
Deep Probabilistic Surrogate Networks for Universal Simulator Approximation

Andreas Munk

Dept. of Computer Science
University of British Columbia
Vancouver, B.C.
Canada
amunk@cs.ubc.ca

Adam Ścibior

Dept. of Computer Science
University of British Columbia
Vancouver, B.C.
Canada
ascibior@cs.ubc.ca

Atılım Güneş Baydin

Dept. of Engineering Science
University of Oxford
Oxford, OX2 6PN
United Kingdom
gunes@robots.ox.ac.uk

Andrew Stewart

Convergent Manufacturing
Technologies Inc.
Vancouver, B.C.
Canada
andrew.stewart@convergent.ca

Goran Fernlund

Dept. of Materials Engineering
University of British Columbia
Convergent Manufacturing
Technologies Inc.
University of British Columbia
Vancouver, B.C.
Canada
goran.fernlund@convergent.ca

Anoush Poursartip

Dept. of Materials Engineering
University of British Columbia
Convergent Manufacturing
Technologies Inc.
University of British Columbia
Vancouver, B.C.
Canada
anoush.poursartip@ubc.ca

Frank Wood

Dept. of Computer Science
University of British Columbia
Vancouver, B.C.
Canada
fwood@cs.ubc.ca

Abstract

We present a framework for automatically structuring and training fast, approximate, deep neural surrogates of existing stochastic simulators. Unlike traditional approaches to surrogate modeling, our surrogates retain the interpretable structure of the reference simulators. The particular way we achieve this allows us to replace the reference simulator with the surrogate when undertaking amortized inference in the probabilistic programming sense. The fidelity and speed of our surrogates allow for not only faster “forward” stochastic simulation but also for accurate and substantially faster inference. We support these claims via experiments that in-

volve a commercial composite-materials curing simulator. Employing our surrogate modeling technique makes inference an order of magnitude faster, opening up the possibility of doing simulator-based, non-invasive, just-in-time parts quality testing; in this case inferring safety-critical latent internal temperature profiles of composite materials undergoing curing from surface temperature profile measurements.

1 Introduction

Simulators are accurate generative models that encode the knowledge of domain experts. Whether in aeronautical engineering (Wu et al., 2018), nonlinear flow physics (Veldman et al., 2007), or stochastic generative modeling (Heinecke et al., 2014; Endeve et al., 2012; Raberto et al., 2001; Perdikaris et al., 2016), they play

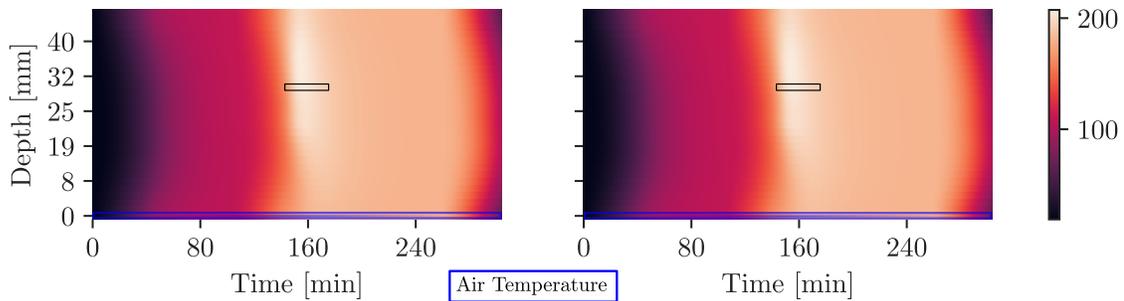


Figure 1: Illustration of a process simulation of composite materials, which we denote $\mathbf{x}_{\text{RAVEN}}$. Each subfigure shows a temperature profile measured in degrees Celsius as a function of time along the x axis and depth along the y -axis. (Left) shows the actual simulated heating process. (Right) shows the same process but originating from our *probabilistic surrogate network* trained to match the original simulator. Noticeably, the process in (right) is **26.8 times faster** than the process in (left). Later we perform inference in this process, where we seek to infer the expected temperature in the time window $[155, 165]$ min at depth 30 mm represented by the black box conditioned on measuring the temperature at the bottom surface and air temperature represented by the blue boxes.

an important role in design, diagnosis, and manufacturing. Unfortunately, detailed simulators are often computationally expensive, ruling them out for just-in-time uses. This problem is exacerbated in stochastic simulators as these often need to be run many times to produce convergent expectations.

A natural solution to this problem, known as surrogate modeling, is to construct a fast approximation to some reference simulator. Surrogate modeling has found success in applications in various fields including computational fluid dynamics (CFD) (Glaz et al., 2010; Yamazaki and Mavriplis, 2013; Biannic et al., 2016), aerospace engineering (Jeong et al., 2005), material science (Rikards et al., 2004) and quantum chemistry (Gilmer et al., 2017).

In this paper we develop a novel method for automatically structuring and training surrogates for arbitrary stochastic simulators, leveraging deep generative neural networks (Hu et al., 2017). Unlike prior art, our surrogate automatically generates all latent variables in the simulator regardless of the language in which the simulator was written. We achieve that by simultaneously learning to approximate the distributions and the control flow of the original simulator. That we do this affords maximal interpretability as it allows the user to inspect, should they wish, any and all of the latent variables at all times during its execution. Our method can handle simulators with arbitrarily many random variables and arbitrary dependency structure between them, including the existence of latent variables being conditional on the values of other variables. Due to this flexibility we describe our surrogates as universal simulator approximators and call them *probabilistic surrogate networks* (PSNs).

Faster simulation via surrogate modeling is itself useful. However, this speed has arguably even greater impact on the development of practical applications by enabling faster “inverting” of simulators via probabilistic programming techniques.

Here inverting a simulator means to perform Bayesian inference over its inputs and latent variables given observed values of its output. This definition blurs the line between stochastic simulators and probabilistic models and is arguably a key point of probabilistic programming (van de Meent et al., 2018). In this paper we specifically build on the *inference compilation* (IC) framework of Le et al. (2017) and its PyProb (Baydin and Le, 2018) realization. PyProb enables Bayesian inference in stochastic simulators written in existing programming languages via built-in inference engines, only requiring the user to annotate random variables in the original simulator source. The built-in inference engine communicates with and controls the simulator through a standardized interface that ensures compatibility across languages. This process is illustrated in Figure 2 and is explained elsewhere in full technical abstraction (van de Meent et al., 2018, chapt. 6).

We design our PSNs to be compatible with this interface, which has two advantages. First, it lets us easily train PSNs using existing simulators written in standard programming languages such as C. This is crucial for practical applications, as reimplementing existing simulators in special-purpose modeling languages is undesirable. Second, this design choice lets us automatically use PSNs for efficient amortized inference. This is achieved by directly substituting the surrogate for the simulator and having the former communicate with the inference engine over the same interface in-

stead (illustrated in Figure 2). As having to run the simulator is nearly always the computational bottleneck in amortized inference, this leads to significantly sped up inference on average.

The non-trivial, real-world application we illustrate our method on relates to just-in-time validation of a composite material cure. In particular we learn a surrogate for the commercial heat-transfer finite element analysis simulator, depicted in Figure 1, that is used to model the cure cycle for Boeing’s composite aircraft parts (e.g. wings). As a proof of concept we show how to use this simulator to estimate the temperature at a point in a simplified design that cannot be accessed non-invasively. We call this a “virtual thermocouple” and its “reading,” achieved through examination of the posterior state of the simulator virtual machine given observable features of the thermal environment, is critical for saying whether the part is safe or not.

To summarize, we make two key methodological contributions in this paper.

1. We develop a universal method for constructing fast surrogates for stochastic simulators and
2. We show how to use our surrogates to speedup inference in stochastic simulators.

Our theoretical contributions include both a novel method for projecting a nonparametric stochastic process onto a parametric stochastic process and transposing time and space in a surrogate for a more parameter-efficient process representation. The application itself is a contribution too.

2 Background

2.1 Surrogate Modeling

Surrogate modeling aims to replace an accurate but slow model/simulator with a faster, approximate “surrogate”. It is fundamentally a regression problem, where the surrogate predicts the output of the model for a given input. Currently, the most commonly used methods for constructing deterministic surrogate models (Razavi et al., 2012) include Kriging (Simpson et al., 2001; Sacks et al., 1989), support vector machines (SVMs) (Willcox and Megretski, 2005), radial basis functions (RBFs) (Hussain et al., 2002; Regis and Shoemaker, 2007; Mullur and Messac, 2006), and neural networks (NNs) (Tompson et al., 2017; Alam et al., 2004; Khu and Werner, 2003; Gilmer et al., 2017), while methods like the stochastic Kriging (Hamdia et al., 2017) allow for stochastic surrogate modeling.

Surrogate modeling methods can be divided into *local*, where the surrogate only needs to be accurate for a specific range of input values, and *global*, where the surrogate needs to be accurate across all the possible input values. Local approximation is easier, but in many applications global approximation is required, because the input range is not known in advance.

Our PSNs are stochastic and global, designed to construct multipurpose surrogate models, where it is not known a priori what questions will be asked of the surrogate or what the inputs to it will be. We denote the simulator as a joint distribution $p(\mathbf{x})$, where \mathbf{x} are all the random variables defined by the simulator. This joint distribution can be specified using a probabilistic program.

2.2 Probabilistic Programming

The probabilistic programming paradigm equates a generative model with a program written in a probabilistic programming language (PPL). An inference backend then takes the program and the observed data and generates inference results, usually in a form of samples from the posterior distribution. PPLs can be broadly divided (van de Meent et al., 2018) into restricted, which limit the set of expressible models to ensure that particular inference algorithms can be efficiently applied (Lunn et al., 2009; Minka et al., 2018; Milch et al., 2005; Carpenter et al., 2017; Tran et al., 2016), and unrestricted (universal), which allow arbitrary models at the expense of making inference less efficient (Goodman et al., 2008; Mansinghka et al., 2014; Wood et al., 2014; Pfeffer, 2009; Goodman and Stuhlmüller, 2014; Bingham et al., 2018). For our purposes it is particularly important to note that extending an existing Turing complete programming language with operations for sampling and conditioning results in a universal PPL (Gordon et al., 2014). For this reason any existing stochastic simulator, written in any language, after some limited annotations of the source code becomes a program in a universal PPL (Baydin et al. (2019, 2018a); Baydin and Le (2018); Baydin et al. (2018b)). PSN targets universal PPLs, so we focus our discussion here on those.

For our purposes the crucial concept is that of a *trace* of a probabilistic program, which is a sequence of pairs (x_{a_t}, a_t) for $t = 1, \dots, T$, where $a_t \in \mathcal{A}$ is an address (Wingate et al., 2011; van de Meent et al., 2018) of a random variable and x_{a_t} its value. $\mathcal{A} = \{\alpha_1, \alpha_2, \dots\}$ is a countable set of possible addresses and within each trace all addresses are guaranteed to be different. The purpose of the addresses is to identify the same random variables across different execution traces to facilitate efficient inference. The trace length T can vary between different traces of the same program

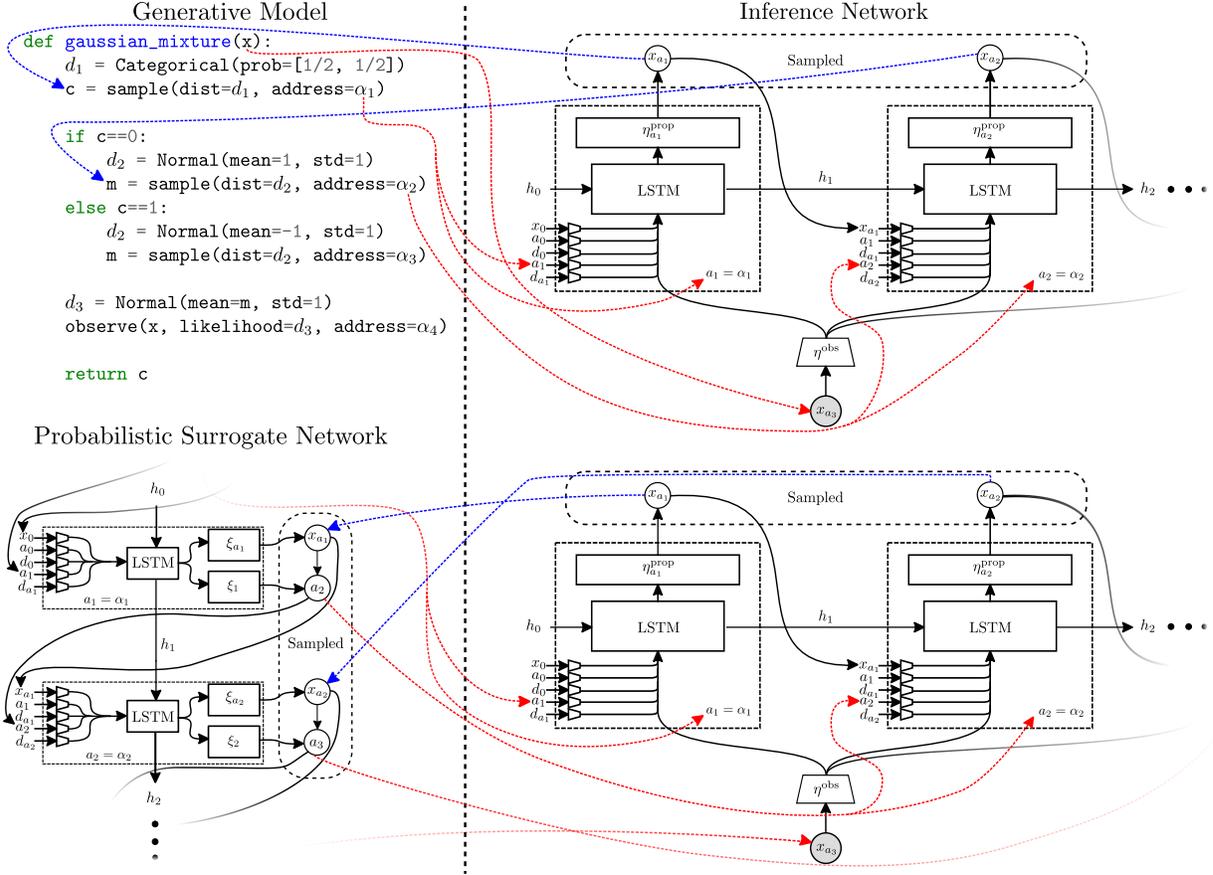


Figure 2: Illustration of the communication between the inference network (right) obtained from *inference compilation* and a generative model $p(\mathbf{x})$ defined as a program or a PSN approximating it (left). Every time the program executes a sampling operation, it sends the distribution d and the address a to the inference network (red lines) and suspends its execution. The inference network then generates a value x and sends it back to the program (blue arrows), which resumes its execution using the obtained value as a result of the call to `sample`. PSN approximates the program, trying to emit the same messages the program would produce. It conforms to the same interface, predicting addresses and distributions for all the random variables in the program, retaining full interpretability and enabling inference using the same inference network without any modifications to the overall setup.

and is generally unbounded but finite. We define $\mathbf{x} = (x_{a_1}, \dots, x_{a_T})$ and $\mathbf{a} = (a_1, \dots, a_T)$. As an example, the program shown in Figure 2 always generates traces of length $T = 3$, with \mathbf{a} being either $(\alpha_1, \alpha_2, \alpha_4)$ or $(\alpha_1, \alpha_3, \alpha_4)$.

Every probabilistic program specifies a joint distribution over the space of traces, denoted as

$$p(\mathbf{x}, \mathbf{a}) = \prod_{t=1}^T p(a_t | x_{a_1:a_{t-1}}, a_{1:t-1}) p(x_{a_t} | x_{a_1:a_{t-1}}, a_{1:t}), \quad (1)$$

where for each t , $p(x_{a_t} | x_{a_1:a_{t-1}}, a_{1:t})$ is specified by the distribution passed to the relevant `sample` or `observe` statement in the program. In probabilistic programs \mathbf{a} is always deterministic conditionally on \mathbf{x} , but we allow it to be stochastic to facilitate the construction

in Section 3.

The subset of \mathbf{x} generated using `observe` statements is designated \mathbf{x}_{obs} and the values for these variables are always provided as observed data. The remaining variables are designated as latent $\mathbf{x}_{\text{lat}} = \mathbf{x} \setminus \mathbf{x}_{\text{obs}}$ and the goal of inference is to compute the posterior distribution $p(\mathbf{x}_{\text{lat}} | \mathbf{x}_{\text{obs}})$. Inference in PPLs can be performed in many different ways, but we focus on a particular approach called *inference compilation*.

2.3 Inference Compilation

Inference compilation (IC) (Le et al., 2017) is an amortized algorithm for performing inference in probabilistic programs using sequential importance sampling (SIS) (Wood et al., 2014). It works by constructing

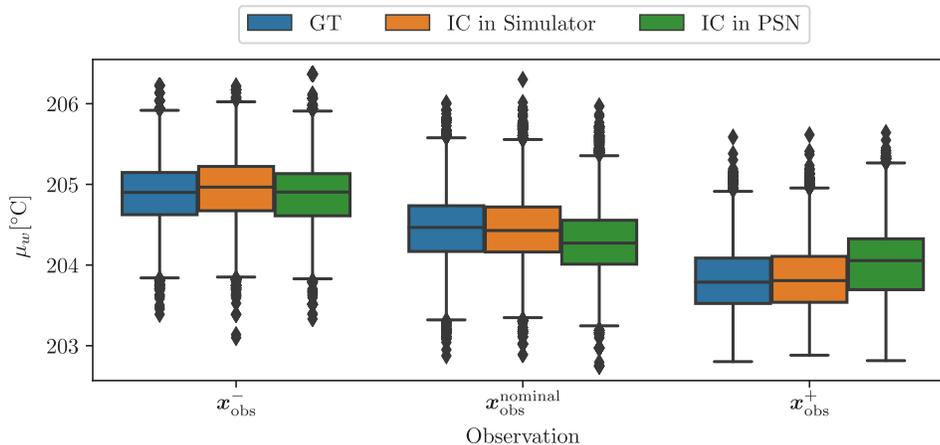


Figure 3: Each boxplot represents the posterior distribution (conditioned on either $\mathbf{x}_{\text{obs}}^-$, $\mathbf{x}_{\text{obs}}^{\text{nominal}}$, or $\mathbf{x}_{\text{obs}}^+$) over $f(\mathbf{x}) = \mu_w$, where μ_w is the empirical mean across the time window $w = [155 \text{ min}, 165 \text{ min}]$ and at a fixed depth 30 mm, see black boxes in Figure 1. We compare three different methods for estimating the posterior; ground truth (GT), which uses regular SIS where $q(\mathbf{x}_{\text{lat}}|\mathbf{x}_{\text{obs}}) = p(\mathbf{x}_{\text{lat}})$, using IC in the simulator and using the same IC inference network in PSN. We observe that the posteriors change as a function of the observation, while each posterior match for the same observation (with the PSN posterior slightly higher for $\mathbf{x}_{\text{obs}}^+$).

an *inference network*, which constructs proposal distributions for all the latent random variables in the program, based on the observed variables.

At the most basic level IC is a self-normalizing importance sampler targeting $p(\mathbf{x}_{\text{lat}}|\mathbf{x}_{\text{obs}})$ using a proposal distribution $q(\mathbf{x}_{\text{lat}}|\mathbf{x}_{\text{obs}})$. It draws K samples $\mathbf{x}_{\text{lat}}^k \stackrel{i.i.d.}{\sim} q(\mathbf{x}_{\text{lat}}|\mathbf{x}_{\text{obs}})$, computes the weights $w^k = \frac{p(\mathbf{x}_{\text{lat}}^k|\mathbf{x}_{\text{obs}})}{q(\mathbf{x}_{\text{lat}}^k|\mathbf{x}_{\text{obs}})}$, and approximates the posterior as

$$p(\mathbf{x}_{\text{lat}}|\mathbf{x}_{\text{obs}}) \approx \frac{\sum_{k=1}^K w^k \delta(\mathbf{x}_{\text{lat}}^k - \mathbf{x}_{\text{lat}})}{\sum_{k=1}^K w^k}. \quad (2)$$

The proposal q factorizes in t just like p . Subsequent conditional distributions in q are constructed using a recurrent deep neural network, the inference network. Specifically,

$$q(\mathbf{x}_{\text{lat}}|\mathbf{x}_{\text{obs}}; \phi) = \prod_{\mathbf{x}_{a_t}^{\text{lat}} \in \mathbf{x}_{\text{lat}}} q(x_{a_t}^{\text{lat}}|\eta_{a_t}(x_{<a_t}^{\text{lat}}, \mathbf{x}_{\text{obs}}, \phi)),$$

where $x_{<a_t}^{\text{lat}} = \{x_{a_t'}|x_{a_t'} \in \mathbf{x}_{\text{lat}}, t' < t\}$, ϕ are the parameters of the inference network, and $\eta_{a_t}(\cdot)$ is the parameters associated with q computed by the neural network.

For our purposes it is particularly important to emphasize how the inference network interacts with the probabilistic program. This is depicted in Figure 2. The program and the network communicate by exchanging messages with the program sending a distribution and an address and receiving back a sampled value. This

interface enables a level of modularity that we take advantage of. Our PSN, as described in Section 3, replaces the original probabilistic program while implementing the same interface, which makes it fully compatible with IC.

In IC the proposal $q(\mathbf{x}_{\text{lat}}|\mathbf{x}_{\text{obs}}; \phi)$ is trained to match the true posterior $p(\mathbf{x}_{\text{lat}}|\mathbf{x}_{\text{obs}}) \propto p(\mathbf{x})$, where the distance between p and q is measured using the Kullback–Leibler (KL) divergence $\text{KL}(p \parallel q)$. In order to match $p(\mathbf{x}_{\text{lat}}|\mathbf{x}_{\text{obs}})$ for all possible \mathbf{x}_{obs} the expected KL divergence under the marginal $p(\mathbf{x}_{\text{obs}})$ is minimized,

$$\begin{aligned} \mathcal{L}_{\text{IC}}(\phi) &= \mathbb{E}_{p(\mathbf{x}_{\text{obs}})} [\text{KL}(p(\mathbf{x}_{\text{lat}}|\mathbf{x}_{\text{obs}}) \parallel q(\mathbf{x}_{\text{lat}}|\mathbf{x}_{\text{obs}}; \phi))] \\ &= -\mathbb{E}_{p(\mathbf{x})} [\log q(\mathbf{x}_{\text{lat}}|\mathbf{x}_{\text{obs}}; \phi)] + \text{const}. \end{aligned} \quad (3)$$

This objective is optimized by running the probabilistic program many times to obtain a collection of traces and maximizing the likelihood under q .

The overall architecture of the inference network uses a Long Short Term Memory (LSTM) core (Hochreiter and Schmidhuber, 1997) as well as embeddings of the addresses, distributions types, and the observations $\eta^{\text{obs}}(\mathbf{x}_{\text{obs}})$. These embeddings are fed to the LSTM core whose output is further fed to a “proposal layer” $\eta_{a_t}^{\text{prop}}$, that produces the parameters to the proposal distribution $q(x_{a_t}^{\text{lat}}|\eta_{a_t}(x_{<a_t}^{\text{lat}}, \mathbf{x}_{\text{obs}}, \phi))$ unique to each address a_t .

We emphasize that in IC both the program and the network are used at inference time. In particular it is not possible to use the inference network without the program, as the network needs to receive the informa-

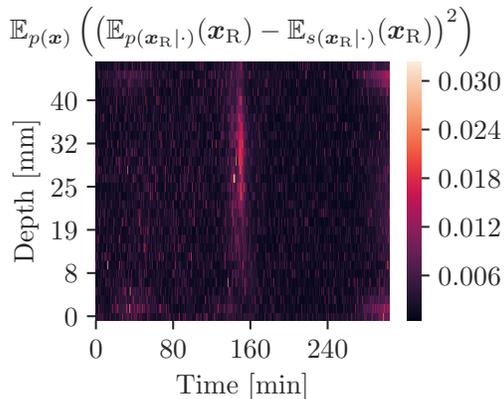


Figure 4: Expected squared difference between the output ($\mathbf{x}_R = \mathbf{x}_{\text{RAVEN}}$) from the PSN and output from the simulator. We observe small errors throughout, with small peaks around time 160 min and towards the end of the heating process.

tion about the address transitions.

3 Probabilistic Surrogate Networks

We introduce probabilistic surrogate networks (PSNs) with the context of probabilistic programming as defined in Section 2.2, emphasizing again that any existing stochastic simulator, written in any programming language, can be turned into a probabilistic program by adding a small amount of annotations to the source code. We construct PSNs to model a distribution over the trace space that factorizes exactly as the distribution of the original program specified in Eq. (1). Specifically, the distribution represented by PSN is defined as

$$s(\mathbf{x}, \mathbf{a}; \theta) = \prod_{t=1}^T s(x_{a_t} | \xi_{a_t}(x_{a_1:a_{t-1}}, a_{1:t}, \theta)) \times s(a_t | \xi_{t-1}(x_{a_1:a_{t-1}}, a_{1:t-1}, \theta)), \quad (4)$$

where $\xi_{a_t}(\cdot)$ is a neural network (NN) that maps to the parameters of the surrogate density at address a_t , and $\xi_t(\cdot)$ is another NN associated with the t th surrogate address transition. We let θ describe all parameters in the PSN, where typically factors in Eq. (4) only use a subset of these.

Structuring the surrogate this way retains the whole structure of the latent space, which means that it can be used to answer any queries about the distribution of any variables in the program. On top of that, because the surrogate also models address transitions of the original program, it can be used with the inference network to speedup inference with IC. We can use exactly

the standard inference network trained on the original program without any modifications. Moreover, as we show below, PSN can be trained using the same collection of traces used to train the inference network, so training a PSN on top of the inference network is very cheap.

3.1 Training Procedure

The PSN $s(\mathbf{x}, \mathbf{a}; \theta)$ is trained to be close to $p(\mathbf{x}, \mathbf{a})$ in terms of the KL-divergence,

$$\begin{aligned} \mathcal{L}(\theta) &= \text{KL}(p(\mathbf{x}, \mathbf{a}) \parallel s(\mathbf{x}, \mathbf{a}; \theta)) \\ &= -\mathbb{E}_{p(\mathbf{x}, \mathbf{a})}[\log s(\mathbf{x}, \mathbf{a}; \theta)] + \text{const} \end{aligned}$$

We can minimize $\mathcal{L}(\theta)$ using the unbiased gradient estimator,

$$\nabla_{\theta} \mathcal{L}(\theta) \approx -\frac{1}{N} \sum_{n=1}^N \nabla_{\theta} \log s(\mathbf{x}^n, \mathbf{a}^n; \theta),$$

where each trace is sampled from the simulator ($\mathbf{x}^n, \mathbf{a}^n \sim p(\mathbf{x}, \mathbf{a})$). We can therefore consider the N sampled traces as our “dataset”. The same collection of traces can be used to train PSN and the inference network.

A subtle point concerning this training procedure is that the number of possible transition addresses can be infinite, so we need to be careful when specifying a parameterization of this distribution. We choose to truncate this distribution, for each address α only allowing transitions to a finite collection of addresses $C_{\alpha} = \{\alpha_1, \dots, \alpha_N\}$. We only allow transitions that were observed in one of the sampled traces, that is $\alpha_i \in C_{\alpha}$ if and only if we have sampled a trace in which for some t , $a_t = \alpha$ and $a_{t+1} = \alpha_i$. Since C_{α} is finite, and typically small, we parameterize it using a categorical distribution.

3.1.1 Surrogate Network Architecture

The PSN architecture is dynamically constructed during training and uses an LSTM core as well as embeddings of the addresses and distributions types. This process is driven by the program, where the embeddings are fed to the LSTM core whose output is further fed to the “distributions layers” ξ_{a_t} and ξ_t , that for each unique address a_t produces the parameters for $s(x_{a_t} | \xi_{a_t}(x_{a_1:a_{t-1}}, a_{1:t}, \theta))$ and $s(a_{t+1} | \xi_t(x_{a_1:a_t}, a_{1:t}, \theta))$ respectively. Whenever a new address is encountered during training (as we go through the “dataset”), new embeddings and distributions layers ξ are constructed, which can then be accessed later in a lookup table.

When replacing the simulator x_0 with the PSN, it is initialized using embeddings x_0 , d_0 , and a_0 which are

Table 1: Runtime [traces/s] comparisons. We calculate the number of traces produced per second when (1) running just the simulator or PSN and (2) when performing SIS in either model. We see a slowdown in traces per second for the PSN when performing inference, as the inference engine adds additional overhead which proves to be a bottleneck. However, as the simulator is considerably slower, it remains the computational bottleneck during inference.

	Simulator ($t_{\text{sim}}[\text{s}]$)	PSN ($t_{\text{PSN}}[\text{s}]$)	Speedup [$t_{\text{PSN}}/t_{\text{sim}}$]
Sampling from generative model	0.4	10.7	26.8
Sampling using IC	0.4	4.2	10.5

typically set to zero, but could be learnable parameters. The unique first address a_1 (which is guaranteed to be unique as the program has to start somewhere) is fed to the PSN and the surrogate program starts its execution. At each subsequent time step t the PSN produces a sample x_{a_t} and address a_{t+1} , which then propagates the PSN forward where the structure is build in a “just-in-time” manner until an “end execution” address is sampled.

3.2 Performing inference

As the PSN mirrors the execution of the original simulator, including the address transitions as described above, the PSN has the capability to replicate the communication interface between the PSN (now the program) and various inference engines found in probabilistic programming, in particular IC. Figure 2 illustrates exactly how inference in PSN corresponds to inference in the model using IC (and therefore the same inference network). As the execution of the simulator tends to be the computational bottleneck in inference, replacing the simulator with a PSN can yield significant speedups.

4 Experiments

We showcase the capabilities of PSNs using a real-world process simulation of composite materials for which the PSNs accurately learn to approximate the generative model. We then show that performing inference in the original generative model and surrogate model respectively yields indistinguishable posteriors, with inference in the PSN being an order of magnitude faster than performing inference in the original simulator and thereby allows for faster amortized inference.

4.1 Process simulation of composite materials

At its most basic level, composite manufacturing involves an uncured composite material being laid up onto a tool, of known material, which are then placed in an autoclave where a predefined pressure and heat-

ing cycle is imposed. Advanced composite materials are composed of a relatively inert reinforcing fibre combined with an initially fluid resin (Tuttle, 2003). The combined material will undergo some imposed cycle to transform the fluid resin into a stiff solid; typically, this involves an autoclave cure which will drive the resin’s exothermic polymerization from a collection of monomers into a fully-connected macromolecule (Pascault et al., 2002).

The current state-of-the-art in process simulation involves deterministically solving the thermo-chemical behaviour of hybrid composite structures under multi-stage processing. The specific simulator used in this work, RAVEN, is used in the aerospace and automotive industries to evaluate key performance metrics for part design with the ultimate goal of decreasing manufacturing cost (Convergent Manufacturing Technologies, 2019). However, simulation of composites processing remains a technical challenge due to the complex physical transformation of the material and the often uncertain boundary conditions (Zobeiry et al., 2018). Full 3D physics-based simulation of large structures can take on the order of weeks to run leading to less efficient designs. Therefore, taking advantage of symmetries, 1D through thickness simulations provide useful insights into part response and can be used as an accurate approximation to the full 3D simulation when analyzed far from the material edges. For a 1D analysis, the heat transfer problem can be solved if the heating cycle, the heat transfer coefficients, and the composite and tool material properties are known (Rasekh et al., 2004). However, while these 1D simulation are significantly faster to compute than their 3D counterparts, at several seconds per run, they remain a bottleneck for inference purposes.

The specific problem at hand is the measurement of the internal temperature of a composite material during processing. Typical processing conditions require tight tolerances, less than a few degrees Celsius (Hexcel Composites, 2010), to ensure the composite will reach a high enough temperature to yield a desired performance metric, while simultaneously dissipating the exothermic reaction energy so that the polymer does

Table 2: We estimate $\hat{\mu}_w \approx \mathbb{E}_{p(\mathbf{x}_{\text{lat}}|\mathbf{x}_{\text{obs}})}[\mu_w]$ under the posterior using SIS denoted GT, IC in simulator and IC in PSN using 15000 traces, and report the associated effective sample size (ESS). We provide six different estimations; three posteriors and three observations $\mathbf{x}_{\text{obs}}^-$, $\mathbf{x}_{\text{obs}}^{\text{nominal}}$, and $\mathbf{x}_{\text{obs}}^+$. We observe that the PSN estimates matches that of the GT and IC in simulator, albeit with a slight overestimation when conditioning on $\mathbf{x}_{\text{obs}}^+$.

	$\mathbf{x}_{\text{obs}}^-$		$\mathbf{x}_{\text{obs}}^{\text{nominal}}$		$\mathbf{x}_{\text{obs}}^+$	
	$\hat{\mu}_w$	ESS	$\hat{\mu}_w$	ESS	$\hat{\mu}_w$	ESS
GT	204.90	259	204.46	304	203.82	399
IC in Simulator	204.94	378	204.44	390	203.84	235
IC in PSN	204.87	399	204.31	186	204.02	121

not degrade. Physical observations of the material’s internal temperature during manufacturing are expensive and manufacturers would prefer to infer the internal state of the material given less expensive external measurements. Inferring the part’s internal temperature from external observations of the surface temperature using imposed latent variables is of great value to advanced composite manufacturing industry.

To solve this task we consider a generative model describing the heating process written in PyProb with the aim of inferring the latent internal temperatures of the part using IC. As even the 1D simulator imposes a computational bottleneck, we will use PSN to speedup inference while preserving highly accurate inference.

The generative model defines a joint distribution over heat coefficients, thicknesses and various temperatures associated with the heating process. We leave out the explicit details of the generative model for proprietary reasons and will discuss the problem purely in terms of the description of the variables.

We continue by denoting the set of latent input parameters $\mathbf{x}_{\text{input}}$ (heat coefficients and thicknesses), the observed temperature configuration $\mathbf{x}_{\text{temp.config}}$, observed air temperatures \mathbf{x}_{air} , observed tool temperature (measured at the bottom surface of the tool) $\mathbf{x}_{\text{temp.bot}}$ (see blue boxes in Figure 1), and latent internal temperature profiles $\mathbf{x}_{\text{temp.internal}}$. We summarize the latent variables $\mathbf{x}_{\text{lat}} = \{\mathbf{x}_{\text{input}}, \mathbf{x}_{\text{temp.internal}}\}$ and the observed variables $\mathbf{x}_{\text{obs}} = \{\mathbf{x}_{\text{temp.config}}, \mathbf{x}_{\text{air}}, \mathbf{x}_{\text{temp.bot}}\}$ such that $\mathbf{x} = \mathbf{x}_{\text{obs}} \cup \mathbf{x}_{\text{lat}}$.

To evaluate the quality of the PSN we consider the conditional expectation of $\mathbf{x}_{\text{RAVEN}} = \{\mathbf{x}_{\text{temp.internal}}, \mathbf{x}_{\text{temp.bot}}\}$ conditioned on $\mathbf{x}_{\text{settings}} = \{\mathbf{x}_{\text{temp.config}}, \mathbf{x}_{\text{input}}\}$ under (1) the model $\mathbb{E}_{p(\mathbf{x}_{\text{RAVEN}}|\mathbf{x}_{\text{settings}})}[\mathbf{x}_{\text{RAVEN}}]$ (expected RAVEN output given the fixed input parameters $\mathbf{x}_{\text{input}}$ and temperature configuration $\mathbf{x}_{\text{temp.config}}$) and (2) the PSN $\mathbb{E}_s(\mathbf{x}_{\text{RAVEN}}|\mathbf{x}_{\text{settings}})[\mathbf{x}_{\text{RAVEN}}]$. Figure 1 shows that our PSN produces outputs indistinguishable from those from the simulator, and Figure 4 confirms

negligible expected squared errors. Small peaks are, however, observed at around time $t = 160$ min, as well as towards the end of the heating process, which is where the internal temperature exhibits the most rapid changes.

To evaluate the quality of performing inference (using IC) in the PSN we consider the function $f(\mathbf{x}_{\text{lat}}) = \mu_w$, where μ_w is the empirical mean of $\mathbf{x}_{\text{RAVEN}}$ across the time window $w = [155 \text{ min}, 165 \text{ min}]$ (chosen to be close to peak temperatures) and at a fixed depth 30 mm (chosen to be somewhere near the upper quarter of the tool/material), see Figure 1.

We then estimate $\hat{\mu}_w \approx \mathbb{E}_{p(\mathbf{x}_{\text{lat}}|\mathbf{x}_{\text{obs}})}[f(\mathbf{x}_{\text{lat}})] = \mathbb{E}_{p(\mathbf{x}_{\text{lat}}|\mathbf{x}_{\text{obs}})}[\mu_w]$ using IC with the same inference network $q(\mathbf{x}_{\text{lat}}|\mathbf{x}_{\text{obs}})$ used for performing inference in both the surrogate and the model. As a ground truth posterior, we employ SIS where the proposal distribution is the prior $q(\mathbf{x}_{\text{lat}}|\mathbf{x}_{\text{obs}}) = p(\mathbf{x}_{\text{lat}})$ and denote it GT. To evaluate the effect of amortized inference we consider conditioning on three different observations $\mathbf{x}_{\text{obs}}^-$, $\mathbf{x}_{\text{obs}}^{\text{nominal}}$, and $\mathbf{x}_{\text{obs}}^+$ each corresponding to an observation produced by the simulator with input values and temperature settings well below, equal to, and well above the nominal values respectively.

In all cases inference is performed using 15000 traces and we summarize the results in Table 2, which shows that performing inference in the PSN yields the same results as inference in the simulator (with the exception that when observing $\mathbf{x}_{\text{obs}}^+$ the PSN slightly overestimates $\hat{\mu}_w$). Further, to get a sense of how our traces are distributed we show in Figure 3 boxplots representing the posterior distribution from which we estimate $\mathbb{E}_{p(\mathbf{x}_{\text{lat}}|\mathbf{x}_{\text{obs}})}[\mu_w]$. The results confirm that inference in the PSN yields similar posteriors and expectations compared to inference in the simulator (with the posterior shifted slightly when observing $\mathbf{x}_{\text{obs}}^+$).

We also find that posterior inference is sensitive to different observations with a resolution of about $\sim 0.5^\circ\text{C}$, which is required for the purpose of assessing whether the internal heat conditions of the material fall within some specified tolerances. The advantage of using the

PSN is that we maintain high accuracy in the posterior estimations with a speedup factor of **10.5**, see Table 1. Further, in cases where we simply seek to produce faster simulations (not for the sake of inference), the PSN provides an even greater speedup factor of **26.8**. The additional speedup is due to dropping the overhead of needing to perform inference.

5 Related Work

As far as the authors of this paper are aware, PSNs is the first framework for learning surrogate models in a model-agnostic way. While the idea of learning trace execution using LSTMs has been seen before, e.g., neural programmer-interpreters (NPI) (Reed and De Freitas, 2015), our approach differs in two key areas: (1) NPIs are trained to predict the sequence of called subroutines used to solve specific tasks like sorting or image rotation. As such there exists no model which NPIs explicitly aim to replicate, and so NPIs are not suitable for inference as the tasks need to be known a priori. (2) Where PSNs model the entire simulator, NPIs make no attempt to abstract away the predicted subroutines – i.e. if any subroutine causes a computational bottleneck, NPIs cannot decrease the computational cost.

6 Conclusions

We have proposed a novel approach to surrogate modeling that models not only the distributions in stochastic simulators but the stochastic structure of the simulator itself. We call our surrogates *probabilistic surrogate networks* and have discussed how our approach is necessary for producing surrogates for arbitrary simulators with e.g. stochastic control flow. Using a real-world process simulation of composite materials as an example, we show that our approach provides significant computational speedups in inference problems relying on evaluating the joint distribution while preserving highly accurate inference results indistinguishable from the ground truth. We believe that our approach can have major impact across academic and industrial fields making substantial use of computationally costly simulators, where fast repeated inference would provide new analytical opportunities.

Acknowledgements

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), the Canada CIFAR AI Chairs Program, Compute Canada, Intel, and DARPA under its D3M and LWLL programs.

References

- Alam, F. M., McNaught, K. R., and Ringrose, T. J. (2004). A comparison of experimental designs in the development of a neural network simulation meta-model. *Simulation Modelling Practice and Theory*, 12(7-8):559–578.
- Baydin, A. G., Heinrich, L., Bhimji, W., Gram-Hansen, B., Louppe, G., Shao, L., Cranmer, K., Wood, F., et al. (2018a). Efficient probabilistic inference in the quest for physics beyond the standard model. *arXiv preprint arXiv:1807.07706*.
- Baydin, A. G. and Le, T. A. (2018). *pyprob*. <https://github.com/probprog/pyprob>.
- Baydin, A. G., Le, T. A., Heinrich, L., Bhimji, W., Cranmer, K., and Wood, F. (2018b). *ppx*. <https://github.com/pyprob/ppx>.
- Baydin, A. G., Shao, L., Bhimji, W., Heinrich, L., Meadows, L., Liu, J., Munk, A., Naderiparizi, S., Gram-Hansen, B., Louppe, G., et al. (2019). Etalumis: Bringing probabilistic programming to scientific simulators at scale. *arXiv preprint arXiv:1907.03382*.
- Biannic, J., Hardier, G., Roos, C., Seren, C., and Verdier, L. (2016). Surrogate models for aircraft flight control: some off-line and embedded applications. *AerospaceLab Journal*, (12):pages–1.
- Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P., Horsfall, P., and Goodman, N. D. (2018). Pyro: Deep universal probabilistic programming. *Journal of Machine Learning Research*.
- Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., and Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of statistical software*, 76(1).
- Convergent Manufacturing Technologies (2019). RAVEN Simulation Software. Technical report, Vancouver.
- Endeve, E., Cardall, C. Y., Budiardja, R. D., Beck, S. W., Bejnood, A., Toedte, R. J., Mezzacappa, A., and Blondin, J. M. (2012). Turbulent magnetic field amplification from spiral sasi modes: Implications for core-collapse supernovae and proto-neutron star magnetization. *Astrophysical Journal*, 751(1):26.
- Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017). Neural message passing for quantum chemistry. *34th International Conference on Machine Learning, Icm1 2017*, 3:2053–2070.
- Glaz, B., Liu, L., and Friedmann, P. P. (2010). Reduced-order nonlinear unsteady aerodynamic

- modeling using a surrogate-based recurrence framework. *Aiaa Journal*, 48(10):2418–2429.
- Goodman, N. D., Mansinghka, V. K., Roy, D., Bonawitz, K., and Tenenbaum, J. B. (2008). Church: A language for generative models. *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence, Uai 2008*, pages 220–229.
- Goodman, N. D. and Stuhlmüller, A. (2014). The Design and Implementation of Probabilistic Programming Languages. <http://dippl.org>.
- Gordon, A. D., Henzinger, T. A., Nori, A. V., and Rajamani, S. K. (2014). Probabilistic programming. *Future of Software Engineering, Fose 2014 - Proceedings*, pages 167–181.
- Hamdia, K. M., Silani, M., Zhuang, X., He, P., and Rabczuk, T. (2017). Stochastic analysis of the fracture toughness of polymeric nanoparticle composites using polynomial chaos expansions. *International Journal of Fracture*, 206(2):215–227.
- Heinecke, A., Breuer, A., Rettenberger, S., Bader, M., Gabriel, A. A., Pelties, C., Bode, A., Barth, W., Liao, X. K., Vaidyanathan, K., Smelyanskiy, M., and Dubey, P. (2014). Petascale high order dynamic rupture earthquake simulations on heterogeneous supercomputers. *International Conference for High Performance Computing, Networking, Storage and Analysis, Sc*, 2015-(January):7012188, 3–14.
- Hexcel Composites (2010). HexPly 8552 product data. Technical report, Stamford.
- Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8):1735–1780.
- Hu, Z., Yang, Z., Salakhutdinov, R., and Xing, E. P. (2017). On unifying deep generative models. *arXiv preprint arXiv:1706.00550*.
- Hussain, M. F., Barton, R. R., and Joshi, S. B. (2002). Metamodeling: radial basis functions, versus polynomials. *European Journal of Operational Research*, 138(1):142–154.
- Jeong, S., Murayama, M., and Yamamoto, K. (2005). Efficient optimization design method using kriging model. *Journal of aircraft*, 42(2):413–420.
- Khu, S.-T. and Werner, M. G. (2003). Reduction of monte-carlo simulation runs for uncertainty estimation in hydrological modelling. *Hydrology and Earth System Sciences Discussions*, 7(5):680–692.
- Le, T. A., Baydin, A. G., and Wood, F. (2017). Inference compilation and universal probabilistic programming. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1338–1348, Fort Lauderdale, FL, USA. PMLR.
- Lunn, D., Spiegelhalter, D., Thomas, A., and Best, N. (2009). The bugs project: Evolution, critique and future directions. *Statistics in medicine*, 28(25):3049–3067.
- Mansinghka, V., Selsam, D., and Perov, Y. (2014). Venture: a higher-order probabilistic programming platform with programmable inference. page 78.
- Milch, B., Marthi, B., Russell, S., Sontag, D., Ong, D. L., and Kolobov, A. (2005). Blog: Probabilistic models with unknown objects. *Ijcai International Joint Conference on Artificial Intelligence*, pages 1352–1359.
- Minka, T., Winn, J., Guiver, J., Zaykov, Y., Fabian, D., and Bronskill, J. (2018). /Infer.NET 0.3. Microsoft Research Cambridge. <http://dotnet.github.io/infer>.
- Mullur, A. A. and Messac, A. (2006). Metamodeling using extended radial basis functions: a comparative approach. *Engineering with Computers*, 21(3):203.
- Pascault, J.-P., Sautereau, H., Verdu, J., and Williams, R. J. (2002). *Thermosetting polymers*, volume 477. Marcel Dekker New York.
- Perdikaris, P., Grinberg, L., and Karniadakis, G. E. (2016). Multiscale modeling and simulation of brain blood flow. *Physics of Fluids*, 28(2):021304.
- Pfeffer, A. (2009). Figaro: An object-oriented probabilistic programming language. *Charles River Analytics Technical Report*, 137:96.
- Raberto, M., Cincotti, S., Focardi, S. M., and Marchesi, M. (2001). Agent-based simulation of a financial market. *Physica A: Statistical Mechanics and Its Applications*, 299(1-2):319–327.
- Rasekh, A., Vaziri, R., and Poursartip, A. (2004). Simple Techniques for Thermal Analysis of the Processing of Composite Structures. *36th International SAMPE Technical Conference*, page 11.
- Razavi, S., Tolson, B. A., and Burn, D. H. (2012). Review of surrogate modeling in water resources. *Water Resources Research*, 48(7):W07401.
- Reed, S. and De Freitas, N. (2015). Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279*.
- Regis, R. G. and Shoemaker, C. A. (2007). A stochastic radial basis function method for the global optimization of expensive functions. *INFORMS Journal on Computing*, 19(4):497–509.
- Rikards, R., Abramovich, H., Auzins, J., Korjakins, A., Ozolinsh, O., Kalnins, K., and Green, T. (2004).

- Surrogate models for optimum design of stiffened composite shells. *Composite Structures*, 63(2):243–251.
- Sacks, J., Welch, W. J., Mitchell, T. J., and Wynn, H. P. (1989). Design and analysis of computer experiments. *Statistical science*, pages 409–423.
- Simpson, T. W., Mauery, T. M., Korte, J. J., and Mistree, F. (2001). Kriging models for global approximation in simulation-based multidisciplinary design optimization. *Aiaa Journal*, 39(12):2233–2241.
- Tompson, J., Schlachter, K., Sprechmann, P., and Perlin, K. (2017). Accelerating eulerian fluid simulation with convolutional networks. *34th International Conference on Machine Learning, Icm1 2017*, 7:5258–5267.
- Tran, D., Kucukelbir, A., Dieng, A. B., Rudolph, M., Liang, D., and Blei, D. M. (2016). Edward: A library for probabilistic modeling, inference, and criticism. *arXiv preprint arXiv:1610.09787*.
- Tuttle, M. E. (2003). *Structural analysis of polymeric composite materials*. Crc Press.
- van de Meent, J.-W., Paige, B., Yang, H., and Wood, F. (2018). An introduction to probabilistic programming. *arXiv preprint arXiv:1809.10756*.
- Veldman, A. E., Gerrits, J., Luppens, R., Helder, J. A., and Vreeburg, J. P. (2007). The numerical simulation of liquid sloshing on board spacecraft. *Journal of Computational Physics*, 224(1):82–99.
- Willcox, K. and Megretski, A. (2005). Fourier series for accurate, stable, reduced-order models in large-scale linear applications. *SIAM Journal on Scientific Computing*, 26(3):944–962.
- Wingate, D., Stuhlmüller, A., and Goodman, N. D. (2011). Lightweight implementations of probabilistic programming languages via transformational compilation. *Journal of Machine Learning Research*, 15:770–778.
- Wood, F., Meent, J. W., and Mansinghka, V. (2014). A new approach to probabilistic programming inference. In *Artificial Intelligence and Statistics*, pages 1024–1032.
- Wu, Z., Huang, Y., Chen, X., Zhang, X., and Yao, W. (2018). Surrogate modeling for liquid-gas interface determination under microgravity. *Acta Astronautica*, 152:71 – 77.
- Yamazaki, W. and Mavriplis, D. J. (2013). Derivative-enhanced variable fidelity surrogate modeling for aerodynamic functions. *Aiaa Journal*, 51(1):126–137.
- Zobeiry, N., Park, J., and Poursartip, A. (2018). An infrared thermography-based method for the evaluation of the thermal response of tooling for composites manufacturing. *Journal of Composite Materials*, page 002199831879844.