

SOFTWARE ENGINEERING GROUP PROJECT

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

KidneyCaliper

Automated Deep-Learning-Based Workflow
for Kidney Pathologists

Authors:

Andy Wang
James Ball
Jess Lally
Paweł Kroll
Stefan Radziuk
Sudarshan Sreeram

Supervisor:

Dr Bernhard Kainz

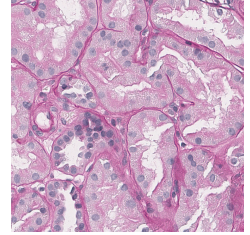
Clients:

Dr Candice Roufosse
Callum Arthurs

1 Executive Summary

1.1 Background

A kidney biopsy is a procedure to extract kidney tissue for examination under a microscope. Typically, kidney pathologists look down the microscope and write reports, marking notable features-of-interest. Kidney transplant recipients may suffer from post-surgery complications due to the donor organ being a *poor fit*. These reports form the basis of a grading process to predict the survival rate of the donor organ in the recipient or to understand why one failed.



Kidney whole slide image close-up

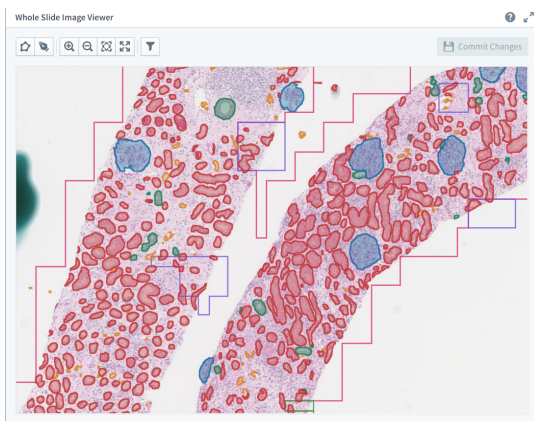
As digital pathology becomes more popular, there is an increasing demand for tools that try to automate a pathologist’s tedious workflow with automatic measurements and feature recognition, especially when 24% of pathologists are having to outsource work weekly[1]. A *slide scanner* is a type of microscope that takes scans of a slide and stitches them together to form a very high-resolution image — a whole slide image (WSI). Ideally, this digital workflow would evolve into a system where the whole slide image is submitted to an algorithm that generates annotations and measurements that pathologists can then include in their reports.

Past Master’s projects, by Grzegorz Sarapata[2] and Jiamin Fu[3], made the essential first step towards creating such a system, which was the creation of two deep-learning models that produce reasonably accurate labelling on kidney whole slide images. This project, aptly named **KidneyCaliper**, takes this a step further and builds upon their models for use outside of a theoretical setting to deliver a fully-automated workflow for kidney pathologists. The new workflow is dramatically faster, doesn’t require a programming background, and provides much more useful analysis compared with what was possible with the models on their own.

1.2 Solution

KidneyCaliper facilitates the submission, processing, and analysis of kidney whole slide images automatically, significantly saving the amount of time that researchers and clinicians need to spend generating analysis for reports. Kidney pathologists can submit many high-resolution whole slide images to be processed outside of work hours without human involvement, and have annotations and statistics ready the next day. This is in stark contrast with the previous workflow, which required significant human effort during work hours to achieve a comparable result.

In addition to the reduced manual workload, **KidneyCaliper** provides detailed information about kidney whole slide images that is practically impossible to gather manually. Image-specific statistics such as the size of each feature-of-interest, along with the areas of their maximum inscribed circles, are examples of measurements that **KidneyCaliper** can generate which have been proven useful in prior research to estimate how well a kidney will function[4]. Whole slide images that were processed historically can also be compared with one another, providing both a macro perspective on the trends seen across many biopsies, and new insights into the differences in the biopsy stage for successful and unsuccessful kidney transplants.



Whole slide image and annotation editing suite

Whilst kidney pathologists have reported that the automatically generated feature-of-interest annotations have been accurate, we have integrated a whole slide image and annotation editing suite to mitigate any potential errors. Any annotations that have been misclassified, are missing, or are otherwise faulty can be easily corrected, and all statistics for the new annotations are updated as soon as modifications are made.

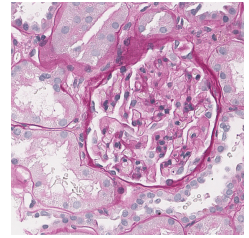
Overall, **KidneyCaliper** dramatically speeds up a pathologist’s workflow and provides valuable statistics that were previously unrealistic to obtain. The rich insights into patient biopsies provide researchers with new information, as well as the ability to find new relationships between data found in a kidney biopsy and a successful kidney transplant. This means **KidneyCaliper** can have a legitimate impact on improving the survival rate of the donor kidney in the recipient.

2 Introduction

2.1 Motivation

Kidney biopsies are often taken from donor kidneys to predict the likelihood of a successful transplant. The biopsy is stained and scanned by a *slide scanner* device to produce a very high-magnification (40X) *whole slide image*. Kidney pathologists then provide visual analysis of this whole slide image, or alternatively analyse the biopsy manually by viewing the slide down a microscope. The count and morphology (shape and size) of glomeruli present in the scan are key measurements used to evaluate the health of the donor kidney. Glomeruli are clusters of blood capillaries, used to filter waste and excess fluids from blood[5].

However, annotating a whole slide image scan can be extremely time-consuming and error-prone. It can take around 45 minutes to annotate and count all the glomeruli, and even longer to annotate all of the features-of-interest. Manually annotating scans is also not very reproducible, and it can be difficult to obtain accurate glomeruli counts[6]. Digital tools for automatically annotating kidney scans could lead to a significant workflow speedup for pathologists, and increase the accuracy of the statistics gathered from the scans[7][8].



Glomeruli visible on kidney biopsy scan

While this would be potentially useful for NHS clinicians as well, our project focused more on improving the workflow for pathologists working in research. Research has previously indicated that other statistics may also be useful for predicting the outcome of transplanted kidneys, particularly the size of glomeruli[4]. These statistics are extremely difficult to calculate by hand, so a digital application that generates these statistics automatically could be very useful and would allow researchers to investigate new methods for determining the likely outcome of kidney transplants.

2.2 Objectives

The overall objective for the project was to combine the two deep-learning models developed by Sarapata[2] and Fu[3] into a single pipeline, providing kidney pathologists with a usable workflow to automatically annotate kidney scans. Users should be able to upload whole slide image scans to the application, which will run each model in turn, generating the annotations and statistics for each scan.

The first model identifies different regions of the kidney within the scan, i.e. the medulla (the inner region of the kidney) and the cortex (the outer region of the kidney), as well as any non-kidney tissue in the scan. The cortex region is primarily used for analysis, so it is important that pathologists can filter annotations and calculate statistics for the different regions separately.

The second model annotates the different features-of-interest present on the kidney scan, such as arteries, glomeruli, and tubules (small tubes carrying fluid and nutrients in the kidneys)[9].

To effectively combine the two models in our application, we needed to adapt the models to be production-ready. One issue was that processing very large whole slide images (which would be used in practice) was very slow, so we needed to make improvements to the models to make the application usable on slower machines, which would be more typical of NHS devices in a clinical setting. Our clients also expressed their desire for the application to allow users to submit scans in batches that can be annotated overnight, which would also make their workflow more efficient.

To make the output of the two models usable for the pathologists, we also needed to develop an interface to allow users to view the produced annotations, make corrections, and view the statistics generated for each scan. We also wanted to automatically generate statistics that could not be previously calculated by hand, to allow for richer insights into biopsy scans.

Another important objective for the project was to make sure that it would be possible to locally host the application, so that it would be only accessible to devices within a local network. This is necessary because the application would be running on private NHS patient data, which must not be accessible to devices outside the NHS network, as detailed in section 6.2. However, to allow our clients to demo the application to other pathologists, we have provided a [publicly accessible version](#), currently hosted on a DoC VM[10].

2.3 Key Achievements

- Significant decrease in the processing time of the two deep-learning models (section 3.3).
- Batch-submission of whole slide images (section 3.2).
- Automatic generation of statistics and graphs (section 3.4).
- Interface for making corrections to generated annotations (section 3.5).
- Support for exporting statistics and annotations into pre-existing software (section 3.4.1).

On a GPU-equipped laptop of ours, the first model (cortex-medulla classifier) was around twice as fast with our improvements, and the second model (semantic segmentation) was around 40 times faster, as shown in Figure 13. As well as decreasing the processing time on powerful machines that researchers are likely to have access to, this has also made the application more usable on slower machines. This makes it flexible for machines with limited GPU capabilities, which are more typical of the NHS devices in clinical settings.

We were also able to implement batch-submission of whole slide images, to allow users to upload multiple kidney scans at once. This was particularly important to the researchers as it would allow them to process scans overnight, significantly speeding up their workflow.

Not only were we able to generate the same statistics that pathologists currently measure from kidney scans by hand, we were also able to generate new, potentially useful measurements and statistics, such as the areas of different vessels. Users are able to plot graphs of any measurement against any other measurement, allowing them to investigate these new statistics. We also support comparison of statistics from historical kidney scans, which will be useful for researchers to compare biopsies taken at different times from the same patient, to investigate how transplanted kidneys have changed with time.

In addition, we have created a simple interface to make minor corrections to the produced annotations, allowing them to add new annotations for vessels that were missed, delete incorrect annotations, and allow them to change the labels of annotations that were incorrectly identified. This should also speed up the pathologists' workflow as there is no need to move the annotations to a different program (such as QuPath[11]) to make these changes.

Finally, we have also created support for the pre-existing software that pathologists are currently using for whole slide image annotation, such as QuPath. All annotations produced by **KidneyCaliper** can be exported in the GeoJSON[12] format to be viewed in QuPath, and any statistics can be exported in CSV format to be compatible with various data analysis tools. This was important as it means the annotations can be used in industry-standard software.

3 Design and Implementation

3.1 Design Overview and Motivation

KidneyCaliper is a web-based application. It is inherently cross-platform and eliminates the need for the user to install additional software. Moreover, this choice provided us freedom to choose from a range of out-of-the-box solutions for displaying and annotating digital kidney images. Figure 1 showcases a high-level overview of **KidneyCaliper**'s architecture; several components of it will be detailed in subsequent sections.

Due to the numerous privacy concerns that arise when dealing with medical data of patients, we identified that our clients would likely have to self-host **KidneyCaliper** either on their own machines or in their clinic's intranet, where patient data is stored on trusted servers. Considering the context of this use-case, we decided to minimise the setup overhead of both the server and client. To achieve this, we chose to use an SQLite[13] database and simple file system storage for uploaded and processed data. Other database and data storage solutions we explored, such as PostgreSQL[14] and cloud file storage, pose additional setup overhead. In the event database performance ever becomes a concern, we allow for an easy transition to different database solutions through our server configuration file. This was made possible through our use

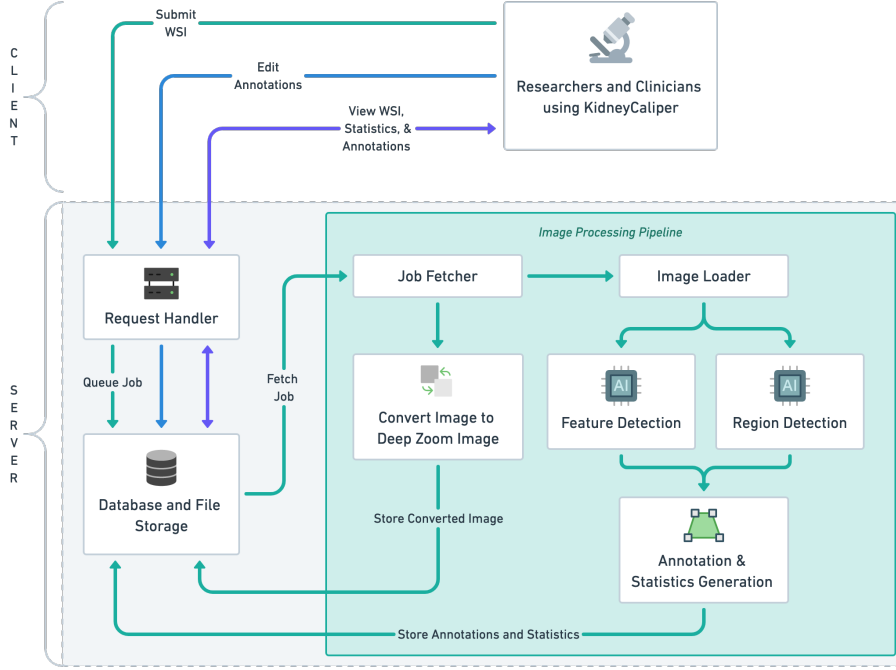


Figure 1: System Architecture Diagram

of SQLAlchemy[15], a Python library that abstracts interfacing with the underlying database. Starting the client is as simple as opening a tab in the browser with the right address.

For the frontend, we use React[16] with TypeScript. We chose to go with TypeScript over JavaScript for several reasons: improved code completion, better code readability and comprehension of purpose and functionality, and the strong typing makes unit testing superfluous to an extent. We did, however, encounter an issue when working with a third-party library, Annotorious[17], that didn’t offer type definitions; this is detailed in section 3.4. For the user interface, we used Palantir’s Blueprint[18], a component library that’s feature-rich and tailored for data-intensive applications. We paired this with React Mosaic[19], a tiling window manager for the web, which offers an experience that’s similar to what one would expect with a traditional desktop application.

React uses client-side rendering by default. The benefits of server-side rendering (SSR) aren’t fully realised in the use-case environment of **KidneyCaliper**. In fact, optimising it for SSR would introduce a complexity overhead and shift the client’s processing load to the server, ultimately impacting maintainability.

3.2 Frontend Interface & Experience

Frontend components are structured to be reusable and the component hierarchy follows a structure that’s similar to that of Atomic Design[20]; here, however, the *atoms* and *molecules* are defined by the component libraries. This strategy reduces duplication across the codebase and promotes good organisation that mitigates the overhead that future developers may incur in picking up where we left off.

The interface layout is tailored to be responsive and configurable on desktop screen sizes; in fact, the window panes can be rearranged through a drag-and-drop action for users on vertically-oriented monitors as shown in Figure 2. These panes can also be resized based on the user’s pane-of-focus by dragging the split handle in the gutter; this is particularly useful in the result page where the user may want to focus more on the image viewer than the statistics pane and vice versa. The use of panes establishes separate contexts in a single



Figure 2: Vertical Monitor Orientation

window, which provides a more cohesive user workflow compared to that offered by a tabbed layout, pop-up modals or floating information boxes.

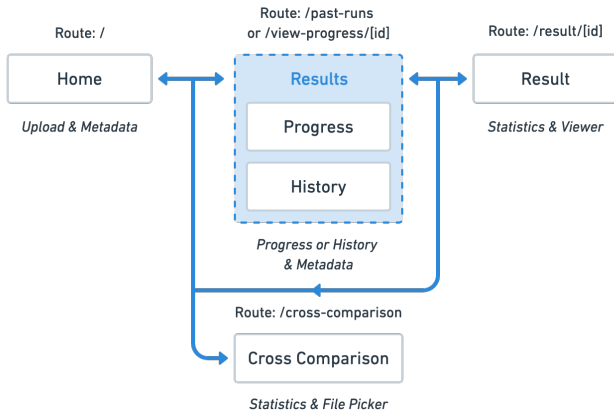


Figure 3: Map of User Workflow

mentioned earlier as shown in Figure 4. For example, the user can right-click a chart on the statistics page to save it as an image or toggle a fullscreen view. The numerous help dialogues we’ve added to the panes guide the user in learning how to use the platform and to understand its capabilities. When the user uploads a kidney image, a pane is revealed with its thumbnail and a metadata table. Part of a good user workflow is keeping the user informed. Further, from a user-experience standpoint, it’s crucial to make the users feel safe by not having their actions lead to irreversible consequences. To address this, we added confirmation dialogues and warning callouts to help avoid accidental clicks. We also employ a non-destructive way of editing annotations, where the original file is duplicated with the user’s changes applied on top, and the file is listed on the results page with an (*edited*) badge.

Another important feature that we’ve added to improve the user experience is allowing users to upload and submit whole slide images in batches. Images are still processed one-by-one, but this was important as processing these whole slide images can be very slow, particularly on machines with limited GPU capacity. This allows users to submit scans to be processed and annotated outside of working hours, making their workflow more efficient.

We paid attention in trying to abide by industry-standard UX research and practices to structure our UI through articles and guides, particularly those on web usability, from the Nielsen Norman group[21]. Although our current interface presents the opportunity for additional attention-to-detail, quality-of-life improvements, we think that it’s well-rounded given the project’s time frame.

3.3 Parallel Processing Pipeline

Both the cortex-medulla classifier and semantic segmentation models internally divide a whole slide scan into overlapping square tiles, and each tile is processed independently. These per-tile processing results are then gathered to form predictions on kidney regions and features.

We have inherited the tiling, per-tile processing, and result merging code from the research works[2][3]. Each tile undergoes image pre-processing, deep learning model prediction, and model output post-processing steps. The deep learning model is GPU-accelerated, whereas the pre and post-processing steps are CPU-based. However, the research work did not exploit parallel computation capabilities in modern multicore CPUs.

Our initial prototype in Streamlit, discussed in section 3.5, had three key page contexts: upload, progress, and result. We found this contributes to a straightforward user workflow and looked to maintain a similar structure. Currently, the user workflow is structured as shown in Figure 3. The introduction of numerous features didn’t complicate or deviate from the simple, intended workflow. This was possible because the complexity associated with those features was extracted into context menus, pop-up confirmation dialogues, inline tooltips, help modals, drop-down menus, collapsible sections, non-ideal state placeholders and more!

Throughout the interface, we have implemented quality-of-life features with the interface elements

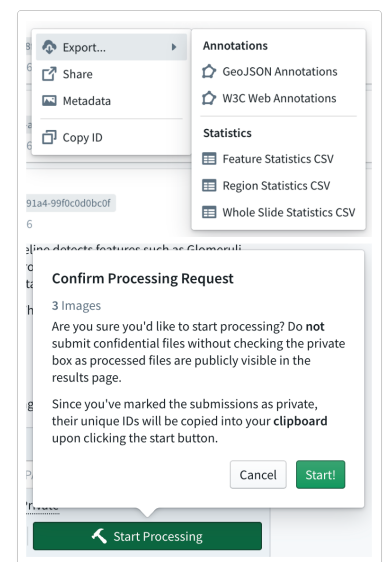
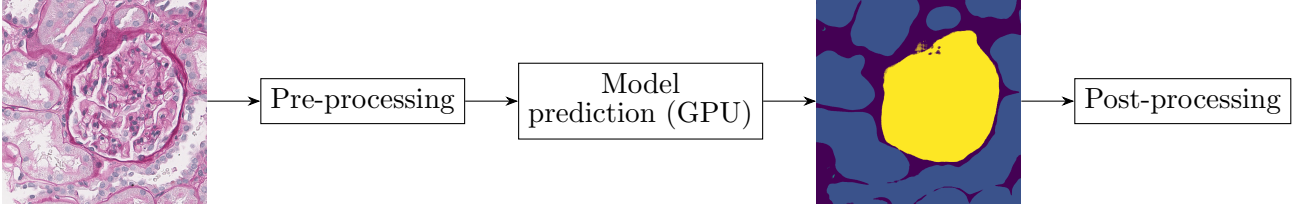
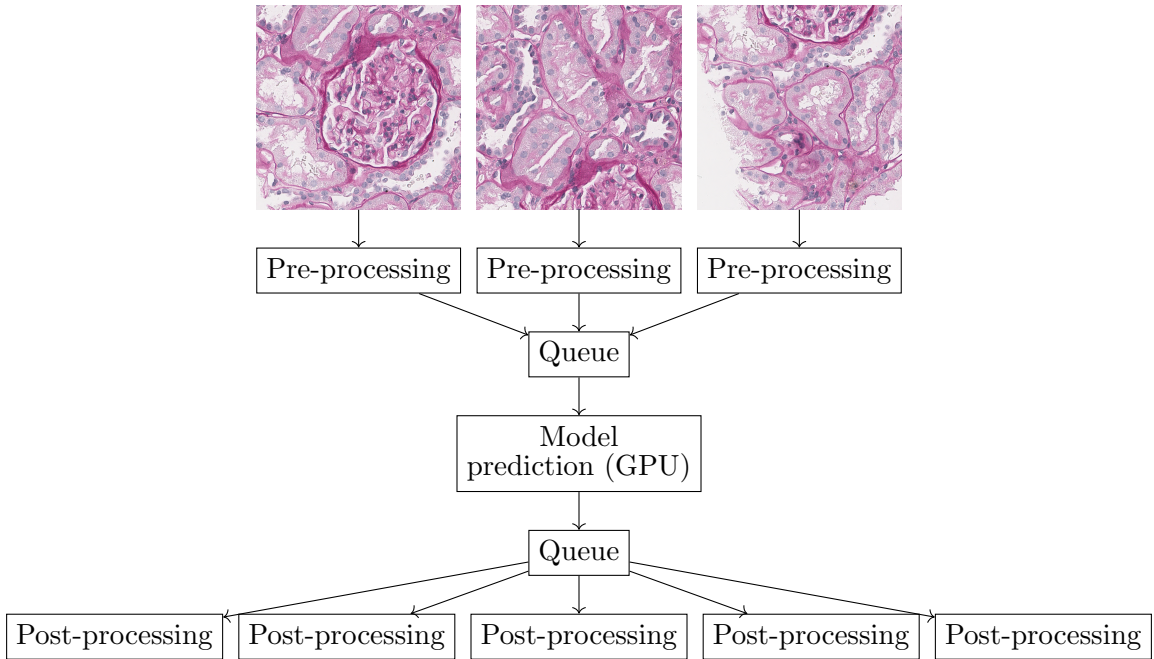


Figure 4: Hidden Interface Elements



Since the processing steps of each tile are isolated, our program appears to be easily parallelisable. Generally, this sort of pattern can be parallelised with a process pool. Due to the differences in the computational models of GPUs and CPUs, this will not work well: while we may have multiple CPU cores for pre and post-processing, the number of GPUs available is not guaranteed to be the same (likely far fewer or only one). Naïvely feeding tasks into a process pool means that there will be multiple instances of the model prediction attempting to run simultaneously on a single GPU. While it is technically possible to run multiple GPU subroutines (CUDA kernels) in parallel on a single GPU[22], this will unlikely bring any positive performance impact as each kernel is already automatically parallelised.

Instead, we opted for an asynchronous three-stage pipeline to efficiently utilise both CPUs and GPUs according to their computation model. Multiple pre and post-processing workers are spawned while only one CPU process by default is responsible for invoking the GPU-based model prediction. The stages are connected by multi-producer multi-consumer queues to maximise worker utilisation; a new unit of work will be picked up as soon as the previous result is placed into the downstream queue. The queues are bounded to prevent intermediate results from piling up in memory.



Note that the number of workers for pre-processing may be different from the number of workers for post-processing. Indeed the numbers are different for the two models. This is because they have different complexities. The ratio between pre and post-processing workers are tuned to reduce CPU bottlenecking as much as possible. Although we expect a researcher’s workstation to have only one GPU installed, our code is structured such that it’ll be extremely easy to add multi-GPU support to further exploit parallelism.

3.4 Annotations and Statistics Generation

3.4.1 GeoJSON and QuPath

In the first iteration, the researchers that we spoke with were clear that the most important thing to implement early on was the ability for annotated features-of-interest to be visualised in industry-standard pathology software, QuPath[11]. QuPath is an open-source program that digital pathologists are already

familiar with, making it an obvious choice to design for in the early prototypes, as recreating a similar interface tailored for kidney pathology would take significantly longer than a single iteration to develop.

QuPath allows pathologists to both view whole slide images, and import annotations that are displayed over the top. The format QuPath uses is called GeoJSON, which is typically used for encoding geographic data structures[12], but it is also perfectly acceptable for other high-resolution image viewers such as QuPath. GeoJSON uses a geometrical format to describe features or shapes, so we must generate geometrical representations of all the features-of-interest present in the whole slide images.

After the image processing pipeline, the semantic segmentation model produces an integer label for every pixel in the original image. This is used to classify each pixel with a feature-of-interest, such as a glomerulus, tubulus, or artery. We need to group these clusters of labels together and then convert the clusters into a geometric representation that can be overlaid as a vector image on top of the kidney whole slide image. Once we have a geometric representation, it is easy to generate various measurements for the annotation.

After some research, it was clear that the Rasterio Python library[23] was a great choice to both cluster the classifications, and generate geometry from them. `rasterio.features.shapes` in particular, does exactly what was required, and converts the classification output into Rasterio shapes, which can then be trivially converted into a geometry format supported by GeoJSON. With some minor post-processing, where colours and feature-of-interest classification names are added to the JSON for QuPath, the GeoJSON can be easily visualised on top of the whole slide image in QuPath, as seen in Figure 5.

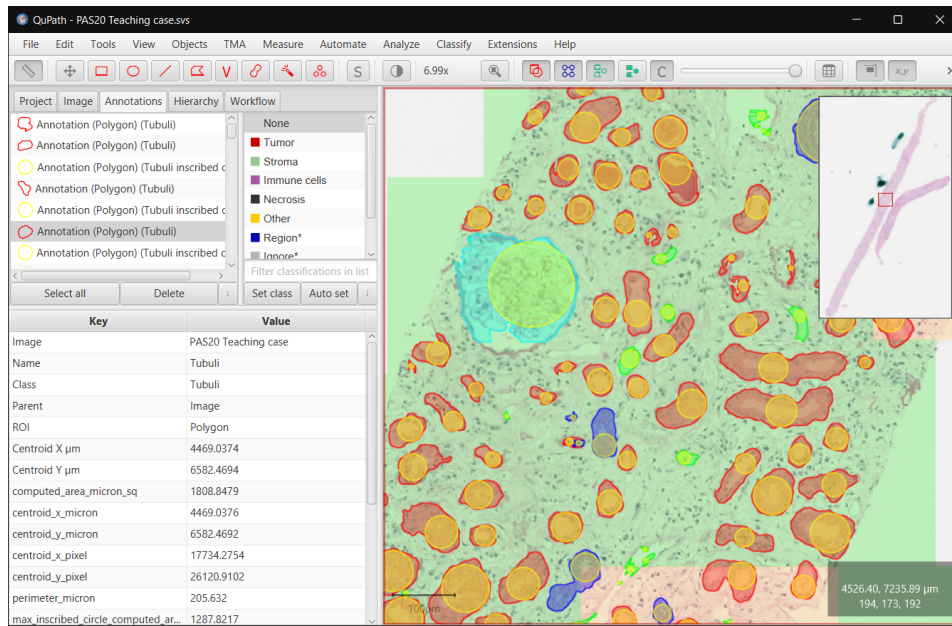


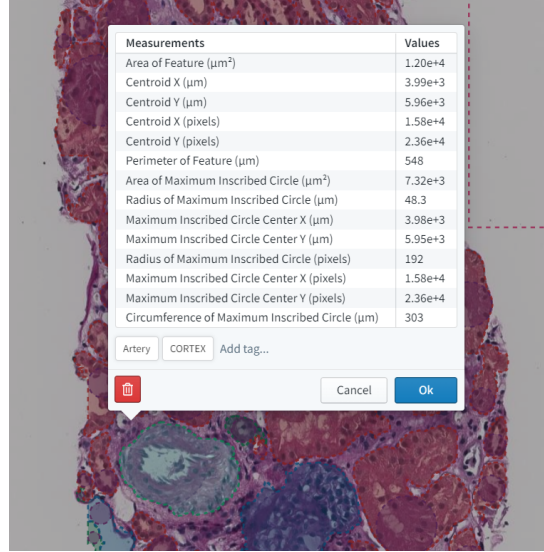
Figure 5: Annotations Visualised in QuPath

In later iterations, we re-use this geometric representation for a slightly different annotation format called the W3C Web Annotation Data Model[24], as discussed later. Support was kept for QuPath even after creating a whole slide image viewer that was more tailored to kidney pathologists so that users had a choice between the familiarity of industry-standard software, or the extra features that **KidneyCaliper**'s interface provides.

3.4.2 Generating Measurements

Now that shapes had been created using Rasterio[23], it was easy to generate basic measurements such as the area, perimeter, and centroid. A full list of measurements can be seen in the final interface in Figure 6. Kidney pathologists suggested that the area of the maximum inscribed circle would be a useful measurement

in addition, which would be more complicated to calculate. Polylabel is an algorithm for calculating this, which finds the point inside a polygon that is furthest away from any edges, which is the centre of the maximum inscribed circle[25].



Measurements	Values
Area of Feature (μm^2)	1.20e+4
Centroid X (μm)	3.99e+3
Centroid Y (μm)	5.96e+3
Centroid X (pixels)	1.58e+4
Centroid Y (pixels)	2.36e+4
Perimeter of Feature (μm)	548
Area of Maximum Inscribed Circle (μm^2)	7.32e+3
Radius of Maximum Inscribed Circle (μm)	48.3
Maximum Inscribed Circle Center X (μm)	3.98e+3
Maximum Inscribed Circle Center Y (μm)	5.95e+3
Radius of Maximum Inscribed Circle (pixels)	192
Maximum Inscribed Circle Center X (pixels)	1.58e+4
Maximum Inscribed Circle Center Y (pixels)	2.36e+4
Circumference of Maximum Inscribed Circle (μm)	303

Antery CORTEX Add tag... Cancel Ok

Figure 6: Annotation Measurement List

Polylabel also computes the radius of the inscribed circle, which is required to calculate the area, but unfortunately this value is rarely returned in polylabel implementations[26]. One such implementation that gave us the radius worked well but was written in pure Python, which considerably slowed post-processing since this must be computed for thousands of annotations. We opted instead to create Python bindings for an existing implementation[26] in Rust that we modified to return both the centre and radius of the maximum inscribed circle[27]. The Rust implementation was over 35 times faster than the Python-only library, which made a significant improvement on post-processing times, speeding it up by over a minute on more complicated images. As we now have measurements for the maximum inscribed circle, we decided to add this as an additional annotation for each other annotation.

Measurements themselves, once generated, can be stored within the annotation format being used. Both GeoJSON[12] and W3C Web Annotations[24] have places to store custom measurements that comply with their specifications, allowing them to be visualised in QuPath[11] and **KidneyCaliper**'s interface. Despite this, a bug was discovered in QuPath that meant custom measurements were not imported correctly. To remedy this, we issued a pull request that fixes the bug, which has now been implemented in QuPath v0.3.1[28].

CSV files are also generated at the same time which contain the same measurements for each feature-of-interest. This was another feature suggested in user feedback that was easy to implement and allows pathologists to make flexible use of the data that we generate, rather than relying only on the graphs in **KidneyCaliper**'s interface.

3.4.3 Presenting Statistics on the Frontend

We used the Recharts[29] library to visualise the generated measurements. It integrates well with React[16] and its graphics style fits nicely with the rest of the site.

We provide two types of measurement visualisation: individual slide statistics (Figure 7) and cross-slide comparisons (Figure 8). Measurements gathered from a slide are displayed alongside the slide viewer and editor. Here, the user can see summary information such as the distribution of feature types, and produce plots of any two measurements of a particular feature type against each other.

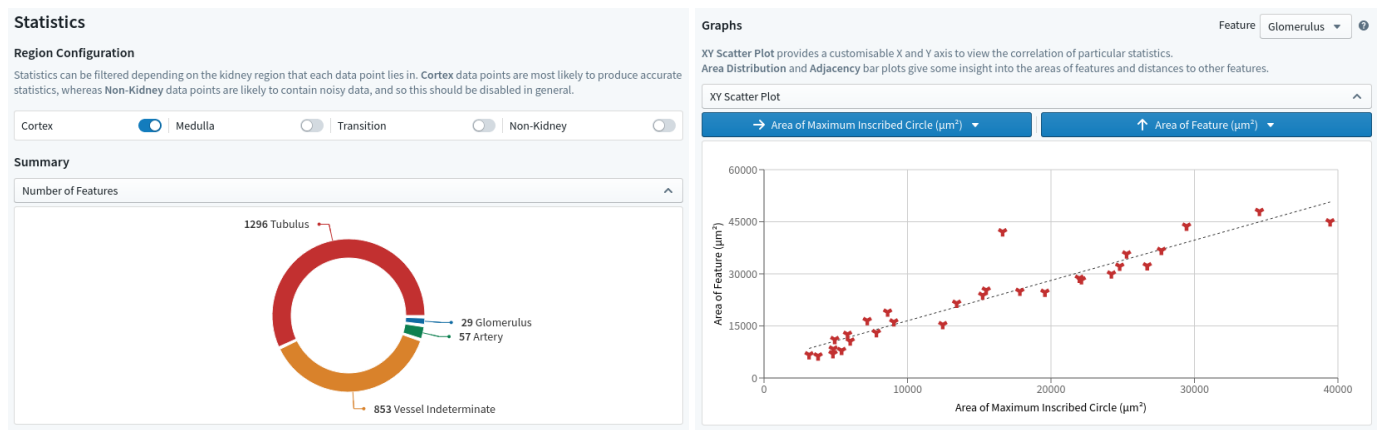


Figure 7: Individual slide statistics

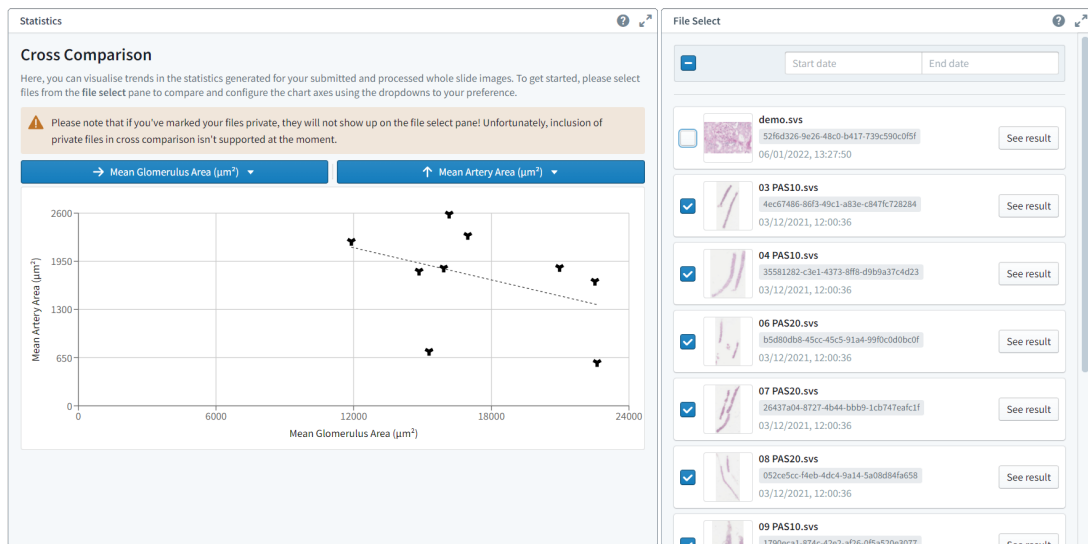


Figure 8: Cross comparison

Cross-slide comparisons are based on aggregated measurements (such as mean, median, sum and counts) of the various data from each slide. Again, the user can plot any category data against each other. Given the sheer number of data categories available, the axis selection is a nested drop-down (Figure 9) to give a structured view of them. This drop-down is dynamically generated from the columns in the CSV file so it will be very easy to track new data in the future.

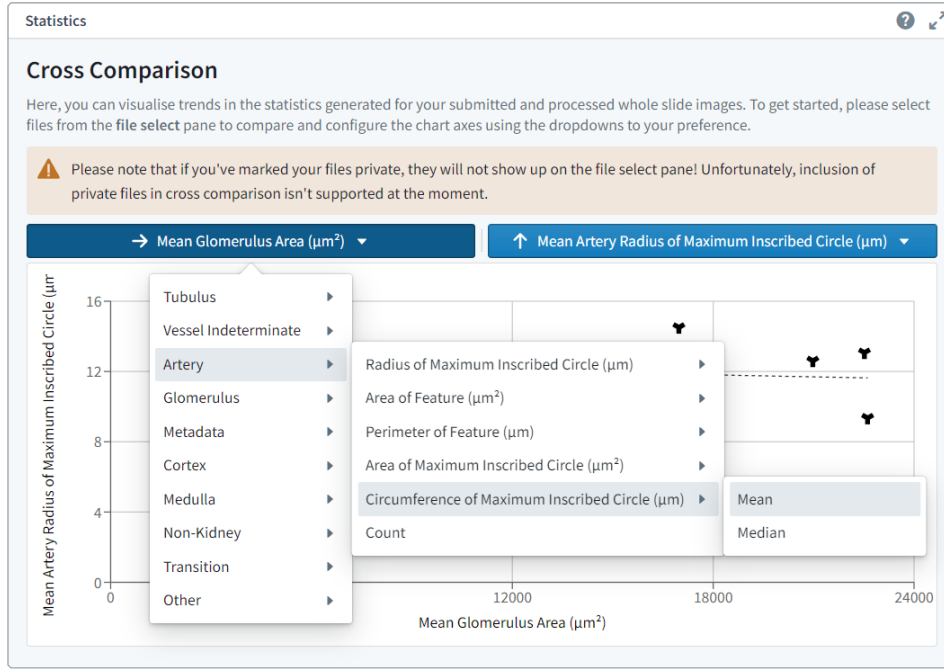


Figure 9: Axis selection

Under cross-slide comparison, each scan provides one data point representing the aggregated measurement from that scan. Using a file picker, the user can select which scans' data they'd like to include. We expect **KidneyCaliper** to gather a large amount of data through long term use, hence the file picker can filter by upload date to prevent old data from cluttering up the interface.

Both of these data visualisation interfaces provide the user with an enormous amount of flexibility and freedom. We want to empower our users with an exploration tool, a torch into the unknown, so that they can display potentially new scientific results and correlations. Research is highly non-linear, and any interesting observation - sometimes quite minor - can become a lead to a major discovery. Researchers build a mental record of these observations from working with a large amount of data for a considerable amount of time. **KidneyCaliper**'s visualisation tools facilitate the discovery of these potential leads and present them in a structured, tangible manner that can be reproduced.

3.5 Viewing and Editing Annotations

We required a specialised plugin to allow very large whole slide images to be viewed in our application. There were two different open-source JavaScript libraries that we could have used: Leaflet[30], which is usually used for maps but would be able to load our tiled whole slide images as well, and OpenSeadragon[31], which is a lightweight tool used in education and medical research. Although the two tools are similar and come with a wide range of plugins, we found that only OpenSeadragon is compatible with Annotorious[17], an image annotation library with a specific OpenSeadragon plugin[32]. Annotorious would allow us to overlay annotations over the basic OpenSeadragon image viewer, and comes with its own wide range of plugins to allow for annotation editing. Because of this, and because we found complaints from users that the zooming, panning, and dragging functionality in Leaflet was not as fluid as OpenSeadragon, we chose to use OpenSeadragon as the image viewer for our application.

The Annotorious plugin supports annotations in the W3C Web Annotation model[24] instead of the GeoJSON[12] format supported by QuPath, which is produced by the two models. Therefore, we had to convert back-and-forth between these two formats to view annotations, although this was not too challenging as they are both JSON formats. Both formats also allow metadata about the annotations to be stored within the annotations themselves. Labels, indicating both the kidney region and the feature-of-interest, allow annotations to be filterable in our interface, as seen in Figure 10, and can also be changed in the case of a misclassification by the models. Measurements can also be stored, allowing these to be viewed by users on our interface.

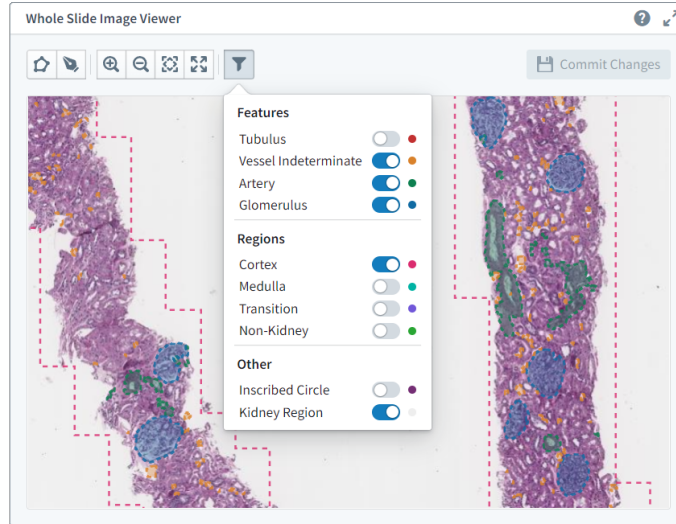


Figure 10: Annotation filter menu

As well as allowing annotations to be viewed on the OpenSeadragon image viewer, Annotorious (with its additional plugins) also comes with functionality to create new polygon and freehand annotations, and edit the labels of current annotations. When new annotations are created, measurements are automatically generated. However, Annotorious does not provide a backend for updating changes for the annotations, so we propagate these changes to the W3C, GeoJSON, and CSV files when a user saves their changes, using JS event handlers.

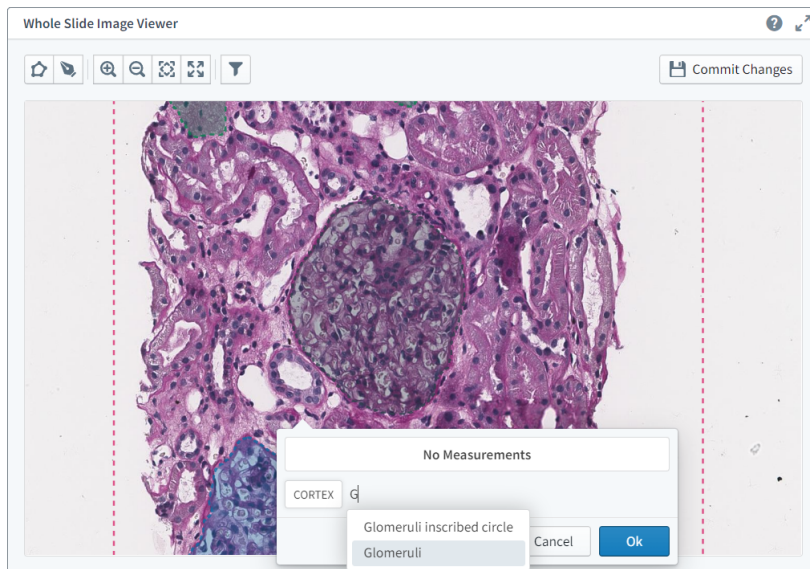


Figure 11: New annotation, just before being created

3.6 Moving Away From Streamlit

Streamlit[33] is a free and open-source Python web framework that allows programmers to prototype data-based web apps. It has a wide variety of pre-built components that allow developers to make user interfaces in a short amount of time. It is also often used to build data-driven dashboards and machine learning web apps. The process of deploying a web app using Streamlit is straightforward and quick, which made it a perfect choice for our group initially.

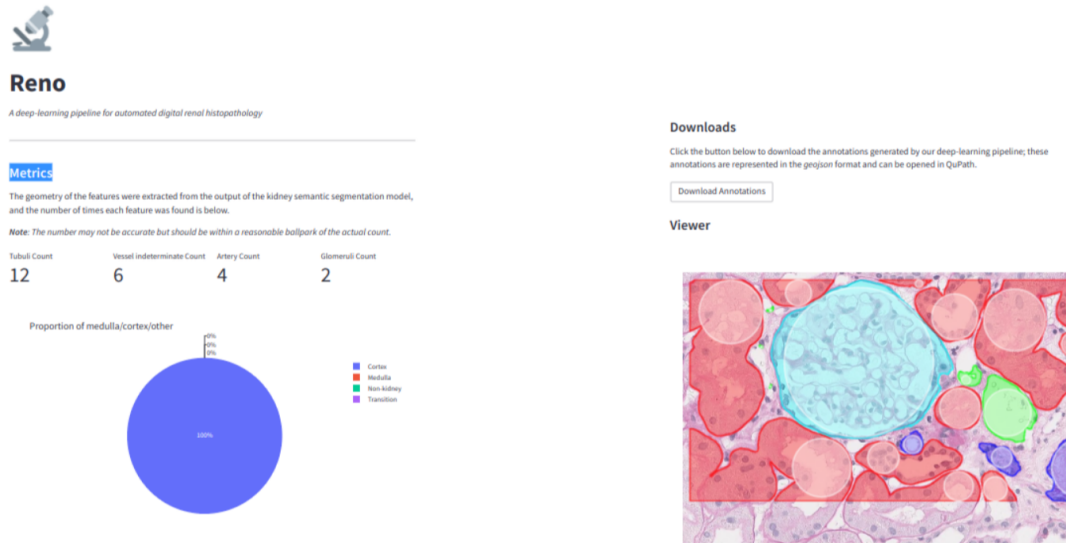


Figure 12: A prototype of **KidneyCaliper** implemented using Streamlit

By using Streamlit we were able to easily develop our web app while writing minimal frontend code since we could use the pre-built components it provides. It meant that we could focus on developing our backend pipeline without spending too much time on our web app code. Streamlit helped us a lot in the first iteration of our project but we quickly discovered several drawbacks.

Firstly, adding a new component to Streamlit is not necessarily straightforward. When creating a new Streamlit component, a lot of boilerplate code must be written. In the long run, if we were to keep developing our Streamlit architecture we would have to dedicate significant time on creating and maintaining custom components.

Secondly, Streamlit does not support multi-page websites and custom HTTP endpoints. This makes routing impossible and limits the complexity of web applications that can be built with the framework. We quickly realised that the app that we were trying to build will be more complex than a standard one-page Streamlit web app. For example, when using Streamlit, a pathologist would not be able to share a URL of the results of one of their kidney images with their colleagues, something they indicated would be crucial.

In conclusion, we realised that Streamlit is simply not built for production-level, complex web applications. After the first iteration, we had to make a difficult decision to either keep Streamlit or change our architecture to a more standard React-based[16] frontend with a FastAPI[34] backend. It was a challenging choice because of the time pressure of the project. Changing our architecture would allow us to introduce more complex features faster and we would not have to worry about Streamlit limitations. However, we estimated that migrating to the new architecture would take us about a week, which was a significant amount of time. Despite this, we decided that changing our architecture would be the safer option since we would not have to worry about being restricted in what we can do with our web app. In the end, the migration took us more than a week, but it was worth it, as we ended up developing a lot of features that would not be possible to implement with Streamlit.

3.7 Clarification: outline of modifications to the deep learning model code

The code in modules `models.cortex_medulla_classifier` and `kidney_semantic_segmentation` is based on previous work by Sarapata[2] and Fu[3], respectively. To avoid ambiguity regarding the attribution, an outline of modifications we made to their original code over the course of the project is listed below.

- Removal of training and visualisation utilities
- Changing the interface from a CLI to an API to allow library-style usage

- Returning raw annotation geometry as result, rather than plotting them
- Generating feature and region statistics
- Passing in parameters and file paths as a configuration object read from file rather than using hard-coded values
- Factoring out elements present in both modules
- Various optimisations (See section 5.3 for details)

We did not make any modifications to the pre-trained model files. Other parts of the project (i.e. the frontend, the web server, integrating the two models into an image processing pipeline) were written by us from scratch.

4 Group Work Methodology / Project Management

4.1 Workflow

We followed the Kanban methodology for managing the work in our group. We found that Kanban aided us in making the right decisions about prioritising work and identifying bottlenecks. We used GitHub’s integrated Project Boards feature[35] to create a Kanban board that updated accordingly each time we modified an issue or pull request in our repository. This meant we did not have to go out of our way to maintain an up-to-date board.

In addition to GitHub’s Project Boards, we heavily relied on their Actions feature to implement continuous testing and continuous deployment for our project. We used a CSG-issued Cloud VM[10] as the continuous deployment target (reachable via <http://kidneycaliper.lucidifai.com/>). We found that having an instance of the app always available and up-to-date with the latest changes helped us easily get feedback from our clients and supervisor.

We decided early on in the project to introduce an obligatory pull request review policy: any changes merged into the `master` branch had to be reviewed by at least one person other than the pull request author. While this introduced some slowdowns to the development process, we found that the benefits of reviews outweighed the cons. Reviews helped us fix bugs early on and offered a chance to refine design decisions before the code was merged. The process also helped everyone on the team get familiar with all parts of the codebase, even those they did not actively develop.

4.2 Weekly Meetings / Iteration Structure

The meetings that structured each of the project iterations were as follows:

- **Weekly internal team meetings at 10 am on Mondays**
We used those to decide what work we needed to do before next week and assign tasks.
- **Weekly client meetings at 1 pm on Thursdays**
During those meetings we discussed our ideas with Dr Candice Roufousse, a Clinical Senior Lecturer in the Department of Immunology and Inflammation, and Callum Arthurs, a Digital Pathology Research Assistant at Imperial. Their experience in histopathology and digital medical imaging proved essential to making well-informed decisions about the scope of our product and prioritising various features over others.
- **Fortnightly supervisor meetings before each checkpoint**
During the checkpoint meetings, we discussed our progress and upcoming iteration goals with our supervisor, Dr Bernhard Kainz.

We found this schedule worked quite well, as it allowed us to work on a given set of features from Monday to Thursday, then verify we were on the right track by demoing them to our clients on Thursday, then adjust if necessary and finalise them by next Monday.

5 Evaluation

5.1 Testing

5.1.1 Image Processing Pipeline Testing

We tested any new backend code we wrote using unit tests with Python’s `unittest` framework. The code for interfacing with the neural networks by Sarapata and Fu[2][3] did not come with tests relevant to the parts of their codebase we ended up using. We decided that writing unit tests for the existing code would not be a good use of our time and also out of the scope of this project. However, we still wanted to test this code, since we were going to modify it for the purposes of optimisation and interfacing with our code. Therefore, we decided that the most suitable way to test it was integration testing.

Our integration tests involved sending requests to a test instance of the server that starts the deep learning pipeline, checking the pipeline has been initiated, and checking the results are well-formatted once the pipeline has finished.

5.1.2 Annotation Format Conversion Testing

We were also able to thoroughly test the code used for converting between GeoJSON[12] and W3C web[24] annotation formats, using Python’s `unittest` framework. This was particularly useful as it was critical that annotations were formatted correctly, or changes to the annotations on our interface might not be saved or exported correctly, breaking compatibility with industry-standard software. These tests were part of the CI pipeline for the project, helping us to easily spot bugs in the codebase.

5.2 User Evaluation

We found the feedback from our users to be invaluable over each iteration in crafting the application to their intended workflow. Our user’s evaluation of the end product is listed below; we feel happy to have accomplished our goal of making a positive impact in their workflows.

“Some of the tasks that I perform are time consuming and mundane, like counting glomeruli. There are also untapped clinically useful data that we’re missing out on due to human limitations. For example, research has shown that the average glomeruli and tubuli size are related to patient outcomes, but we’re not able to provide a reliable estimate of these in clinical practice. **KidneyCaliper** could save me time by performing those mundane tasks for me, and it could also measure and provide statistics on all kidney structures, providing patients with the best possible information on the state of their kidneys”

— *Dr Candice Roufosse*

“The **KidneyCaliper** application has essentially sped up my workflow by 1000% because previously we had a set of deep learning models that were optimised from a research point of view but not from a production point of view. So, when I provided this to the team, they initially sped up the models by about 1000 fold. Slides that would take 20 minutes to run can be analysed in under a minute, and this has empowered my whole workflow. Now I’m able to use the platform to upload my data, and this is patient data for about 400 patients and it’s soon to be a multi-centre study with thousands of patients. This is going to provide me with a tonne of data to analyse further down the line.”

— *Callum Arthurs*

5.3 Optimisations

Here is a comparison of the processing speeds on a typical whole slide scan (17996x44931 pixels, 180MB Aperio SVS format), on a laptop with i7-8570H with 12 threads, connected to an Nvidia GTX 980 GPU (released in 2014).

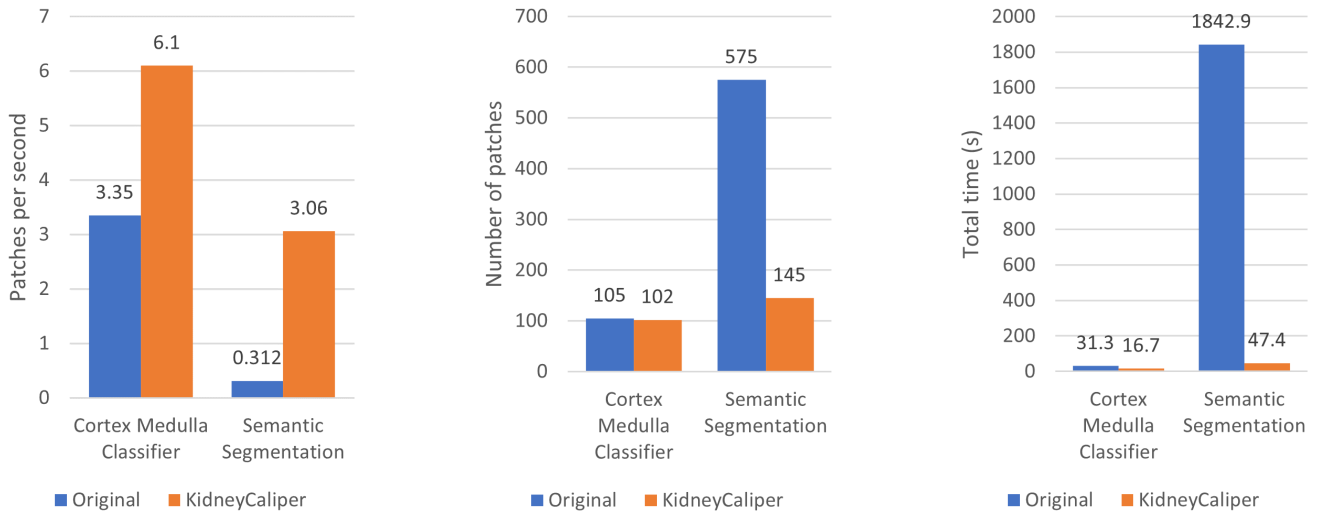


Figure 13: Processing improvements to a typical whole slide image

Thanks to parallelisation, we have drastically improved the processing speeds of both models. The cortex-medulla classifier saw an 82% improvement, from 3.35 patches/s to 6.10 patches/s. The semantic segmentation part, which already has a more complex deep learning model and post-processing needs, enjoyed an almost 9-fold speedup, from 0.312 patches/s to 3.06 patches/s. This improvement is also amplified by an improved thresholding algorithm which means the models have far fewer patches (empty spaces) to process; this is especially beneficial to semantic segmentation, which reduced the amount of work by 75%.

The overall processing time for this slide has been cut from more than 30 minutes to between 1-2 minutes. Naturally, this figure will vary depending on the hardware in use and the size of the scan, but the improvement will be for sure reflected in real workflows.

These speedups are not merely improvements, but a critical factor in the ultimate viability of **KidneyCaliper** for our users. A lab may produce some 400 whole slide scans per day. If each were to take around 30 minutes to process, then the rate of processing results will be lower than the rate of incoming scans, meaning some slides will have to be skipped, and the insights contained within forever buried. With our optimisations and reasonable hardware, researchers can leave **KidneyCaliper** to run overnight and have all the results from the previous day ready by the morning. This also means that our users will not need a dedicated server to run **KidneyCaliper**, as installing an Nvidia GPU produced within the last decade onto their existing workstation will be sufficient.

5.4 Potential Extensions

5.4.1 In-Browser Deep-Learning Pipeline

One idea that we considered was the possibility of removing the entire backend of our solution as an extension project and allowing it to run in the browser. This would theoretically be possible using a JavaScript library such as Tensorflow.js[36] to run the deep-learning models in-browser. It would also be possible to convert the existing models that were written in PyTorch[37] into Tensorflow.js models.

The major benefit of removing any backend is that any patient or personal data is kept completely offline and never sent to an external server. This makes it much more appropriate when used in a clinical setting, where NHS patient data is very tightly locked down. All the images we are using are anonymised and have been approved for use in this project, but if that were not the case, a solution that is completely local would remove any potential privacy-related issues.

We ultimately decided against implementing this as it would require re-implementing the image processing pipeline from scratch, and we could expect performance to be much worse due to multi-processing limitations

in-browser. Moreover, there are not as many high-quality scientific libraries available for JavaScript as there are for Python, which would make writing the code for processing model output more challenging. Since few pathologists will be using it at once, and due to personal data concerns, **KidneyCaliper** has been designed to be primarily run on a local network, rather than being accessible from anywhere. This means that in the intended use-case, private patient data will not be sent to an external server anyway, reducing the importance of having everything in-browser.

5.4.2 Active Learning Loop

Another possible way to extend **KidneyCaliper** is to use it as a part of an active learning loop. Our work provides an easy-to-use interface, which professionals can use to correct the labels outputted by the neural networks. If desired, the corrections made by pathologists can then be included in a dataset used to retrain the networks.

One of the reasons we did not further explore this path is that, due to medical data protection policies, it is usually not acceptable to use the whole slide images that clinicians and researchers work with to train a model.

5.4.3 User Authentication

From the beginning of the project, we realised that user authentication would be a ‘nice-to-have’ but would not be very beneficial to the end-user. User authentication would allow only specific people to access specific whole slide images, making them inaccessible to anybody else. This seems beneficial, particularly in our case where patient data may be uploaded, but in reality, the pathologists we spoke with were very clear that they expect **KidneyCaliper** to be used in a locally-hosted environment. There will be very few pathologists working on the same instance of **KidneyCaliper**, and so they can be trusted to use the software correctly and all users have full access to the data. This meant this was a low-priority feature to implement, as we would have not had the time to implement more important features like the comparison of statistics on historical runs.

Despite this, user authentication would be the next thing we implement as it makes **KidneyCaliper** more appropriate for general use. As an interim solution, we have added the option to upload private images which will not be displayed on the ‘past runs’ page, but are still accessible via a UUID that is copied to your clipboard on submission. This is a primitive ‘user authentication’ system that works well for the current pathologist’s workflow.

6 Ethical Considerations

6.1 A GDPR non-compliant Streamlit

While developing our prototype in the first iteration, we were disappointed to find out that Streamlit Inc. collects telemetry containing personal information (such as IP addresses) from both the server on which Streamlit is deployed, as well as from the browsers of all visitors to the Streamlit-based website. The visitors are never informed of the data collection. Streamlit does inform the server administrator of data collection during its installation and the method to disable it (by editing a configuration file after installation); however, the user has no choice but to agree to telemetry collection, and positive action is needed to disable it. If the collection is enabled on the server-side, then Streamlit will also collect telemetry from visitors to the site and send the data to Streamlit Inc. The visitor will not see a privacy notice unless they scroll to the bottom, click to Streamlit’s homepage streamlit.io, scroll to the bottom again and click privacy policy.

It’s important to note that although Streamlit is open-source software that can be hosted by anyone on any server, all telemetries are sent to Streamlit Inc., a for-profit business, not the hoster.

To collect and process personal information in compliance with UK data protection laws (primarily the UK GDPR), Streamlit Inc. must rely on one or more of the six lawful bases laid out in the statute. Only two can be potentially applicable to Streamlit: consent and legitimate interest. Commonly, software telemetry collection relies on consent, which has a specific meaning within GDPR[38]:

‘consent’ of the data subject means any freely given, specific, informed and unambiguous indication of the data subject’s wishes by which he or she, by a statement or by clear affirmative action, signifies agreement to the processing of personal data relating to him or her

Opt-in by default collection does not constitute valid consent. As such, we have made a pull request[39] to Streamlit, making telemetry disabled by default instead. In the same PR, we have asked Streamlit which legal basis, if not consent, is used to process personal information. Streamlit rejected our PR and cited legitimate interest as their legal basis for processing telemetries.

Whether legitimate interest is applicable here is a subjective matter with a balancing exercise required. We will note that even if telemetry collection constitutes legitimate interest, Streamlit Inc. is still required to inform users that their personal information is being processed, the legal basis for processing, and that users may opt out. This is usually contained in a document named privacy notice. However, as mentioned above, for visitors to a self-hosted Streamlit site, this is not at all clearly presented.

While the data displayed through Streamlit is never sent to Streamlit Inc., we are nevertheless dissatisfied with Streamlit’s approach to user privacy. Given the stringent data security context we are working in, this was one of the drivers for us to move away from Streamlit.

6.2 Patient data handling

Since the beginning, we knew that we will have to be extremely cautious with the way that we are using data, because of the medical nature of our project. As we mentioned before, our group focused on research rather than clinical usage of our software. Therefore we did not have to follow all of the restrictions that a medical device would have to follow[40]. Instead, we only had to follow the research ethics granted by IRAS for the project[41].

According to those rules, all data that is used in the research, outside of the clinic has to be pseudo-anonymised. It should be noted that the only patient data we are using in our project are kidney images. We do not collect any other patient data such as names, NHS numbers, or any other identifiable information. Researchers that we worked with obtained the renal transplant biopsy FFPE blocks from the Imperial College Healthcare NHS Trust Tissue Bank[42] (REC 21/ES/0080) (approved project number R18040) and scanned them using a Leica Aperio CS2 slide scanner. All of the kidney images were anonymised before we were able to use them. The names have been randomly generated and the labels have been removed from the slides making all of the images anonymous. Therefore we could lawfully use them during the development process.

KidneyCaliper is currently hosted on the Department of Computing Virtual Machine, which is not a part of the NHS trust network. As a result, any user that wants to upload their images to the website has to anonymise them. However, **KidneyCaliper** can also be hosted by researchers locally. In that case, if pathologists run our software on their machines in a secure NHS trust network (e.g. clinic or hospital) then they may not have to anonymise their data as our system can be self-contained.

References

- [1] S. Bainbridge, R. Cake, M. Meredith, P. Furness, and B. Gordon, “Testing times to come? an evaluation of pathology capacity across the UK,” Nov. 2016.
- [2] G. Sarapata, B. Kainz, and C. Roufousse, “Deep learning-based segmentation and quantification in kidney transplant pathology,” Master’s thesis, Imperial College London, Sep 2021.

- [3] J. Fu, B. Kainz, and C. Roufosse, "Mapping needle core biopsies to kidney zones using swin transformer," Master's thesis, Imperial College London, Sep 2021.
- [4] A. Denic, J. Mathew, V. V. Nagineni, R. H. Thompson, B. C. Leibovich, L. O. Lerman, J. C. Lieske, M. P. Alexander, J. J. Augustine, W. K. Kremers, and A. D. Rule, "Clinical and pathology findings associate consistently with larger glomerular volume," *Journal of the American Society of Nephrology*, vol. 29, no. 7, 2018.
- [5] "Cleveland Clinic - Glomerular Diseases." <https://my.clevelandclinic.org/health/diseases/5993-glomerular-diseases>.
- [6] V. G. Puelles and J. F. Bertram, "Counting glomeruli and podocytes: rationale and methodologies," *Current opinion in nephrology and hypertension*, vol. 24, no. 3, pp. 224–30, 2015.
- [7] A. Z. Rosenberg, M. Palmer, L. Merlino, J. P. Troost, A. Gasim, S. Bagnasco, C. Avila-Casado, D. Johnstone, J. B. Hodgins, and C. Conway, "The application of digital pathology to improve accuracy in glomerular enumeration in renal biopsies," *PLoS One*, vol. 11, no. 6, 2016.
- [8] Y. Ozluk, P. L. Blanco, M. Mengel, K. Solez, P. F. Halloran, and B. Sis, "Superiority of virtual microscopy versus light microscopy in transplantation pathology," *Clinical transplantation*, vol. 26, no. 2, pp. 336–344, 2012.
- [9] "Renal Tubule - National Cancer Institute." <https://www.cancer.gov/publications/dictionaries/cancer-terms/def/renal-tubule>.
- [10] Computing Support Group, Department of Computing, Imperial College London, "Private IaaS Cloud." <https://www.imperial.ac.uk/computing/people/csg/services/cloud/>.
- [11] P. Bankhead, M. B. Loughrey, J. A. Fernández, Y. Dombrowski, D. G. McArd, P. D. Dunne, S. McQuaid, R. T. Gray, L. J. Murray, H. G. Coleman, J. A. James, M. Salto-Tellez, and P. W. Hamilton, "Qupath: Open source software for digital pathology image analysis," *Scientific Reports*, vol. 7, no. 1, p. 16878, 2017.
- [12] "GeoJSON - format for encoding a variety of geographic data structures." <https://geojson.org/>.
- [13] "SQLite Home Page." <https://www.sqlite.org/>.
- [14] "PostgreSQL: The World's Most Advanced Open Source Relational Database." <https://www.postgresql.org/>.
- [15] "SQLAlchemy - The Database Toolkit for Python." <https://www.sqlalchemy.org/>.
- [16] "React - A JavaScript library for building user interfaces." <https://reactjs.org/>.
- [17] "Annotorious." <https://recogito.github.io/annotorious/>.
- [18] "Blueprint - A React-based UI toolkit for the web." <https://blueprintjs.com/>.
- [19] K. Verdick, "react-mosaic." <https://github.com/nomcopter/react-mosaic>, 01 2022.
- [20] B. Frost, "Atomic design." <https://bradfrost.com/blog/post/atomic-web-design/>, 06 2013.
- [21] "Nielsen Norman Group: UX Training, Consulting, & Research." <https://www.nngroup.com/>, 2019.
- [22] "CUDA C++ programming guide." <https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#concurrent-kernel-execution>.
- [23] "Rasterio Python library GitHub repository." <https://github.com/rasterio/rasterio>.
- [24] R. Sanderson, P. Ciccarese, and B. Young, "Web annotation data model," *W3C Recommendation*, Feb. 2017.

- [25] “Polylabel - A fast algorithm for finding the pole of inaccessibility of a polygon.” <https://github.com/mapbox/polylabel>.
- [26] “Polylabel - A fast algorithm for finding the pole of inaccessibility of a polygon.” <https://github.com/mapbox/polylabel>.
- [27] “Pylylabel - A Rust implementation of the Polylabel algorithm.” <https://pypi.org/project/pylylabel/>.
- [28] J. Ball, “Include measurements when creating annotations from imported geojson data #835.” <https://github.com/qupath/qupath/pull/835>.
- [29] “Recharts - Redefined chart library built with React and D3.” <https://github.com/recharts/recharts>.
- [30] V. Agafonkin, “Leaflet.” <https://leafletjs.com/>.
- [31] “OpenSeadragon.” <https://openseadragon.github.io/>.
- [32] “Getting Started with the OpenSeadragon Plugin.” <https://recogito.github.io/annotorious/getting-started/osd-plugin/>.
- [33] “Streamlit - The fastest way to build and share data apps.” <https://streamlit.io/>.
- [34] “FastAPI framework, high performance, easy to learn, fast to code, ready for production.” <https://fastapi.tiangolo.com/>.
- [35] “About project boards – GitHub Docs.” <https://docs.github.com/en/issues/organizing-your-work-with-project-boards/managing-project-boards/about-project-boards>.
- [36] “TensorFlow.js is a library for machine learning in JavaScript.” <https://www.tensorflow.org/js>.
- [37] “PyTorch Python library.” <https://pytorch.org/>.
- [38] “Regulation (EU) 2016/679 of the European Parliament and of the Council,” *OJ*, vol. L 119, p. 1–88, 2016.
- [39] A. Wang, “Stop collecting telemetry by default #3952.” <https://github.com/streamlit/streamlit/pull/3952>.
- [40] “A guide to good practice for digital and data-driven health technologies.” <https://www.gov.uk/government/publications/code-of-conduct-for-data-driven-health-and-care-technology/initial-code-of-conduct-for-data-driven-health-and-care-technology>.
- [41] “Intergrated Research Application System.” <https://www.myresearchproject.org.uk/>.
- [42] “Imperial College Health Tissue Bank.” <https://www.imperial.ac.uk/tissuebank>.