Research Statement
Chang Liu

My research interests are in the areas of deep learning, security, and programming languages, as well as their intersection. I have explored how deep learning is susceptible to adversarial manipulation, and how to defend against related attacks. I have also explored how deep learning can yield highly effective solutions to traditional programming languages problems, such as code similarity detection and software synthesis. Also, I have looked at how programming languages techniques can be leveraged to improve the efficiency and effectiveness of cryptography-based secure computation. In all of these works, I have drawn ideas from multiple research areas to yield breakthrough solutions.

In the following, I will provide an overview of my research on (1) security for machine learning systems; (2) deep learning-based program analysis and synthesis; and (3) combining programming language and cryptography to enable efficient, privacy-preserving computation. I will conclude with a discussion of future research directions which aim to further build synergies among PL, ML, and security.

1 Security of Machine Learning Systems

Deep learning has made huge advances in various domains, including computer vision and natural language processing. There is an increasing trend to apply deep learning to safety- and security-sensitive applications, such as autonomous driving [4] and facial biometric authentication [22, 23]. Thus, it is important to understand the security of deep learning in the presence of attackers. My research advances the field in two directions: (1) how to perform practical attacks; and (2) how to craft rigorously analyzed defenses. In the following, I highlight some of my work.

Attacking image classifiers under a black-box threat model. Recent work has discovered that deep learning systems, especially image classification systems, can be easily fooled by adversarial examples [9, 24]. That is, given a machine learning model and a source image, an attacker can add a human-imperceptible perturbation to the image that can mislead the model to predict an irrelevant label. To launch the attack, the attacker typically requires knowledge of the architecture and the parameters of the victim model. However, this information is hard to acquire, and thus it is unclear how dangerous adversarial attacks are in practice.

I studied the problem of whether we can launch the attack under a black-box threat model. That is, the attacker is assumed to have no knowledge about the victim model at all, but is only provided with a query interface. According to a phenomenon documented as transferability [20, 21], an adversarial example generated against one model remains adversarial to another model. Thus, an attacker can train a model on his side, and rely on transferability to generate adversarial attacks that can transfer to the victim black-box model.

However, this simple approach is not effective for generating targeted adversarial examples: the generated attack generally cannot be recognized as the target label specified by the attacker. We develop a novel technique to generate the targeted attack instances not only against one model, but also against many others. Thus the generated targeted attacks are guaranteed to transfer among multiple models on the attacker’s side, and thus are more likely to transfer to the victim black-box model as well. Using this approach, we successfully generated targeted adversarial examples to attack a commercial website, i.e., clarifai.com. Ours is the first work [17] demonstrating that black-box attacks are possible even when the attacker has no knowledge of and no access to either the victim system or the training dataset during the adversarial example generation phase. Our work [17] is regarded as the state-of-the-art black-box attacks [8, 25].

Defense against data poisoning attackers. We now consider an attacker who can compromise machine learning systems through data poisoning, i.e., injecting maliciously crafted samples into the training dataset. The need for a large amount of data to train effective models typically demands that practitioners hire crowd-sourced workers or an in-house data collecting team. Such a practice also leaves a door to data poisoning attack.

We studied how to robustly learn a linear regression model under a data poisoning attacker [15]. That is, how to train an effective model when a percentage of the training data can be arbitrarily manipulated by an attacker. Our basic idea is to use a combination of dimension reduction based on Principle Component Analysis (PCA) and a linear regressor. However, both of these two procedures can be the target of the data poisoning attackers as well. Existing works have considered robust PCA and robust linear regression separately, and mostly assume that the mean and co-variance of pristine data’s distribution is known, which is unrealistic in practice.
Our work avoids all such assumptions and provides defenses with a formal guarantee about the worst-case performance. Our contributions are three-fold: (1) we proved a necessary-and-sufficient condition to characterize when the dimension reduction can be perfectly solved regardless of the presence of data poisoning; (2) we proposed a Trimmed Principle Component Regression (T-PCR) algorithm and proved a formal guarantee on the worst case performance of the model learned using T-PCR; and (3) we demonstrated that our implementation is both efficient and effective. Our paper [15] won the best paper award of AISec 2017.

2 Deep Learning-based Program Analysis and Synthesis

Software plays a central role in the computing world. Modern security research and practice heavily relies on program analysis approaches to detect, analyze, and eventually fix software vulnerabilities. Despite the decades of development of formal methods to rigorously analyze and process programs, some problems are hard to define formally and have defied a general solution. For example, there is an increasing need to detect whether two pieces of a program are similar or not, but a definition of program similarity is typically application dependent, and so the solution for one application may not solve another. Moreover, a long-standing open problem is to translate natural-language descriptions into programs, but since the semantics of natural language is quite complex, formally defining the translation problem is hard. In these examples, machine learning can provide a means to tackle the problem without the need of a formal definition. In the following, I highlight my work on binary code similarity detection and program synthesis.

Neural network-based code similarity detection. Code similarity detection is a core problem for many applications, such as plagiarism detection and vulnerability detection. Existing approaches to code similarity detection typically rely on manually crafted heuristics. In particular, most existing approaches represent a binary program as its control-flow-graph (CFG), and employ a graph matching-based algorithm to detect whether two CFGs are similar or not. We observe two inevitable drawbacks to such approaches: (1) one heuristic cannot fit all application scenarios; and (2) graph matching-based algorithms are slow.

To overcome these issues, we proposed the first neural network-based approach to solve the code similarity detection problem [26]. In particular, we employed a neural network to convert a CFG into a numeric vector called an embedding. Detecting whether two pieces of code are similar can be solved by computing the distance between their embeddings and checking if the distance is small, i.e., below a threshold.

The key challenge is how to effectively train the network. We developed a Siamese network-based training approach. In particular, instead of using only one network, we combined two neural networks sharing the same parameters. We construct a training data set of binary code pairs. Each pair is labeled as positive if the two pieces of code are compiled from the same source, or negative otherwise. In doing so, we can train the network to distinguish similar code pairs from dissimilar code pairs. Also, our approach can be adapted to different application scenarios by providing different training samples to label similar and dissimilar code pairs. Our approach can improve prior art’s efficiency by 3 to 4 orders of magnitude, and also identify 200% more vulnerabilities (42 versus 14 among top 50 results).

Program synthesis. Programming is on its way to becoming ubiquitous not only for well-trained program developers, but also for non-professionals, such as data analysts in non-IT industry or even general users. As a result, inventing novel programming methods to facilitate program development is attracting increasing interests. One direction of my research is to study how to translate a natural language description into a program. We develop novel deep learning architectures to capture the syntactic features of target programming languages as well as properties of the input natural language.

For example, IFTTT.com provides an interface to allow users to specify a trigger and an action to form a recipe, so that when the trigger event happens, the action is automatically performed. A recipe is also called an If-Then program, and the If-Then program synthesis task is to translate the natural language description of a recipe into the recipe itself. We proposed a latent attention network to achieve a test accuracy of 87.5% [5], outperforming not only the prior art by 5 points, but also all followup work [1, 28] on the same dataset by 9 points.

Translating natural language into SQL queries is attracting attention from both academia and industry. Prior solutions mainly studied data sets with at most a few thousand samples, so it was unclear whether they can generalize to large, unseen scenarios. Recently, Zhong et al. [29] provided the first large scale natural language to SQL dataset, called WikiSQL. We proposed a sketch-based SQL synthesis approach outperforming the prior art on this dataset by 9
points [27]. We are at the top of the leadership board of WikiSQL.\footnote{https://github.com/salesforce/WikiSQL}

**Remarks.** So far, my works mainly focus on applying deep learning to solve PL problems. While deep learning has improved the solutions to many problems that can be empirically evaluated, many other scenarios require a certificate of “free-of-bug”. Most of them have to resort to a formal approach such as verification or type systems. In Section 4, I will highlight my future research to apply PL techniques to improve deep learning research.

# 3 Trace Oblivious Computation

My PhD research significantly advances a direction called trace oblivious computation, which is motivated by privacy-preserving computing. Consider the problem of computing on data owned by two or more parties who do not trust each other. Conceptually, privacy-preserving computing systems solve this problem by instantiating a “secure” abstract machine consisting of a CPU and encrypted memory, so that an adversary cannot learn information through either the computation within the CPU or the data in the memory. Unfortunately, evidence has shown that side channels (e.g. memory accesses, timing, and termination) in such a “secure” abstract machine may potentially leak highly sensitive information, including cryptographic keys that form the root of trust.

Along with my collaborators, I introduced trace oblivious computation. It constitutes a programming language approach to formally enforce that no information about sensitive information leaks through execution traces that characterize observable side channels. My work introduces and lays the foundation for three research directions: (1) secure and efficient co-processors for enabling trace oblivious computation with a compiler; (2) efficient RAM-model secure computation; and (3) verifying the security of an efficient ORAM implementation.

**Memory trace oblivious (MTO): a theoretical foundation.** Consider a physical attacker that can observe all communications outside the processor. We can rely on cryptographic Oblivious RAM (ORAM) protocols to ensure that neither the data nor the addresses transmitted on the bus leak sensitive information to the attacker, while paying the cost of a poly-logarithmic overhead (in the memory size). We observed that a program with memory access pattern not depending on secret inputs could avoid paying the ORAM overhead. From this observation, I developed a theory to describe programs that enjoy the memory trace oblivious property [13] while making selective use of ORAM, i.e., only when needed. Our paper won the 2013 NSA Best Scientific Cybersecurity Paper Award.

**GhostRider: A MTO secure co-processor.** Based on our theoretical results, we built a hardware-compiler co-designed system, called GhostRider [12]. GhostRider customizes a RISC-V ORAM processor (implemented as an FPGA), and includes a compiler to take advantage of the customized hardware. We implemented several optimizations to ensure compiled programs are efficient, using a novel typed assembly language to ensure the optimized code remains MTO. Evaluation results on several benchmark programs showed that GhostRider achieved up to $10^6 \times$ speedup over previous oblivious systems such as Phantom [18]. The paper [12] won the Best Paper Award at ASPLOS 2015.

**ObliVM: practical RAM-model secure computation.** Secure computation (SC) provides a cryptography-based alternative to the privacy-preserving computing problem. To leverage a SC solution in a practical setting, a computation which is typically written in a RAM-model language needs be compiled into a circuit. Prior work mostly incurs an overhead linear to the memory size to each random memory access, and is thus impractical. To mitigate the issue, my collaborators and I developed ObliVM [14, 16]. ObliVM is a programming framework to enable efficient translation from a program written in a RAM-based language into an efficient SC protocol. ObliVM provides efficient implementations for several frequently used sub-modules, such as ORAM and oblivious data structures, and the ObliVM compiler can detect the most efficient sub-modules to use during compilation by analyzing the source code.

ObliVM greatly improves the developers’ efficiency and the generated protocols’ efficiency. For example, the secure ridge regression implementation [19] took the authors 15 person-weeks to develop, while its ObliVM counterpart took only 2 person-hours, and achieved $3 \times$ smaller circuits sizes. On some standard benchmarks, ObliVM can achieve a speedup of up to $10^6 \times$ from the previous state-of-the-art system. ObliVM [16] won the First Place in the Best Applied Cyber Security Research Paper Competition at CSAW 2015. I have open-sourced ObliVM at http://www.oblivm.com, and it has enabled researchers to efficiently prototype and evaluate their ideas [3, 10].
A theory to enforce the security of ORAM implementation. My earlier research treated ORAM as a black-box primitive. While we built ObliVM we started to consider opening the black-box to verify the security of an ORAM implementation. We have developed a novel formal language and type system to do this [7]. The main challenge is developing a sufficient type-enforced invariant for reasoning about the independence of random numbers used by the ORAM algorithm (and those of related, oblivious data structures) while enabling sufficient expressiveness. This work provides the first foundation on which to formally verify the security of an efficient ORAM implementation.

4 Ongoing and Future Direction

In the future, I plan to continue investigating different issues at the intersection of deep learning, security, and programming languages. In the following, I highlight a subset of topics that attract me the most.

Novel programming interfaces to accelerate deep learning research. The recent rapid advancement in deep learning is largely due to the availability of several programming frameworks, such as Tensorflow and PyTorch. I am interested in developing the next generation of languages to support more advanced deep learning research. As an example, existing programming frameworks may not be the most suitable to support architecture search. In fact, most recent advances in deep learning are due to novel designs of neural network architectures. It is thus desirable to search for the best architecture automatically, given a dataset. I observe that one requirement of an architecture search algorithm is to ensure that every instance in the architecture search space is well-formed, e.g., the dimensions can match. Existing frameworks are mostly implemented in Python, a dynamically typed language, and thus developers typically need to implement additional functionality for well-formedness checking. This process can be tedious and error-prone. Thus, including well-formedness checking in the design of a type system can relieve developers from establishing well-formedness manually. I plan to explore more requirements for the next generation of deep learning programming frameworks as well as design choices to fulfill the demands.

Toward synthesizing complex programs using neural network approaches. Existing neural programming-by-example research mostly focuses on synthesizing simple programs ranging from array manipulations to sorting algorithms, from basic arithmetic to string operations. I am interested in considering principled solutions to more challenging problems. In my first attempt [6], I established a problem to learning a parser for a programming language whose grammar is unknown, and showed that it is more challenging than those that can be handled by existing solutions: all existing solutions failed with a 0% test accuracy. By tackling this task, I observed three key ingredients of an effective and generic solution: (1) modeling the program space using a domain-specific machine; (2) using a neural network to represent the program operating the machine; and (3) employing reinforcement learning to train the neural program. This combined approach can effectively tackle the parser synthesis task: our approach can achieve 100% test accuracy on 100× longer inputs. I plan to introduce more challenging program synthesis tasks in the future, examine the effectiveness of such a program synthesis strategy, and propose novel, principled solutions.

Verifiably robust neural networks against adversarial examples. Although many defenses against adversarial examples have been proposed, most of them were eventually broken. It is thus desirable to have a defense that can be rigorously proven to be robust against adversarial examples. Due to the non-linear and non-convex nature of deep neural networks, such an analysis had been difficult. Recently, have researchers started considering how to train a neural network while making the verification easy at the same time [2, 11]. However, these ideas were only evaluated on the MNIST dataset, i.e., the simplest dataset for deep learning. I am interested in extending this line of work to handle a more complex dataset. For example, there has been no work that can provide verifiably robust models against adversarial examples on CIFAR. One phenomenon I observe is that, different from MNIST, most robust models may not have to (or be able to) achieve an almost perfect training accuracy (e.g., ≥ 98%) on CIFAR. In this case, allowing a small portion of the training data to be misclassified may help to achieve better robustness for the rest. This idea is similar to the soft-margin trick for SVM and robust training against poisoning attacks in my work [15]. I thus plan to adopt similar ideas to see if we can build and verify a CIFAR model against adversarial examples. In a longer term, I am interested in building verifiable defense for more complex dataset and tasks.
References


