Topo Health Portfolio

Nicholas Dzomba, Ben Holmgren, Joaquin Monterrosa

Introduction

Motivation

According to the American Cancer Society, lung cancer is the leading cause of cancer-related death in the United States, making up over 25% of all cancer deaths. They estimate that in 2021, there will be 235,760 new cases of lung cancer and 131,880 deaths from lung cancer in the United States. In one's lifetime, the odds of getting lung cancer are 1 in 15 for men and 1 in 17 for women [1]. Worst of all, 75% of cases are diagnosed at stage 3 or 4 [2], which is problematic because early detection can improve 5-year survival rates anywhere from 2-10 times [3].

There are two primary factors to this problem. First, radiologists are extremely expensive. The average radiologist makes over $250,000 a year, or $138 an hour [4]. This problem is amplified for rural communities, which struggle to get access to radiologists. Once they do get access, there are typically higher expenses for them as well. In particular, the Northwest has a severe lack of access to quality screenings and care, as there are no National Cancer Institute designated cancer centers in Montana, Alaska, Idaho, Wyoming, North Dakota, or South Dakota [5].

Secondly, the accuracy of screenings conducted by radiologists is imperfect. Studies have shown that conventional radiologists maintain a 0.75 specificity rate when diagnosing lung cancer in CT scans [21]. Specificity describes the rate at which a radiologist accurately diagnoses a cancer-positive patient. Based on our calculations in Appendix A, radiologists maintain a 0.79 general accuracy rate. General accuracy describes the rate at which a radiologist accurately diagnoses any patient regardless of whether they do or do not have cancer. Ultimately, these accuracy rates indicate that a significant number of lung cancer cases fail to be caught especially in the early stages.

This is where Topo Health comes in. Topo Health is a state-of-the-art AI model used to diagnose lung cancer wrapped in an intuitive, secure web app. The application uses a two-dimensional convolutional neural network, with convolutions occurring slice-by-slice in the processed CT scan. Our neural network, implemented using tensorflow, achieves an 82% accuracy rating, performing slightly better in practice than we would expect from a human radiologist. On top of that, it can significantly ease the expense of screenings. Per scan, Topo Health takes 90 seconds on average to preprocess the scan and complete a diagnosis, which we would expect to greatly surmount the speed of a conventional radiologist. If Topo Health can streamline the work of a radiologist by two hours it could save around $276 per screening, and if it can entirely replace a radiologist it has the potential to save over $500 per screening. As there were 262,700 lung

cancer screenings in the United States in 2015 [8], Topo Health's potential impact on accuracy and savings will be substantial.

As for the broader impact, lung cancer screenings have an annual cost of $78.8 million to Americans each year [9]. It has also been estimated that lung cancer as a whole costs the United States 21.3 billion dollars annually [10]. Most importantly, there are an average of 131,880 lung cancer deaths in the United States each year. Topo Health can help to reduce that number by providing cheaper, more accessible, and more accurate screenings.

Research Methods

Our methods involved testing multiple learning models to diagnose lung cancer in CT scans. In particular, we implemented a 3-dimensional convolutional neural network, a 2-dimensional convolutional neural network, and a k-nearest-neighbor model using techniques from computational topology.

The most widely studied choice in the literature is a convolutional neural network (CNN). CNNs are the foremost type of neural network for image classification problems. In some cases, deep convolutional networks have performed extraordinarily well at diagnosing lung cancer in CT scans, achieving accuracy rates exceeding 90%[16][17]. We study two different CNNs, one which performs 3D convolutions and one which performs 2D convolutions on each slice in a DICOM image. Each model makes use of Tensorflow implementations, which is among the preeminent Python libraries for open source neural networks. We investigated both variants of CNNs using an automated hypertuning process, which optimizes our networks according to depth in convolutional layers and the filters applied.

We also investigated the use of topological data analysis (TDA) techniques for lung cancer diagnosis in a CT scan. TDA techniques have achieved high efficacy when detecting cancer in histopathology slide data[22], but remain relatively untested in the domain of CT scan cancer diagnosis. More specifically, we used a Vietoris-Rips filtration on the point cloud generated from the 3D mesh of each lung. This outputs a persistence diagram representing topological descriptors of the lung. Persistence diagrams are then used in a k-nearest-neighbor model (KNN), where distance is computed using the bottleneck distance. Though intriguing and potentially novel in this domain, we eventually abandoned TDA models in our experimentation due to their high computational complexity.

Finally, to make our machine learning possible, the scan data for our experimentation was pulled from the open source data provided by The Cancer Imaging Archive and the Kaggle 2017 Data Science Bowl. To evaluate the performance of our models, we establish their total accuracy in providing a correct diagnosis, and their sensitivity. We then compare models to the sensitivity and accuracy rates of conventional radiologists.

Hypothesis

Our hypothesis is that the 3D convolutional neural network will outperform the average radiologist, since it has shown the most promising results in the literature for lung cancer screenings. We are not sure how the TDA model will perform, but it is not unreasonable to expect that it may be comparable to the averages obtained by conventional radiologists.

Engineering Product

Our engineering product, titled "Topo Health", is an intuitive & secure web app built around the AI model discussed above. This product allows radiology technicians to automatically process lung cancer scans. The web app works as follows:

Step 1: A technician will complete a CT scan.

Step 2: The technician will upload the scan to our intuitive, secure web app.

Step 3: The scan will be processed through our state-of-the-art diagnostic AI model.

Step 4: After a few minutes the technician will get back the results from their diagnosis.

While this is the core of our app's functionality, it doesn't paint the whole picture of what our app does. First, the Topo Health app will be purchased by clinics and used by radiology technicians. We're not sure if it will replace the role of a radiologist or be used as a tool alongside a radiologist. Either way, the app has an administrative view for the clinic along with a technician provisioning system. After logging in, technicians can find past scans and results using a myriad of search and filter tools. Additionally, the app is HIPAA-ready, meaning that all protected health information (PHI) is transferred and stored securely. Last and most importantly, this product has been built with the input of our target users: technicians. For this reason, we conducted a customer discovery sprint to learn about the specific needs and use cases of our target customer before building the app. Details about this customer discovery sprint can be found in Appendix D.

Qualifications

Every member of our team brought their own unique set of abilities integral to the success of our project. Ben Holmgren is pursuing a dual degree in mathematics and computer science with a data science minor. Over the past four years, he has been conducting research in computational topology. His exploits have led him to publish papers at international conferences, establishing novel theoretical results aimed at improving the broad viability of TDA. In addition, he has helped to maintain the open-source R "TDA" package, troubleshooting issues and co-creating an

NSF workshop to teach topological data analysis using R to undergraduates throughout the country. Ben is our group's topological data analysis (TDA) and general computational geometry expert, and he led the development of our TDA models and the conversion of a DICOM image into usable data.

Nicholas Dzomba is also a double major in mathematics and computer science with a minor in data science. He has a strong background in machine learning, including experience using deep learning with Los Alamos National Laboratory. He and Ben spearheaded our group's efforts in developing the Convolutional Neural Networks (CNNs), which were tested alongside our TDA model.

Joaquin Monterrosa is pursuing a double major in business accounting and computer science. He is our group's expert in business viability and in web development and was responsible for the success of our model's web interface. He has four years of experience as a venture consultant at the Blackstone Launchpad and two years of experience doing freelance web-app development.

Our resumes are presented below:

Benjamin Holmgren

Email: benjamin.holmgren1@student.montana.edu Phone: (406)-599-4614
Website: https://benholmgren.github.io/ben-holmgren/ Address: 86 Scobie Way, Bozeman, MT

Education

Fall 2018-Present **Dual Degrees in Computer Science and Mathematics**, Montana State University, Bozeman, MT
- Data Science Minor
- 3.95 GPA, 3.98 CS GPA, President's List

Experience & Service

2018-Present **Undergraduate Researcher**, *Computational Geometry & Topology Group*, Montana State University
- President of MSU computational geometry & topology club, assisted in organizing 2021 CG fall workshop
- Presented my research in an international conference and in venues throughout the US
- Served as a referee for the Canadian Conference on Computational Geometry

Spring 2021 **Teaching Assistant**, *CSCI 276 - Discrete Mathematics*, Montana State University
- Substitute lecturer for classes of roughly 60 undergraduates
- Led help sessions for students and graded assignments

Selected Intellectual Contributions

Discrete Morse Theory *If You Must Choose Among Your Children, Pick the Right One*

We improve the computational complexity of a fundamental problem in discrete Morse theory. Published in CCCG.
Talk: https://www.youtube.com/watch?v=kHpD-J4EzI8&t=608s
Citation: Holmgren, B., McCoy, B.K., Fasy, B.T., & Millman, D.L. (2020). If You Must Choose Among Your Children, Pick the Right One. *CCCG20*.

Co-Organizer of NSF Workshop *Topology For Data Science 2020* (Postponed)

Created an open source tutorial project in 2018 on techniques in TDA which grew into an NSF workshop in 2020. To be held in Spring 2021. The MSU news wrote an article about my work here.
Original tutorial project and work with the R package was presented in the MSU Undergraduate Research Celebration 2019 Poster Session: *Updating the R Package 'TDA'*.

National Conference on Undergraduate Research 2020 (Given in April 2021)

Poster Title: *Using Hasse Diagrams to Compute a Gradient Vector Field,* outlines my contributions in discrete Morse theory and its applications in efficient data simplification and processing.

Fréchet Properties *Path-Connectivity of Spaces of Graphs under the Fréchet Distance*

Collaborators: Fasy, B.T., Majhi, S., Wenk, C. We prove an array of topological properties of the Fréchet distance, which is particularly useful when analyzing network data. To appear in EuroCG 2022.

Visualization Project *Poking a Simplicial Complex* Collaborators: Marco Huot, Brad McCoy

Multidisciplinary Project to visualize Morse theory as part of a thesis project for students in the art department.

Honors & Awards

- MSU Cameron Presidential Scholar, John C. Felton & John L Magaret Scholarships
- 2019/20 SOC Undergrad. Researcher of the year, 2020/21 MSU Math Outstanding Scholar award
- Phi Kappa Phi & Pi Mu Epsilon Honors Societies

Skills & Interests

- Python, C/C++, Java, R, Matlab, Git, Latex, TCP/IP, OpenGL, Security, HTML/Javascript/CSS
- Algorithms, Topology, Abstract Algebra, Alpine Climbing, Rich Teamwork

Nicholas Dzomba

Contact

1500 Topaz Dr.
Missoula, MT, 59808
USA

(406)546-2367

nicustm@gmail.com

Skills

Technical Skills

Python
Java
C/C++
R
MATLAB
NumPy
MatPlotLib
PyTorch
PyQt
Arduino/Raspberry Pi
SQL/MySQL
Git/Github
LaTeX
Linux Shell
Adobe Creative Cloud
Microsoft Office

Soft Skills

Problem Solving
Technical Writing
Technical Presenting
Teaching
Communication
Public Speaking
Collaborative Work
Independent Work

Summary

Computer Science and Mathematics double major, data science minor, in fourth year at Montana State University. Research/Work experience in multiple fields, upper level coursework includes probability theory, dynamical systems, applied mathematics, artificial intelligence, database systems, numerical analysis, mathematical imaging and advanced linear algebra. Also capable in Python, Java, C, R, and MATLAB programming.

Education

2018 - **B. Sc. in Computer Science and Mathematics, Minor in Data Science** — Montana State University

3.71 GPA. Expected spring 2022 graduation. Courses taken include applied mathematics, dynamical systems, numerical analysis, advanced linear algebra, mathematical imaging, robotics, database systems as well as Python, Java, R, and C/C++ programming. Fall 2021 coursework includes artificial intelligence, UX design, computational biology, probability theory, and capstone project using machine learning and topological data analysis to improve cancer detection.

Work Experience

2021 **Los Alamos National Laboratory** — Los Alamos, NM

Los Alamos Dynamics Summer School Fellow

Fellowship as part of 2021 Los Alamos Dynamics Summer School, on Waveometry: Multi-Layer Input Deep Learning for Ultrasonic Wavefield Measurements project. Duties included problem solving, convolutional neural networks with PyTorch, Python and MATLAB programming, technical writing and presenting.

2021 **Montana State University** — Bozeman, MT

Computer Science Teaching Assistant

Teaching assistant for a lower level computer science course in Spring 2021 and Fall 2021. Duties include running a lab section, assisting students and grading.

2020 **Montana Space Grant Consortium** — Bozeman, MT

MSU BOREALIS Summer Intern

Part of a team of interns working on developing for and organizing high altitude weather balloon launches over summer 2020. Work included high altitude ballooning live landing prediction software with PyQt and autonomously steering parachute.

Research Experience

2021 **Los Alamos Dynamics Summer School** — Los Alamos National Laboratory

Aforementioned fellowship as part of 2021 Los Alamos Dynamics Summer School. Research to be presented at 2022 International Modal Analysis Conference.

2018 - 2019 **Computational Topology and Geometry Research Group** — Montana State University

Includes spending summer 2019 working on experiments with sphere stratification for shape reconstruction and auditing graduate level computational geometry course. Duties included Python programming and problem solving.

Awards and Honors

2019-2021 **Outstanding Student Award** — MSU Department of Mathematics

$1500 scholarship for 2019-2020, 2020-2021, and 2021-2022 academic years.

2020-2021 **Sonderegger Tutor** — MSU Department of Computer Science

$1000 scholarship for 2020-2021 academic year for helping tutor computer science students.

2018-2020 **President's and Dean's List** — Montana State University

President's List Fall 2018, Spring 2020, Dean's List Spring, Fall 2019

JOAQUIN MONTERROSA

SKILLS

- UX Research
- Java
- C
- Python
- Javascript
- PHP
- MySQL
- SQLite
- Non-Relational databases
- XML
- HTML
- CSS
- Bootstrap
- Database Structuring
- React
- Firebase
- Web Development
- Mobile Development
- Fluent in Spanish
- Project Management
- Agile Methods
- Graphic Design
- Lo and Hi Fidelity Prototyping
- Excel Power User
- Access Power User

ACHIEVEMENTS

- Won the MSU $50K Startup Competition
- Vice-president of the Pi Kappa Alpha fraternity (110-man chapter)
- Won Techstars' International startup competition at UCLA
- Headed a $3,000 blood cancer philanthropy event.
- Brandon Speth $1,000 Award for Passion, Poise, and Charisma
- Most Innovative Startup, $1000 award
- 1st Place in Elementary School Science Fair (my proudest accomplishment)

WORK EXPERIENCE

Software Engineer at NASA
SUMMER 2020

- Worked on the Kennedy Space Center Launch Control Systems
- Strengthened and expedited hardware configuration update procedures
- Updated launch dashboard
- Trained to adhere to the highest security and testing standards
- Worked with a large, complex software lifecycle
- Learned a ton about space exploration and other engineering realms

IT Analyst at Moss Adams (Accounting Firm, Silicon Valley Office)
SUMMER 2019

- Audited IT systems
- Advised risk mitigation strategies for IT systems
- Advised application and database process improvements
- Advised improvements to software development lifecycles

Business Strategist and Cofounder of Freats LLC
2017 - PRESENT

- Scrum Master
- Strategize long-term activities and goals
- Software development
- Legal and tax oversight

Venture Consultant at Blackstone Launchpad
2017 - PRESENT

- Accelerate and navigate the ideation process of early-stage startups
- Inject agile principles into startups and their founders
- Advise entrepreneurs in navigating the software development and production process
- Growth strategies

EDUCATION

Montana State University: Computer Science & Business Accounting
2017 - PRESENT

- 4.0 GPA
- Full-Ride Presidential Scholar in the Honors College
- Ressmeyer Scholarship for Computer Science Advancement
- Harold and Reta Haynes Business Scholarship
- Jake Jabs Entrepreneurship Scholarship
- President of campus Entrepreneurship Club
- U.S. Bank of Bozeman Accounting Scholarship
- Freshmen Business Mentor (Leadership Position In College of Business)

REFERENCES

J.D SUSAN DANA | DR. DANIEL DEFRANCE | DR. MARYANN CUMMINGS

Reference contacts available upon request.

Background

The process of diagnosing lung cancer is almost universally conducted in person, and the significant costs associated with diagnosis are rooted in the historic reliance on an in-person diagnosis from a radiologist. To address this problem, Topo Health sought to utilize significant advancements that have been made in the artificial intelligence community to automate and streamline lung cancer diagnosis. In recent years, techniques have emerged simultaneously in the topological data analysis (TDA) and deep learning research communities which have shown great promise in cancer detection. Our technology brought these advancements demonstrated in the literature to fruition, with the hope of making an impact in the lives of the cancer afflicted community. We implemented models informed from academic research in order to make these recent advancements in AI accessible to technicians through an intuitive web application. In the future, with the addition of technologies like Topo Health, it is reasonable to expect that automating lung cancer diagnosis could result in substantial monetary savings in the healthcare sector. This could make a meaningful contribution to the democratization of cancer treatment, and most importantly, could save countless lives by expediting & improving the accuracy of diagnosis.

Due to the importance of automatic cancer detection, this problem has been studied extensively. Significant strides have been made independently in both the TDA and deep learning research communities in recent years. Both fields provide unique benefits in addressing the problem, and we brought forward lessons from each area of research in our final product. Conveniently, when using techniques from either field, the workflow is largely the same. The data pipeline consists of identifying data in the form of a CT scan, converting the scan to a grayscale image, and then transforming pixel data into a matrix. (Often, given the resolution of the image, one may want to reduce the resolution for time and space concerns when handling the matrix in a model.) Such matrices can either simply consist of the original grayscale images in the scan, or can be binary given some threshold value in the grayscale image. After data has been successfully preprocessed, it is either fed into a trained machine learning model, or it is processed into a TDA filtration. Both models provide a functional summary of key features in the scan, and based on these summaries, an automatic estimation is made to diagnose an image for lung cancer.

To begin, advances in TDA are worth bringing into the discussion as a viable set of techniques in detecting cancerous regions of a CT scan. Motivated by the ubiquitous importance of shape in data, TDA is an attempt to broadly categorize connectedness within a point cloud. To do so, a field called persistent homology provides the gateway between discrete data and a measurable, continuous space. We attempted to use techniques from persistent homology to quantify the shape in lung cancer CT scans. Namely, we utilized the Dionysus library to implement the Vietoris-Rips filtration [13][14]. In doing so, generated spatial summaries of the connectivity of features within a CT-scan, which are the primary features of interest in diagnosing cancer in DICOM image data and have been useful in the literature[15][7]. Using these spatial summaries

(called persistence diagrams), we can compute measures of distance between the shape of given scans, and based on these distances in shape, can estimate the likelihood with which a given scan presents cancer. We computed distance between persistence diagrams through a metric known as the bottleneck distance, which computes the cost of an optimal matching between two persistence diagrams, and is available in the Python Persim library.

By computing a persistence diagram for a large training set of images, and comparing diagrams between images known to have cancer at varying degrees of severity, we were able to 'train' a supervised model. Then, in using TDA for our engineering product, our initial plan was to integrate the model into our web application by conducting a filtration on images provided by a user, and then simply conducting a nearest-neighbor process in order to provide a diagnosis for the given image. Though this was not our end approach due to time complexity concerns, our implementation was fully able to be integrated in such a web based setting.

Additionally, we felt that TDA was a worthy subject for experimentation in this setting because of its inherent avoidance of the 'black box' tendencies of neural networks. In the future, using techniques in TDA could allow us to not simply stop at a diagnosis. Rather, we could further use distance measures between persistence diagrams to flag connected components within a given scan that are likely to present cancerous nodules. After flagging these specified connected components, we could identify the corresponding pixels in an original image, and highlight these pixels in an output provided for the radiologist. Remarkably, works in the literature have gone even further, and have used acquired persistence diagrams as input into a Cox proportional hazards model to estimate the survivability of the lung cancer presented in a given scan [7]. This is an immense level of diagnostic power to provide to a technician. It is the hope of the authors, as well as the research community more broadly, that such information could provide a greater level of clarity for technicians to provide the best possible treatment for their patients. Though this was out of the scope of our project, it is certainly an intriguing set of features to consider adding in the future.

Another promising advancement in image processing, and the logical next technique for us to try, is the convolutional neural network (CNN). A convolutional neural network is an artificial neural network with additional layers, most notably a convolutional layer. In the convolutional layer, instead of having sets of weights and a weighted sum that a typical layer would have, it instead performs the convolution operation. Standing alone, advancements in TDA make this an incredible moment to tackle the problem of automating cancer diagnosis. But simultaneously, an equally if not even more impressive revolution has transpired in the deep learning research community in recent years using CNNs. Spearheading this effort has been Mozzayir Etemadi and Ulas Bagci of the Northwestern University Feinberg School of Medicine. Etemadi and Bagci alongside other collaborators have successfully implemented a deep learning model to diagnose lung cancer extraordinarily effectively, averaging a 94% success rate in detecting early stage cancer alone. In general, the duo have created models able to diagnose lung cancer in a CT scan

with 95% accuracy. Currently, this is the most effective known method to automatically diagnose lung cancer [16][17]. Most successfully, this has occurred through a 3-layer neural network, and has presented an extremely low rate of false positives of only around 1.2%. Engineers affiliated with Google have jumped on this progress, and have begun collaborating with researchers in the field to develop a preliminary google lung cancer AI. Though Topo Health did not achieve this level of performance, the momentum behind CNNs in the realm of automated cancer detection makes our relative success unsurprising. Additionally, in the future Topo Health is interested in attempting to collaborate with these researchers, and we applied to adopt the preliminary Google lung cancer model into our framework with the hope of testing their model, and of bringing forward these technologies to benefit healthcare in Montana.

Not all of the mentioned technologies were used in the final product delivered by our team. However, in gaining familiarity with each of the primary schools of thought in the field, we achieved a good level of experimentation with this range of models. We not only tested the success rate of each technique, but we also analyzed the practical performance of these techniques in a web-based setting. Thankfully, due to the formidable open source movement in the machine learning research community, the vast majority of these complex technologies were readily implementable using existing free libraries. Consequently, the data pipeline and preprocessing presented by far the largest unknown in this development effort. Our final preprocessing was extremely robust, and adapted from the preprocessing implementation used by the winners of the 2017 Kaggle Data Science Bowl [23]. Once completed, the mentioned techniques were inserted interchangeably into our pipeline, allowing us to much more easily experiment with a large number of models.

It is worth mentioning that it is extremely unlikely that another AI will emerge and be more successful than those developed by the deep learning research originally stemming from the Northwestern School of Medicine. Certainly, the odds were never in our favor to be the ones to do it. Nevertheless, achieving a success rate comparable to that of a typical radiologist could have significant implications for the radiology workflow when coupled with a smoothly running web application. For that accomplishment alone, we feel that our engineering product can be justly measured as a technology with the potential to leave a serious, positive impact.

Work Schedule

We used Notion as our scheduling and scrum software. Below, is an exported representation of our work schedule:

Work Schedule

Name	Assign	Date	Property	Status
Customer Discovery Sprint	Joaquin Monterrosa Ben Nic Dzomba	@January 23, 2022 → January 29, 2022		Completed
Find Initial Data	Ben Nic Dzomba	@January 30, 2022 → February 5, 2022		Completed
Complete Branding Guide	Joaquin Monterrosa	@January 30, 2022 → February 5, 2022		Completed
Low Fidelity Mockups	Joaquin Monterrosa	@January 30, 2022 → February 5, 2022		Completed
Create Initial DICOM Data Pipeline	Ben	@February 6, 2022 → February 16, 2022		Completed
Build Uploader Section	Joaquin Monterrosa	@February 6, 2022 → February 16, 2022		Completed
Data Pipeline with STL	Ben Nic Dzomba	@February 17, 2022 → February 26, 2022		Completed
Setup Bubble (Normal) Database	Joaquin Monterrosa	@February 17, 2022 → February 26, 2022		Completed
Setup Xano (Secure) Database	Joaquin Monterrosa	@February 17, 2022 → February 26, 2022		Completed
Preliminary CNN	Ben Nic Dzomba	@February 27, 2022 → March 5, 2022		Completed
Preliminary TDA with STL	Ben	@February 27, 2022 → March 5, 2022		Completed
Create all other necessary Xano Endpoints	Joaquin Monterrosa	@February 27, 2022 → March 5, 2022		Completed
Get .dcm Files To Upload to Xano Endpoint	Joaquin Monterrosa	@February 27, 2022 → March 5, 2022		Completed
Create Results Section	Joaquin Monterrosa	@March 6, 2022 → March 12, 2022		Completed
Connect Xano Database and Enpoints To UI	Joaquin Monterrosa	@March 6, 2022 → March 12, 2022		Completed
Final Data Pipeline	Ben	@March 6, 2022 → March 12, 2022		Completed

Name	Assign	Date	Property	Status
Create Patients Section	Joaquin Monterrosa	@March 13, 2022 → March 19, 2022		Completed
Write Flask Endpoint File	Joaquin Monterrosa	@March 13, 2022 → March 19, 2022		Completed
Create Docker Image and Get It Set Up On Google Cloud Run	Joaquin Monterrosa, Nic Dzomba	@March 13, 2022 → March 19, 2022		Completed
Create Add Scan Webook-like Functionality	Joaquin Monterrosa	@March 13, 2022 → March 19, 2022		Completed
Connect The UI and The Model	Joaquin Monterrosa	@March 13, 2022 → March 19, 2022		Completed
KNN for TDA	Ben	@March 13, 2022 → March 19, 2022		Completed
TDA Optimization for Runtime and Accuracy (With limited success)	Ben	@March 13, 2022 → March 19, 2022		Completed
TDA Model for Numpy	Ben	@March 13, 2022 → March 19, 2022		Completed
Adjust Numpy data formatting to fit CNNs	Ben	@March 20, 2022 → March 26, 2022		Completed
Implement First Iteration of CNNs	Ben	@March 20, 2022 → March 26, 2022		Completed
Add The User Authentication Logic	Joaquin Monterrosa	@March 20, 2022 → March 26, 2022		Completed
Create Profile UI and Functionality	Joaquin Monterrosa	@March 20, 2022 → March 26, 2022		Completed
Create Add Patient Functionality	Joaquin Monterrosa	@March 20, 2022 → March 26, 2022		Completed
Add Search and Filter Functionality To Patients Section	Joaquin Monterrosa	@March 20, 2022 → March 26, 2022		Completed

Name	Assign	Date	Property	Status
Add Search and Filter Functionality To Results Section	Joaquin Monterrosa	@March 20, 2022 → March 26, 2022		Completed
Get 3d CNN running	Ben	@March 27, 2022 → April 2, 2022		Completed
Add abstraction to Hypertune CNNs	Ben	@March 27, 2022 → April 2, 2022		Completed
Configurations for Local Training	Ben	@March 27, 2022 → April 2, 2022		Completed
Add Auxiliary Functionality	Joaquin Monterrosa	@March 27, 2022 → April 2, 2022		Completed
Create Admin Dashboard	Joaquin Monterrosa	@March 27, 2022 → April 2, 2022		Completed
Create Add Patient Functionality	Joaquin Monterrosa	@March 27, 2022 → April 2, 2022		Completed
Get 2D CNN Model Working On Cloud Run	Joaquin Monterrosa	@March 27, 2022 → April 2, 2022		Completed
Integrating CNN with Application	Ben Joaquin Monterrosa	@April 3, 2022 → April 9, 2022		Completed
Saving Better Models and wiring up to cloud run environment	Ben	@April 3, 2022 → April 9, 2022		Completed
Configuring Training on Google Cloud (For higher cardinality data)	Ben	@April 3, 2022 → April 9, 2022		Completed
Add Design Touch	Joaquin Monterrosa	@April 3, 2022 → April 9, 2022		Completed
Brand The App	Joaquin Monterrosa	@April 3, 2022 → April 9, 2022		Completed
Add 3D Lung Animation To Marketing Page	Joaquin Monterrosa	@April 3, 2022 → April 9, 2022		Completed
Create Marketing Page	Joaquin Monterrosa	@April 3, 2022 → April 9, 2022		Completed

Name	Assign	Date	Property	Status
Finalizing Cross Validation and Statistical Sigificance	Ⓑ Ben	@April 10, 2022 → April 16, 2022		Completed
Add Auto-Avatar Functionality	 Joaquin Monterrosa	@April 10, 2022 → April 16, 2022		Completed
Hook Up Domain and Deploy	 Joaquin Monterrosa	@April 10, 2022 → April 16, 2022		Completed
Fix The Outstanding Bugs	 Joaquin Monterrosa	@April 10, 2022 → April 16, 2022		Completed
Test The App	 Joaquin Monterrosa	@April 10, 2022 → April 16, 2022		Completed
Squash All Bugs That Are Revealed In Testing	 Joaquin Monterrosa	@April 10, 2022 → April 16, 2022		Completed
Update Portfolio	 Joaquin Monterrosa	@April 17, 2022 → April 23, 2022		Completed
Make Poster	 Joaquin Monterrosa	@April 17, 2022 → April 23, 2022		Completed

For a more detailed look at our timeline you can see a Gantt chart, calendar view, and scrum board view on our Notion page here:

https://eight-net-3e0.notion.site/41613df3b71146c8a148421934ef1250?v=400567023aac4b41b39cae2766e150df

We've also included our old work schedule projection in Appendix F for comparison.

Our work schedule shows each of our tasks, who was assigned to each milestone, and when we worked on each milestone. In general, Ben and Nic worked on the AI models while Joaquin worked on the application. A more detailed view of who worked on what can be found in Appendix I.

While this chart details our work schedule from a high-level, we adopted an agile lifecycle approach to guide our day-to-day workflow. Specifically, we used a Scrum framework. This approach best fit our project because Scrum allowed for a flexible and iterative workflow. This iteration was necessary for the development of both our AI model and the encompassing web-app. Three different methods were used to create an accurate AI model and we had to iterate our application based on which method performed best. Additionally, developing the web-app required the ability to iterate based on input from our customer segment: radiologists and technicians.

Joaquin assumed the role of Scrum Master, ensuring that we abided by the Scrum framework. Our sprints were two-weeks in length and we held weekly stand-ups every Sunday where we reviewed the progress of the stories currently under development. At the beginning of each two-week sprint, we took an epic or two from the backlog and broke it down into smaller stories that could be accomplished in the next two-week sprint. We moved the stories we estimated we could complete in two weeks into the "in-progress" stream. Initially, our backlog consisted of the milestone's outlined in our work schedule. Although these milestones don't fit the formal definition of an epic, they served the purpose of an epic well. We used Notion's board view (which is similar to a Trello board) as our scrum board software. We did not use story points to estimate task lengths due to the limited time frame of this project.

Lastly, it's important to note that before we wrote a single line of code we performed a customer discovery sprint. From the outset, we realized that we had the computer science experience to develop Topo Health but we had no experience as radiologists or oncologists (the primary users of our app). For this reason, we vowed to understand our end-users by conducting a customer discovery sprint before building the product. Information about this customer discovery sprint and the outcomes of this process can be found in Appendix D.

Requirements

Functional Requirements

The main function of the Topo Health app is to process CT Scans with our AI model in order to diagnose lung cancer. To make this happen, our app has successfully achieved the following functional requirements outlined at the beginning of this project. Our app is able to:

- Upload CT Scans
- Preprocess .dcm files into a format compatible with the AI Model
- Analyze the scan with an accurate AI model
- Output diagnostic results
- Interpret diagnostic results
- Display interpreted results

Another crucial functionality for Topo Health is searching through diagnosis history. A technician should be able to search for past lung cancer diagnoses. This functionality was originally intended to be accomplished through fuzzy searching but after talking to our customer segment, we realized that standard search functionality offering multiple search and filter options was more useful than fuzzy searching.

As with most apps, user authentication and profile creation is a rudimentary functionality. For Topo Health, our app provides this onboarding functionality for clinics, technicians, and app admins. App admins manage the provisioning of clinics, clinics manage the provisioning of technicians, and technicians are the primary users of our automated diagnostic functionality. For this onboarding functionality the app:

- Allows clinics, technicians, and app admin to sign up for the first time
- Allows clinics, technicians, and app admin to create their profiles (to a limited degree)
- Authenticates clinics, technicians, and app admin
- Allows app admin to provision clinic accounts
- Allow clinics to provision technician accounts

Before implementation, we intended on allowing app admin to add info to clinic profiles and allow clinics to add info to technician profiles but after better understanding our user flow, we realized that this functionality wasn't necessary.

Lastly, we included standard auxiliary functionality that comes with most web-apps. This auxiliary functionality includes:

- Universal access to terms of service and privacy policy

    - Access to frequently asked questions (FAQs)
    - The ability to easily report bugs and other errors

Another feature we considered implementing assuming we had the time was “quick report generation”. Quick report generation would allow radiologists or technicians to quickly generate diagnostic report templates based on our AI model’s diagnosis. During our customer discovery sprint, we learned that radiologists already have templates like these in most charting software so we decided to eliminate the idea.

For more context, refer to our use-case diagram in the “Architectural Design Documents” section.

Nonfunctional Requirements

During our planning phase, we broke up the non-functional requirements of the app into seven sections and detailed the nonfunctional requirements for each section:

- *Security/Regulatory*
    - *Topo Health will need to be unofficially compliant with HIPAA standards for privacy & security of medical information, in particular regarding Private Health Information (PHI). Additionally, ADA standards for accessibility will need to be met as Topo Health may be working with public entities. For general security, there will be protected passwords that are salted and hashed.*
- *Capacity/Storage*
    - *CT Scans are typically stored in their original state as .DCM files, which average around 200 MB per file. Our model will output PNG files, which are on average 0.4 MB. As a result, the storage of the PNG files and other information that will be stored in text form is negligible.*
    - *If we consider all 260,000 scans conducted yearly in the United States, it would require 52 TB of storage. If we just estimate the number of scans conducted in Montana each year by taking the number of scans in the United States in proportion to Montana’s population, it will take roughly 152 GB to store a year’s worth of scans.*
    - *For this project, we will be targeting 152 GB of storage, with scalability in mind.*
- *Compatibility*
    - *Topo Health will be run as a desktop web application, meaning our website has to be compatible with most desktop browsers. We will be targeting compatibility with all desktop browsers with over 1000 px screen width. Additionally, Topo Health will have to be compatible with all the most common web browsers, so we will be targeting compatibility with Chrome, Firefox, and Edge.*
- *Reliability/Availability*

    - *We intend for Topo Health to be available 24/7/365, as there is no reason for there to be significant downtime for a web application.*
    - *To ensure reliability, we will have tech support available, as well as feedback and bug reporting.*
- *Maintainability*
    - *We are targeting less than 5 hours of maintenance a week.*
- *Scalability*
    - *We intend for Topo Health to be highly scalable in two different ways. Firstly, it needs to scale to different clinics & technicians. Secondly, we would like Topo Health to be able to scale to implement new diagnostic tools in the future, alongside our initial lung cancer diagnosis model.*
- *Usability*
    - *70% of users should be able to intuitively use the applications their first time without prior instruction.*
    - *Usability standards & principles should be satisfied.*

To truly understand the usability of our app we conducted a usability study. The outcome of this study showed that 90% of first-time users were able to use our app without any guidance. More detail on this study can be found in Appendix C.

In the end, we were able to successfully achieve all of the requirements except for two. We were not able to meet all ADA accessibility standards due to a lack of time and resources. We underestimated the amount of effort that is required to meet all ADA accessibility standards. We also weren't able to guarantee 152 gigabytes of storage due to the cost of such a plan. That being said, we did build the app in a scalable manner so that if we could afford a 152 GB plan we could scale the app with a click of a button.

Performance Requirements

Our performance requirements were focused on two areas. First, the application needed to be able to quickly and effectively handle the expected amount of web traffic at any time. With 260,000 lung cancer scans being done in the United States annually, we expected there to be around 712 scans per day on average. We assumed around 350 scans would be analyzed concurrently at peak use. Thus we were targeting optimal performance to still hold at 350 concurrent users.

The second area of concern was the performance of the model. First, we wanted the model to provide rapid feedback, so we were targeting a processing time of fewer than five minutes from file upload to result. Secondly, the model & results must be accurate. We were originally targeting an accuracy greater than 65% but mid-way through implementation, we increased our target to greater than 75%.

Ultimately, we weren't able to verify our concurrency target of 350 users because load-testing the application required too much time and money. That being said, our platforms and architecture should theoretically be able to handle that number of concurrent users. As for the performance of the model, we were successfully able to achieve a processing time of 90 seconds on average and we were able to implement an AI model with 82% general accuracy.

Interface Requirements

The Topo Health app consists of five interfaces:

- The **user** of the web application.
- The **client** which handles all frontend processing for the web application.
- The **server** which provides all backend processing for the web application.
- The **model** which is the AI model that generates the diagnostics based on the scans. The model is stored within the server as another backend function.
- The relational **database** where data is stored.
    - There is a **normal database** where non-sensitive information is stored
    - There is a **secure database** where personal identifiable information (PII), and protected health information (PHI) is stored.

These interfaces communicate with each other in the following ways:

- The **user** interacts with the **client** via common web event handling.
- The **client** uploads scan data to the **server** via HTML POST requests.
- The **client** retrieves interpreted results from the **server** via HTML GET requests.
- The **server** preprocesses .dmc CT scan files into a digestible format and inputs that data into **model**.
- The **model** outputs raw results to an "interpretation function" on the **server**.
- The **server** uploads data to the **database** via database queries.
- The **database** passes data to the **server** via database queries.
- The **client** uploads data to the **database** via database queries.
- The **database** passes data to the **client** via requests and database queries.

For a more detailed description of how these components interface with each other, refer to the component diagram in the "Architectural Design Documents" section.

To learn about other key features outside of our specified requirements, reference Appendix H

Architectural Design Documents

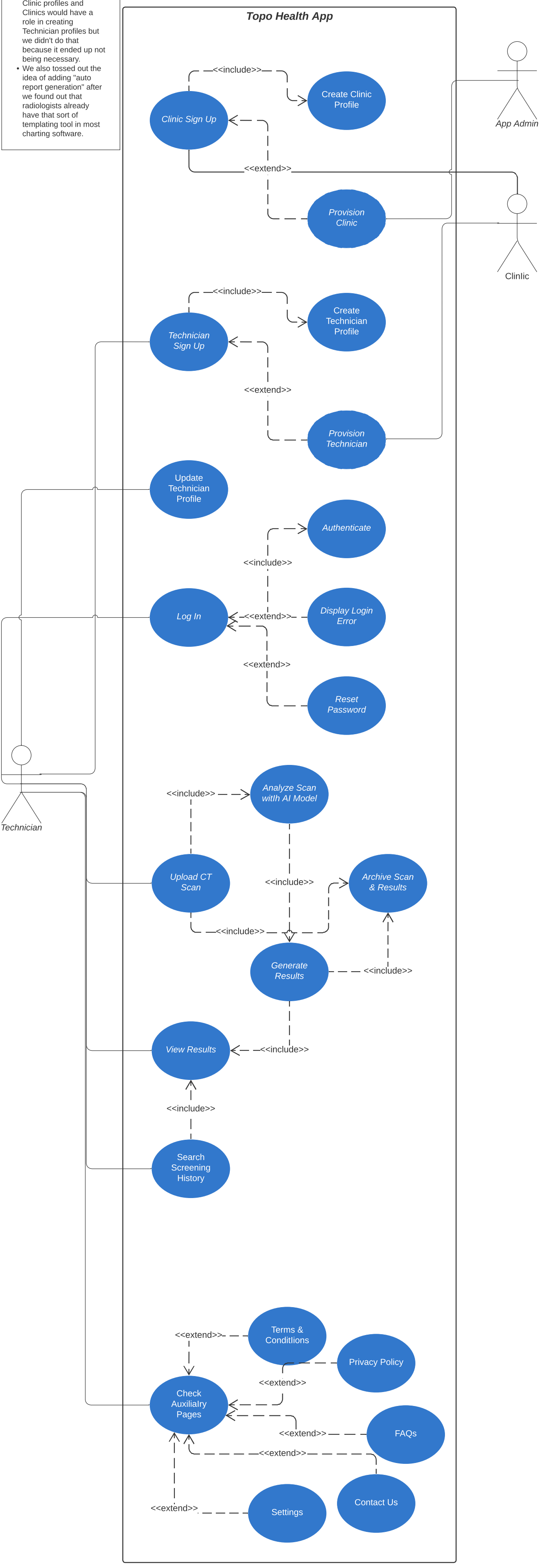
To better understand our proposed architecture, we've created the following behavioral and structural UML diagrams:

Topo Health Use-Case Diagram

Topo Health | April 28, 2022

Changes & Justifications

- Originally, we thought that App admin would have a role in making Clinic profiles and Clinics would have a role in creating Technician profiles but we didn't do that because it ended up not being necessary.
- We also tossed out the idea of adding "auto report generation" after we found out that radiologists already have that sort of templating tool in most charting software.

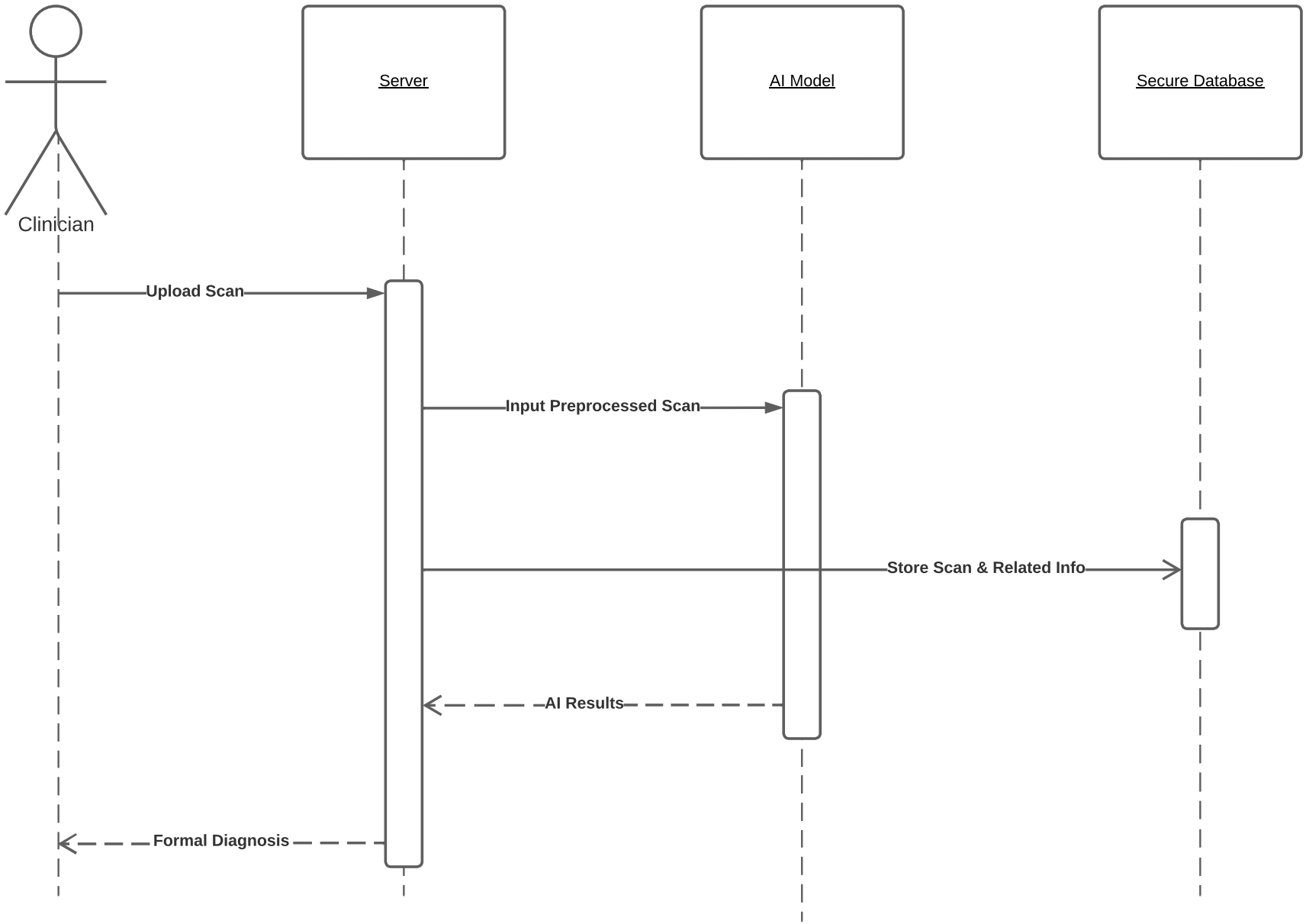


Clinician Uploading Scan

Topo Health | April 28, 2022

Key

Note: Secure Database is the encryped, HIPAA CompliaInt database The database service we're using also handles encrypted file storage



Topo Health Component Diagram

Topo Health | April 28, 2022

Changes & Justificlation
- We didn't make a dedicated "Review Screen" since it made sense to display all result data in the search scans screen.
- We realized we didn't want to use the normal database to send out "patient tokens". Instead it was easier and more secure to request patient data directly from the secure database.
- Added a profile screen to the UI
- We changed the AI model in this diagram from the TDA model to the 2D CNN model since it outperformed the TDA model. Also, the previous diagram had a "Discretionary Model" which we decided not to impliment

<<Database>>
Normal Database

<<Component>>
User Authenticlation

User Login Authentication

User Creation Verification

Normal Database Data

<<UI>>
User Interface

<<UI>>
Login

<<UI>>
Signup

<<UI>>
Search Patients Screen

<<UI>>
Profile Screen

<<UI>>
Search Scans Screen

<<UI>>
Upload Screen

The PHI & PII Data is used throughout most of the UI and all its subcomponents. It includes patient data, scan data, result data, etc.

<<Server>>
Server

.dmc Scan

<<Function>>
.dcm Scan Prepossesing

Preprocessed Scan

<<Function>>
Generate Boolean Results

Scan Results

Model Diagnosis

<<Database>>
Secure Database

New PHI & PII Data

PHI & PII Data

We implemented and tested three different ML models (2D CNN, 3D CNN, and TDA) and the 2D CNN performed best so we used that for the production model

<<Function>>
AI Model (2D CNN)

<<Function>>
Tensorflow Keras Neural Network

Topo Health - ER Diagram

Joaquin Monterrosa | April 28, 2022

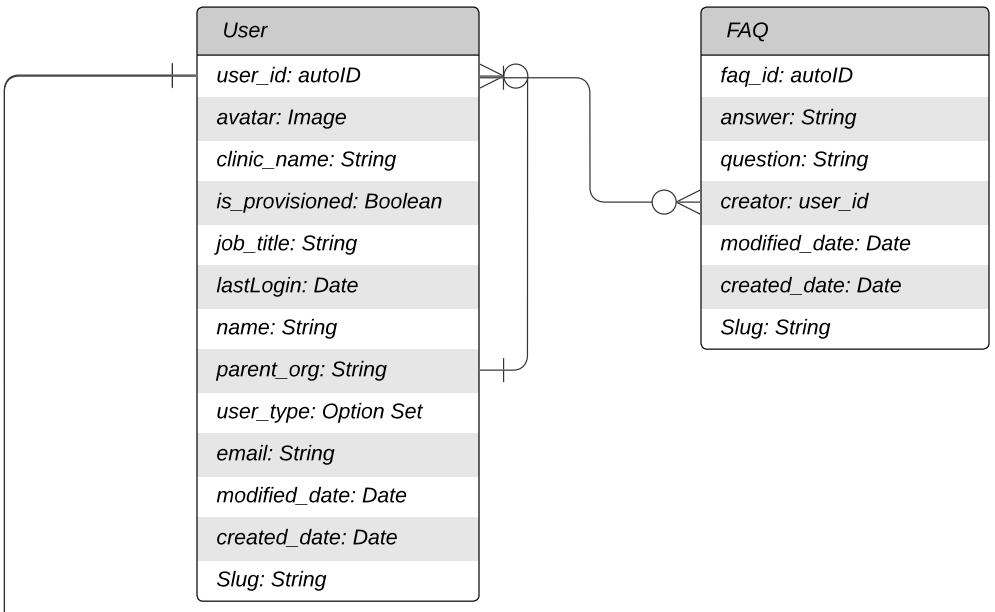
Notes

The User entity uses single-table inheritence to represent the parent-child relationship between a User and Clinics, Clinitians, and App Admins

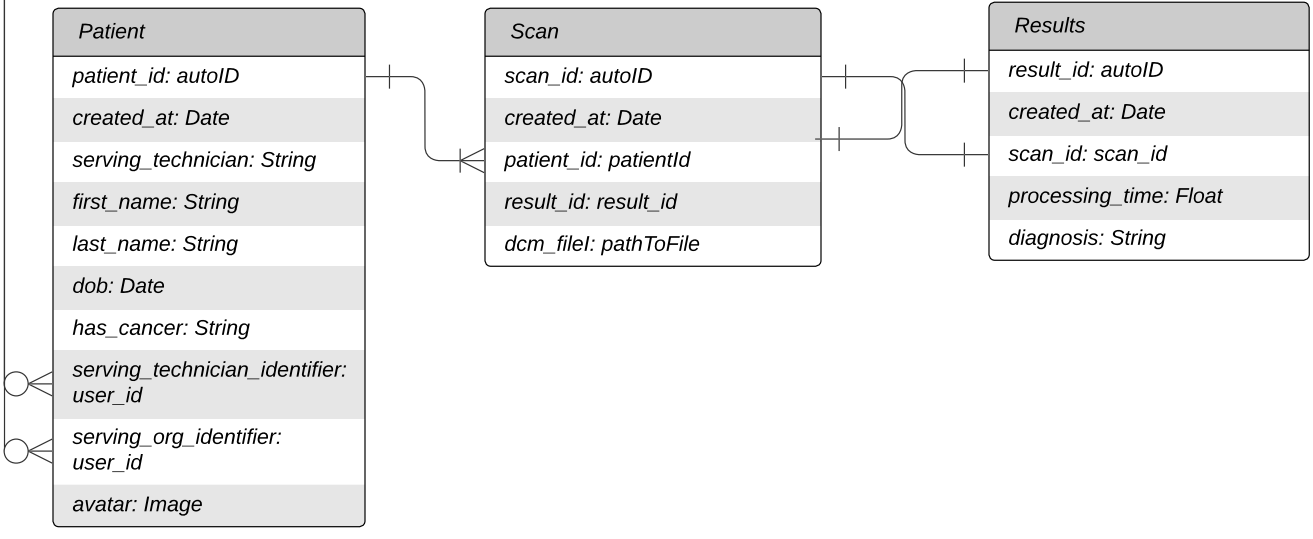
PII stands for Personal Identifiable Information and PHI stands for Protected Health Information

Changes
None of the tables changed but the fields within the tables did change as we built out the app. The other important thing to note is that we are using temporary tables to track the status of .dcm files being processed but due to their effemiral nature we didn't include them in this ER diagram

Normal Database (No PII or PHI Data)



Secure Database (HIPAA Compliant)



Algorithm 1 Vietoris-Rips Filtration

Require: $X = \{x_1, x_2, ...x_k\} \subset \mathbb{R}^n$, a point cloud embedded in $\mathbb{R}^n$

Ensure: $P = \{(b_1, d_1), (b_2, d_2), ..., (b_k, d_k)\}$, a list of birth death pairs corresponding to L under the Vietoris-Rips Filtration, aka the persistence barcode

$B \leftarrow \{\mathbb{B}_1(x_1, 0), \mathbb{B}_2(x_2, 0), ..., \mathbb{B}_k(x_k, 0)\}$ ▷ List of radius 0 n-balls at $x \in X$

$P \leftarrow \{(\emptyset, \emptyset), (\emptyset, \emptyset), ...(\emptyset, \emptyset)\}$ ▷ Empty list of birth-death pairs

for $r \in \mathbb{R}$ **do**
  $B \leftarrow \{\mathbb{B}_1(x_1, r), \mathbb{B}_2(x_2, r), ..., \mathbb{B}_k(x_k, r)\}$
  for $i \in \{1, 2, ..., k\}$ **do**
    for $j \in \{2, ..., k\}$ **do**
      if $\mathbb{B}_i \cap \mathbb{B}_j \neq \emptyset$ **then**
        $(b_i, d_i) \in P \leftarrow (0, r)$
        $B \leftarrow B \setminus \mathbb{B}_i$
      end if
    end for
  end for
end for
return P

Above is the theoretical algorithm implementing the Vietoris-Rips filtration, perhaps the most foundational persistent homology technique. This will be our method of obtaining topological descriptors of point cloud data, in the form of a discretization of a CT scan. In practice, of course the filtration occurs over a sparse representation of $\mathbb{R}$, and terminates once the final birth-death tuple has been updated. For the sake of brevity, we include the theoretical algorithm to convey the core mathematical principles at hand. It is also worth mentioning that the true implementation of the Vietoris-Rips filtration records all homology groups arising throughout the inflation of $n-$balls, whereas our implementation only records the birth and death of $0-$dimensional connected components. For the purpose of recording the shape of potentially cancerous features in a scan, only recording connected components is sufficient.

Algorithm 2 K-Nearest Neighbor for Persistence

Require: $P = \{P_1, P_2, ...P_n\}$, a list of persistence barcodes, $C = \{C_1, C_2, ..., C_n\}$, a list of Boolean classifications, P_q, a query barcode, and $k \in \mathbb{N}$

Ensure: True or False classification for P_q

```
N ← ∅                                   ▷ Initialize list of nearest neighbor indices
count = 0
for i ∈ {1, 2, ..., k} do
    if d_W(P_m, P_q) = argmin(d_W(P, P_q)) then
        N ← m
        P ← P \ P_m
    end if
end for
for i ∈ N do
    if C_i = True then
        count + +
    end if
end for
if count ≥ n/2 then return True
else
     return False
end if
```

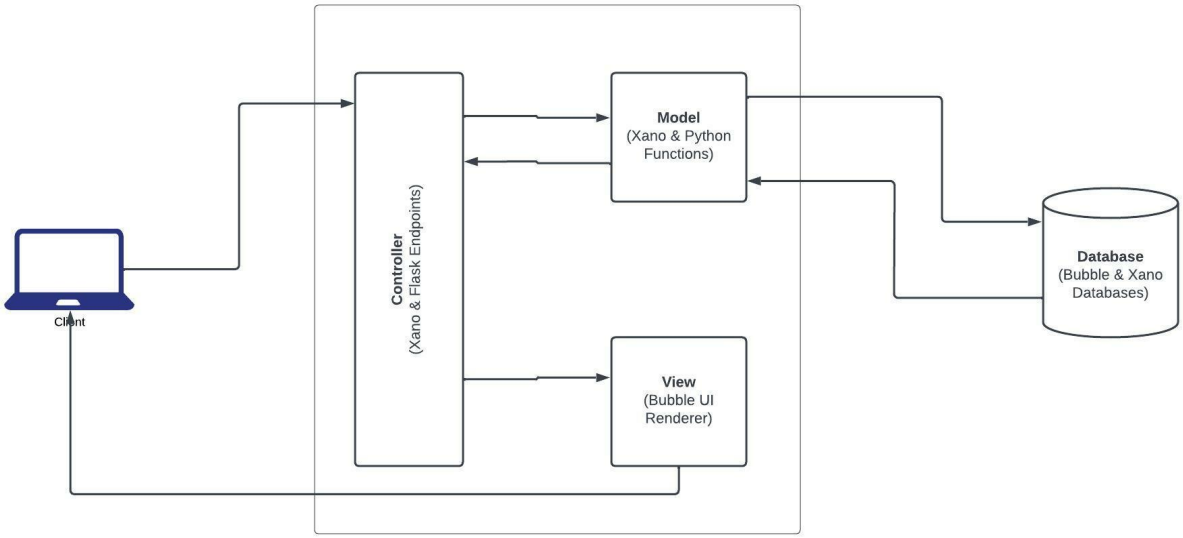
In order to classify the cancer diagnosis of a given unknown CT scan, we feed the resulting persistence barcode through a k-nearest neighbor implementation. In this way, we find the k nearest classifications of a query barcode, and classify this barcode with whichever Boolean value is most prevalent in its neighbors. Distance is defined by the Wasserstein distance, a common metric between persistence barcodes. The Wasserstein distance is defined by comparing the probability distributions constructed by a kernel density estimation among persistence barcodes. This is our chosen distance measure as a holistic categorization of distance among the whole of a barcode, and is particularly well established in the TDA research community.

This architecture was created meticulously and deliberately. For example, in the use-case diagram and the sequence diagram, we decided to sacrifice scalability for security and control when we decided that each technician needs to be manually provisioned by a clinic and each clinic needs to be provisioned by an app admin. The alternative would have been to make each user fully responsible for their own account creation.

Another trade-off we decided to make is depicted in the sequence diagram along with the component diagram. We decided to outsource the preprocessing computation and the delegation of database storage to the server instead of the client. In this case, we traded a few seconds of additional computation time (the time it takes to outsource this functionality over the network) in order to take a load off the front-end and make it asynchronous.

As for the database schema, we decided to represent the parent-child inheritance relationship between the abstract User class and it's concrete children with single-table inheritance. This will mean that the table will have many null fields but it will be faster to query. Because we don't expect memory space in this table to be a problem, this tradeoff made sense.

It's important to note that we also use a Model Controller View (MVC) design pattern to compute and display information on the user interface.

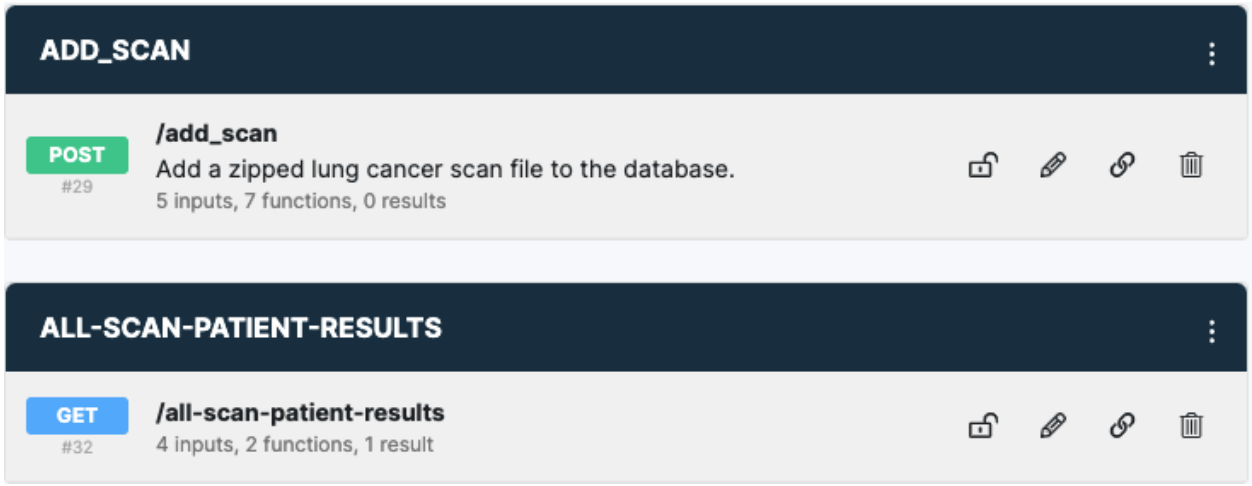


We used a unique tech stack that combines low-code platforms with custom code. Because of this, our MVC design pattern consists of various platforms and scripts instead of traditional classes and objects.

Our controller consists of a suite of endpoints some of which are hosted Xano and some of which are hosted in a Flask script on Google Cloud Run. Here is an example of some of these endpoints:

```
@app.route('/api/v1/diagnose', methods=['POST'])
def diagnose():
   ##scanURL = request.form['scan_url']
   ##fileName = request.form['file_name']
   content = request.get_json(force = True)
   scanURL = content['scan_url']
   fileName = content['file_name']
   scanStorage = "scanFolder"
   pathToScan = "./" + scanStorage + "/" + fileName

   ### download the scan from the given URL ###
   wget.download(scanURL, out=scanStorage)

   ### send the scan through the dianosis ML model ##
   theResult = processScan.full_predict_new(pathToScan, 'scan_out', 'scan_out.npy',
'./Basic-CNN.model')

   ### delete the downloaded scan to remove scan file clutter ###
   os.remove("scan_out.npy")

   for rmvFile in os.listdir('scan_out/' + os.listdir('./scan_out')[0]):
       os.remove(os.path.join('scan_out/' + os.listdir('./scan_out')[0], rmvFile))

   for f in os.listdir(scanStorage):
       os.remove(os.path.join(scanStorage, f))

   return theResult
```

(One of the Flask Endpoint)



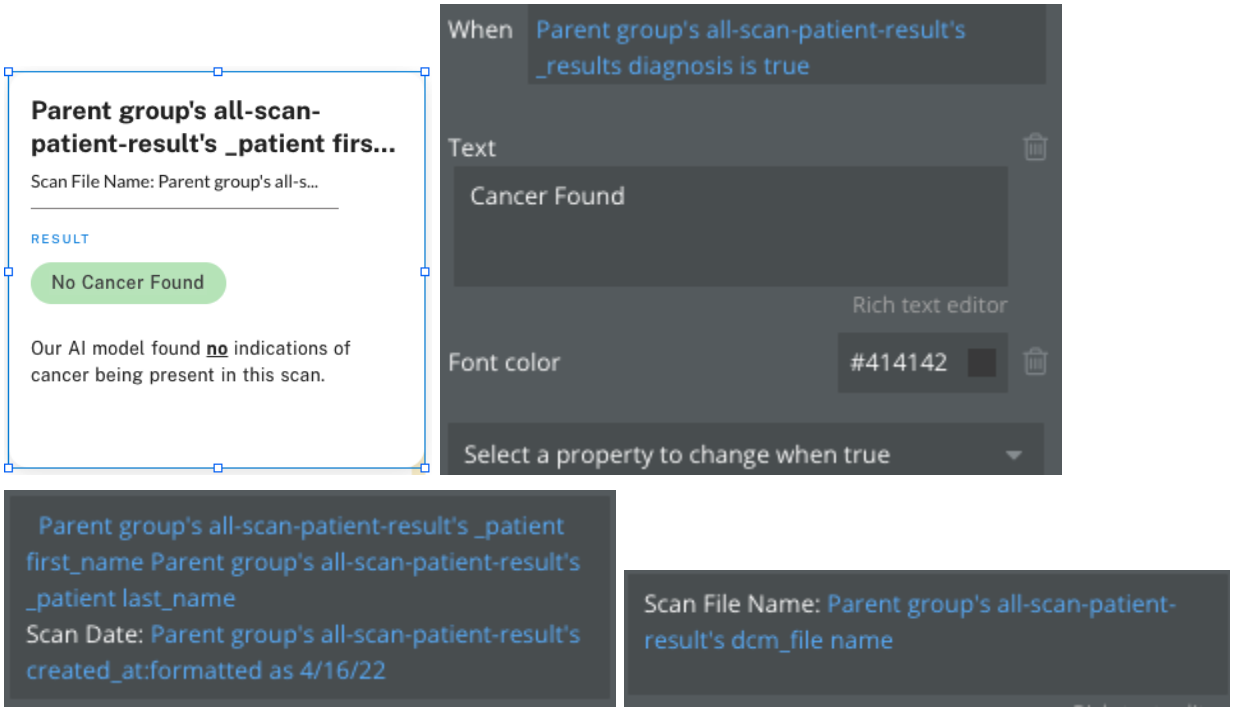
(Two of the Xano Endpoints)

Our model consists of various Xano and Python functions. Here is the code from our Python function that preprocesses and diagnoses the lung cancer scan:

```
import zipfile

import numpy as np

import Preprocess as p
import tensorflow as tf
from tensorflow.keras import layers, models
import os


def unzip(filepath, outpath):
   with zipfile.ZipFile(filepath, 'r') as zip_ref:
       zip_ref.extractall(outpath)


def stage_np(in_np, model_shape):
   new_np = np.resize(in_np, (1, model_shape[1], model_shape[2], model_shape[3]))
   return new_np

# do full prediction given new input
# In: input .zip, output for unzipped .dcms, output for preprocessed np array, model
def full_predict_new(zipfile_in, zipfile_out, npy_save, model_path):
   unzip(zipfile_in, zipfile_out)
   zipfile_OG_dir = os.listdir(zipfile_out)[0]
   p.save_npy(zipfile_out + '/' + zipfile_OG_dir, npy_save)
```

```
in_np = np.load(npy_save)
# To plot fully preprocessed lung, uncomment
#pad_value = 170
#p.plot_3d((~(in_np[0] == pad_value)).astype(int), 0)
model = tf.keras.models.load_model(model_path)
new_np = stage_np(in_np, model.input_shape)
prediction = model.predict([new_np])

outputs = ['false', 'true']
return outputs[int(prediction[0][0])]
```

Our view is implemented with Bubble's UI renderer. Here is an example of the UI component that presents the results of a diagnosed scan:



Nevertheless, our pattern abides by the traditional MVC principles. When a client triggers an action or makes a request, our controller handles that request by passing it to the appropriate function in our model. After the model completes the request, it sends the resulting data back to the controller. Once the controller has the resulting data, it passes that data to the view to render as a UI component on the client's browser.

Alternative design patterns we considered can be found in Appendix J.

Development Standards

We abided by the ACM/IEEE Software Engineering Code of Ethics and Professional Practice [18]. Additionally, we abided by some of the W3C standards for web design and applications [19].

As for our technology stack, here are the technologies, frameworks, and platforms that were used to build the Topo Health App:

- **Front-End:** Bubble.io [20]
- **Server:** Google Cloud Platform
    - All backend scripts were written in Python. For the model, the Dionysus library was used for the Vietoris-Rips filtration, GUDHI was used for the cubical complex filtration, and TensorFlow was used for deep learning.
- **Secure Database and File Storage:** Xano
- **Normal Database:** Bubble.io Database
- **User Authentication:** Bubble.io User Authentication
- **UX Mockups:** Static Bubble.io

During the planning stages of this app, we thought we were going to use Amazon Web Services (AWS) instead of Google Cloud Platform (GCP) but later we went with GCP because it had better support for machine learning.

Results

In the end, we implemented an AI model to detect lung cancer and successfully hosted this model online in an intuitive, secure web application. At the beginning of the project, we had set a benchmark for our best model to achieve 65% accuracy with a processing time less than five minutes per scan. Not only was this benchmark surpassed, but the general accuracy of our winning AI model performed better than the general accuracy of the average radiologist. For more details on how we determined the general accuracy of the average radiologist, see Appendix A. We also had set the goal of predicting the survivability of cancer. Predicting survivability was reliant on a Cox Proportional Hazards model which is only accomplished through the use of TDA. Since the 2D CNN ultimately beat out our TDA model, survivability was no longer possible to implement for our application. For more details on specific model architectures, statistical significance and preprocessing, see Appendix B.

Ultimately, we achieved the following machine learning results:

Model Results:

1. We found that both a 2D and a 3D CNN outperform or are comparable to our benchmark of a conventional radiologist, and that our TDA model falls short of this success.

2. The 2D CNN was our most accurate model, achieving a general accuracy rate of 82% and a sensitivity rate of 70%, performing slightly better on general accuracy than a human radiologist (79.2% accurate) and slightly worse than a human radiologist on sensitivity (75% sensitivity).
3. The 3D CNN achieved comparable results, demonstrating 78% accuracy and a 72% sensitivity rating.
4. Both outperformed our TDA model substantially, which achieved only a 63% accuracy rating in the limited tests that were conducted. However, it is worth noting that significant pruning measures were necessary to make the TDA model computable in a reasonable timeframe, which likely hampered the success of the model greatly.

As for the web-app, we were successfully able to develop a comprehensive product that can be used by technicians, clinics, and app admin. In addition to all the features of the app discussed in Appendices H and E, we were able to make the app intuitive and build with HIPAA compliance in mind.

Results From The Web Application

1. We were able to make the app intuitive enough for 90% of first-time users to use the app without any guidance. For more information on how we conducted this usability study, reference Appendix C.
2. We successfully implemented a "HIPAA-Ready" web app. We're not naive enough to believe our app is fully ready for a HIPAA audit but by storing all of our personal identifiable information (PII) and personal health information (PHI) on a HIPAA compliant database (separate from the normal database), our app is in a good position to achieve HIPAA compliance in the future.

Broadly speaking, we achieved the expected result of building a product to streamline, democratize, and generally improve the process of lung cancer diagnosis from the current radiology workflow. Admittedly, our models fall short of achieving an accuracy rate as high as 95% which is the ceiling reached in the deep learning research community However, we feel proud with our achievement of creating a AI model that performs slightly better than the average radiologist and wrapping that AI in a web-app ready for use by technicians.

Source Code

Preprocessing

Our full preprocessing implementation was adapted/taken from [23]. The code was originally written in 2017, so many small tweaks needed to be made throughout the preprocessing

implementation to make the code up to date. This probably represented about 85% original code and 15% ours, so for brevity most of the preprocessing was left out. However, I will include the full batch preprocessing implementation, which we wrote ourselves. This preprocessing implementation was our third attempt at gaining a robust input to the model, and was undoubtedly crucial for our success.

First, we begin with quite a lot of imports needed to preprocess the data. (I'll just include them because they're referenced throughout, and are necessary to read through the source code.)

```
import numpy as np
import pandas as pd
import pydicom
import os
import scipy.ndimage
import matplotlib.pyplot as plt

from skimage import measure, morphology
from PIL import Image
from mpl_toolkits.mplot3d.art3d import Poly3DCollection
from scipy.ndimage.morphology import
binary_dilation,generate_binary_structure
from skimage.morphology import convex_hull_image
from scipy.ndimage.interpolation import zoom
from scipy.io import loadmat
import warnings
from multiprocessing import Pool, cpu_count
from functools import partial
import time
import matplotlib.pyplot as plt
import csv
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
from sklearn.model_selection import KFold, cross_val_score
```

This small function is useful to plot the 3D dicom, just to see what it looked like before preprocessing.

```
def view_unprocessed(path, thresh):
    patient_img, patient_spacing = get_pixels_hu(load_scan(path))
    plot_3d(patient_img, threshold=thresh)
```

Then, here is the function that preps our full input directory of dicoms, saving the output as numpy arrays for us to load into a model.

```
def full_prep(path):
    # list of already prepared dicoms
    prepped = [f[:-10] for f in os.listdir('../input/staging_processed')]
    times = []
    num_unprepped = len(os.listdir(path)) - len(prepped)
    for f in os.listdir(path):
        if f not in prepped:
            start = time.time()
            print("Preprocessing", f)
            cur_path = path + f + '/'
            save_npy(cur_path, f)
            end = time.time()
            times.append(end-start)
            print(f, "Done in ", end - start, "Sec")
            ave_time = sum(times)/len(times)
            num_unprepped -= 1
            print("ETA: ", ave_time * num_unprepped, "Sec")
```

This script allows us to read in a CSV with class values (‘0’ or ‘1’) and output an NP array in the expected format for Tensorflow.

```
def init_classes(path, input_length):
    rows = []
    with open(path, newline='') as f:
        reader = csv.reader(f)
        count = 0
        for row in reader:
            if count < input_length:
                if row[0] + '_clean.npy' in
os.listdir('../input/processed/'):
                    print(row[0])
                    rows.append([row[0], int(row[1])])
                    count += 1
        return np.asarray(rows)
```

Also, for Tensorflow to work, we needed to have every input image be the same shape. This script finds the maximal shape in our set of images (saved as numpy arrays) in order to change the shape of every image to make the shape equivalent. This is done by padding every image

with zero vectors, if in any dimension the shape was originally smaller.

```
def find_max_shape(path, classes):
    max_i, max_j, max_k = 0, 0, 0
    for i in range(len(classes)):
        cur_np = np.load(path + classes[i][0] + '_clean.npy')
        if cur_np.shape[1] > max_i:
            max_i = cur_np.shape[1]
        if cur_np.shape[2] > max_j:
            max_j = cur_np.shape[2]
        if cur_np.shape[3] > max_k:
            max_k = cur_np.shape[3]
    return max_i, max_j, max_k
```

Then we had to actually resize all of our data. This is the implementation to do so, and to split the data at an 80/20 ratio for training and testing sets. This hardcoded version was useful (and often actually completely necessary) when we would run into kernel restart issues because of the volume of our data. For some reason the hard coded splits would often work, whereas the sklearn cross validation splits somehow pushed the system over the edge.

```
def stage_data_hardcoded(input_length):
    classes = init_classes('../input/stage1_labels.csv', input_length)
    train_classes, test_classes = classes[:int(0.8*input_length), :],
classes[int(0.8*input_length):, :]
    train_data_arr, test_data_arr, train_class_arr, test_class_arr = [],
[], [], []
    max_shape = find_max_shape('../input/processed/', classes)
    for i in range(input_length):
        if i < len(train_classes):
            cur_np = np.load('../input/processed/' + classes[i][0] +
'_clean.npy')
            # pad with zeros to fit max_shape
            cur_np = np.resize(cur_np, (1, max_shape[0], max_shape[1],
max_shape[2]))
            train_data_arr.append(cur_np)
            train_class_arr.append([int(train_classes[i][1])])
        # classification in  test_classes, load test set
        else:
            cur_np = np.load('../input/processed/' + classes[i][0] +
'_clean.npy')
            cur_np = np.resize(cur_np, (1, max_shape[0], max_shape[1],
max_shape[2]))
```

```
            cur_arr = cur_np.tolist()
            test_data_arr.append(cur_arr)
            test_class_arr.append([int(test_classes[i -
len(train_classes)][1])])


    print("MAX SHAPE:", max_shape)

    train_data = np.concatenate(train_data_arr)
    print(train_data.shape)
    train_class = np.array(train_class_arr).reshape(-1, 1)
    print(train_class.shape)
    test_data = np.concatenate(test_data_arr)
    print(test_data.shape)
    test_class = np.array(test_class_arr).reshape(-1, 1)
    print(test_class.shape)

    return train_data, train_class, test_data, test_class, max_shape
```

Here is the data staging without hard coded splits. This is built to later use test set/training set splitting from sklearn.

```
# stage data/classes into one large np array for cross validation
def stage_data(input_length):
    classes = init_classes('../input/stage1_labels.csv', input_length)
    data_arr = []
    class_arr = []
    max_shape = find_max_shape('../input/processed/', classes)
    for i in range(input_length):
        cur_np = np.load('../input/processed/' + classes[i][0] +
'_clean.npy')
        # pad with zeros to fit max_shape
        cur_np = np.resize(cur_np, (1, max_shape[0], max_shape[1],
max_shape[2]))
        data_arr.append(cur_np)
        class_arr.append([int(classes[i][1])])


    print("MAX SHAPE:", max_shape)

    out_data = np.concatenate(data_arr)
    print(out_data.shape)
```

```
    out_classes = np.array(class_arr).reshape(-1, 1)
    print(out_classes.shape)

    return out_data, out_classes, max_shape
```

And finally, here is the 2D CNN implementation, which calls in a parameter for virtually every aspect of the model architecture for hypertuning.

```
def init_2d_model(input_length, num_layers, filters, window, activ_fcn,
pool_size, cross_validation=False, k_fold_splits=0):


    # for hardcoded data staging (80-20 ratio)
    if not cross_validation:
        train_data, train_class, test_data, test_class, max_shape =
stage_data_hardcoded(input_length)

        model = models.Sequential()

        for i in range(num_layers):
            model.add(layers.Conv2D(filters, window, activation=activ_fcn,
input_shape=max_shape))
            model.add(layers.MaxPooling2D(pool_size))
        model.add(layers.Conv2D(2 * filters, window, activation=activ_fcn))

        model.add(layers.Flatten())
        model.add(layers.Dense(64, activation='relu'))
        model.add(layers.Dense(10))

        # Compile the model
        print("Compiling Model")
        model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                      metrics=['accuracy'])
        model.summary()

        with tf.device("/cpu:0"):
            print("Evaluate Model Fitness")
```

```
            history = model.fit(train_data, train_class, epochs=10,
                                validation_data=(test_data, test_class))
            # Uncomment to save model
            #model.save('./output/Test-CNN.model')

        plt.plot(history.history['accuracy'], label='accuracy')
        plt.plot(history.history['val_accuracy'], label='val_accuracy')
        plt.xlabel('Epoch')
        plt.ylabel('Accuracy')
        plt.ylim([0.5, 1])
        plt.legend(loc='lower right')
        test_loss, test_acc = model.evaluate(test_data, test_class,
verbose=2)


    # for cross validation
    elif cross_validation:
        all_data, all_class, max_shape = stage_data(input_length)

        model = models.Sequential()

        for i in range(num_layers):
            model.add(layers.Conv2D(filters, window, activation=activ_fcn,
input_shape=max_shape))
            model.add(layers.MaxPooling2D(pool_size))
        model.add(layers.Conv2D(2 * filters, window, activation=activ_fcn))

        model.add(layers.Flatten())
        model.add(layers.Dense(64, activation='relu'))
        model.add(layers.Dense(10))

        # Compile the model
        print("Compiling Model")
        model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                      metrics=['accuracy'])
        model.summary()

        k_fold = KFold(k_fold_splits)
        test_acc = 0
```

```
        for train_idcs, test_idcs in k_fold.split(all_data):
            train_data_arr = [all_data[i].tolist() for i in train_idcs]
            train_class_arr = [all_class[i] for i in train_idcs]

            test_data_arr = [all_data[j].tolist() for j in test_idcs]
            test_class_arr = [all_class[j] for j in test_idcs]

            train_data, train_class = np.array(train_data_arr),
np.array(train_class_arr).reshape(-1, 1)
            test_data, test_class = np.array(test_data_arr),
np.array(test_class_arr).reshape(-1, 1)

            print(train_data.shape)
            print(train_class.shape)

            history = model.fit(train_data, train_class, epochs=10,
validation_data=(test_data, test_class))

            # uncomment for plotting
            #plt.plot(history.history['accuracy'], label='accuracy')
            #plt.plot(history.history['val_accuracy'],
label='val_accuracy')
            #plt.xlabel('Epoch')
            #plt.ylabel('Accuracy')
            #plt.ylim([0.5, 1])
            #plt.legend(loc='lower right')


            loss, acc = model.evaluate(test_data, test_class, verbose=2)

            if acc > test_acc:
                test_acc = acc

    return test_acc
```

In order to use a 3D CNN, which expects data of one more dimension, we needed to reshape our data to be compatible with 3D convolutions. We just did this by adding a dimension corresponding to the 0/1 boolean value saying whether or not that pixel was in the binary mask associated with lung tissue.

```
def stage_3d_data(input_length):
    train_data, train_class, test_data, test_class, max_shape =
stage_data(input_length)
    new_train_data, new_test_data = [], []
    for lung in train_data:
        train_lung = (~(lung[0] == 170)).astype(int)
        train_combined = np.concatenate(lung.astype(int), train_lung)
        new_train_data.append(train_combined)
    for lung in test_data:
        test_lung = (~(lung[0] == 170)).astype(int)
        test_combined = np.concatenate(lung.astype(int), test_lung)
        new_test_data.append(test_combined)

    out_train_data = np.concatenate(new_train_data)
    out_test_data = np.concatenate(new_test_data)


    return out_train_data, train_class, out_test_data, test_class
```

Then we implemented the 3D CNN using Tensorflow. Currently, this is a hardcoded model, and we never abstracted away parameters in the 3D CNN in the way that was done with the 2D case. This choice was due to runtime and computability issues, we were limited to a low number of filters for our model to compile and train correctly (due to the size of our data) even when using environments in Google cloud.

```
def init_3d_model(input_length):
    train_data, train_class, test_data, test_class, max_shape =
stage_data(input_length)

    model = models.Sequential()
    model.add(layers.Conv3D(2, (3, 3, 3), activation='relu',
input_shape=(max_shape[0], max_shape[1], max_shape[2], 1)))
    model.add(layers.MaxPooling3D((2, 2, 2)))
    model.add(layers.Conv3D(2, (3, 3, 3), activation='relu'))
    model.add(layers.MaxPooling3D((2, 2, 2)))
    model.add(layers.Conv3D(4, (3, 3, 3), activation='relu'))

    model.add(layers.Flatten())
    model.add(layers.Dense(4, activation='relu'))
    model.add(layers.Dense(10))
```

```
    # Compile the model
    print("Compiling Model")
    model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                  metrics=['accuracy'])
    model.summary()
    with tf.device("/cpu:0"):
        # Uncomment to save model
        # model.save('./output/Basic-CNN.model')
        print("Evaluate Model Fitness")

        history = model.fit(train_data[:, :, :, :, np.newaxis],
train_class, epochs=10,
                            validation_data=(test_data[:, :, :, :,
np.newaxis], test_class))

        plt.plot(history.history['accuracy'], label='accuracy')
        plt.plot(history.history['val_accuracy'], label='val_accuracy')
        plt.xlabel('Epoch')
        plt.ylabel('Accuracy')
        plt.ylim([0.5, 1])
        plt.legend(loc='lower right')

        test_loss, test_acc = model.evaluate(test_data, test_class,
verbose=2)
        return test_acc
```

Below is our hypertuning script that was used to optimize the performance of our 2D CNN.

```
import CNN

# optimize performance of 2d CNN
# IN: CNN dimension (2 or 3), max number of layers, list of filter numbers,
# list of windows, list of activation fcns, list of pool sizes
# OUT: CNN optimal parameters and resulting accuracy
def tune_CNN(max_depth, num_filters, windows, activ_fcns, pools,
CNN_dim=2):
    accuracy = 0
    params = []
    for depth in range(max_depth):
```

```
        for f in num_filters:
            for window in windows:
                for fcn in activ_fcns:
                    for pool in pools:
                        if CNN_dim == 2:
                            cur_acc = CNN.init_2d_model(50, depth, f,
window, fcn, pool)
                        else:
                            cur_acc = CNN.init_3d_model(50, depth, f,
window, fcn, pool)

                        if cur_acc > accuracy:
                            accuracy = cur_acc
                            params = [depth, f, window, fcn, pool]
    return accuracy, params
```

Now, in what follows we will include our full implementation of a TDA-based model, which despite being unsuccessful in the end represented a major endeavor. For starters, here is a list of all of the modules we used in our TDA Python environment.

```
import math

import dionysus as d
import rpy2 as r
from sklearn.neighbors import KernelDensity
import numpy as np
from sklearn.metrics import DistanceMetric
import open3d as o3d
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import similaritymeasures as sim
```

In the beginning of the project, we actually were initially storing all of our processed lung data as STL files, because it was a format that Nic had familiarity with, and it was nice to visualize. We pretty quickly moved over to just using saved .npy files, but here is the initial conversion of STL files to a numpy pointcloud for input into a Vietoris-Rips filtration.

```
def stl_to_input(stl_path):
    mesh = o3d.io.read_triangle_mesh(stl_path)
    pcd = o3d.geometry.PointCloud()
```

```
    pcd.points = mesh.vertices
    arr_points = np.asarray(pcd.points)
    return arr_points
```

For time complexity reasons, we also very quickly implemented methods to thin our input with a discretization parameter.

```
# take as input np array cloud and a natural, keeping every nth entry
def thin_input(cloud, thinning_param):
    del_idcs = []
    x, y, z = [], [], []
    for i in range(len(cloud)):
        if (i % thinning_param) == 0:
            x.append(cloud[i][0])
            y.append(cloud[i][1])
            z.append(cloud[i][2])
    thinned_cloud = np.column_stack([x, y, z])
    return thinned_cloud
```

Here is a nice, short function to visualize the point cloud data we were working with, to view the lung data (and thinned versions of it).

```
def visualize_cloud_3d(cloud):
    arr_points = np.asarray(cloud)
    print(cloud)
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    x, y, z = [], [], []
    for i in range(len(arr_points)):
        x.append(arr_points[i][0])
        y.append(arr_points[i][1])
        z.append(arr_points[i][2])
    ax.scatter(x, y, z, c=z, alpha=1)
    plt.show()
```

This function does the same thing as the above, but with a saved STL.

```
def visualize_stl(stl_path):
    mesh = o3d.io.read_triangle_mesh('./stl/cube.stl')
    pcd = o3d.geometry.PointCloud()
    pcd.points = mesh.vertices
    arr_points = np.asarray(pcd.points)
```

```
    # print(arr_points)
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
    x, y, z = [], [], []
    for i in range(len(arr_points)):
        x.append(arr_points[i][0])
        y.append(arr_points[i][1])
        z.append(arr_points[i][2])
    ax.scatter(x, y, z, c=z, alpha=1)
    plt.show()
```

While generating persistence diagrams, it is also useful to view them, which is done below.

```
def show_pd(dgms):
    d.plot.plot_diagram(dgms[0], show=True)
```

Here is the short function where the magic is happening from a TDA perspective: we generate persistence diagrams here from the Vietoris-Rips filtration, halting the filtration at 3-dimensional simplices (tetrahedra) and halting the expansion of balls after a radius of 2. There are some serious choices here between performance and efficacy! The default is what we found to strike a nice balance.

```
# Run Vietoris-Rips Filtration on np array
def run_rips(cloud, img_idx, max_dim=3, max_rad=2):
    print('Generating filtration for img', img_idx)
    # max dimension = 3, max radius = 2
    f = d.fill_rips(cloud, max_dim, max_rad)
    p = d.homology_persistence(f)
    dgms = d.init_diagrams(p, f)
    # for visualization
    d.plot.plot_diagram(dgms[0], show=True)
    return dgms
```

Also implemented is a height filtration, though this is a less natural filtration for the use case. Nonetheless, we tried it to see if there would be any performance benefits.

```
def run_height(cloud):
    f_lower_star = d.fill_freudenthal(cloud)
    f_upper_star = d.fill_freudenthal(cloud, reverse=True)
    p = d.homology_persistence(f_lower_star)
    dgms = d.init_diagrams(p, f_lower_star)
```

```
    return dgms
```

We also constructed a kernel density based distance between persistence diagrams in the 0th homology dimension. This was for computational complexity purposes when constructing our distance matrix to do k-nearest-neighbors. In theory, our implementation pairing a one dimensional Gaussian kernel computation with a linear Frechet distance should've been linear in time complexity. In practice, the benefits were outweighed by the still relatively high computational costs of computing Gaussian kernels for each point in the persistence diagram, and summing them together. When compared to the built in and ultra-optimized implementations for Wasserstein and Bottleneck distance that we used, the time difference was insignificant. As seems to be the story with TDA in this project, this method was abandoned, though it represented an interesting attempt.

```
def kde_dist(x, y):
    kde1 = KernelDensity(kernel='tophat', bandwidth=0.2).fit(x)
    kde2 = KernelDensity(kernel='tophat', bandwidth=0.2).fit(y)
    density1 = kde1.score_samples(x)
    density2 = kde2.score_samples(y)

    disc1 = []
    disc2 = []
    # init proper dimensional vectors
    for i in range(len(x)):
        disc1.append([x[i][1], density1[i]])

    for j in range(len(y)):
        disc2.append([y[j][1], density2[j]])

    dist = sim.frechet_dist(np.array(disc1), np.array(disc2))

    return dist
```

Additionally, here are the two standard distances used in TDA between persistence diagrams. We in the end opted for the bottleneck distance, which is the induced L-infinity norm among persistence diagrams. The prior code snippet is an attempt to approximate the Wasserstein distance for only connected components in linear time.

```
# Won't work without a bijection.
def wass_dist(diag1, diag2):
    return d.wasserstein_distance(diag1, diag2)
```

```
def bottle_dist(diag1, diag2):
    return d.bottleneck_distance(diag1, diag2)
```

After implementing the required persistent homology techniques, we had to implement a way to write them in a CSV format for reference later. In what follows, we provide an implementation to do so, both for saved numpy arrays as input and for STLs as input in the preliminary version.

```
import csv
import math
import Persist as p
import os
import numpy as np


# batch write pd for already saved npy arrs
def batch_write_pd_diag(disc, max_dim, max_rad, input_path):
    count = 0
    for f in os.listdir(input_path):
        count += 1
        name = input_path + '/' + f
        lung_np = np.load(name)
        lung = (~(lung_np[0] == 170)).astype(int)
        lung_list = []
        for i in range(lung.shape[0]):
            for j in range(lung.shape[1]):
                for k in range(lung.shape[2]):
                    if lung[i][j][k] == 1:
                        lung_list.append([float(i), float(j), float(k)])
        lung_shaped = np.asarray(lung_list)
        thinned_lung = p.thin_input(lung_shaped, disc)
        diag = p.run_rips(thinned_lung, max_dim, max_rad, count)

        gen_pd_arrs(diag, count, f[:-10])


# input discretization param, dimension bound, radius of balls bound
def batch_write_pd(disc, max_dim, max_rad, len_stl):
    for i in range(53, len_stl):
        cur_name = "./input/stl/NSCLC/LUNG1-" + f"{i:03}.stl"
```

```
        # handle deleted entries due to image corruption
        if os.path.isfile(cur_name):
            in_file_name = "./input/stl/NSCLC/LUNG1-" + f"{i:03}.stl"
            lung_pts = p.stl_to_input(in_file_name)
            thinned_lung = p.thin_input(lung_pts, disc)
            diag = p.run_rips(thinned_lung, max_dim, max_rad, i)

            gen_pd_arrs(diag, i)
        else:
            pass


def gen_pd_arrs(diag, i, name='', path_0="./output/out_stage/out_dgms_0/",
path_1="./output/out_diag/out_dgms_1/",
path_2="./output/out_diag/out_dgms_2/"):
    path_0 += name
    path_1 += name
    path_2 += name
    arr_diag_0 = []
    arr_diag_1 = []
    arr_diag_2 = []

    # remove all points dying at infinity
    for point in diag[0]:
        if point.death != math.inf:
            arr_diag_0.append([point.birth, point.death])

    for point in diag[1]:
        if point.death != math.inf:
            arr_diag_1.append([point.birth, point.death])

    for point in diag[2]:
        if point.death != math.inf:
            arr_diag_2.append([point.birth, point.death])

    out_file_name = path_0 + f"{i:03}"
    write_pd_arr(arr_diag_0, out_file_name)

    out_file_name = path_1 + f"{i:03}"
    write_pd_arr(arr_diag_1, out_file_name)

    out_file_name = path_2 + f"{i:03}"
    write_pd_arr(arr_diag_2, out_file_name)
```

```
# persistence diagram array save to csv (hardcoded for rips-filtration)
def write_pd_arr(persist_arr, out_file):

    with open(out_file, 'w') as csvfile:
        # creating a csv writer object
        csvwriter = csv.writer(csvfile)

        # writing the data rows
        csvwriter.writerows(persist_arr)
```

Here, we implement a short script to classify a new scan according to a computed distance matrix.

```
import Persist as p
import CSV_Writer as csvw
import KNN as knn


# give path to CT scan (str), thinning param, max dimension (default=3)
# max filtration radius (default=5), img index (default=1),
def classify_new(stl_path, thin=10, max_dim=3, max_rad=5, idx=1,
class_dim=2):
    diag = run_pipeline(stl_path, thin, max_dim, max_rad, idx)
    knn.k_nearest_bottle(10, diag, class_dim)


def run_pipeline(stl_path, thin, max_dim=3, max_rad=5, idx=1):
    scan_arr = p.stl_to_input(stl_path)
    thinned_arr = p.thin_input(scan_arr, thin)
    diag = p.run_rips(thinned_arr, max_dim, max_rad, idx)
    # store in corresponding csvs
    #csvw.gen_pd_arrs(diag, 1, 'new')
    return diag
```

Finally for ease of computation, we save a distance matrix of bottleneck distances among persistence diagrams, and refer to that to find the k nearest neighbors of a given persistence diagram.

```
import os.path
import numpy as np
import Persist as p
import csv
import persim
import CSV_Writer as csvw
import math


# Print distances, given number of dgms and
# dist = 'kde' or 'bottle' for bottleneck dist or kde + frechet dist
def gen_dist_mtx(dgms_path, dgms_len, dist):
    dist_mtx = []
    dgms = []
    for f in os.listdir(dgms_path):
        cur_name = dgms_path + '/' + f
        if os.path.isfile(cur_name):
            file = open(cur_name, "r")
            csv_reader = csv.reader(file)
            dgm = []
            for row in csv_reader:
                dgm.append(row)
            dgms.append(dgm)
        else:
            pass
    # kde + frechet distance, using our persist module
    if dist == 'kde':
        for i in range(dgms_len):
            print("NOW COMPUTING DISTANCES FOR DGM ", i)
            # for each dgm, compute distance to all other diagrams
            for j in range(dgms_len):
                dist = p.kde_dist(dgms[i], dgms[j])
                print("Dist(", i, j, ")", dist)
    # bottleneck distance, using persim module
    elif dist == 'bottle' or 'wass':
        for i in range(dgms_len):
            print("NOW COMPUTING DISTANCES FOR DGM ", i)
            cur_i = np.asarray(dgms[i])
            x = cur_i.astype(np.float)
            #maintain distances to append to dist matrix
            dists_to_x = []
```

```
            for j in range(dgms_len):
                cur_j = np.asarray(dgms[j])
                y = cur_j.astype(np.float)
                if j != i:
                    print("ON ITER: i = ", i, "j = ", j)
                    if dist == 'bottle':
                        if len(x) != 0 and len(y) != 0:
                            d = persim.bottleneck(x, y)
                            print(d)
                            dists_to_x.append(d)
                        else:
                            dists_to_x.append(math.inf)
                    elif dist == 'wass' and (len(x) != 0 and len(y) != 0):
                        Ms = range(5, 100, 2)
                        ds = [persim.sliced_wasserstein(x, y, M=M) for M in
Ms]
                        print(ds[99])
                        dists_to_x.append(ds[99])
            # append to matrix
            dist_mtx.append(dists_to_x)

    # write matrix to file
    out_fname = "./output/out_diag/" + dist + "_mtx1"
    csvw.write_pd_arr(dist_mtx, out_fname)
```

To conclude the source code section, we include select key methods from the web app. To begin, below is the Python script running in a docker container which unzips a new DICOM image, stages the dicom in the fully preprocessed numpy format, and then conducts the full cancer diagnosis.

```
import zipfile
import numpy as np
import Preprocess as p
import tensorflow as tf
from tensorflow.keras import layers, models
import os


def unzip(filepath, outpath):
    with zipfile.ZipFile(filepath, 'r') as zip_ref:
        zip_ref.extractall(outpath)
```

```
def stage_np(in_np, model_shape):
    new_np = np.resize(in_np, (1, model_shape[1], model_shape[2],
model_shape[3]))
    return new_np

# do full prediction given new input
# In: input .zip, output for unzipped .dcms, output for preprocessed np
array, model
def full_predict_new(zipfile_in, zipfile_out, npy_save, model_path):
    model = tf.keras.models.load_model(model_path)
    unzip(zipfile_in, zipfile_out)
    zipfile_OG_dir = os.listdir(zipfile_out)[0]
    p.save_npy(zipfile_out + '/' + zipfile_OG_dir, npy_save)
    in_np = np.load(npy_save)
    # To plot fully preprocessed lung, uncomment
    #pad_value = 170
    #p.plot_3d((~(in_np[0] == pad_value)).astype(int), 0)
    new_np = stage_np(in_np, model.input_shape)
    prediction = model.predict([new_np])

    outputs = ['false', 'true']
    classidx = np.argmax(prediction, axis=1)
    if classidx > 0:
        return outputs[1]
    else:
        return outputs[0]



def diagnose(scanURL, fileName):
    scanStorage = "scanFolder"
    pathToScan = "./" + scanStorage + "/" + fileName

    ### download the scan from the given URL ###
    wget.download(scanURL, out=scanStorage)

    ### send the scan through the dianosis ML model ##
    theResult = processScan.full_predict_new(pathToScan, 'scan_out',
'scan_out.npy', './Basic-CNN.model')

    ### delete the downloaded scan to remove scan file clutter ###
```

```
    os.remove("scan_out.npy")

    for rmvFile in os.listdir('scan_out/' + os.listdir('./scan_out')[0]):
        os.remove(os.path.join('scan_out/' + os.listdir('./scan_out')[0],
rmvFile))

    for f in os.listdir(scanStorage):
        os.remove(os.path.join(scanStorage, f))

    return theResult
```

This is a POST endpoint that receives zipped dicom scan files ready for analysis. We download the scan given a URI, and run the scan through the model to receive a “true” or “false” diagnosis. Finally, to maintain the state in the application before the method is called, we remove the downloaded scan from the file structure.

```
@app.route('/api/v1/diagnose', methods=['POST'])
def diagnose():
    ##scanURL = request.form['scan_url']
    ##fileName = request.form['file_name']
    content = request.get_json(force = True)
    scanURL = content['scan_url']
    fileName = content['file_name']
    scanStorage = "scanFolder"
    pathToScan = "./" + scanStorage + "/" + fileName

    ### download the scan from the given URL ###
    wget.download(scanURL, out=scanStorage)

    ### send the scan through the dianosis ML model ##
    theResult = processScan.full_predict_new(pathToScan, 'scan_out',
'scan_out.npy', './2d-CNN.model')

    ### delete the downloaded scan to remove scan file clutter ###
    os.remove("scan_out.npy")

    for rmvFile in os.listdir('scan_out/' + os.listdir('./scan_out')[0]):
        os.remove(os.path.join('scan_out/' + os.listdir('./scan_out')[0],
rmvFile))
```

```
    for f in os.listdir(scanStorage):
        os.remove(os.path.join(scanStorage, f))

    return theResult
```

Check Out The Finished Product

To check out the finished product, reference the Appendix E to view the product captures or follow the instructions in Appendix G to test out the app yourself.

References

[1] *Lung cancer statistics: How common is lung cancer?* American Cancer Society. (n.d.). Retrieved September 30, 2021, from https://www.cancer.org/cancer/lung-cancer/about/key-statistics.html.

[2] Blandin Knight, S., Crosbie, P. A., Balata, H., Chudziak, J., Hussell, T., & Dive, C. (2017, September). *Progress and prospects of early detection in lung cancer*. Open biology. Retrieved September 30, 2021, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5627048/.

[3] Wexier, A. (2021, February 10). *Lung cancer stages: Survival rate, prognosis, and more*. Medical News Today. Retrieved September 30, 2021, from https://www.medicalnewstoday.com/articles/316198#survival-rates.

[4] *Salary: Radiologist | glassdoor*. Glassdoor. (n.d.). Retrieved October 22, 2021, from https://www.glassdoor.com/Salaries/radiologist-salary-SRCH_KO0,11_IP3.htm.

[5] *Find an NCI-designated cancer center*. National Cancer Institute. (n.d.). Retrieved October 22, 2021, from https://www.cancer.gov/research/infrastructure/cancer-centers/find.

[6] Del Ciello, A., Franchi, P., Contegiacomo, A., Cicchetti, G., Bonomo, L., & Larici, A. R. (2017). *Missed lung cancer: When, where, and why?* Diagnostic and interventional radiology (Ankara, Turkey). Retrieved September 30, 2021, from https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5338577/.

[7] Somasundaram, E., Litzler, A., Wadhwa, R. R., & Scott, J. G. (2020, January 1). *Persistent homology of tumor ct scans predicts survival in lung cancer*. medRxiv. Retrieved September 30, 2021, from https://www.medrxiv.org/content/10.1101/2020.12.06.20244863v1.full.

[8] Simon, S. (2017, February 2). *Lung cancer screening rates remain low*. American Cancer Society. Retrieved October 22, 2021, from https://www.cancer.org/latest-news/lung-cancer-screening-rates-remain-low.html.

[9] Rice, S. (2015, February 6). *Medicare will pay for lung CT scans for cancer screening*. Modern Healthcare. Retrieved October 22, 2021, from https://www.modernhealthcare.com/article/20150205/NEWS/150209967/medicare-will-pay-for-lung-ct-scans-for-cancer-screening.

[10] Islami F, Miller KD, Siegel RL, et al. National and State Estimates of Lost Earnings From Cancer Deaths in the United States. *JAMA Oncol.* 2019;5(9):e191460. doi:10.1001/jamaoncol.2019.1460

[11] Henderson JT, Webber EM, Sawaya GF. Screening for Ovarian Cancer: Updated Evidence Report and Systematic Review for the US Preventive Services Task Force. *JAMA.* 2018;319(6):595–606. doi:10.1001/jama.2017.21421

[12] *How to check for ovarian cancer: Ovarian cancer screening*. American Cancer Society. (n.d.). Retrieved October 22, 2021, from https://www.cancer.org/cancer/ovarian-cancer/detection-diagnosis-staging/detection.html.

[13] "Gudhi Library." *GUDHI*, https://gudhi.inria.fr/python/3.0.0/cubical_complex_user.html.

[14] "Vietoris–Rips Complexes." *Vietoris–Rips Complexes*, https://mrzv.org/software/dionysus2/tutorial/rips.html.

[15] Bukkuri, Anuraag, et al. "Applications of Topological Data Analysis in Oncology." *Frontiers*, Frontiers, n.d, https://www.frontiersin.org/articles/10.3389/frai.2021.659037/full.

[16] Svoboda, Elizabeth. "Artificial Intelligence Is Improving the Detection of Lung Cancer." *Nature News*, Nature Publishing Group, 18 Nov. 2020, https://www.nature.com/articles/d41586-020-03157-9.

[17] Ardila D, Kiraly AP, Bharadwaj S, Choi B;Reicher JJ, Peng L, Tse D, Etemadi M, Ye W, Corrado G, Naidich DP, Shetty S, "End-to-End Lung Cancer Screening with Three-Dimensional Deep Learning on Low-Dose Chest Computed Tomography." *Nature Medicine*, U.S. National Library of Medicine, https://pubmed.ncbi.nlm.nih.gov/31110349/.

[18] Software engineering code - ACM ethics. ACM Ethics - The Official Site of the Association for Computing Machinery's Committee on Professional Ethics. (2018, December 19). Retrieved October 23, 2021, from https://ethics.acm.org/code-of-ethics/software-engineering-code/.

[19] W3C. (n.d.). Retrieved October 23, 2021, from https://www.w3.org/standards/webdesign/.

[20] *The best way to build web apps without code*. Bubble. (n.d.). Retrieved October 23, 2021, from https://bubble.io/.

[21] Al Mohammad, B, et al. “Radiologist Performance in the Detection of Lung Cancer Using CT.” *Clinical Radiology*, The Royal College of Radiologists, 8 Oct. 2018, https://www.clinicalradiologyonline.net/article/S0009-9260(18)30574-9/fulltext#:~:text=When%20planning%20for%20a%20large,similar%20to%20cancer%20center%20radiologists.

[22] Lawson, Peter, et al. “Persistent Homology for the Quantitative Evaluation of Architectural Features in Prostate Cancer Histology.” *Nature News*, Nature Publishing Group, 4 Feb. 2019, https://www.nature.com/articles/s41598-018-36798-y.

[23] Liao, Fangzhou, et al. “Evaluate the Malignancy of Pulmonary Nodules Using the 3d Deep Leaky Noisy-or Network.” *ArXiv.org*, 22 Nov. 2017, https://arxiv.org/abs/1711.08324.

[24] “Notebook on Nbviewer.” *Jupyter Notebook Viewer*, https://nbviewer.org/github/bckenstler/dsb17-walkthrough/blob/master/Part%201.%20DSB17%20Preprocessing.ipynb.

Appendix A: Calculating the Benchmark

In this section we discuss our methods for preparing a benchmark with which to compare artificial intelligence to a human radiologist. Namely, due to the demonstrated sensitivity and specificity rates determined in [21], we compute an expected accuracy rate for conventional radiologists, assuming that they observe cancer positive or cancer negative patients with the same frequency as is presented in our data. That is, in the Kaggle data science bowl dataset used for training our neural networks, 70% of patients were classified as not having cancer, and 30% of patients were cancer positive. This leaves the simple calculation:

$$(sensitivity \ * \ positive\,ratio) \ + \ (specificity \ * \ negative\,ratio)$$

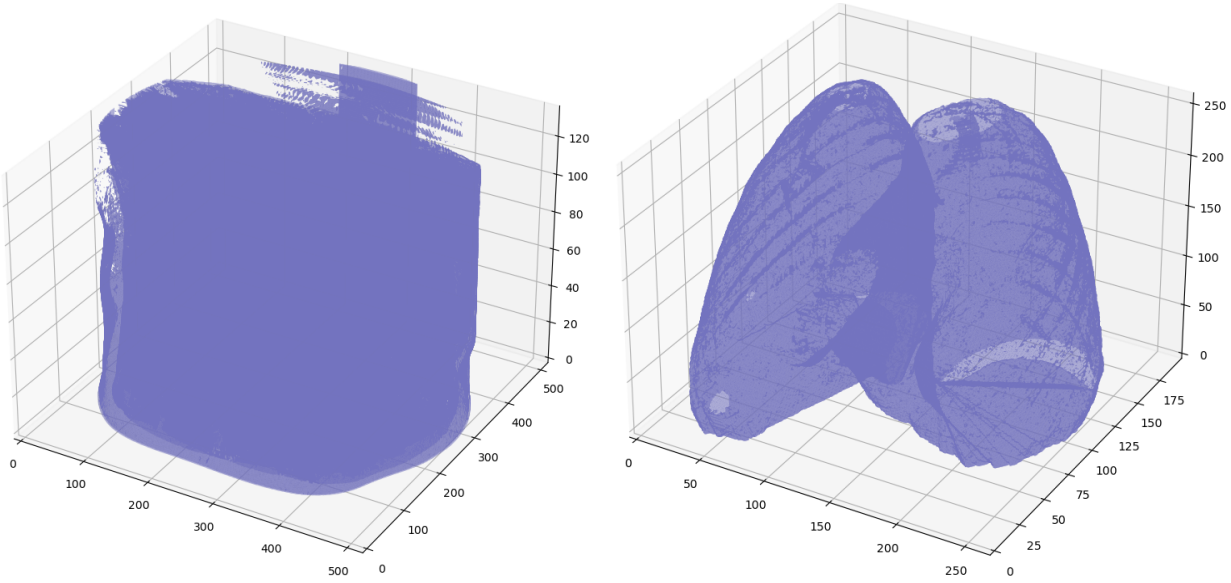
$$= \ 0.75 \ * \ 0.7 \ + \ 0.82 \ * \ 0.3 \ = \ 0.792 \ = \ expected\,accuracy$$

Admittedly, our metric is resting on the major assumption that radiologists in practice experience roughly 70% of patients being healthy and 30% of patients presenting lung cancer. We believe that this is a reasonable assumption given that the Kaggle data science bowl strives for datasets that are representative of the corresponding phenomenon in practice. However, such a jump should undoubtedly be noted when assessing our benchmark for performance.

Appendix B: Model Architecture, Preprocessing, and Statistical Significance

This section covers the preprocessing techniques used, the model architectures we used for a 2D and 3D CNN, and the statistical significance of our achieved accuracies.

Pre-Processing: Our implementation was adapted from the winning preprocessing implementation of the 2017 Kaggle data science bowl. After successfully downloading the DICOM dataset, we convert pixel data to Hounsfield Units, allowing us to select regions of a scan according to radiodensity. We then use careful thresholding to identify background pixels (water and air in the lung) and build a binary mask accordingly. Then, a handful of gaps inevitably will exist in the subsequent lung mesh, which we fill. Finally, to remove noisy esophageal structures, we separate each half of the lung, and delete the leftover networks. As a final clean up step, we fill in the resulting excess gaps in 2D binary mask slices for our image, and obtain after some careful thresholding a clean picture of only the lung and interior nodules. The process is extremely robust, and is undoubtedly a primary factor in the success of our implemented models.



(a) Before preprocessing at lung threshold (b) After preprocessing at lung threshold

2D CNN: After accomplishing a trustworthy preprocessing framework for our dicom data, our most successful model implementation was a two-dimensional convolutional neural network, with convolutions being conducted slice by slice in the overall image of the lung. This was implemented using the standard Tensorflow Keras CNN framework. After hypertuning, we found that the model performed best with four hidden layers, each with 32 filters, two dense layers, and the Relu activation function. In support of the accuracy result achieved by the CNN, our model was trained on a batch of 200 dicom images using five-fold cross validation, (with 80/20 training/test set splits) with a mean accuracy of 82%. This winning model was saved and is the functioning classifier in our web application. It is worth noting that this model cannot be

trained on a standard laptop, and we were forced to depend on the free trial of Google cloud's AI platform to gain sufficient computing power.

3D CNN: We achieved slightly less, though comparable, success with a three-dimensional convolutional neural network, again using tensorflow. The strongest 3D CNN we could muster achieved a 79.2% accuracy, and had three hidden layers and two dense layers. Despite achieving close results to the 2D CNN, the three-dimensional implementation had clear shortcomings in time complexity, which is expected as a consequence of its three-dimensional convolutions. In order to train the 3D CNN on 200 images, the authors were forced to quickly use a free trial of Google cloud's AI platform, with a VM running 16 CPUs each with 128 GB of RAM.

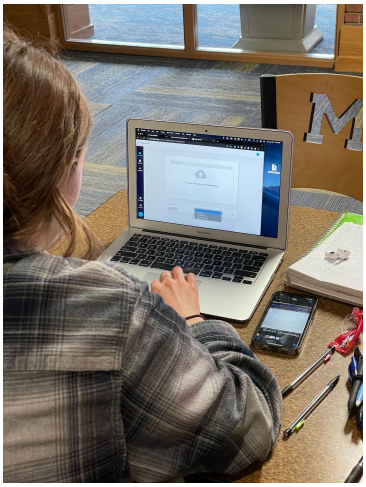
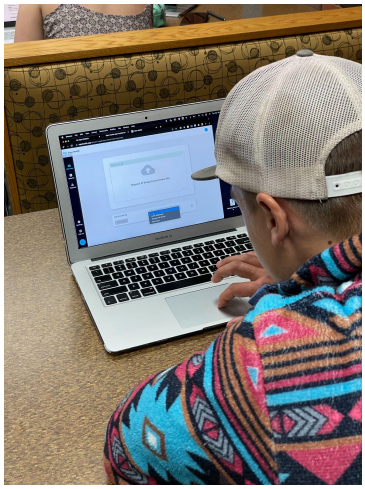
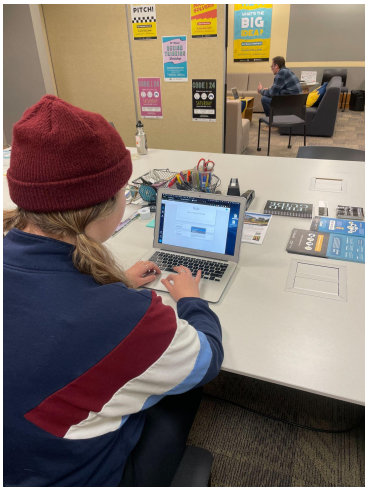
TDA: We achieved the least success, in time complexity and in accuracy (63%), from a k-nearest neighbor model derived from bottleneck distances between the persistence diagrams of a Vietoris-Rips filtration on the preprocessed point data. This was done once for each homology group (0,1 and 2), using a batch of 100 scans, and setting K=5. In particular, we were interested in the 2nd homology group, which collects so-called 'voids' in data. In this way, TDA would directly portray the presence or absence of nodules in a scan through collecting a series of pockets. The reasoning behind this relatively quick abandonment of TDA came as a result of its time complexity. Conducting a Vietoris-Rips filtration on a representative number of points is an extremely computationally expensive process, and actually required leaving the model running for multiple days (for 400 scans) over spring break. Obtaining a filtration, even on only 100 images, requires significant pruning of a dataset. Because nodules make up such a small subset of the total points in a point cloud, thinning our data to a computationally reasonable degree meant forgoing the actual topological features we were trying to detect in the first place.

Statistical Significance: The statistical significance of these models also suffers from the sheer volume of our data, which is challenging to process with the standard laptop on large scales. That said, we are able to bolster our results for the 2D CNN as maintained by 5-fold cross validation, occurring in 2 batches each of 200 images. Higher cardinality training sets and cross validation parameters were attempted, but exhausted the compute resources of Google's AI platform (dangerously close to the end of our free trial for the $1500/month compute resources). These models should be taken with a grain of salt, and given more time and computational resources we expect that these results could be bolstered further.

Appendix C: Usability Study

In order to test usability and intuitiveness of our app, we conducted a usability study where we prompted 20 people who had never used the app to accomplish the task of analyzing a CT Scan and telling us what the results of that scan were. 90% of these participants successfully

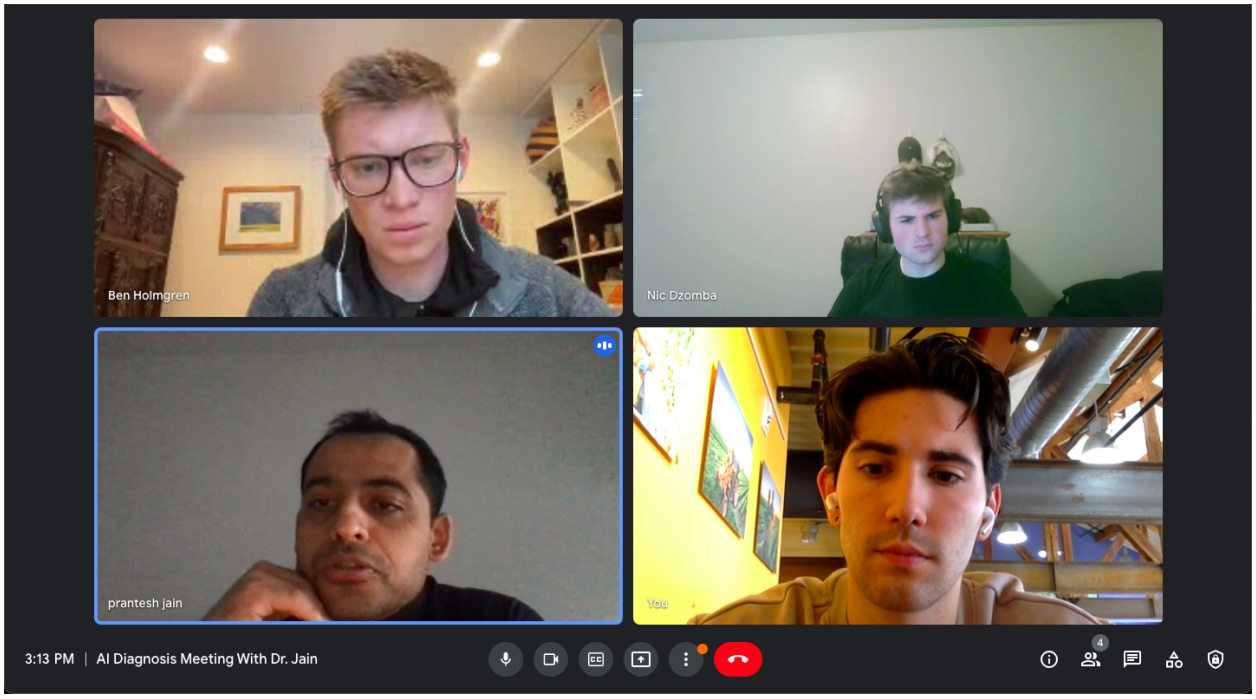
accomplished this task without us saying a word to them through the process. A few participants even remarked, “That was it? That was super simple.” at the end of the test.

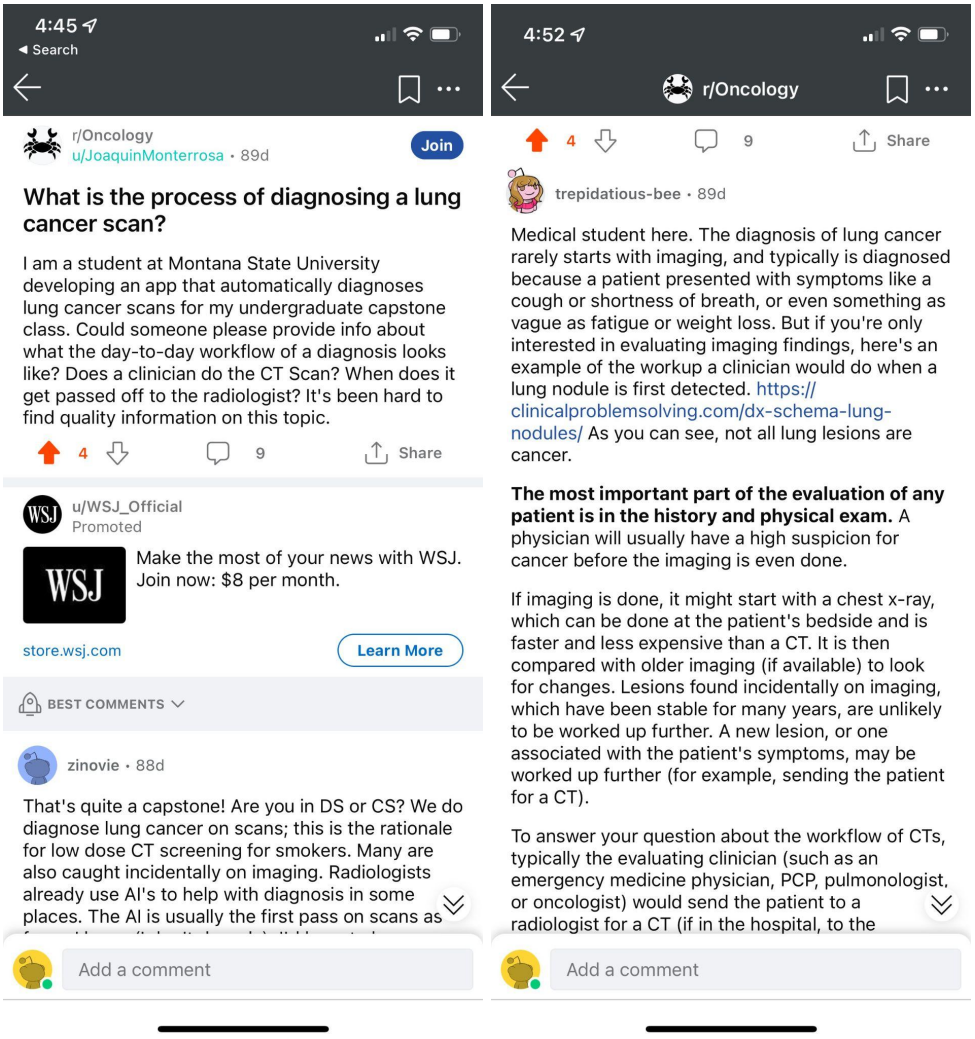




Appendix D: Customer Discovery Sprint

To understand the end users of the Topo Health App, we conducted a week-long customer discovery sprint where we interviewed radiologists, radiology technicians, and oncologists. Throughout this process we wanted to answer two main questions. We wanted to understand the entire process of lung cancer diagnosis and treatment in detail and we wanted feedback as to how we could build the app in the optimal way for day-to-day use by radiologists.

It was a bit difficult to find and interview radiologists and oncologists but ultimately we were able to interview four radiologists and two oncologists (one of whom does research directly related to AI diagnosis of lung cancer). We used a variety of interview methods. We created reddit threads, we had conversations over the phone, and we conducted Zoom interviews.


Ben Holmgren
Nic Dzomba
prantesh jain
You
3:13 PM | AI Diagnosis Meeting With Dr. Jain

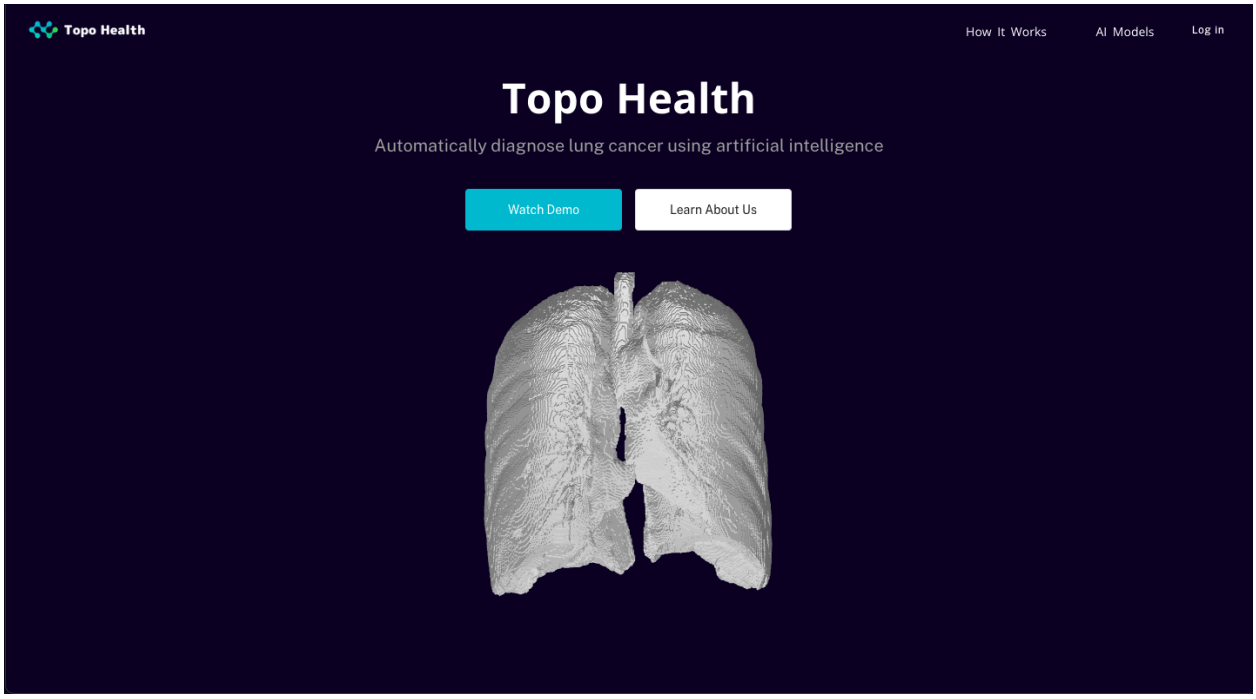


Ultimately, we were able to answer our two questions. We gained a detailed understanding of the lung cancer diagnosis and treatment process and we received feedback informing us how to best design the Topo Health app for day-to-day use by radiology technicians.

Appendix E: Product Captures

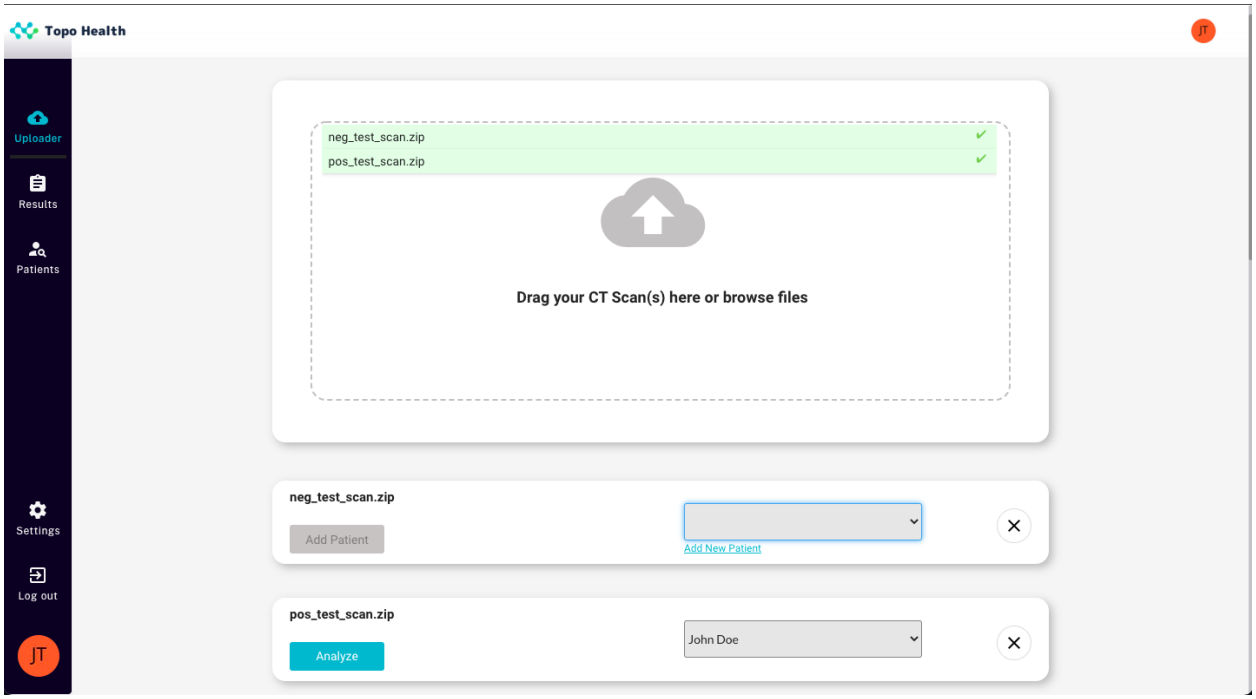
Here are captures of the Topo Health App and short explanations of the functionality represented by each screenshot. You can also reference Appendix G if you want to test out our app for yourself.

Marketing Page



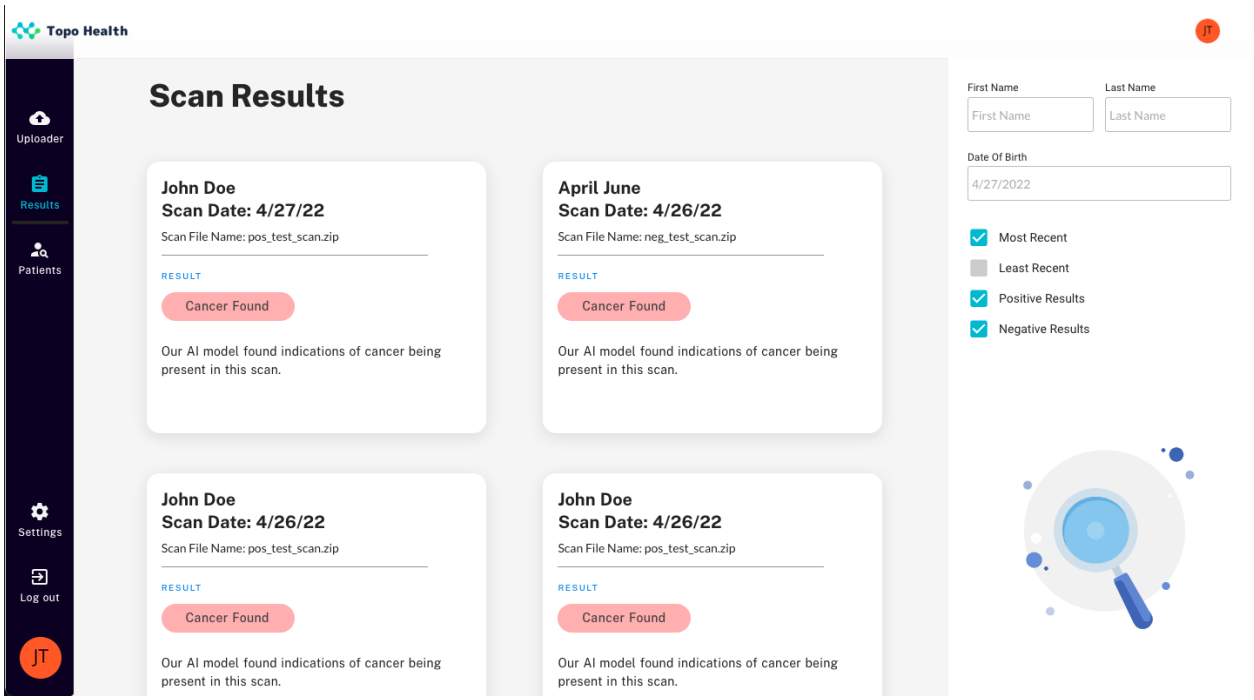
Our marketing page is meant to show off the Topo Health app to anyone who is interested in learning about the app. We walk through how the app works and we explain the accuracy our AI models. Additionally, visitors can watch a short demo of our app or read this portfolio by clicking the top two buttons on the page.

Uploader Section



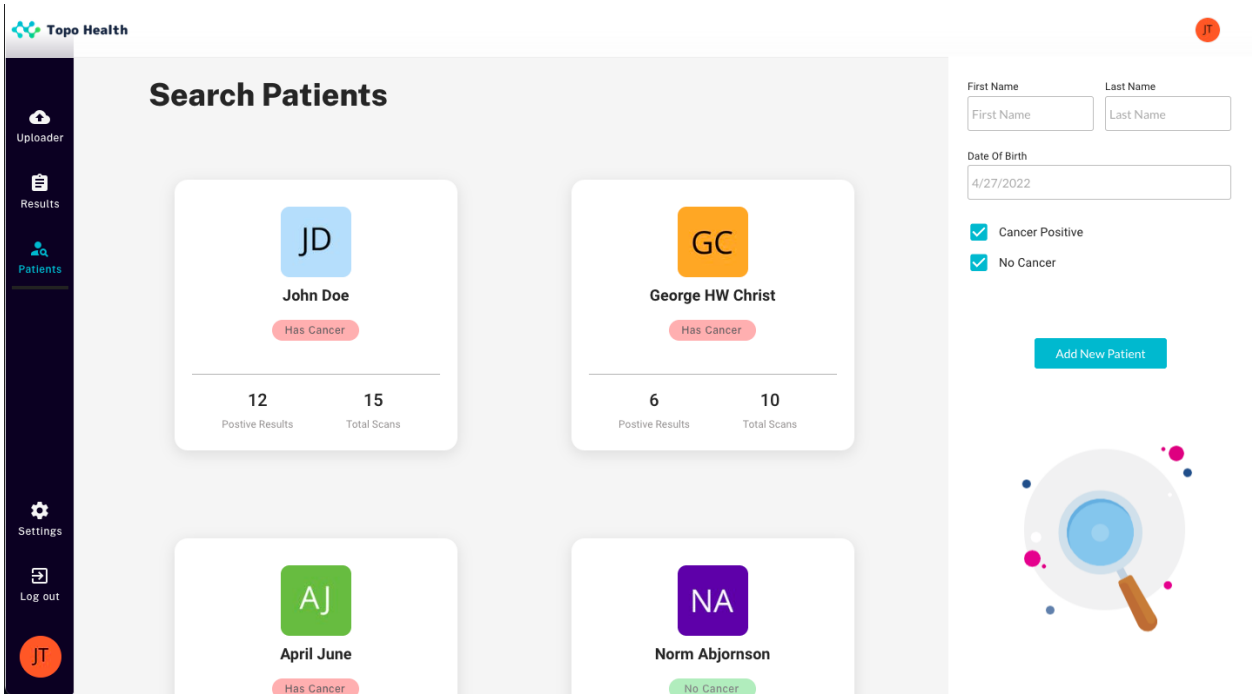
When you log in as a technician, you're immediately directed to the "Uploader Section" of the app. From here you can drag and drop multiple zipped CT scan files into the uploader window, assign a patient to those scans and analyze them.

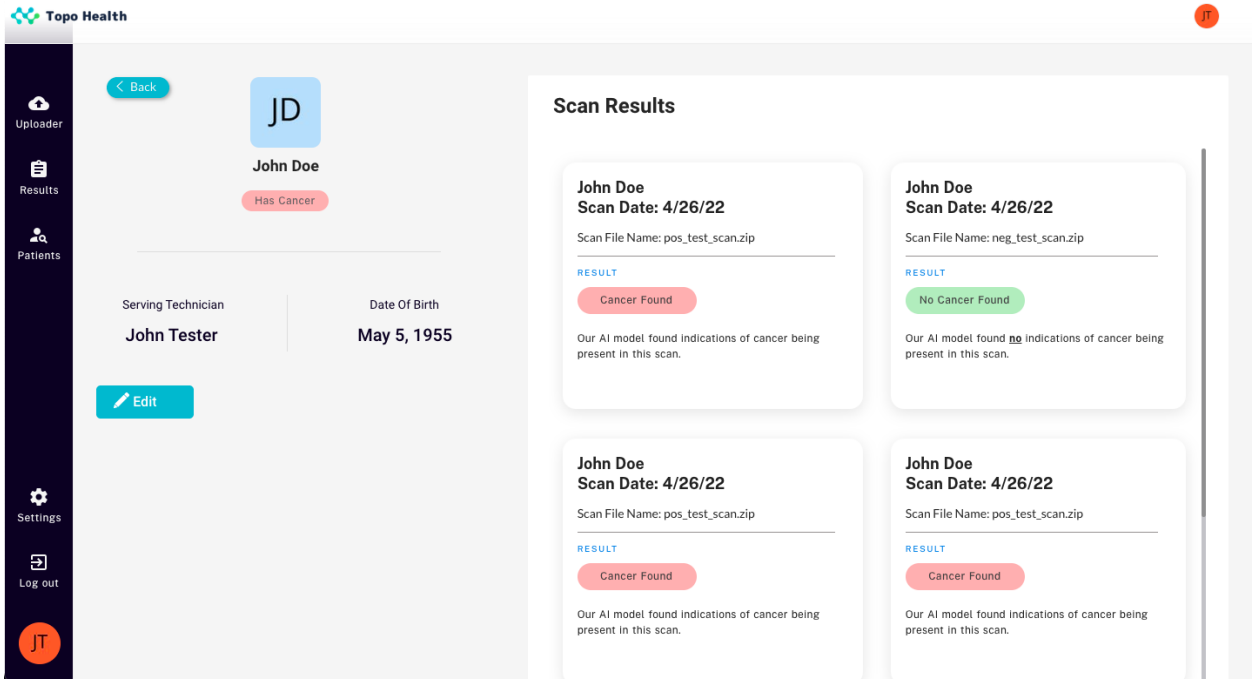
Results Section



As a technician in the "Results Section", you can view past scans and their results. You can use a variety of search and filter tools to find the exact scan that you're looking for.

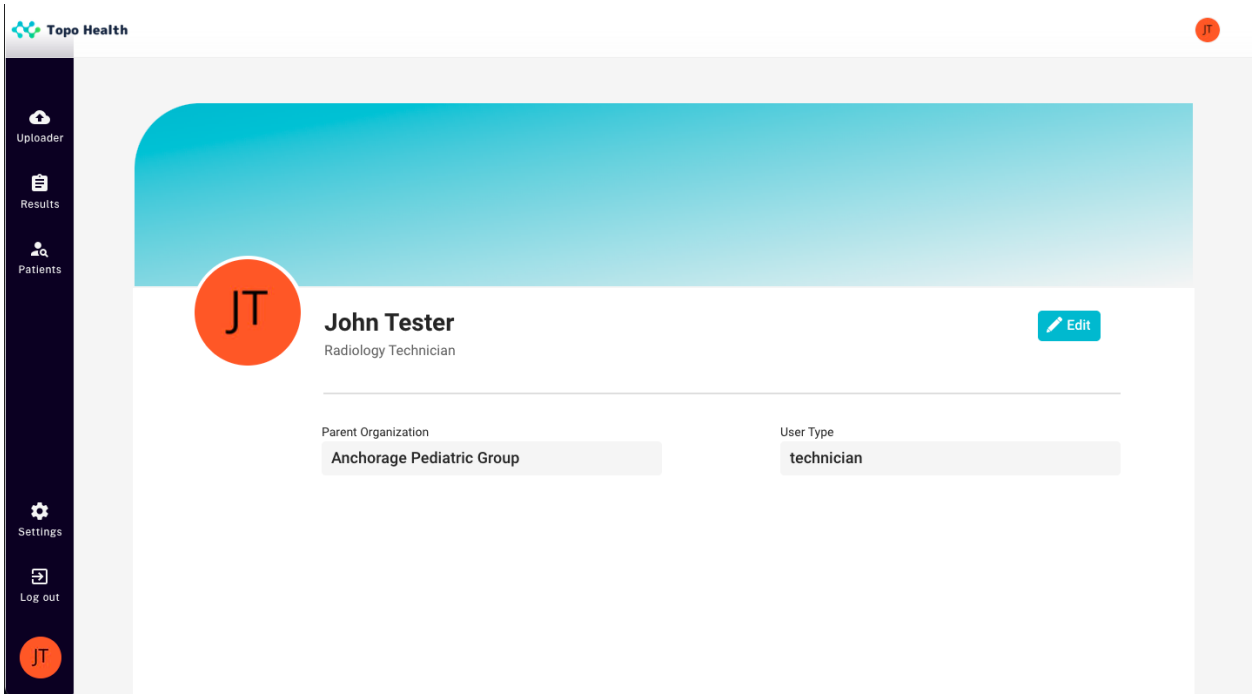
Patients Section





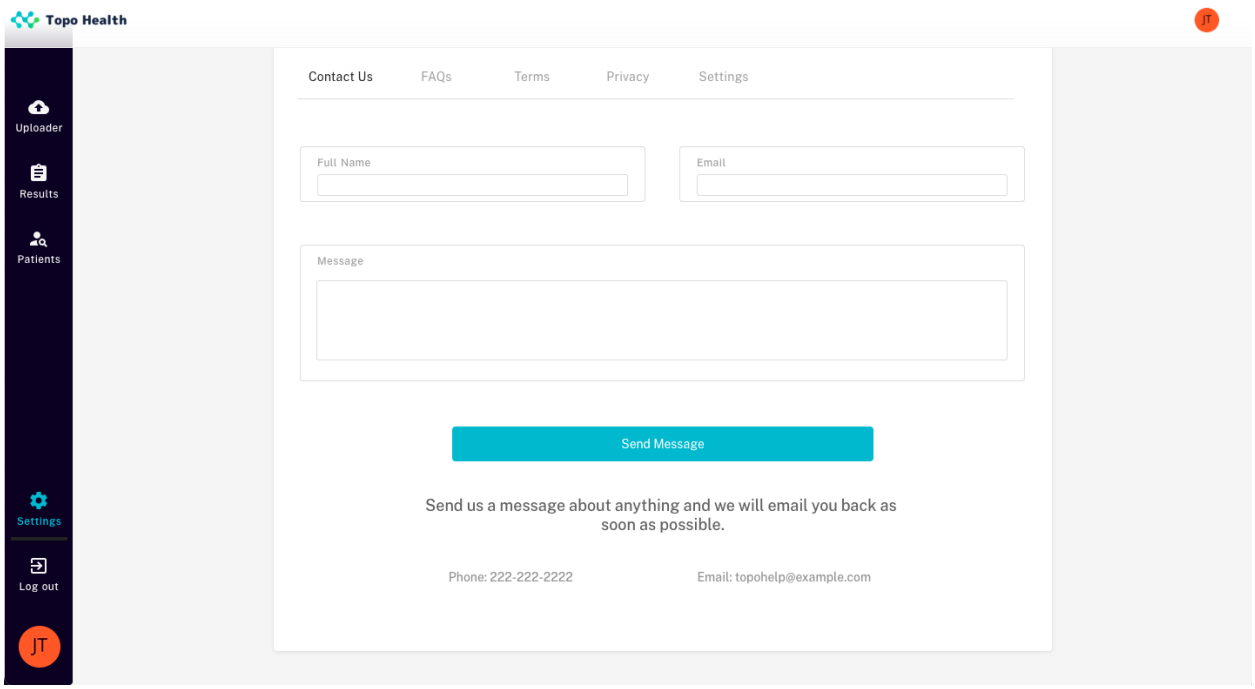
As a technician in the “Patients Section”, you can view all the patients belonging to your clinic. You can use a variety of search and filter tools to find the exact patient that you’re looking for. Additionally, you can create a new patient in this section and you can click into a patient to view and update patient information and to view scans associated with that patient.

Profile Section



As a technician in the “Profile Section”, you can view and edit your profile.

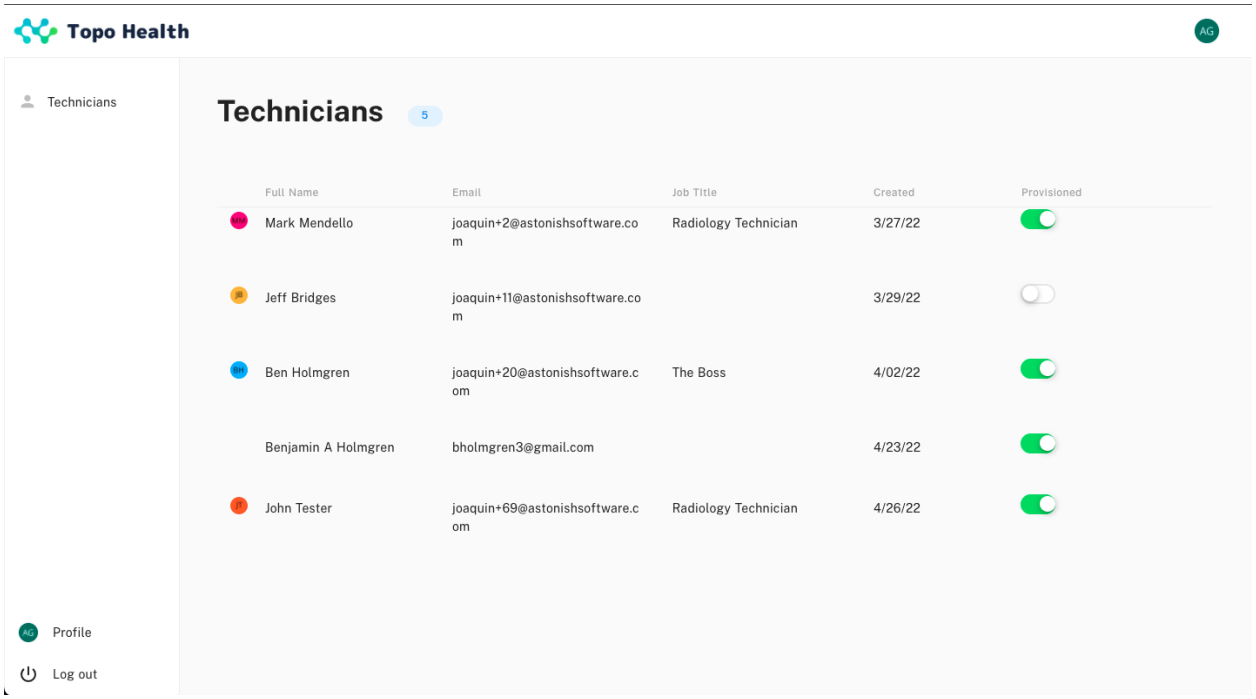
Settings Section



As a technician in the “Settings Section”, you can contact us (the app admin) with any question or to report a bug. You can also read up on frequently asked questions, you can view the terms

and conditions and privacy policy, and you can access additional settings by clicking on the "settings" tab.

Clinic Admin Dashboard



As a clinic, you can view all the technicians that are currently working under your organization. Each technician must be provisioned before they can access the app or your sensitive patient information.

Appendix F: Old Work Schedule

Naturally, our work schedule changed as we started to work on the project. The old work schedule simply lists the high-level tasks that we could predict. Our actual work schedule broke these high-level tasks (epics) down into smaller tasks (stories) that could be accomplished in our sprints. Below, is an exported representation of our old work schedule:

Old Work Schedule

Name	Assign	Date	Property	Status
Cushion		@May 1, 2022 → May 7, 2022		Not started
Testing	Joaquin Monterrosa Nic Dzomba Ben	@April 17, 2022 → April 23, 2022		Not started
Testing	Joaquin Monterrosa Nic Dzomba Ben	@April 24, 2022 → April 30, 2022		Not started
General Market Research	Joaquin Monterrosa	@January 23, 2022 → January 29, 2022		Not started
Low Fidelity Mockups	Joaquin Monterrosa	@January 23, 2022 → January 29, 2022		Not started
High Fidelity Mockup	Joaquin Monterrosa	@January 30, 2022 → February 5, 2022		Not started
Implementing Databases and Storage	Joaquin Monterrosa	@March 20, 2022 → March 26, 2022		Not started
Build Static Front-End	Joaquin Monterrosa	@February 6, 2022 → February 12, 2022		Not started
Implement Server	Joaquin Monterrosa	@April 3, 2022 → April 9, 2022		Not started
Write back-end functions	Ben Nic Dzomba	@April 3, 2022 → April 9, 2022		Not started
Make Front-end functional	Joaquin Monterrosa	@February 13, 2022 → March 12, 2022		Not started
Connect Databases to Front-end	Joaquin Monterrosa	@March 27, 2022 → April 2, 2022		Not started
Connect Server to Front-end	Joaquin Monterrosa	@April 10, 2022 → April 16, 2022		Not started
Preprocess .dcm images	Ben	@January 23, 2022 → January 29, 2022		Not started
Implement & Test TDA	Ben	@January 30, 2022 → February 19, 2022		Not started
Implement & Test CNN	Nic Dzomba	@January 30, 2022 → February 12, 2022	2 weeks	Not started
Implement & Test Deep Learning	Nic Dzomba Ben	@February 13, 2022 → March 5, 2022		Not started
Implement Winning Model On Server	Nic Dzomba Ben	@March 6, 2022 → March 12, 2022		Not started

Name	Assign	Date	Property	Status
Model Results → Interpreted Results	N Nic Dzomba Ⓑ Ben	@March 20, 2022 → April 2, 2022		Not started
Develop Prognosis Template	N Nic Dzomba	@April 10, 2022 → April 16, 2022		Not started
Spring Break (Cushion)		@March 13, 2022 → March 19, 2022		Not started

Appendix G: Test Out The App

To test out the app, click the link below and follow the instructions presented:

https://eight-net-3e0.notion.site/Test-Out-Topo-Health-51bb452a8db24612b28773b905caacdf

Appendix H: Key Features

Some key features of the Topo Health app include:

HIPAA Database

We're not naive enough to believe our app is fully ready for a HIPAA audit but by storing all of our personal identifiable information (PII) and personal health information (PHI) on a HIPAA compliant database (separate from the normal database), our app is in a good position to achieve HIPAA compliance in the future.

Multi-Scan Uploading

Our "uploader window" allows for multiple scans to be uploaded and analyzed at one time. This is important because radiologists told us that they would like to analyze scans in batches.

Strong User Access Managment

All technicians must first be provisioned by their parent clinic before they can access the app. Clinics can control this provisioning through an admin dashboard on the app. Additionally, clinics must be provisioned by app admin using the app admin dashboard. All in all, this allows for strong user access management which is a vital part of the Topo Health app.

Scalable Capacity

Topo Health has been built to scale by using platforms that allow for elastic scaling. We use AWS (through Bubble) to host our front-end and normal database and we use Google Cloud Platform to host our AI models.

Search and Filter Options For Scans and Patients

Technicians can search and filter for patients and past scans using a variety of hand-picked search and filter options.

CT Scan Storage

Topo Health allows users to securely store their CT scans on our database. This is a valuable feature because CT scans are large files and they can be hard to send and store via traditional platforms.

Fast Diagnosis

Our winning 2D CNN model is hosted and can execute rapidly (~90 seconds per scan) on a cloud server.

Appendix I: Who Did What?

Joaquin Monterrosa

Joaquin was responsible for the entire product side of the Topo Health. He developed the web-app, configured the Google Cloud Platform, designed the databases, added all the necessary API endpoints, created the branding, designed the UI and userflow, conducted usability testing, and did anything else related to the product. He also served as the Scrum Master for our agile development lifecycle.

Ben Holmgren

Ben was responsible for the machine learning and data management side of Topo Health. He constructed the pipeline for dicom datasets, found and adapted the preprocessing implementation, and constructed the TDA model and CNNs. When the models weren't possible to train on large datasets locally due to high compute costs, he configured a Google cloud environment to train the models. He also dealt with the statistical significance of the final chosen CNN, implementing cross validation, and integrated the saved model into Google cloud to be run in the final web application.

Nic Dzomba

Nic was responsible for working alongside Ben on the preprocessing of the dicom datasets and the machine learning implementation. Additionally he did the initial set up & configuration of the Google cloud environment.

Appendix J: Alternative Design Patterns

We used a unique tech stack that combined low-code platforms with custom code. Because of this, we didn't use a traditional object-oriented design and therefore didn't include a class diagram in the architecture section of this portfolio. That being said, we did design a class

diagram depicting how our app could be implemented in an object-oriented manner. This alternative architecture is shown below.

Notes

- create___Profile() methods also include update functionality
- This class diagram is primarily conceptual. These classes willl be implemented using a web-framework not a strict OO language.
- Concrete user classes are created using a Factory Pattern
- Result Class uses a decorator pattern to add additional result fields and functionality to positive results.
- All getters and setters are assumed methods in each class

Topo Health - class diagram

Joaquin Monterrosa | April 28, 2022