Final Report Amended

1

P

Pascale Tan Peck Hui <Pascale.tan@ssmc.com>
Thu 1/6/2022 1:09 PM

To: Tam Zher Min

- External Email -

Dear Zher Min,

I'm fine with the report. Well done!

Thanks and best regards
Pascale

EG3611A Final Report

Name: Tam Zher Min

Company: Systems on Silicon Manufacturing Company (SSMC)

Supervisor: Ms. Pascale Tan

Internship Period: Aug 23 2021 — Jan 7 2022

Table of Contents

Acknowledgements

I would like to express my deepest appreciation to SSMC for providing me this internship opportunity alongside my supervisor, Ms. Pascale, as well as my colleagues in the QRA department, Mr. Wen De and Ms. Nedlin, for their continued support throughout my internship journey. Special gratitude to my supervisor for vetting my presentations and reports as well as appreciating my efforts and trusting my abilities to advance the project independently.

Not forgetting my mentors from NUS, Professor Vincent, who attended my interim and final presentations and provided valuable comments as well as my CELC tutor, Dr. Lira, for the constant feedback and detailed deliverable breakdowns for this industrial attachment module.

Last but not least, I would also wish to profess my gratitude to whomever that may stumble upon this report, be it any future employers, employees or interns that may read this report to further understand the thought processes behind this major project that I had embarked on and hopefully, to either further extend this project or build upon the experiences gained from deploying such a system into SSMC's existing processes towards increasing automation.

1.0 Introduction

1.1 Internship Overview

This final report marks the end of my four-month journey at Systems on Silicon Manufacturing Company (SSMC) as a machine learning engineer, responsible for not only training up machine learning models that could classify images accurately, but also deploying them into a system that integrates with SSMC's existing infrastructure.

In this report, I detail down a mix of both the technical implementation and logic behind my decisions alongside some of the challenges that I have faced along the way. Overall, this report will largely focus on the second part of my internship, serving as a continuation to my interim report.

Zooming out to the macro picture, these final months at SSMC saw me writing the entire code base from ground up independently for my "Automatic Defect Classification System", or ADCS, for brevity. Creating the system from scratch was no easy feat, demanding heavy investments in time and effort planning, solving, and testing the system. What was produced at the end is a seamless flow of data and logic, painstakingly enabled by my 2000 lines of code [1].

1.2 Recap of First Half of Internship

To recap on some of the details previously presented in the interim report, the first half of this internship leaned heavily on understanding the domain of semiconductor quality assurance alongside learning about the technical implementations of machine learning onto such problems.

Noted in the interim report, the overarching problem statement for this entire project was always this: to "set up or create an Automatic Defect Classification (ADC) system" that can classify and filter out defects found either on the back or edges of completed wafers (excluding wafer frontside). In the first half, I managed to train performant convolutional neural networks (CNNs) that could accurately classify defects found on wafer edges. Naturally, in the second half, the wafer backsides are tackled similarly as well using CNNs.

As a sidenote, at the end of the interim report, the concept of MLOps, or machine learning operations, was brought up. This concept ties in tightly with the "system" I was tasked to create. Although the word "system" may seem simplistic on first glance, it actually entails an interconnected web of components with deep and hidden layers that are often abstracted away from the end-user in order to *appear* simplistic. In many ways, the system that I have constructed draws ideas, guidelines and practices from the larger concept of MLOps, which is often said to be "a set of principles that aims to deploy and maintain machine learning models in production reliably and efficiently" [2]. Undoubtedly, in a company with greater emphasis on employing machine learning to solve their business problems, their heavier investments in money and talent in developing a team of machine learning engineers and software developers would mean that their systems would be a lot more in line with the idea of MLOps.

Nonetheless, I personally feel that I have ticked several of the boxes [3] that MLOps demand from a system, such as "deployment and automation", "reproducibility of models and predictions", "diagnostics" and of course, "business uses". In doing so, I can confidently say that I am proud of my creation, as much as it could be a challenge to fully appreciate the intricacies that I have considered when architecting such a system.

2.0 New Project Requirements

With regards to the second half of the internship, I came up with a few key problems that required solving. By breaking down the rather vague requirement to “create a system”, it enables tasks to be clearer and more defined, in turn aiding the approach and solutions I employ.

First and foremost, right after solving the classification of wafer edges, the immediate next step would be to solve for the classification of wafer backside. Once I have produced satisfactory models for these two key problems, that is where the “system” part comes in.

To build a system that can deploy both models (or more) simultaneously and to utilise them to classify a stream of images as input and, no doubt, a set of predictions as outputs, there are several components that are critical in achieving this goal.

These include the flow, structure and interface of the entire system. Flow would refer to the flow of data, the files and images, from the starting point to the ending point. Flow also refers to the flow of the system itself, the systematic and predictable steps that the system would take in order to process the data and produce sensible results.

Structure, on the other hand, pertains mainly to the locations of the critical files and folders that the system would rely on in order to run properly. Structure is highly important because in any system, there will be some reliance in the way that the system components are stationed at to be able to maximise automation and minimise reliance on manual intervention.

Finally, to tie the system together, there needs to be an interface for end-users to interact with and operate the system in a user-friendly manner. This is where I would detail down the evolution from the most minimal command-line interface (CLI), perfect for developers, to a much more familiar graphical user interface (GUI), furnished with everyone’s favourite buttons and text boxes.

All of these three high-level concepts required detailed planning on my part, thinking ahead for any potential problems that may surface arising either from system errors or human errors.

3.0 Backside ADC

In a lot of positive ways, the solutions employed to solve for backside classification are essentially identical to that of the edge classification problem detailed in the interim report. As such, much of the implementation details would be skipped in this final report, offering heavier emphasis on the system design instead.

Essentially, training and testing images are obtained primitively by screenshotting a virtual desktop. Model architectures that have been pre-trained on millions of images are then utilised to speed up the learning of the backside problem statement. Specifically, instead of VGG16, which was my model of choice previously, MobileNetV2 was used instead, which offers improvements in speed and memory at the expense of very slight dips in accuracies. These models, again, are convolutional neural networks, which are deep learning models that work amazingly well on image data.

Similar to edge classification, backside classification also faces the same problems in training a performant model, namely managing the data well. This includes balancing the dataset in order to avoid producing a biased model that predicts the majority class more often than it should. The amount of data was also limited, which further brings out the advantages of using pre-trained models, since they learn well even when presented with merely tens or hundreds of samples.

With that said, new problems also arose. Previously, for edge classification, the problem was simpler as it was binary: does the image exhibit signs of chipping or not. On the flipside, backside classification increases the number of classes to predict from 2 to 5. As seen from the 5 defect classes below in Figure 1, the machine learning model faces a harder set of samples to learn from. Moreover, because sometimes a particular class could resemble another, it adds another layer of complexity for the model to differentiate from.

Figure 1: The 5 Defect Classes of Wafer Backsides

Nonetheless, the final trained models were still extremely satisfactory, averaging 97% out of sample accuracy for the best models. With both the backside and edge models achieving such accuracies, tying them together in a system proved to be the most important yet hardest part of this project.

4.0 ADC System Design and Architecture

As mentioned in section 2.0, the system design encompasses three main sub-components: the flow, the structure and the interface. The details of the structure of the code base and folder arrangements will be skipped as they do not require the most complicated of architecture and planning. Rather, the emphasis in this section would be on the flow and interfacing of the system since they entail a greater amount of depth and technical challenge.

4.1 Data Flow

4.1.1 Current State

With regards to the flow of data, the data at hand refers to firstly, the wafer scans, which will be in large numbers if the machines are activated at 100%, and secondly a unique text file with a format called 'KLA'. These KLA files contain valuable information about a batch of wafers, or a wafer lot, such as what the scanning machines initially classified them to be, the filenames, the location and size of the defects, and so on.

Crucially, these KLA files are read by SSMC's software called Klarity Defect in order to present the found defects in a user-friendly manner. The problem is, as mentioned in the interim report, the fact that the scanning machines often generate an immense amount of noise, or false positives, images that the machines think exhibit defects but in actuality, do not.

Referring to Figure 2 below, the flow of the KLA files and the wafer scans currently follows a simple path, from scanning machine to an intermediate folder location before being read by Klarity Defect and cleared from the folder. To be able to intercept and filter off the noise generated by the scanning machines, my system will have to gather the outputs from the machine, process and classify them, and finally send them back to the location that the software reads from.

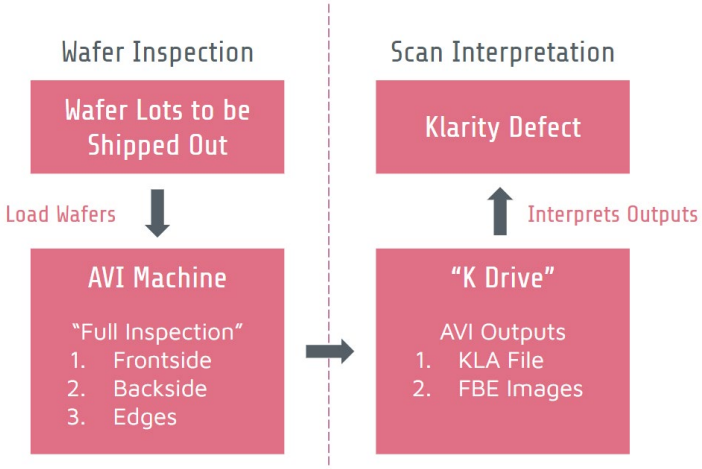

Figure 2: Current State of Data Flow

4.1.2 Future State with ADCS Integration

Referring again to Figure 2 above, the best point to intercept this flow is at the dotted lines. By introducing additional steps in this chain, the ADCS will be able to classify and filter the images just before Klarity Defect reads the relevant images. Figure 3 below shows the extended data flow after the ADCS is integrated with existing infrastructure.

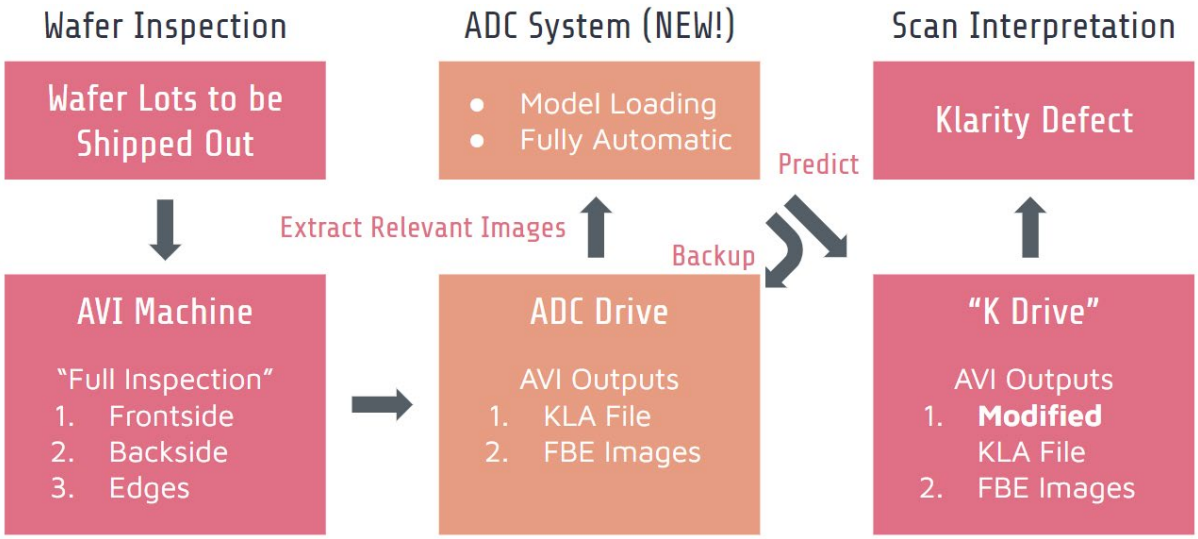

Figure 3: Future State of Data Flow with ADCS Implementation

Along the stream of KLA file and wafer scan outputs generated by the scanning machine, these outputs will first be redirected to another separate folder different from the one Klarity Defect is reading from. This allows the ADCS to gather the inputs safely and to leverage on the trained models to classify them correctly.

Finally, after classification is done, the KLA file is modified to reflect the "correct" defect classes predicted by the ADCS. Along with only images with defects, they will all be transferred to the folder that Klarity Defect reads from. The result is a set of filtered images and modified KLA files that cuts the memory usage significantly. This ties back to the original underlying impetus of this project, since scanning wafers at its current state result in extreme storage wastage due to the noise generated.

4.2 System Flow

As depicted clearly in Figure 4 below, the system flow chart paints the systematic steps that the ADCS would take for each cycle. Making it a continuous loop ensures that the system can be run constantly, which is needed in almost all production processes as it is an indication of a well-utilized and efficient process. In the case of wafer production, wafers are fabricated round the clock, and likewise, quality assurance for these wafers should also be a non-stop process. Unfortunately, to reemphasize, this is not the case currently due to the amount of noise generated that would heavily strain the storage systems.

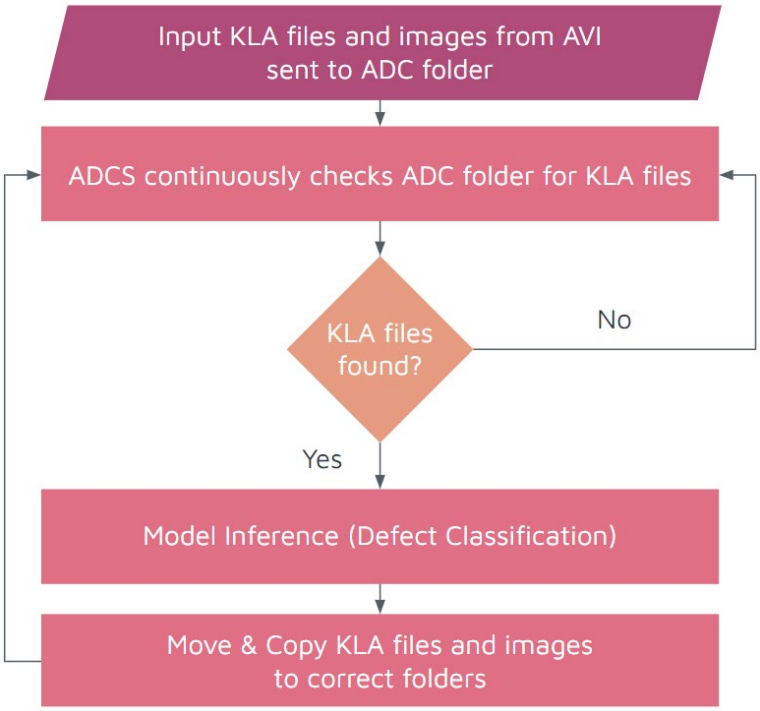

Figure 4: System Flow Chart

The model inference step consists of smaller steps that further break down and enable the wafer scans to be classified systematically. The files and images follow a single-directional pipeline, passing through the models before being streamed into the correct folders both for backup and for interpretation by Klarity Defect. At the end, the entire loop repeats itself, ensuring a continuous pipeline that further extends the existing process infrastructure in place.

4.3 User Interface

Finally, the last major section in terms of the system architecture is interfacing. Specifically, an interface for end-users instead of one merely for the developers of the system. The goal of an interface is to be understandable and friendly to use for the majority of people. To achieve this, understanding what users are already familiar with is paramount to a good user experience.

In my case, this includes using design themes and styles often used in Windows applications such as certain text boxes and buttons coupled with clear description of their functionalities. However, before even drafting such a complex graphical user interface (GUI), the prototype first needs to work.

Hence, in the initial stages, a command-line interface (CLI) was used instead. This entails starting and stopping the system, logging and debugging the system, all through a text-based interface most familiar to developers. The reason for such a simplistic prototype is obvious: it does not require any building of the interface itself, yet all information necessary for testing the system presents itself, albeit looking very visually unappealing. With that said, it was clear to me that such a foreign interface would not be in the final product for the learning curve was simply too steep.

Therefore, after ensuring that the system's core functionalities work as intended, the planning and building of the GUI took place. Planning of the interface was the easiest part — leveraging design mock-up tools such as Figma [4] to draft up the dimensions and placements of each of the UI components. These tools also include more advanced functions such as mocking up the flow of the interface, say, what happens when the "SAVE" button is clicked, allowing me to have a clear understanding of the intended outcomes actions that each component would deliver. These are all crucial in translating design into code, the most tedious part of all.

To give a rather general overview of the technical implementation of the user-interface, the Python library used was called "Tkinter" [5]. This UI library offers the main benefit of being a pre-installed library within the Python application, thereby simplifying the ADCS' installation process. However, countless obstacles were faced throughout the code-writing.

One key challenge was the fact that user-interfaces are often a continuous loop by itself, continuously drawing itself repeatedly at tens of frames per second to be able to be "displayed". This is separate from the "main loop", which is the loop referred to in section 4.2, the "system flow". Unfortunately, trying to run two things (or more) at the same time in programming presents a major problem. The problem is that both of these two different loops have to run concurrently, or simultaneously, yet at the same time, communicate with each other and pass information back and forth. This is necessary because the user-interface must first receive the information from the main loop before it is able to display that information to the screen, or the interface. Likewise, any user interactions with the interface, for example, a user clicking a button, is information that needs to be sent from the UI loop to the main loop to be processed.

Thankfully, with the help of countless hours of digging the vast wealth of knowledge on the internet, I stumbled upon an implementation almost identical to my desired outcome [6]. With that, I broke down his code and built upon it, extending the functionalities bit by bit until it looks almost identical to the mock-up done in Figma [7]. It is without a doubt that much of the nitty gritty details were left out, but the overarching point of this section is to simply emphasize the difficulty of creating a user-interface that, to most, might look completely basic and easy to build.

Nonetheless, the final product, as presented in Figure 5 below, offers a very functional interface that allows users to tune and save certain settings and to start and stop the system as they see fit. Moreover, system updates and critical information are also logged to the interface which allows users and developers alike to understand what is happening throughout the system's runtime, as well as to aid in bug fixing, should any arise.

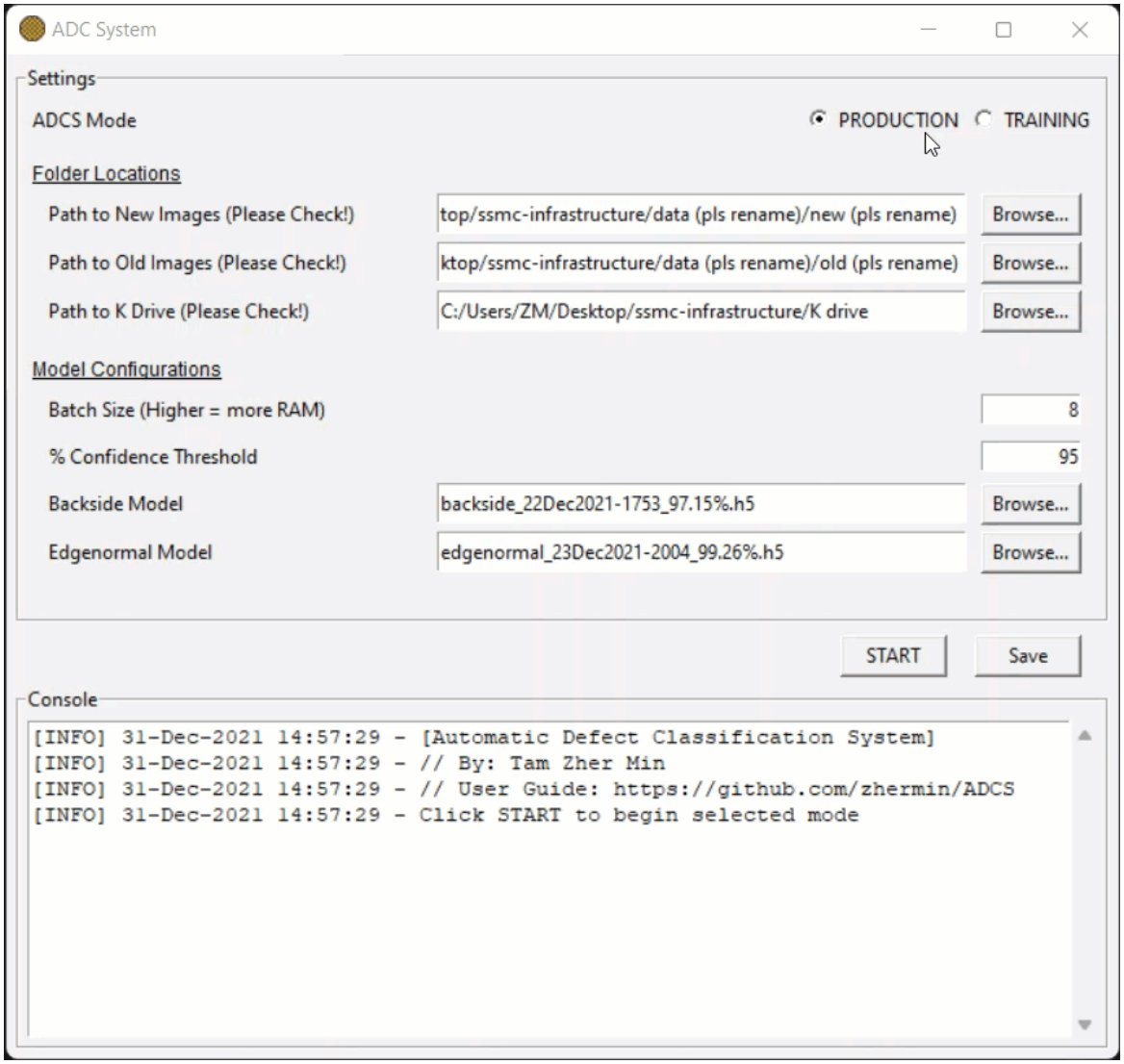

Figure 5: The ADCS Graphical User Interface

5.0 Business Value-Add

All in all, this project was birthed from the underlying desire to not only work towards automating more parts of the process pipeline in SSMC, but critically, to significantly clean up the sheer amount of unnecessary noise generated by the scanning machines.

This is because everything has a certain cost to a company, be it manpower, time or space, although in this case, space would refer more so to the digital storage space of servers and databases. In an enterprise setting, storage spaces do not come cheap, and with a highly scaled-up production facility such as SSMC running its machines 24 hours a day, 365 days a year, the amount of generated data can chew through storage quotas extremely easily. Moreover, in the case of the scanning machines, the outputs happen to be images that undoubtedly take up significantly more space than hundreds of text files combined for just one image.

Thus, the current situation prior to implementing such a system was to simply avoid scanning most of the wafers, but rather, to only selectively scan for a few of them when necessary. This means that much of the possibility of detecting defects early are nought, since the defects, if any, would only be discovered through manual human inspections instead.

Therefore, with this project, two key improvements were proposed and created by me. Firstly, as was mentioned in the interim report, I had proposed to completely disable the scanning of Edge Tops specifically. This was because I had analysed and discovered that Edge Top scans not only take up a huge amount of space, but that they offer close to zero ability in finding defects on the wafer. What often ends up happening is that almost every single Edge Top scan would turn out to be noise, or false positives, due to the high sensitivity and limitations of the scanning machines. By disabling this feature, and instead focusing only on the backside and direct edges of the wafers, it was calculated to cut the space usage by at least 66% or even higher depending on the number of Edge Top scans generated. Yet, at the same time, the ability to find actual defects on the wafers remain the same, since reliance on the backside and direct edges is sufficient.

Secondly, my ADCS further solves this problem by filtering out all other noise generated by the scans from the wafer backside and direct edges using machine learning to do the classification automatically and intelligently. The resultant output is a highly refined set of images containing only defects, with no noise from the back or edges, and completely no images from Edge Tops. This translates to upwards of 80% to 90%, and potentially even 100% of the noise to be filtered out from the back and edge scans. A 100% reduction in space requirements is possible due to the simple fact that certain wafers could have completely no defects at all. In such cases, the ADCS would simple filter everything out.

Overall, on average, both of these solutions can offer an average of around 85% total reduction in the number of images stored in SSMC's servers for each wafer scanned. Depending on how much space each of the type of scans take up, for example, Edge Top scans taking up more space than Edge Normal scans, the percentage reduction could be even higher.

Additionally, if more wafers are scanned in the future, it might mean that most of the wafers scanned have no defects at all. This is because the current practice is to scan defects that are suspected to have problems. With the current state offering such promising reductions in storage requirements, this could mean an even further reduction, resulting in a highly optimized and automated wafer quality assurance workflow.

All of such reductions in storage spaces not only allow more wafers to be scanned and processed, but crucially, they translate directly to money to the company. Considering a storage server with hundreds of terabytes of storage could cost upwards of tens of thousands of dollars to any company out there, not including the maintenance and service costs of such servers, this set of solutions could offer significant cost savings that compounds over the years to come.

Lastly, with this step in the direction towards automation, the success of this project paves the way to further automating either time or storage consuming sections of the wafer fabrication pipeline and spur heavier investments in researching and developing such automation processes. This could open up doors to more specialised automation departments within the company, potentially allowing significant competitive advantages to be gained over other industry players.

6.0 Reflections and Conclusion

With that, here are some closing thoughts over the entirety of this four-month long journey. Personally, I am proud of myself for being able to bring the project from start to end independently. From gathering and understanding the project requirements and purpose, I was able to pinpoint what were the key steps that I had to accomplish in order to achieve the desired outcome.

This began from understanding the wafer fabrication process, especially along the quality assurance pipeline, allowing me to further decipher the business problems presented in order to propose ideas and solutions that could tackle some of these issues. This is because not all solutions have to be a tangible product such as an application or device, but rather, could be proposals to streamline or optimize existing processes.

Nonetheless, the ability to leverage cutting-edge machine learning technologies was another area that I personally was rather contented with. For one, I did not enter this role with much hands-on experience in implementing such solutions. Countless challenges also presented itself, from the understanding of the available solutions out there, to the actual implementation of the models, to gathering and accessing of the data needed. These challenges were thankfully overcome with some creativity and perseverance.

Moreover, when the need to deploy the models to production arose after confirming their business utility, I learnt a lot about the concept of MLOps, which governs the critical question of: what happens after you have trained a performant model? It started with prototyping a web user-interface using a Python library called Streamlit [8], which turned out to be unsuitable due to SSMC's limitations. Once I have realised that deploying the models to the web was to be avoided, I fully committed to creating a system that could function well locally within SSMC's internal servers, leading to the ADCS detailed in this report.

This ADCS was the culmination of several paper drafts and lists of features and potential problems to tackle. Planning for such a rather elaborate system is no doubt likely the most important part, more so than having the ability to actually code the system out. This is because there must be a deep enough understanding of the technological limitations alongside the time constraints presented before the project could be embarked on. For example, although I knew that a GUI was going to be an extremely useful addition to the system, I was prepared to ignore that feature if certain of my earlier steps took longer than expected, say, training a performant backside model or perhaps deploying multiple models at the same time. This would mean that my end-users would have to learn how to operate a CLI instead of a GUI and accept the trade-off of time and ease of implementation for a much less friendly user-interface and steeper learning curve.

Overall, I would say that I have managed to accomplish something significant and can likely offer an appreciable amount of value-add to SSMC's business. My only hope is that SSMC continues to further invest in the area of automation, data science and data engineering, machine learning and software development under the name of research and development, such that current and future employers, employees and interns of SSMC will have a much more friendly and stable infrastructure and environment that understands their needs, allowing them to collaborate effectively and further build upon my work and discover completely new and creative solutions to areas that would very much benefit from these technological advancements.

References

[1] T. Zher Min, "zhermin/ADCS: Automatic Defect Classification System (ADCS) for SSMC," *GitHub*. [Online]. Available: https://github.com/zhermin/ADCS. [Accessed: 05-Jan-2022].

[2] C. Breuel, "ML OPS: Machine learning as an Engineering Discipline," *Medium*, 03-Jan-2020. [Online]. Available: https://towardsdatascience.com/ml-ops-machine-learning-as-an-engineering-discipline-b86ca4874a3f. [Accessed: 05-Jan-2022].

[3] N. Walsh, "The rise of quant-oriented DEVS and the need for standardized MLOps," *Slides*, 03-Feb-2021. [Online]. Available: http://slides.com/walsh/standards-in-ml-ops#/18. [Accessed: 05-Jan-2022].

[4] "The Collaborative Interface Design Tool.," *Figma*. [Online]. Available: https://www.figma.com/. [Accessed: 05-Jan-2022].

[5] "Tkinter - Python interface to TCL/TK," *tkinter - Python interface to Tcl/Tk - Python 3.10.1 documentation*. [Online]. Available: https://docs.python.org/3/library/tkinter.html. [Accessed: 05-Jan-2022].

[6] B. Bertrand, "Logging to a tkinter scrolledtext widget," *Tchut-Tchut Blog*, 28-Dec-2017. [Online]. Available: https://beenje.github.io/blog/posts/logging-to-a-tkinter-scrolledtext-widget/. [Accessed: 05-Jan-2022].

[7] T. Zher Min, "SSMC ADCS GUI," *Figma*, 16-Dec-2021. [Online]. Available: https://www.figma.com/proto/ZTgF32w2j9suCelMuveQil/SSMC-ADCS-GUI?node-id=3%3A370&scaling=contain&page-id=3%3A167&starting-point-node-id=3%3A370. [Accessed: 05-Jan-2022].

[8] "Streamlit • the fastest way to build and share data apps," *Streamlit*. [Online]. Available: https://streamlit.io/. [Accessed: 05-Jan-2022].