SWE-590 TERM PROJECT

ML BASED DIABETIC RETINOPATHY DETECTION APPLICATION

27.12.2022

E.HAKAN IBIL-SADIK KUZU

Introduction

Diabetic retinopathy is a phenomenon seen in the retina of the eye as a complication of diabetes. As a result of diabetes damaging the vascular structure, the vessels that feed the retina cannot fully fulfill their functions and if this process continues, some damage may occur that affects the person's vision. Numbered grading is used in the diagnosis of diabetic retinopathy. According to these values between 0 and 4, the degree of diabetic retinopathy of the person is determined. 0 indicates that there is no evidence of the disease, while 4 indicates the highest degree in the diagnosis of diabetic retinopathy. In this project, we designed a model that processes the fundus image uploaded by the user by using image processing and machine learning algorithms and models in the Cloud environment, then returns this processed image to the user and then makes a diagnosis according to this processed image. Since the aim of the project is to use Cloud instruments effectively and to implement the project on the Cloud platform, the project will be analyzed from this point of view after this section. The design of the report is as follows; In Chapter 1, the Image Processing Module is discussed. In Chapter 2, the Prediction Module that predicts the processed image will be examined, and in Chapter 3, the tests of the created project will be performed. In Section 4, the general architecture will be shown.

Targets

1. Recognise and use cloud instruments efficiently.
2. Identifying the requirements and needs of the project, identifying the Cloud instruments that are suitable to meet these needs and implementing the project in the cloud environment.
3. Performing tests of the project implemented on the cloud.

Features

1. Model training was performed on Google Colab.
2. APTOS-2019-BLINDNESS DETECTION dataset published on Kaggle was used as the dataset.
3. The model is a supervised based machine learning model.

1. Image Processing Module Overview

In order to make an effective prediction, the fundus image uploaded to the system must first be processed with some image processing algorithms. The purpose of this process can be defined as highlighting the regions that are important in diagnosis both when training the machine learning model and when predicting an image in the trained model.

We preferred AWS LAMBDA, which is a serverless solution for image processing. Of course, this solution could have been offered with a virtual machine or container, but the high utilization rate of cloud functions, which is a relatively new technology, and our desire to experience this technology led us to this choice.

Flow

1. The image uploaded to the system on the client side is converted to base64 format.
2. The image converted to base64 format is sent to AWS Lambda endpoint in JSON format.
3. The image in base64 format arriving at the endpoint triggers the image processing function.
4. The 'body' section is taken from the JSON format message, which is called Event and triggers the image processing function, and converted from base64 format to ndarray extension image file.
5. The image converted to ndarray format is processed with certain algorithms.
6. The processed image is converted back to base64 format.
7. The processed image converted to base64 format is returned to the user.

Challenges, Solutions and Contributions

As can be understood from the above process steps, it is mandatory to use a number of libraries for the operation of image processing algorithms and other system requirements. In our project, these are mainly CV2, BASE64, PIL, JSON, NUMPY libraries. AWS Lambda does not support these libraries as pre-built. Extra configuration is required to use these libraries on AWS Lambda. At this point, the Layer feature of AWS Lambda was used. The Layer feature allows libraries that do not come pre-built to be used on Lambda. In order to configure Layer, firstly you need to configure ;

1. Creating virtualenv in local or virtual machine
2. Then upload the requirements of the source code to this virtualenv environment,
3. Isolate the relevant virtualenv environment after making sure that its function is working properly
4. To zip this isolated environment and upload it to file storage environments with AWS CLI commands.

5. Add this uploaded file to AWS Lambda as a layer.

Contributions;

1. AWS Lambda Layer feature was used.
2. EC2 was used to provide Layer requirements and dependencies files were isolated here.
3. Experienced using AWS CLI to upload isolated files to S3 bucket.

2. Prediction Module Overview

The fundus image uploaded to the system is returned to the user after processing on AWS Lambda. After this stage, the processed image needs to be predicted in the tilted model. Cloud platforms allow the user to predict this trained model via an endpoint after training ML Models. A second method is to upload the custom model to the system and move it to the endpoint for prediction. Since we trained the model in Google Colab before our project, the second option was our preferred method. At this point, Cloud platforms provide solutions with various instruments. On the AWS side, this service is SageMaker, and on the GCP side, Vertex AI can be given as examples of these instruments. Since the image processing function is deployed on AWS, our first choice was AWS SageMaker. However, the fact that the trained model was trained with the TensorFlow framework and that some extra adjustments were required for this model to work in SageMaker led us to use Vertex AI, which is more suitable for TensorFlow.

Flow

1. The model in SavedModel Format is uploaded to Google Bucket.
2. The bucket with the relevant custom model is selected via Vertex AI and the model is deployed.
3. The standing model is deployed to the endpoint.
4. The processed image returned to the user is first converted to .PNG format.
5. The image converted to .PNG format is converted to ndarray format.
6. The image converted to ndarray format is converted to JSON.
7. The JSONised message is sent to the Vertex AI endpoint for prediction.
8. The image in ndarray format is predicted in Vertex AI.
9. The prediction is returned to the user.

Challenges, Solutions and Contributions

Since TensorFlow is a framework developed by Google, it can be said that models trained with this framework work more harmoniously on GCP. In Vertex AI, the maximum size of the JSON object sent to the endpoint is limited to 1.5 mb. This situation required us to revise the function on AWS Lambda. Because in its unrevised version, when the image processed on AWS Lambda was converted into a JSON object, it had a size of approximately 5 mb. This problem was solved by reducing the size of the image from 400x400 to 80x80. The model was also retrained.

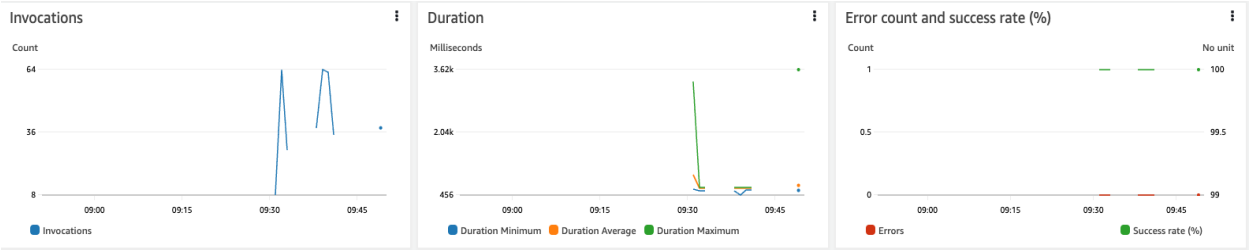
3. Test

We have tested our project modules separately (Image Processing module and Prediction Module) with different loads (n= 10, 100, 200 , 500). We have seen that modules We observed that the performance of the modules did not decrease in the tests and there was no significant change in response times. At this point, we have seen that the auto-scaling feature of AWS Lambda, one of the serverless solutions, stands out significantly. Below are the data obtained as a result of the test and the graphs of the systems.

We also experienced the cold-start situation, which is one of the disadvantages of serverless solutions, in our tests. The response time, which lasted between 650-700 ms on average in the tests, increased to around 3500 ms in cases where we observed cold start. This situation is clearly seen in the images below.

Image Processing Module(AWS Lambda)

n=100;

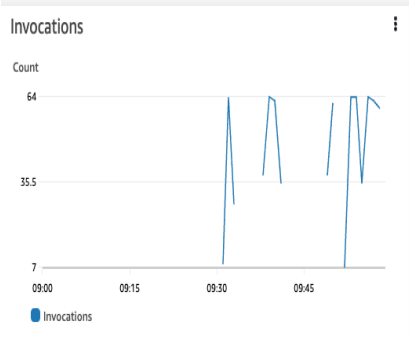


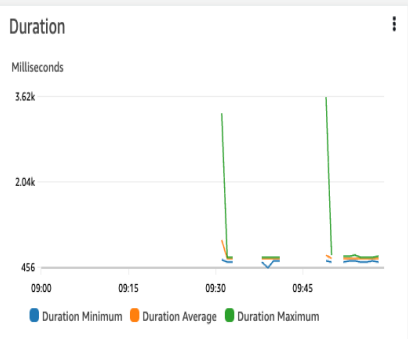
#	Timestamp	RequestID	LogStream	DurationInMS	BilledDurationIn…	MemorySetInMB	MemoryUsedInMB
1	2022-12-26T09:50:21.354Z	da6cec56-d01c-4548-b356-3da96dda28b9	2022/12/26/[$LATEST]f27bfb2dfcf74e2baf1f795367092632	642.6	643.0	128	119
2	2022-12-26T09:50:20.336Z	db17f582-a784-4578-bb54-19d38f877942	2022/12/26/[$LATEST]f27bfb2dfcf74e2baf1f795367092632	573.36	574.0	128	119
3	2022-12-26T09:50:19.434Z	b651aeb8-3746-46cb-94aa-048872b45280	2022/12/26/[$LATEST]f27bfb2dfcf74e2baf1f795367092632	630.41	631.0	128	119
4	2022-12-26T09:50:18.494Z	1b9fd58b-764b-4fb2-b298-1e41c7a3517e	2022/12/26/[$LATEST]f27bfb2dfcf74e2baf1f795367092632	625.28	626.0	128	119
5	2022-12-26T09:50:17.554Z	ff24bd71-1ab6-4b55-bef4-c2dba49a2237	2022/12/26/[$LATEST]f27bfb2dfcf74e2baf1f795367092632	646.84	647.0	128	119
6	2022-12-26T09:50:16.594Z	e18fa7f2-4a26-41fc-8537-4fe614e2de71	2022/12/26/[$LATEST]f27bfb2dfcf74e2baf1f795367092632	617.73	618.0	128	119
7	2022-12-26T09:50:15.636Z	e627a60f-99b8-4e96-b1ee-13cf750da75a	2022/12/26/[$LATEST]f27bfb2dfcf74e2baf1f795367092632	636.65	637.0	128	119
8	2022-12-26T09:50:14.694Z	a1638723-5324-4a4b-a12a-497033e62f5d	2022/12/26/[$LATEST]f27bfb2dfcf74e2baf1f795367092632	624.39	625.0	128	119
9	2022-12-26T09:50:13.794Z	2b05b9e0-d6f9-4318-b802-b451dcf82b19	2022/12/26/[$LATEST]f27bfb2dfcf74e2baf1f795367092632	649.47	650.0	128	119

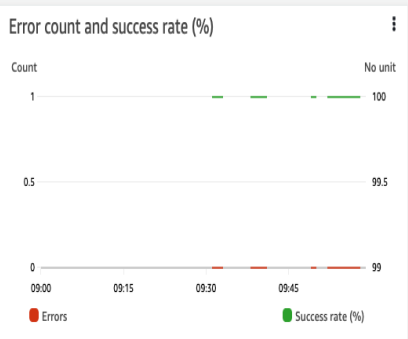
Most expensive invocations in GB-seconds (memory assigned * billed duration)

#	Timestamp	RequestID	LogStream	BilledDurationIn…	MemorySetInMB	BilledDurationInGBSeconds
1	2022-12-26T08:39:53.168Z	8b408bfa-24c3-489f-b575-c276b462791a	2022/12/26/[$LATEST]f23a7fd333fb4b54a2103c21bc587fc0	3802.0	128	0.4753
2	2022-12-26T08:46:55.715Z	92bca106-0ba5-4ec9-9b4b-e3e87a69fde3	2022/12/26/[$LATEST]059858cc3fa24a4bb6eb1af9d45222c5	3627.0	128	0.4534
3	2022-12-26T09:49:25.894Z	f701f40a-9094-4ee1-bf0f-d963687e7677	2022/12/26/[$LATEST]f27bfb2dfcf74e2baf1f795367092632	3619.0	128	0.4524
4	2022-12-26T07:57:16.826Z	46bc53b4-e03f-4c54-9ea1-5d1bf3c91156	2022/12/26/[$LATEST]87e42a712b94418c89ea8084d04c0822	3598.0	128	0.4497
5	2022-12-26T08:10:13.342Z	d182beaf-f977-4189-a4cf-4f9e616f8c8e	2022/12/26/[$LATEST]e4a77950ed864d0694fb1ef1b720f4af	3563.0	128	0.4454
6	2022-12-26T09:31:53.846Z	b70bf9e0-cab6-4bf0-b0fd-39601357dfe5	2022/12/26/[$LATEST]56fdf3dded0a41d0be28328f95e004a1	3315.0	128	0.4144
7	2022-12-26T09:31:54.846Z	db988ee0-1d3b-4a5c-a418-1f8c03984027	2022/12/26/[$LATEST]56fdf3dded0a41d0be28328f95e004a1	707.0	128	0.08837
8	2022-12-26T08:40:13.026Z	64d5ea58-3b1a-4cd0-a45f-b8c22781e8f8	2022/12/26/[$LATEST]f23a7fd333fb4b54a2103c21bc587fc0	677.0	128	0.08463
9	2022-12-26T09:50:12.834Z	91153ee3-9150-4bb2-8377-b3fd526c229b	2022/12/26/[$LATEST]f27bfb2dfcf74e2baf1f795367092632	665.0	128	0.08313

n=200;







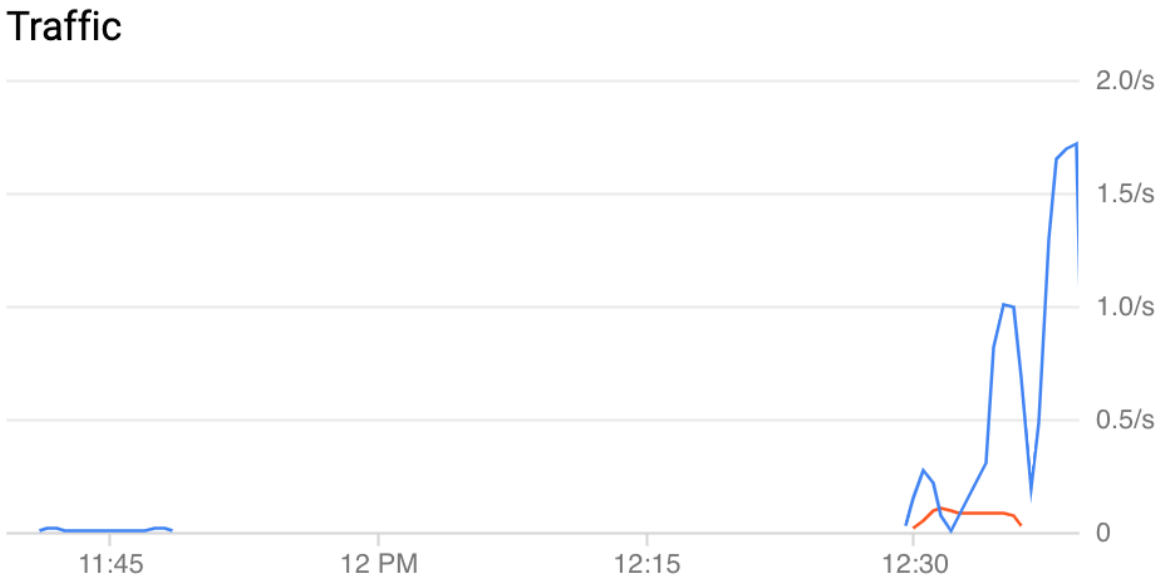
#	Timestamp	RequestID	LogStream	DurationInMS	BilledDurationIn…	MemorySetInMB	MemoryUsedInMB
1	2022-12-26T09:58:55.856Z	c2f33aae-a400-4e24-a6d9-b242050a34fe	2022/12/26/[$LATEST]f27bfb2dfcf74e2baf1f795367092632	613.73	614.0	128	119
2	2022-12-26T09:58:54.976Z	940ded16-9520-4000-8111-c61e2b7f2c97	2022/12/26/[$LATEST]f27bfb2dfcf74e2baf1f795367092632	642.72	643.0	128	119
3	2022-12-26T09:58:53.996Z	5fa25067-21e4-440c-b9d8-b16635ba5ab4	2022/12/26/[$LATEST]f27bfb2dfcf74e2baf1f795367092632	631.57	632.0	128	119
4	2022-12-26T09:58:53.056Z	956072ce-aee8-4246-9fd7-b4f14c4eed08	2022/12/26/[$LATEST]f27bfb2dfcf74e2baf1f795367092632	632.88	633.0	128	119
5	2022-12-26T09:58:52.116Z	3d92c278-3de5-4b57-878b-acff4c692f33	2022/12/26/[$LATEST]f27bfb2dfcf74e2baf1f795367092632	635.18	636.0	128	119
6	2022-12-26T09:58:51.119Z	8b1cb601-03a4-4148-b2ce-7334e31de1b3	2022/12/26/[$LATEST]f27bfb2dfcf74e2baf1f795367092632	587.07	588.0	128	119
7	2022-12-26T09:58:50.236Z	82e3ed41-a37f-46aa-b4a9-e1f5180344e6	2022/12/26/[$LATEST]f27bfb2dfcf74e2baf1f795367092632	635.49	636.0	128	119
8	2022-12-26T09:58:49.296Z	32d1813b-7f00-4540-9d67-2ceeaf486626	2022/12/26/[$LATEST]f27bfb2dfcf74e2baf1f795367092632	628.48	629.0	128	119
9	2022-12-26T09:58:48.356Z	b6e92642-dff0-4faa-95e2-a724dddf67dd	2022/12/26/[$LATEST]f27bfb2dfcf74e2baf1f795367092632	630.58	631.0	128	119

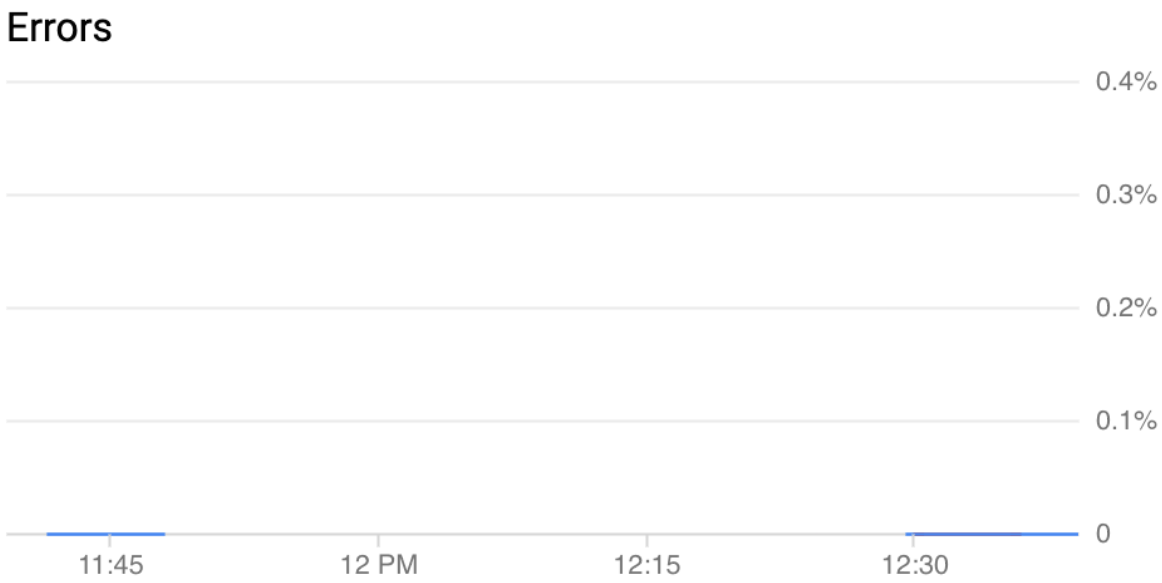
Most expensive invocations in GB-seconds (memory assigned * billed duration)

#	Timestamp	RequestID	LogStream	BilledDurationIn…	MemorySetInMB	BilledDurationInGBSeconds
1	2022-12-26T08:39:53.168Z	8b408bfa-24c3-489f-b575-c276b462791a	2022/12/26/[$LATEST]f23a7fd333fb4b54a2103c21bc587fc0	3802.0	128	0.4753
2	2022-12-26T08:46:55.715Z	92bca106-0ba5-4ec9-9b4b-e3e87a69fde3	2022/12/26/[$LATEST]059858cc3fa24a4bb6eb1af9d45222c5	3627.0	128	0.4534
3	2022-12-26T09:49:25.894Z	f701f40a-9094-4ee1-bf0f-d963687e7677	2022/12/26/[$LATEST]f27bfb2dfcf74e2baf1f795367092632	3619.0	128	0.4524
4	2022-12-26T07:57:16.826Z	46bc53b4-e03f-4c54-9ea1-5d1bf3c91156	2022/12/26/[$LATEST]87e42a712b94418c89ea8084d04c0822	3598.0	128	0.4497
5	2022-12-26T08:10:13.342Z	d182beaf-f977-4189-a4cf-4f9e616f8c8e	2022/12/26/[$LATEST]e4a77950ed864d0694fb1ef1b720f4af	3563.0	128	0.4454
6	2022-12-26T09:31:53.846Z	b70bf9e0-cab6-4bf0-b0fd-39601357dfe5	2022/12/26/[$LATEST]56fdf3dded0a41d0be28328f95e004a1	3315.0	128	0.4144
7	2022-12-26T09:31:54.846Z	db988ee0-1d3b-4a5c-a418-1f8c03984027	2022/12/26/[$LATEST]56fdf3dded0a41d0be28328f95e004a1	707.0	128	0.08837
8	2022-12-26T09:54:05.995Z	c106bbfa-6e6f-48a6-833e-54a8d028a5ad	2022/12/26/[$LATEST]f27bfb2dfcf74e2baf1f795367092632	689.0	128	0.08612
9	2022-12-26T09:50:54.894Z	1c961d00-81a0-4471-bc1d-16adb38bcb26	2022/12/26/[$LATEST]f27bfb2dfcf74e2baf1f795367092632	681.0	128	0.08513

Prediction Module(Google Vertex AI)

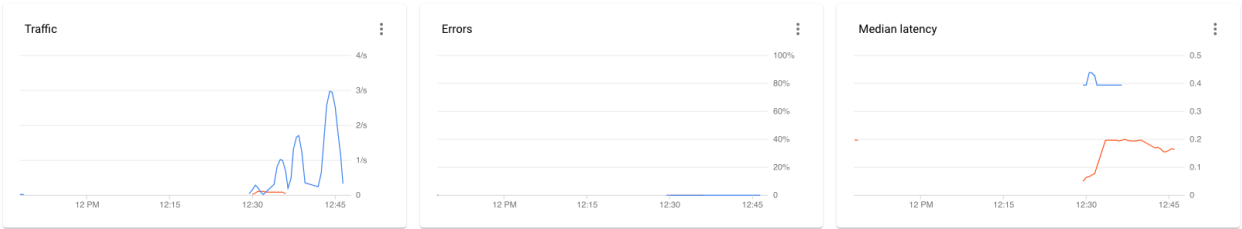
n=200;





Name	↓ Requests	Errors (%)	Latency, median (ms)	Latency, 95% (ms)
Vertex AI API	344	0	191	259
Notebooks API	29	0	397	520
Compute Engine API	5	0	305	502

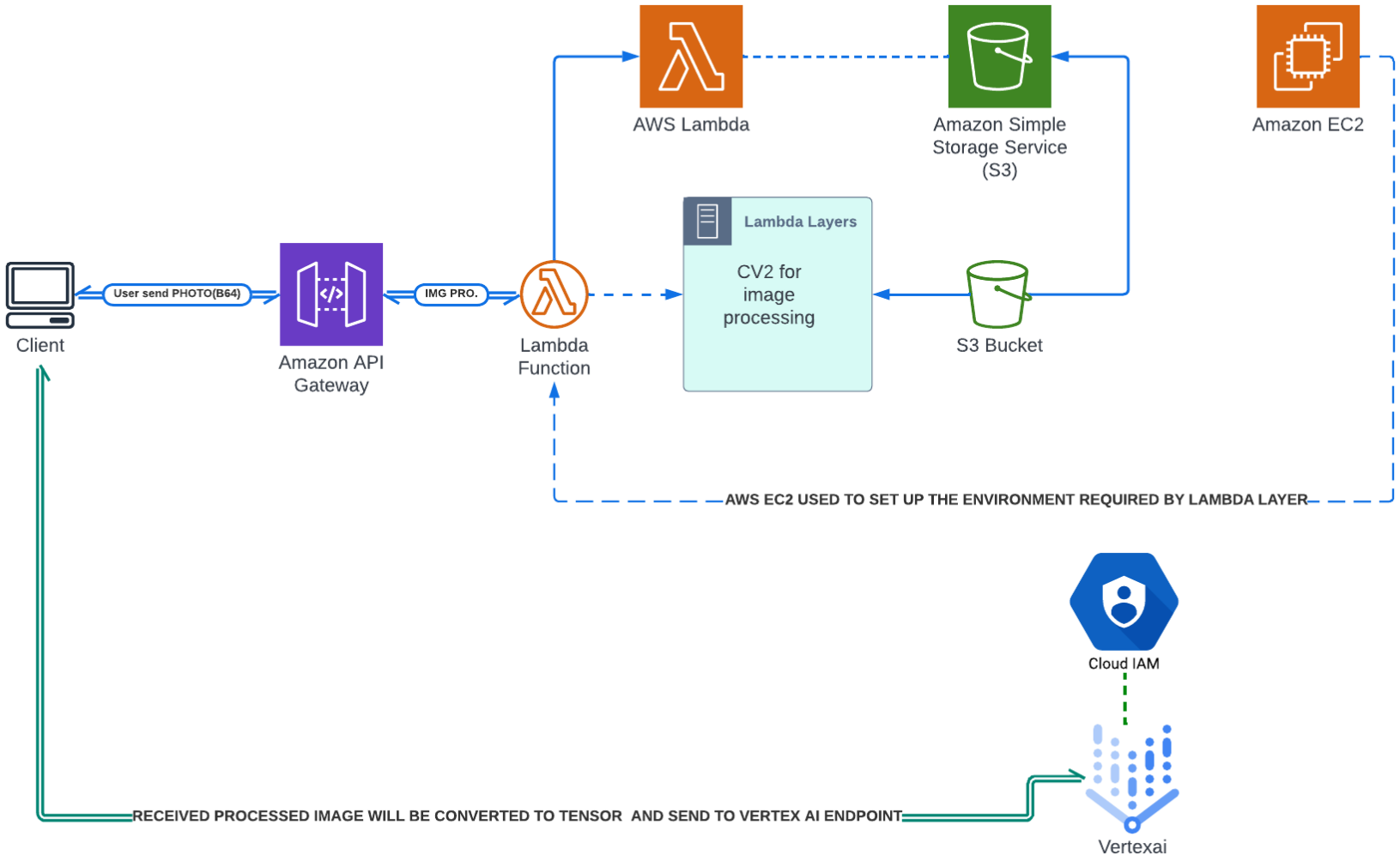
n=500;



Filter Filter

Name	Requests	Errors (%)	Latency, median (ms)	Latency, 95% (ms)
Vertex AI API	843	0	175	255
Notebooks API	29	0	397	520
Compute Engine API	5	0	305	502

4. Architecture



5. Demo and Resources

DEMO VIDEO + CODE RESOURCES LINK: https://github.com/1773-1863/SWE-590-TERM-PROJECT