My research area is in applications of Graph Neural Networks (GNNs) to computer vision and time series problems. GNNs are a type of deep learning model that uses graphs as inputs instead of vectors. They make use of message passing layers which update the node values using its neighbors, and these node representations can go on to be further processed or used directly for prediction. GNNs have seen great success in a variety of tasks, such as computer vision and natural language processing. Recently I have worked on an image keypoint matching problem where the goal is to match keypoints between different images (eg. match the front tire of a truck to front tire of a car). I am currently working on transforming time series with continuous values into small graph representations. Unlike the current body of work which methods descend from the signal processing domain, I am investigating whether graph representations inspired more directly from the time series domain are effective. This involves constructing graphs based on relationships in time instead of physical locality. In this assignment I will talk about software maintenance and evolution, sustainability, and security and privacy and how they apply to my project.

From my understanding, maintenance and evolution is about essentially developing robust software where it is easy to change specific parts with minimum impact on the others. Maintenance refers to the process of fixing bugs and keeping the software operational as is, while evolution refers to somehow changing the software and its capabilities, whether that is adding new features or optimizing old ones. I think this is critical to my work, and although I do not technically deploy any software or have much of a production environment to speak of, programming with these factors in mind has been extremely useful. Designing a new model is complex and many things can change during the process, from the architecture to optimization parameters, datasets, and metrics. Sitting down and clearly separating out these areas and designing the code to be modular has made it much easier to modify and add to each segment. The amount of time saved is probably uncountable, but I feel that designing for maintenance and evolution is not given enough credit. As projects develop, I will often find that the direction no longer fit my original expectations and that my code is quickly becoming rickety and unorganized despite my best efforts. However, it is difficult to convince my supervisor that it is important for me to take a few days to sit down and reorganize the code for future development since there is no obvious forward progression being made when I could just continue hacking things together. I imagine this is only exacerbated in an industry context where there is more pressure to get a product out by a specific deadline.

Sustainability is about designing software to diminish our impact on the environment, physical and social. It is well known by now that machine learning requires intensive computing resources to develop and deploy. Models are getting deeper and are being trained with more data. With that comes more power consumption, more specialized processors which need to be manufactured and which materials need to be mined, and so on. One interesting factor about GNNs is that they are incapable of being too deep. Each layer of a GNN essentially spreads information around a graph, if there are too many layers each node will eventually trend towards the same mean value. This means that GNN models must be relatively small which means they can be trained and used on lower powered processors. This factor combined with GNNs growing success in many areas leads me to hope that GNNs may be a more environmentally sustainable alternative to conventional models. Even so, I expect the gains to be primarily limited to the model itself, unfortunately GNNs still require the large datasets common in machine learning.

Security and privacy of machine learning models has recently become the topic of an interesting debate online and offline. I primarily associate security and privacy with data in machine learning. There are many ways to interpret these terms in relation to data and software engineering in general. To me,

security represents ensuring that the data can only be accessed by authorized used, and privacy refers to protecting the rights of the subjects that appear in the data. These rights can be varied depending on the application, but primarily refer to protecting an individual's privacy preferences and their personally identifiable information. Machine learning models need lots of data in order to achieve good performance, this is undeniable. The exact provenance of this data should be hugely important, after all if you feed a model garbage data it will only produce garbage results. However, I feel that machine learning development (and especially the large corporations making waves recently) has hugely, undeniably, failed in both ensuring the security and privacy of the data they're using. Even besides the recent debate over whether developers have the right to scrape public facing but still copyrighted data to train a commercial product, I think the attitude towards privacy in general has been too cavalier. Recently someone discovered private medical photos had been used to train an image generation model, and its likely thousands more had snuck into the dataset without express permission from the subjects.

While that is an extreme example, it reminds me of when I'm looking through image datasets and finding photos of average people going about their lives. I always wonder, who are they? Do they know that their photo is in here? Would they have felt comfortable knowing a photo of them and their children are being poured over by countless researchers in one of the largest and most well-known image datasets in the world? I have no doubt that if I wanted to, it would be trivial to find out who they are. However, datasets only state that the images are scraped from some website and that the process is within the terms of service. These datasets have no security to speak of, they are public, freely accessible, and the images are completely unprocessed to hide identifying features. I honestly feel like the machine learning field is attempting to just ignore the glaring problems in security and privacy, and instead rely on obscurity. It is after all, unlikely for someone to look through all those photos of people (until they need to), and it is unlikely for them to find any inappropriate or unethically sourced images (until someone does), and it is unlikely for that someone to use these images for unintended purposes (until…). But the truth of the matter is that adding security is very inconvenient. Having these large, easily accessible, labeled datasets has been and will continue to be invaluable for development. Who knows how much research as progressed and improved with the help of these datasets, I know I would not have been able to train a keypoint matching algorithm without them. I think a real solution is much more difficult than people want to implement.

The future of machine learning will be interesting. I have hope that methods will mature and continue to improve people lives, but also misgivings about how the technology will be used to make it worse. However, I do believe that the common discourse around machine learning is hugely overblown. There is this belief going around that machine learning is going to solve everything and be able to do anything, and it is just around the corner! I think such beliefs are nonsense, yes machine learning is improving but quite frankly I don't think we have quite managed to bring it into the "real world" yet. Yes, it may be good at generating text and images, machine learning may be good at beating benchmarks, but almost every case I've heard of that requires adapting to changing, real world, circumstances end in failure. From high performance object detection algorithms failing to identify a semi-truck, to anomaly detection algorithms in nuclear reactors being completely ignored by engineers because it only produces nonsensical outputs, to doctors unwilling to use machine learning due to the uncertainty, machine learning is hardly a silver bullet. I think this will apply to software engineering too. Machine learning might be useful as a tool, but I struggle to see it truly replacing software development. As the field currently is, I don't believe there is such a thing as a perfect model. And that is all the justification I think is needed for the continued existence of software development, there are too many things in the world which cannot be allowed to fail randomly due to some quirk in the training data. Things that people will need to be able to understand and modify without needing to train a whole new and inscrutable model for it. Things that are

simple and straightforward enough where adding machine learning would only be a waste of time and resources.

Even though I believe machine learning isn't the solution to all of life's problems, it is clearly effective and will probably continue to see wide adoption especially in the service sector. Instead of becoming the primary development method, I think data science and machine learning are well on the path to becoming part of software engineering. The development pipeline is a little different from standard software engineering ones, and I have certainly felt the awkwardness of trying to fit that pipeline into a standard Agile format. However, Agile is just a framework, and it has not always existed. To me, the most reasonable approach is to consider machine learning as a branch within software engineering along with front end and back end. Since the overall pipeline is different front-end/back-end development, standard Agile practices might need to be adjusted for the machine learning branch or perhaps some new framework which can elegantly combine all three branches can be developed. I think it is important for machine learning to be considered part of and have close ties with the software engineering branch. In the end, the models will have to be used in the product somehow. Much like how a front-end engineer might not need to know everything about the back-end, but they do need to know enough about the inputs and outputs to put things together properly, the back-end engineers will need to know similar information about the models being used. Machine learning and back-end teams should be communicating and making sure changes that might affect each other are known ahead of time.