Assignment 2
WASP Software Engineering Course

Valency Oscar Colaco

October 2, 2022

Artificial Intelligence (AI) will be integral to all facets of human life in an interconnected future. These AI-based systems are capable of learning from their interactions and can perform a variety of duties ranging from brewing your morning coffee (the way you like it) to transporting you to work (in a self-driving vehicle while you catch up on your favorite show on Disney Plus). Even though advancements in AI have evolved in scale and complexity and can take advantage of new computational support, research shows that these systems are still not fully capable or reliable for real-world operations. The systems need to be dependable and resilient in cases of adversarial inputs or distributional shifts. Unlike the tools used in engineering, machine learning methods do not come with formal safety guarantees. While advances in control theory have allowed complex physical systems to be built with a proven (and guaranteed) low likelihood of failure, equivalent formal safety guarantees for AI do not exist. As a result, many AI systems cannot be completely deployed without the danger of the system encountering an unanticipated event that results in a catastrophic failure (causing loss of life). These risks exponentially increase as AI systems are widely deployed in environments where safety and security are critical.

While machine learning (ML) models are frequently deployed to strengthen cybersecurity in real-time systems by detecting zero-day attacks (AI4security), there are serious concerns regarding security threats towards the ML systems themselves (security4AI). Considering these problems, my research will investigate the dependability issues at the convergence of safety assurance, security analysis, and their complexities. The objective is to advance the knowledge of the interrelationships between safety and security issues in AI systems that utilize machine learning components. The goal is to create techniques and tools that provide evidence for safety assurances in the face of security threats towards ML-based systems.

According to NIST (the National Institute for Standards and Technology), the Software Development Life Cycle (SDLC) is defined as a formal or informal methodology for designing, creating, and maintaining software. An integral part of the SDLC is Testing & Integration in which the software is rigorously checked against requirements and then integrated into a larger system. While exhaustive software testing can be done manually when the specification has a small enough cardinality, in practice, for larger specifications, automated testing methods are used. According to Atlassian, **automated software testing** consists of evaluating the software using multiple tools with varying capabilities, ranging from isolated code correctness checks to simulating a full human-driven manual testing experience. Back when manual testing was the norm, the quality assurance process was slow, expensive and error-prone. Automated testing has the advantages of improved efficiency, productivity, better test coverage, consistent & reliable results, and it also enables continuous testing.

At the core, continuous testing is about three things – testing at earlier stages of the release pipeline (where fixing bugs are less costly), testing more often before release, and testing everywhere across various environments and devices. This process can also be integrated into the Continuous Integration - Continuous Delivery pipeline. Continuous integration is the practice of automatically integrating code changes from multiple developers or contributors into a single software project. This process enables developers to continuously merge code changes into a single repository, after which builds and tests are executed. Continuous delivery is an extension of continuous integration in which the code is automatically built and deployed to the test phase. Continuous deployment (CD) goes one step further than continuous delivery. While the code is built automatically, the update will only be rolled out to your customers if each and every code change passes all stages of your testing pipeline. A failed test will halt the release of the update. In my research, a key goal is to develop tools and techniques to provide safety assurances for ML systems. This involves the creation of formal verification software tools, developed using an agile software development methodology which includes automated testing, and by extension, continuous deployment. While this seems like a good aspect for small teams, in larger organizations, however, CD could be a challenge as it requires a complete overhaul of technical procedures, operational culture, and organizational thinking - and people usually display a strong resistance to change.

This facet of change management is a crucial aspect of project management and could be the difference between a successful and an unsuccessful project. In terms of software engineering, **project management** is the process of using milestones to align the project management aspects and system development lifecycles to report project progress. Project Managers can add value to the SDLC through improved communication between teams, better consistency of results, enhanced quality of code, complete documentation of abstract procedures, and strict compliance with standards like ISO (International Standards Organization) and CMMI (Capability Maturity Model Integration) to ensure stakeholder satisfaction. Agile project management involves the management of iterative software development projects that focus on continuous releases and cyclic updates. This style of project management can help achieve better development speeds, productivity and adaptability to market trends. In comparison, the Waterfall project management approach entails a clearly defined sequence of project phase execution. Once phases are executed, it becomes very costly to go back to fix something. Agile teams follow a similar sequence, but visit all phases in small increments with more frequent feedback loops. My research project follows the same stages in every typical project management process with appropriate constraints placed on time, scope, cost, quality, publication venues and collaborations. Also, my research aims to reduce the risk of harm towards human life in systems that use ML components. This aspect of risk management includes modelling the security threats towards the ML systems and preserving the privacy of the data used for training the model.

In general, **Security** deals with protecting access to critical infrastructure or information from attackers and unauthorized users. **Privacy**, on the other hand deals with the measures implemented to safeguard the confidentiality of the data. In terms of ML, the entire aspect of security and privacy can be specified for each stage (training & testing) of the machine learning pipeline. Data poisoning and backdoors are examples of adversarial attacks that are specific to the training phase. In data poisoning, modified data is injected into the training dataset. The model then uses this tainted data during training and develops a sensitivity to the adversarial perturbations contained within. At inference (test) time, a poisoned model will then act in an unpredictable manner causing all sorts of incorrect predictions, which could be detrimental. A specific form of data poisoning called a backdoor attack involves the adversary introducing static triggers (with fixed patterns and locations) into the training data. After training, the attacker employs such patterns during inference time to cause the ML model to act in a particular way. Attacks that focus on the model after training are known as test phase or 'inference time' attacks. The most common kind is 'model evasion', in which an attacker constructs an adversarial example by gradually adding noise to it to skew the target model's output in the direction of the desired prediction. A different category of inference-time attacks aims to obtain confidential data from the ML model. For instance, membership inference attacks employ a variety of strategies to deceive the target ML model into disclosing its training data. These kinds of attacks can be very harmful if the training data contained sensitive information like credit card numbers or passwords. My research focuses on creating techniques and tools that support the reasoning for safety assurances in ML systems in the presence of security and privacy threats.

When it comes to the **future** of software engineering, while the general feeling is that ML will 'eat' software as most software development will be the training of ML models with only the minimal 'glue' code needed to integrate these models into a larger system, I have a slightly different perspective. In simple terms, software engineering is the design and development of a program to solve a problem. When the problem at hand can be easily analyzed, and the specification is small enough, we don't really need machine learning to achieve the desired outcome. If one were insistent on using a learning algorithm as the underlying de-facto methodology for software development, we would then encounter all the issues currently associated with machine learning in terms of verification (and scalability), combinatorial explosion, robust training, adversarial perturbations, and so on. While the integration of scalable formal verification in the ML model development pipeline could help address these issues, most formal methods currently do not scale to high-dimensional data. For more complex problems, where the underlying logic is too hard to describe, and the specification is large, ML models can be deployed to help address the issue.

While software engineering, data science and machine learning are intricately interlinked, a huge domain known as 'ML for Software' has emerged in recent times. In this domain, ML models are leveraged to assist, speed-up or optimize the software development process. The most recent example would be GitHub's Co-pilot or VS Code's BlackBox Extension in which a sophisticated AI is used to automatically complete a programmer's code or suggest snippets of potential solutions. While these extensions currently cannot replace programmers entirely, they are well on their way towards generating entire programs while a human-in-the-loop oversees the process. Outside software development, AI is dominating all aspects of human life from our search engines, streaming platforms, self-driving cars, autonomous mining, navigation, online security, manufacturing, telecommunications, online shopping and even healthcare! When it comes to the arts, Jukebox is an AI that can write a song and sing it to you, while Midjourney uses diffusion models to create visually stunning images from text descriptions. Considering all these astonishing developments, it is evident that we are moving towards a world in which AI will be used on a daily basis to solve real-world problems, and software engineering and machine learning will just become another part of data science. Finally, we will also start to see the emergence of specialized branches of software development dedicated solely to the creation, implementation, maintenance, and testing of newly researched machine learning models, along with their verification and integration methods.