denotes the empirical p -quantile.

Table 3: Sampling framework for submodel uncertainty in digital-twin setting.

State	Submodels	Output
\mathbf{X}_t	$\hat{\mathbf{S}}_t^{(1)}$	$Y_{t,1}^{(1)} = Y_1(\mathbf{X}_t, \hat{\mathbf{S}}_t^{(1)})$ $Y_{t,2}^{(1)} = Y_2(\mathbf{X}_t, \hat{\mathbf{S}}_t^{(1)})$ \vdots $Y_{t,n}^{(1)} = Y_n(\mathbf{X}_t, \hat{\mathbf{S}}_t^{(1)})$
\mathbf{X}_t	$\hat{\mathbf{S}}_t^{(2)}$	$Y_{t,1}^{(2)} = Y_1(\mathbf{X}_t, \hat{\mathbf{S}}_t^{(2)})$ $Y_{t,2}^{(2)} = Y_2(\mathbf{X}_t, \hat{\mathbf{S}}_t^{(2)})$ \vdots $Y_{t,n}^{(2)} = Y_n(\mathbf{X}_t, \hat{\mathbf{S}}_t^{(2)})$
\vdots	\vdots	\vdots
\mathbf{X}_t	$\hat{\mathbf{S}}_t^{(B')}$	$Y_{t,1}^{(B')} = Y_1(\mathbf{X}_t, \hat{\mathbf{S}}_t^{(B')})$ $Y_{t,2}^{(B')} = Y_2(\mathbf{X}_t, \hat{\mathbf{S}}_t^{(B')})$ \vdots $Y_{t,n}^{(B')} = Y_n(\mathbf{X}_t, \hat{\mathbf{S}}_t^{(B')})$

6.3 Numerical Example

To illustrate the effects of submodel uncertainty in the digital-twin setting, we consider a simulation model of a contact center with two contact groups and three expert groups. Contacts from each group are assumed to arrive according to two independent nonhomogeneous Poisson processes with piecewise-constant arrival rate functions that vary by hour. For each contact group, patience times and service times are jointly distributed with Gamma marginal distributions coupled through a Gaussian copula. The KPI of interest is the expected average waiting time of contacts in the second contact group. Each expert group has two experts. The first expert group serves only the first contact group, the second expert group serves only the second contact group, and the third expert group serves both contact groups with priority given to the first contact group.

A routing mechanism dynamically assigns contacts to experts based on system conditions and is triggered when one of two scenarios occurs: If a contact arrives to find their contact group queue is empty and one or more compatible experts are available, the contact is immediately assigned to one of the available experts. Alternatively, if an expert becomes available when queues are present, a waiting contact is selected to be serviced by that expert. We consider two distinct routing processes: a contact-triggered routing process, denoted by \mathbf{S}_C^c , and an expert-triggered routing process, denoted by \mathbf{S}_E^c . Both \mathbf{S}_C^c and \mathbf{S}_E^c are rules-based functions with non-trivial decision rules. For the frequentist case, we use maximum likelihood estimators (MLEs) for the input submodels and logistic regression models for the routing submodels. For the Bayesian case, we use non-informative priors for the input submodels—which yields the same estimates as the MLEs—and for the routing submodels, we train two Bayesian neural networks, each with one hidden layer with 128 nodes.

The simulation model is run for a single nine-hour day, and the state of the system is recorded at each 30-minute interval for a total of 18 epochs. At the end of the day, data on customer arrivals, patience times, handle times, and routing decisions are collected and used to construct the estimated submodels. In the case without submodel uncertainty, the simulation is run for 100 replications for each of the observed system states. In the presence of submodel uncertainty, 180 configurations are selected, where for each configuration we use one of the five resampled instances of each submodel as denoted by the Latin hypercube (LH) design. Since we have observed 18 states, we pick 2 stacks each with 5 configurations of submodels for each state. To obtain the true KPI value for each state, an additional 1,000 replications are performed and the results treated as the ground-truth benchmark. Figure 5 reports 90% confidence and credible intervals for the mean waiting-time KPI of the second contact group at the end of each epoch under the two experimental settings for both frequentist and Bayesian settings. As observed, the width of the intervals increases as submodel uncertainty is incorporated. For each observed system state, we compute importance scores using the bagging approach with 40 trees and average these scores across all states to obtain an overall importance measure for both the

frequentist and Bayesian settings, as shown in Figure 6. The second contact group’s arrival and patience-and-handle time submodels contribute the most to the overall uncertainty. Therefore one should prioritize enhancing the second contact group’s input modeling in the frequentist setting, for instance, by collecting more data. In the Bayesian setting, the expert-side routing submodel contributes the most to the overall uncertainty, thus collecting more training data or considering a more flexible ML model could potentially reduce the epistemic uncertainty. Calculating the overall bias yields CIs of $(-0.33, -0.12)$ and $(-0.61, -0.38)$ for the frequentist and Bayesian settings, respectively. These results indicate that the use of estimated submodels has led to a potential misalignment in KPI estimation. Whether this degree of state-average bias is practically significant will depend on the business purpose.

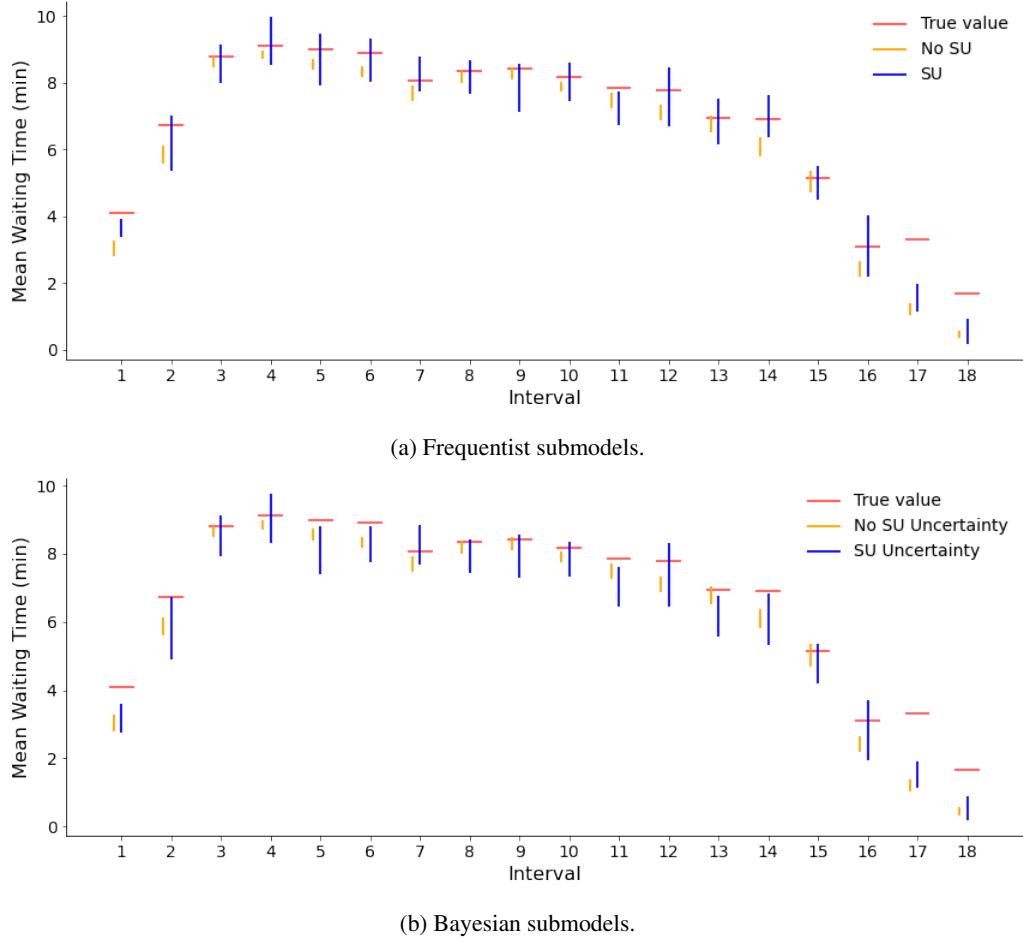
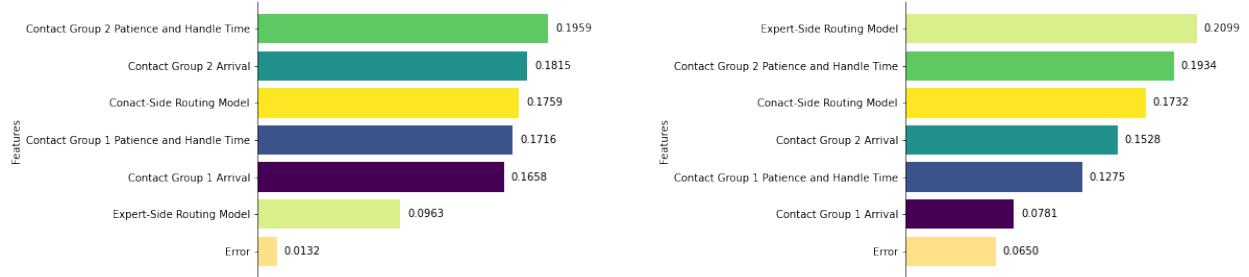


Figure 5: 90% confidence and credible intervals when considering submodel uncertainty and not considering submodel uncertainty for (a) frequentist and (b) Bayesian submodels.

7 Conclusion

This paper develops a general framework for quantifying uncertainty that arises from embedding submodels within stochastic simulation models. We employ bootstrapping and Bayesian model averaging to construct quantile-based confidence and credible intervals that capture both epistemic and aleatoric uncertainty. To understand the drivers of this uncertainty, we use regression-tree-based variable-importance measures to identify the most influential submodels. We further show how the proposed framework can be extended to digital-twin settings with time-varying submodels. Numerical experiments demonstrate that conventional methods may substantially underestimate uncertainty, whereas accounting for submodel uncertainty yields more reliable inference. Future work can investigate the role of uncertainty quantification in model calibration and active learning approaches for dynamically training and updating submodels in digital-twin settings.



(a) Frequentist submodels.

(b) Bayesian submodels.

Figure 6: Importance scores for the submodels in the second experiment for (a) frequentist and (b) Bayesian formulations.

Acknowledgement

This work was supported by the Intuit University Collaboration Program. We also thank Dusan Bosnjakovic for his support throughout the project.

References

- [1] Antonio Alcántara, Carlos Ruiz, and Calvin Tsay. A quantile neural network framework for two-stage stochastic optimization. *Expert Systems with Applications*, page 127876, 2025.
- [2] Saurabh Arora and Prashant Doshi. A survey of inverse reinforcement learning: Challenges, methods and progress. *Artificial Intelligence*, 297:103500, 2021.
- [3] Russell R Barton. Response grid plots for model-agnostic machine learning insight. *IMA Journal of Management Mathematics*, 36(4):623–648, 2025.
- [4] Russell R Barton, Henry Lam, and Eunhye Song. Input uncertainty in stochastic simulation. In *The Palgrave Handbook of Operations Research*, pages 573–620. Springer, Cham, Switzerland, 2022.
- [5] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.
- [6] Wang Cen and Peter J Haas. Nim: Generative neural networks for automated modeling and generation of simulation inputs. *ACM Transactions on Modeling and Computer Simulation*, 33(3):1–26, 2023.
- [7] Wenbo Chen, Mathieu Tanneau, and Pascal Van Hentenryck. Real-time risk analysis with optimization proxies. *Electric Power Systems Research*, 235(1):110822, 2024.
- [8] Stephen E Chick. Input distribution selection for simulation experiments: Accounting for input uncertainty. *Operations Research*, 49(5):744–758, 2001.
- [9] Luc Devroye. Nonuniform random variate generation. *Handbooks in Operations Research and Management Science*, 13:83–121, 2006.
- [10] Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. Chapman and Hall/CRC, New York, NY, 1994.
- [11] Mohammadmahdi Ghasemloo, David J Eckman, and Yaxian Li. Quantifying uncertainty from machine learning surrogate models embedded in simulation models. In *2025 Winter Simulation Conference (WSC)*, pages 3418–3429. IEEE, 2025.
- [12] Michael Grieves and John Vickers. *Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems*. Springer Series in Advanced Manufacturing. Springer, Cham, Switzerland, 2016.
- [13] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics. Springer, New York, 2nd edition, 2009.
- [14] Linyun He, Luke Rhodes-Leader, and Eunhye Song. Digital twin validation with multi-epoch, multi-variate output data. In *2024 Winter Simulation Conference (WSC)*, pages 347–358. IEEE, 2024.
- [15] Linyun He and Eunhye Song. Introductory tutorial: Simulation optimization under input uncertainty. In *2024 Winter Simulation Conference (WSC)*, pages 1338–1352. IEEE, 2024.
- [16] Ziqi Huang, Yang Shen, Jiayi Li, Marcel Fey, and Christian Brecher. A survey on ai-driven digital twins in industry 4.0: Smart manufacturing and advanced robotics. *Sensors*, 21(19):1–23, 2021.
- [17] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: With Applications in R*. Springer Texts in Statistics. Springer, New York, 2nd edition, 2021.

- [18] Soumendra N. Lahiri. *Resampling Methods for Dependent Data*. Springer Series in Statistics. Springer, New York, 2003.
- [19] Henry Lam and Huajie Qian. Subsampling to enhance efficiency in input uncertainty quantification. *Operations Research*, 70(3):1891–1913, 2022.
- [20] Lucas Mentch and Weijia Zhou. Randomization as regularization: A degrees of freedom explanation for random forests. *Journal of Machine Learning Research*, 21(202):1–39, 2020.
- [21] Lucy E Morgan, Barry L Nelson, Andrew C Titman, and David J Worthington. Detecting bias due to input modelling in computer simulation. *European Journal of Operational Research*, 279(3):869–881, 2019.
- [22] Radford M. Neal. *Bayesian Learning for Neural Networks*. PhD thesis, University of Toronto, 2012.
- [23] Barry Nelson and Linda Pei. *Foundations and Methods of Stochastic Simulation*, volume 187. Springer, 2nd edition, 2021.
- [24] Dimitris N. Politis and Joseph P. Romano. The stationary bootstrap. *Journal of the American Statistical Association*, 89(428):1303–1313, 1994.
- [25] Christos Pylianidis, Val Snow, Hiske Overweg, Sjoukje Osinga, John Kean, and Ioannis N Athanasiadis. Simulation-assisted machine learning for operational digital twins. *Environmental Modelling & Software*, 148:105274, 2022.
- [26] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, Cambridge, MA, 2006.
- [27] Luke A Rhodes-Leader and Barry L Nelson. Tracking and detecting systematic errors in digital twins. In *2023 Winter Simulation Conference (WSC)*, pages 492–503. IEEE, 2023.
- [28] Susan M Sanchez, Paul J Sanchez, and Hong Wan. Work smarter, not harder: A tutorial on designing and conducting simulation experiments. In *2020 Winter Simulation Conference (WSC)*, pages 1128–1142. IEEE, 2020.
- [29] Thomas J. Santner, Brian J. Williams, and William I. Notz. *The Design and Analysis of Computer Experiments*. Springer, New York, 2nd edition, 2018.
- [30] David Simchi-Levi, Philip Kaminsky, and Edith Simchi-Levi. *Designing and Managing the Supply Chain: Concepts, Strategies, and Case Studies*. McGraw-Hill, 2007.
- [31] Simio LLC. Simio simulation software. <https://www.simio.com>, 2023. Accessed: 2026-02-13.
- [32] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998.
- [33] Zhou Tang and Ted Westling. Consistency of the bootstrap for asymptotically linear estimators based on machine learning. *arXiv preprint arXiv:2404.03064*, 2024.
- [34] Simon JE Taylor, Charles M Macal, Andrea Matta, Markus Rabe, Susan M Sanchez, and Guodong Shao. Enhancing digital twins with advances in simulation and artificial intelligence: Opportunities and challenges. In *2023 Winter Simulation Conference (WSC)*, pages 3296–3310. IEEE, 2023.
- [35] Adam Thelen, Xiaoge Zhang, Olga Fink, Yan Lu, Sayan Ghosh, Byeng D Youn, Michael D Todd, Sankaran Mahadevan, Chao Hu, and Zhen Hu. A comprehensive review of digital twin — part 2: Roles of uncertainty quantification and optimization, a battery digital twin, and perspectives. *arXiv preprint arXiv:2208.12904*, 2022.
- [36] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*, 2015.
- [37] Jiannan Yang, Robin S. Langley, and Luis Andrade. Digital twins for design in the presence of uncertainties. *Mechanical Systems and Signal Processing*, 179:109338, 2022.
- [38] Jianming Ye. On measuring and correcting the effects of data mining and model selection. *Journal of the American Statistical Association*, 93(441):120–131, 1998.