My primary research interests are at the intersection of artificial intelligence, machine learning, and programming languages. The unifying theme of these research interests is probabilistic modeling: designing systems that represent and automatically reason about probabilistic uncertainty. Modern machine learning heavily emphasizes scaling with data: enormous datasets have led to impressive developments in fields like computer vision and natural language processing. However, there are blind spots in this purely data-driven approach. For instance, many prominent machine learning models are not very interactive: they cannot easily explain how they arrived at a conclusion, or update their predictions given input from a user. I focus on scaling the model: I want to give users the power to describe the state of the world, in all its complexities, and reason about its uncertainties probabilistically. This approach neatly complements data-driven learning algorithms by putting the user back in charge: models will require less data and are more intuitive to understand because the user is in the loop driving the process. There are two key needs in current approaches to probabilistic modeling that I attack in my research:

Richer modeling languages. The world is complex, and describing it requires a rich vocabulary. Every probabilistic model has a modeling language: this is the user interface through which a human being describes the world to the automated reasoning system. There is a need for modeling languages that are (1) accessible to a broad swathe of users, and (2) flexible enough to allow a user to express nuanced information about the world to the system.

Scalable inference algorithms. Once the user has described the world, the next thing they will want is to ask questions: what is the probability, according to my model, that the world will behave a certain way? Answering these queries often takes the form of Bayesian probabilistic inference, and answering such queries is computationally hard in general. There is no universal approach to probabilistic inference, so there is a need for new diverse forms of inference that enable the development of more sophisticated and descriptive models.

My research is at the forefront of the unification of AI with programming languages. The marriage of these two fields is a probabilistic programming language (PPL): a programming language that is itself a probabilistic model. I develop new PPLs: Dice, a PPL that I developed, received an ACM SIGPLAN Distinguished Paper Award. I develop new inference algorithms, drawing on insights from programming languages – such as predicate abstraction [5]-[8] – and insights from AI, such as tractable probabilistic modeling and lifted inference [6],[17]. And finally, I apply PPLs in new and surprising problem domains, such as classical simulation of quantum systems [15]. Taken in sum these advances expand the domain of probabilistic modeling. Long-term, I see PPLs as being a standard tool in the toolbox of every programmer.

Now I will discuss in detail my work on probabilistic programming languages, tractable probabilistic reasoning, and future ambitions.

1 Probabilistic Programming

The world has few experts in probabilistic modeling: deploying probabilistic models still requires advanced AI expertise, and that technology is not accessible to most scientists or engineers. However, the world has many programmers. Probabilistic programming extends programming languages – a concept many people intuitively understand – with notions of probabilistic choice and Bayesian conditioning, allowing programmers to create probabilistic models in a language that they are comfortable with. There are many popular PPLs – such as Stan, BLOG, Pyro, Venture, Church, Anglican, Turing, ProbLog etc. – that are
popular within the machine learning and AI communities. Despite its immense promise, the development of PPLs as a general tool has been plagued by scalability challenges: people often tend to write models that are more complex than the included inference algorithms are capable of automatically reasoning about. Hence, one of the key challenges in making probabilistic programs widely useful is scaling inference.

I will describe the efforts made during my PhD. towards scaling inference to large probabilistic programs. A central theme is combining insights from the PL community – such as abstract interpretation and symbolic model checking – with insights from the AI community – such as knowledge representation and probabilistic graphical models.

The Dice PPL There is a critical deficiency in many PPLs in widespread use today: they cannot handle discreteness. Many problem domains are naturally discrete, including graphs, text, computer network reliability, genomics, and countless more. I developed Dice as a standalone probabilistic programming system in order fill this important gap in the landscape of PPL inference \[10, 11, 13\]. Dice performs exact inference on extremely large programs. An online demo for Dice is available at [dicelang.cs.ucla.edu](http://dicelang.cs.ucla.edu).

Supporting discrete distributions is one of the biggest open challenges in PPLs. We designed Dice to meet this challenge: it is a domain-specific PPL targeted at discrete probability distributions. This focus on discreteness enables an inference algorithm based on weighted model counting, a well-known strategy for exact inference in the discrete graphical models and probabilistic logic programming communities. In contrast to many existing PPL inference algorithms which are either approximate or path-based (work by enumerating all possible paths through the program), Dice factorizes the inference computation. This enables Dice to scale multiple orders of magnitude faster than existing methods on important examples from computer network verification, graph reachability, and text analysis.

During my PhD. I participated in acquiring funding in order to pursue this work. In 2018 I assisted my advisors Todd Millstein and Guy Van den Broeck in writing an NSF grant titled Opening Up the Black Box of Probabilistic Program Inference, a 5-year $947,397 grant. In 2019 and 2020 we received the Facebook Probability in Programming research award to develop Dice. In 2020 I received a UCLA Dissertation Year Fellowship, which covers my tuition and grants a $20,000 stipend.

Impact on Other Fields In order to realize the potential of PPLs as a lingua franca for probabilistic modeling and reasoning, it is essential to reach out and apply it in various domains. Dice focuses on discrete programs, which has enabled us to apply it in new areas:

- **Classically Simulating Quantum Algorithms.** The classical simulation of quantum algorithms has much in common algorithmically with the kinds of computations performed during probabilistic inference. With collaborators Yipeng Huang (Rutgers University) and Margaret Martonosi (Princeton University), we observe that with minor modifications to Dice’s inference algorithm, one can compute the output state vector of a quantum circuit. This creates a bridge between the rich literatures of automated probabilistic reasoning and classical quantum simulation. In Huang et al. \[14, 15\] we exploited this similarity to perform fast and scalable classical quantum simulation, out-performing existing specialized simulators like Google’s Cirq on important variational applications.

- **Probabilistic Model Checking.** The verification community relies on probabilistic reasoning to verify the correctness of probabilistic systems and algorithms. This community has developed a suite of probabilistic reasoning tools independently from the AI community: popular examples include Storm \[3\] and Prism \[16\]. With collaborators Sebastian Junges, Marcell Vazquez-Chanletta, and Sanjit Seshia from UC Berkeley we are exploring applying Dice to the task of performing probabilistic verification in this setting. This work has yielded an automated translation from Prism to Dice, and the initial performance evaluation demonstrates that there are interesting classes of Prism models for which Dice performs inference multiple orders of magnitude faster.

Decomposition by Abstraction Decomposition is a central idea in computer science, and is the heart of many of its most successful algorithms. For example, decomposition is the core idea in Bayesian network algorithms (tree decomposition), constraint satisfaction, and optimization (dual decomposition). Probabilistic programs are not monolithic: they are often made up of smaller, simpler parts. This motivates
decomposing inference queries: combining simple queries on small sub-programs to answer the original harder one. Solving these simpler inference queries is often easier than solving the original program: these smaller programs are easier than the original to reason about in isolation, and furthermore one can deploy heterogeneous inference strategies specialized to each sub-program.

In general automatically decomposing an arbitrary probabilistic program is difficult. Common approaches to decomposition – such as the kind deployed in graphical models – rely heavily on a graph structure that is absent in the context of programs. I asked the question: what is an alternative to a graph for decomposing probabilistic programs? This led to a new method for decomposing probabilistic program inference called decomposition by abstraction [5, 6, 7, 8]. The core idea of this work is to generalize predicate abstraction, an effective form of abstract interpretation from the PL community, into a kind of probabilistic predicate abstraction. An early version of this work was nominated by the UCLA computer science department for the Vienna Center for Logic and Algorithms Outstanding Master Thesis, and I received the UCLA Outstanding Graduating Master’s Student award in recognition of this research project. This work led to direct follow-up work extending it to other domains such as first-order logic [1].

2 Tractable Probabilistic Reasoning

Probabilistic programs enable the production of vastly more complex probabilistic models than previously possible. As these models continue to grow in complexity, there is an increasingly pressing need for modeling frameworks that support tractable probabilistic reasoning: for instance, permit the computation of Bayesian probabilistic inference queries in time that is polynomial in the size of the model. Data structures that support tractable probabilistic reasoning are known as tractable probabilistic models (TPMs). TPMs have become an increasingly important independent area of study, yielding numerous workshops and tutorials at machine learning conferences [1].

TPMs have been studied thus far primarily by the AI community, but the PL community will find the ideas exciting: a TPM can be thought of as a compilation target for probabilistic programs that supports fast (polytime) inference. Dice lays a foundation for the deep connection between TPMs and programming languages by performing inference by compiling (functional) programs into TPMs. Broadening the landscape of TPMs can now be re-imagined as developing new compilation targets for PPLs. I made two main thrusts towards this goal: (1) I developed a new form of lifted inference for probabilistic graphical models, identifying new classes of graphical models for which probabilistic inference is tractable; and (2) I identified deep connections between probabilistic circuits and determinantal point processes, two well studied but distinct TPMs.

Applying Lifted Inference in More Places Most inference algorithms – including the one presented in Dice – rely on forms of probabilistic independence in order to scale. Unfortunately, many interesting probabilistic models do not have much conditional independence, so there is a need for identifying alternative structure to be exploited in order to scale on these models. Lifted inference (LI) algorithms identify symmetry as a key property that enables efficient inference. These methods identify orbits of the distribution: sets of points in the probability space that are guaranteed to have the same probability, enabling inference strategies that scale in the number of distinct orbits. Symmetry is orthogonal to independence: ideally, an inference algorithm can exploit both.

Typically, existing (exact) lifted inference only apply when the probabilistic model is given as a relational representation such as a Markov logic network. This prevents the insights from LI from being applied in other probabilistic modeling frameworks such as probabilistic programs or graphical models. In Holtzen et al. [5, 12] I described the first exact lifted inference algorithm for arbitrary factor graphs, and showed that it can be used to perform exact inference efficiently that cause state-of-the-art exact factor graph inference algorithms to fail. Moreover, in this same line of work I described a new approximate inference algorithm based on Markov-Chain Monte Carlo (MCMC) that provably mixes rapidly in the number of orbits, the first MCMC algorithm for graphical models with this sort of guarantee. This work received invited oral presentations at the Conference on Uncertainty in Artificial Intelligence (2019) and the International Workshop on Statistical Relational AI (2017).

1See the Workshop on Tractable Probabilistic Modeling at ICML or Choi et al. [2].
Towards a Unified Theory of Tractable Probabilistic Modeling  
There are many different kinds of TPMs, which begs the question: is there a universal theory of TPMs? Might it be that all TPMs can be explained as different special cases of a more general class of models? In Zhang et al. [17] we explore the relationship between two well-known but distinct TPMs: determinantal point processes (DPPs) and probabilistic circuits (PCs). We gave a number of separation results, showing that it may be challenging to fully unify these distinct families, and paved the way for future work on unifying these two distinct topics.

3 Future Outlook

The demand for scalable, accessible, and effective probabilistic modeling and inference has never been more pressing. In many settings I see the usage of PPLs as more than just a neat application that saves some effort or work: I see it as fundamentally the right tool for the task, in the same way linear algebra is the right tool for describing quantum mechanics. In order to get to this point from where we are today, we need to improve our fundamentals: here I outline some paths towards this goal based on work I have done.

A Systems Design Approach  
Jax and Tensorflow have revolutionized the machine learning model development workflow through their ease of use and automation. A key part of this success is all the automatic optimizations that happen behind the scenes, allowing modelers to make clean, readable, and intuitive models. Optimizations for PPLs will bring this same success to probabilistic modeling. I am mentoring an undergraduate student (Ellie Cheng) and Master’s student (Meet Teraviya) on developing compiler optimizations for probabilistic programs. These have many similarities with traditional compiler optimizations – such as peephole analysis, code hoisting, etc. – but also some key differences that arise due to the probabilistic semantics. I will bring these well-known classical ideas to probabilistic program compilers.

More compilation targets enable more optimizations: hence, there is a need for further exploration of TPMs. On this front, I aim to (1) develop deeper insights into lifted inference and TPMs that enables their application to broader classes of models, and (2) more deeply integrate these insights into Dice and other commonly-used PPLs.

Abstract by Default  
Predicate abstraction is one of the most successful static program analyses: it is part of an automated pipeline at Microsoft for validating drivers. Long-term I see abstraction as being a default analysis for probabilistic programs, in the sense that, if during inference you are not trying to find a good abstraction, people might wonder why. My work on probabilistic predicate abstractions is only the first step in this direction: long-term, this process needs to become more automated, for instance with better heuristics for selecting predicates. One path to this is to draw on deep research from traditional predicate abstraction and generalize techniques such as counter-example guided abstraction refinement. Generating abstractions more easily is paramount, but another important factor is utilizing them in more effective ways. One path for this is to make inference algorithms that leverage the abstraction: for instance, by using the abstract program itself to guide a sampling process.

From PPL Applications to Fundamentals  
Probabilistic programming languages have many applications, but this is a narrow bar to clear: my goal is for PPLs to be a capable default choice for probabilistic modeling. A key part of achieving this goal is embedding within communities and identifying their needs. Here is a case study in the challenges of making PPLs a fundamental tool. Inverse cognitive reasoning – for instance, plan recognition, inferring the internal goal of an agent given observations of its behavior – is a common task from cognitive psychology and robotics. With collaborators Yibiao Zhao (MIT/UCLA), Tao Gao (MIT/UCLA), and Josh Tenenbaum (MIT) I investigated applying probabilistic programs for plan recognition in the context of robotics [4]. A challenge here was that PPLs could not scale to these plan inference problems due in part to the discreteness naturally present in the probabilistic model: this motivated future work in Dice to tackle this specific need.

Applying PPLs in more diverse domains will continue to drive insights. I see deep connections to be drawn in the fields of probabilistic verification and quantum classical simulation: the work I have participated in thus far are initial forays that I foresee blossoming into large-scale investigations.
References