

AIN SHAMS  
UNIVERSITY



# DISTRIBUTED COMPUTING REPORT

PROF. AYMAN BAHAA  
ENG. MOSTAFA ASHRAF

*document is a detailed document from A-Z for our “CSE 354: Distributed Computing” semester’s project which we will be implementing a Distributed Image Processing System using Cloud Computing*

## **This Project is Brought to You by Team 10**

<b>Mariam Wahdan Allam</b>	<b>20P5500</b>
<b>Malak Mohamed Mahfouz</b>	<b>20P7813</b>
<b>Mohamed Abdelnasser Mohamed</b>	<b>20P9501</b>
<b>Mohamed Khaled Ahmed</b>	<b>20P4277</b>

We sincerely thank Professor "Ayman Bahaa" and Engineer "Mostafa Ashraf" for their unwavering support, direction, and encouragement during this project. Their eagerness kept us motivated and on course and their timely and supportive answers to our questions were priceless. We consider ourselves quite lucky to have had mentors who are so committed, astute, and motivating.

# Table Of Contents

Table Of Contents.....	3
1. Phase 1.....	6
1.1 Scope .....	6
1.2 Project Objectives.....	6
1.3 Requirements.....	7
1.3.1 Functional Requirement.....	7
1.3.2 Non-Functional Requirement.....	7
1.4 Architecture .....	8
1.5 Technologies .....	9
1.6 Image Processing .....	9
1.7 System Components: .....	10
1.8 Responsibilities.....	11
1.9 Detailed Project Plan.....	12
1.9.1 Tasks: .....	12
1.10 System Interaction and Components.....	13
1.10.1 Component Diagram: .....	13
1.10.2 Architecture Diagram: .....	13
1.10.3 Sequence Diagram: .....	14
1.10.4 Network Diagram: .....	15
1.10.5 Gant Chart: .....	15
1.10.6 User Stories:.....	16
1.11 Cost Analysis: .....	17
2. Phase 2.....	19
2.1 How To Run .....	23

2.2 Run .....	26
2.2.1 Home Page.....	26
2.2.2 Click on upload button.....	27
2.2.3 After uploading .....	28
2.2.4 Choose Operation .....	28
2.2.5 Preview Image: We used color inversion operation.....	29
2.2.7 Preview Image: We used edge detection operation. ....	29
2.2.6 Download Image.....	30
2.3 Azure .....	31
2.3.1 : How to create a virtual machine .....	31
2.3.2 First VM (Master Node) .....	33
2.3.4 Third VM .....	35
3. Phase 3.....	36
3.1 Run.....	36
3.1.1 How to run server .....	36
3.1.2 Run Client .....	37
3.2 Code .....	41
3.2.1 Server .....	41
3.2.2 Client .....	42
4. Phase 4.....	43
4.1 End User Guide: .....	43
4.2 Testing.....	44
4.3 Scalability .....	45
4.4 Conclusion.....	58

## List Of Figures

Figure 1RPC Mechanism .....	8
Figure 2 System Component Diagram .....	13
Figure 3 System Architecture Diagram .....	13
Figure 4 System Sequence Diagram.....	14
Figure 5 System Network Diagram .....	15
Figure 6 System Time plan.....	15
Figure 7 Use Case.....	16
Figure 8 Code Normal Run without MPI 1) Touch.....	23
Figure 9 Code Normal Run without MPI 2) code.....	23
Figure 10 Code Normal Run without MPI 3) chmod.....	24
Figure 11 Normal Code Run without MPI 4) python3.....	24
Figure 12 Output From previous run .....	25
Figure 13 Running Code Using MPI For Parallel processing Making 2 instance Of Worker Thread ....	25
Figure 14 System Home Page .....	26
Figure 15 Uploading Image to System .....	27
Figure 16 Image Display inside The Main window Before preprocessing.....	28
Figure 17 Choosing Operation.....	28
Figure 18 Output of using color inversion and Displaying new Image.....	29
Figure 19 Output of using Edge Detection and Displaying new Image.....	29
Figure 20 Downloading Image after Showing the result of image processing .....	30

## 1. Phase 1

### 1.0 Introduction

The development of cloud computing technologies has completely changed the architecture and operation of distributed computing systems. Our goal in this project is to create a distributed image processing system by utilizing contemporary parallel processing frameworks like MPI and OpenCL as well as the capabilities of cloud computing. Python will be used to develop the system, and cloud-based virtual machines will be used to do distributed computing activities. The primary goal of the system is to effectively analyze pictures using parallel computing techniques, allowing for rapid and scalable image analysis. To accomplish this, the system will split up image processing jobs over several cloud virtual computers, guaranteeing optimal performance and robustness against possible hardware malfunctions. As the workload grows, the system's scalability will enable the installation of new virtual machines to meet the increasing needs.

### 1.1 Scope

The project aims to develop a distributed image processing system using cloud computing technologies. The system will leverage cloud based virtual VMs for parallel processing, implementing various image processing algorithms such as filtering, edge detection, and color manipulation.

### 1.2 Project Objectives

- Develop a distributed image processing system that can distribute tasks across multiple VMs.
- Implement various image processing algorithms for filtering, edge detection, and color manipulation.
- Ensure scalability to manage increasing workloads by adding more VMs.
- Design fault tolerance mechanisms to manage node failures and redistribute tasks accordingly.
- Provide UI for uploading images, selecting operations, monitoring task progress, and downloading processed images.

## 1.3 Requirements

### 1.3.1 Functional Requirement

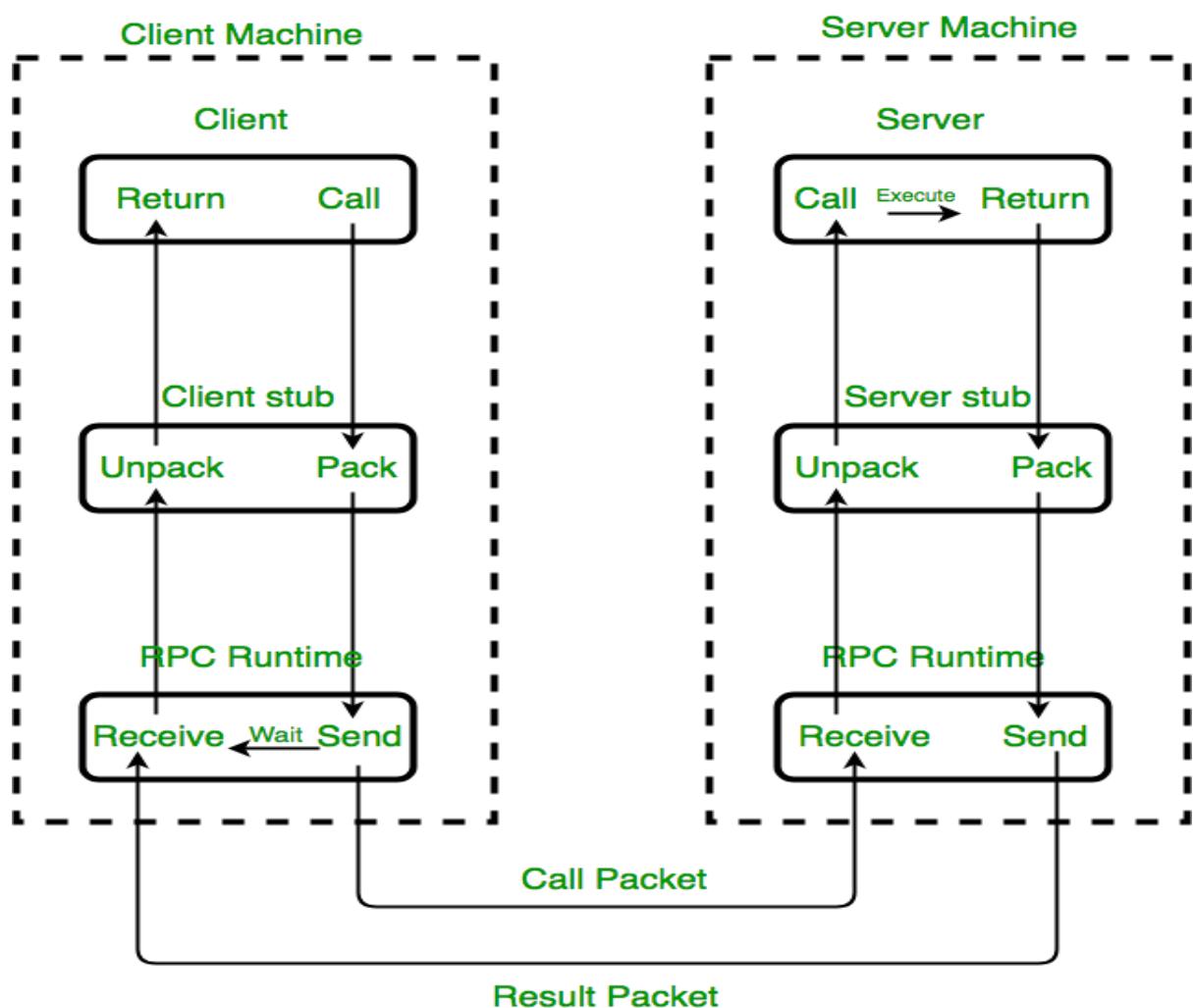
- 1 **Distributed Processing:** Develop a mechanism to distribute tasks across multiple VMs in the cloud.
- 2 **Image Processing Algorithms:** Implement algorithms for filtering, edge detection, and color manipulation using OpenCL or MPI for parallel processing.
- 3 **Scalability:** Design the system to scale by adding more VMs dynamically as workload increases.
- 4 **Fault Tolerance:** Implement procedures to manage node failures and redistribute tasks to operational nodes.
- 5 **UI:** Develop a user interface allowing users to upload images, select processing operations, monitor task progress, and download processed images.

### 1.3.2 Non-Functional Requirement

- 1 **Security:** Ensure data privacy and integrity by implementing appropriate security measures for data transmission and storage.
- 2 **Resource Management:** Manage resources efficiently such as VM instances and storage in the cloud environment.
- 3 **Performance Optimization:** Optimize the system for performance by minimizing latency and maximizing throughput during tasks.
- 4 **Documentation:** Provide comprehensive documentation including user guides, system architecture, and developer documentation.
- 5 **Testing:** Conduct thorough testing to ensure system functionality, reliability, and performance under various conditions.

## 1.4 Architecture

-  **RPC:** A key idea in distributed computing, remote procedure call (RPC) architecture has completely changed the way programs communicate with one another in networked contexts. Fundamentally, Remote Procedure Calls (RPC) make it easier to call procedures or functions on distant systems or servers by simulating local function calls inside the same system. To provide smooth communication and cooperation between dispersed software system components, this abstraction of network communication complexity is essential.



*Figure 1RPC Mechanism*

## 1.5 Technologies

- MPI: For high-performance parallel processing, especially if the processing tasks are computationally intensive.
- RPC: For the internal messaging between the VMs, facilitating the delivery of tasks and ensuring system scalability.
- Docker: To containerize microservices and manage them efficiently, especially in a cloud environment.
- Python and OpenCV: Will be used in image processing tasks.
- Cloud Platform: Microsoft Azure
- OpenCL or MPI: Using either OpenCL or MPI to enable parallel processing of image data across multiple virtual machines, increasing processing speed and efficiency.

## 1.6 Image Processing

1. Edge Detection: An essential algorithm in image processing is called edge detection, and it finds locations in an image where there are abrupt changes in brightness. These points are usually arranged into a collection of curved line segments called edges, which are essential for deciphering the composition of the image's items. The Sobel, Canny, and Laplacian algorithms are examples of common edge detection techniques. Edge detection is essential for tasks requiring the study of the forms and boundaries inside an image since it aids in object recognition, picture segmentation, and feature extraction.
2. Color manipulation is the process of modifying or manipulating an image's colors to improve its visual appeal or draw attention to certain details. Methods might vary from straightforward contrast and brightness tweaks to more intricate color space conversions (e.g., RGB to HSV). In addition to being utilized for artistic objectives in digital photography and video, color manipulation is also employed in jobs requiring precise color detection for categorization and analysis, satellite imaging feature enhancement, and illumination correction.

## **1.7 System Components:**

### **1. *User Interface (UI) Layer:***

Includes functionalities for uploading images, selecting processing operations, monitoring task progress and downloading processed images.

### **2. *System Application Layer:***

- Holds the system's fundamental logic.
- Gathers requests from the user interface layer and coordinates the image processing operations.
- Distributes work among several cloud virtual Machines.
- Puts in place fault tolerance techniques to deal with node failures.

### **3. *Image Processing Algorithms:***

- Contains implementations of various image processing algorithms such as filtering, edge detection, and color manipulation.
- Utilizes libraries like OpenCV or custom implementations using OpenCL or MPI for parallel processing.

### **4. *Cloud infrastructure layer:***

Distributed computing using cloud-based virtual machines (VMs) makes up the makes use of cloud computing platforms like Microsoft Azure.

### **5. *Communication Layer:***

Enables communication between the system's many components. makes use of message transmission systems for asynchronous communication, such as Apache Kafka or RabbitMQ.

## 1.8 Responsibilities

- Team Member 1 (*Mohamed Khaled*): System Design, User Interface Development
- Team Member 2 (Malak Mohamed) :Technology Selection, Image Processing Module Development
- Team Member 3 (Mohamed Adbelnasser) :Cloud Infrastructure Setup, Integration and Testing
- Team Member 4 (Mariam Wahdaan): Fault Tolerance Implementation, Documentation, User Acceptance Testing

## 1.9 Detailed Project Plan

### 1.9.1 Tasks:

#### 1. Requirement Analysis

- Discuss and understand the project requirements as a team.
- Brainstorm ideas and features for the image processing system.

#### 2. System Design

- Collaboratively design the system architecture and user interface layout.
- Define the functionality and features of the system based on the requirements.

#### 3. Technology Selection

- Research and discuss various technologies suitable for implementation.
- Evaluate options based on simplicity, availability of resources, and compatibility.

#### 4. User Interface Development

- Work together to create wireframes and prototypes for the user interface.
- Implement the user interface design using basic web development tools or platforms.

#### 5. Image Processing Module Development

- Collaboratively implement basic image processing functions such as filtering, cropping, and color manipulation.
- Integrate the image processing module with the user interface.

#### 6. Cloud Infrastructure Setup

- Explore and decide on using cloud storage services such as Google Drive or Dropbox for storing images.
- Set up accounts and configure cloud storage options as a team.

#### 7. Fault Tolerance Implementation

- Discuss potential failure points in the system and brainstorm simple error handling mechanisms.
- Implement basic error handling to ensure system stability.

#### 8. Integration and Testing

- Integrate different components of the system and test them together.
- Conduct informal testing among team members to identify and resolve any bugs or issues.

#### 9. Documentation

- Collaboratively create user guides or manuals explaining how to use the system.
- Document any technical details or considerations for future reference as a team.

#### 10. User Acceptance Testing

- Invite fellow students to test the system and provide feedback.
- Make any necessary adjustments based on user input.

## 1.10 System Interaction and Components

### 1.10.1 Component Diagram:

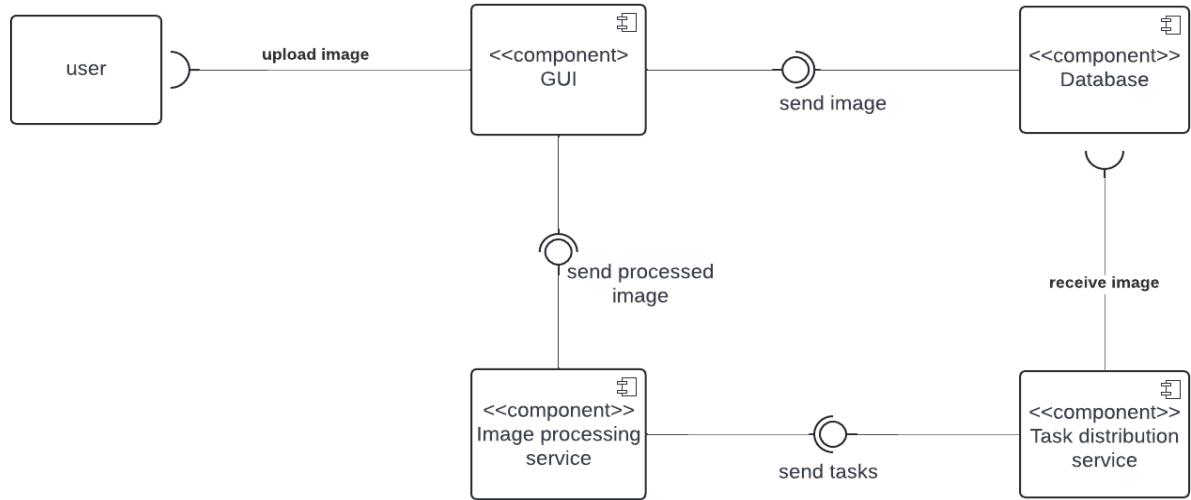


Figure 2 System Component Diagram

### 1.10.2 Architecture Diagram:

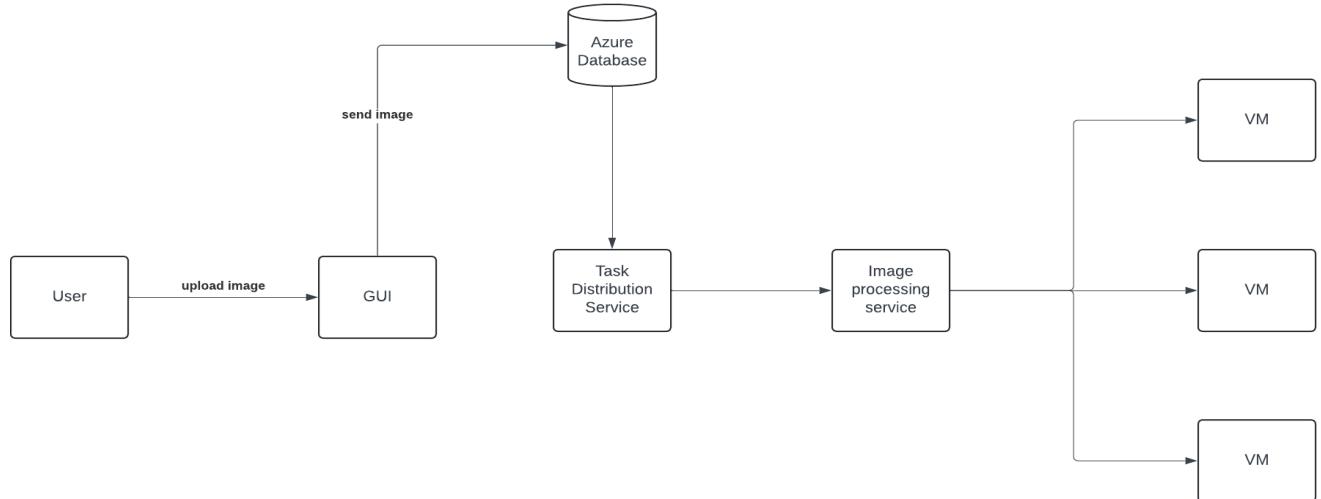


Figure 3 System Architecture Diagram

### 1.10.3 Sequence Diagram:

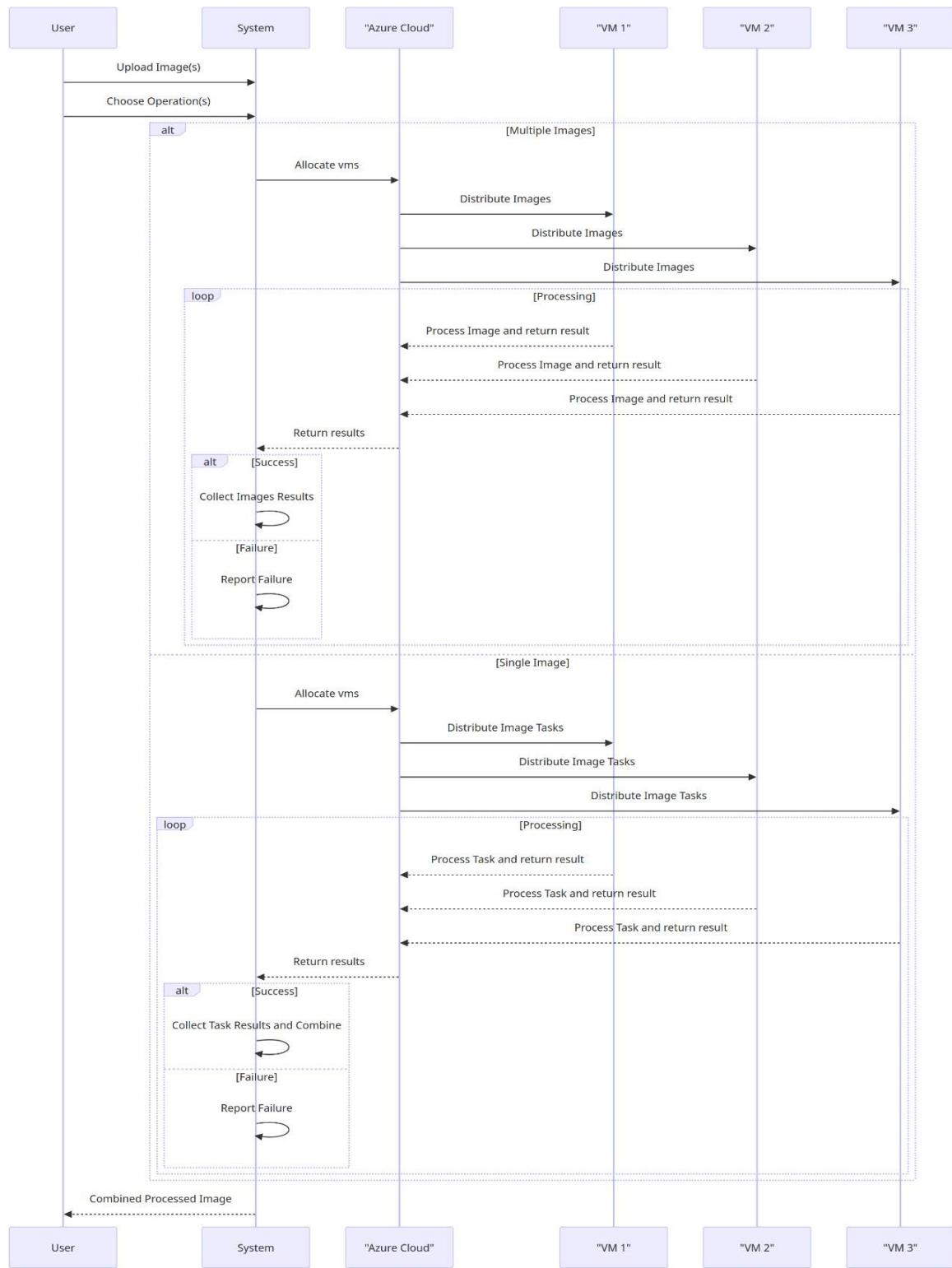


Figure 4 System Sequence Diagram

#### 1.10.4 Network Diagram:

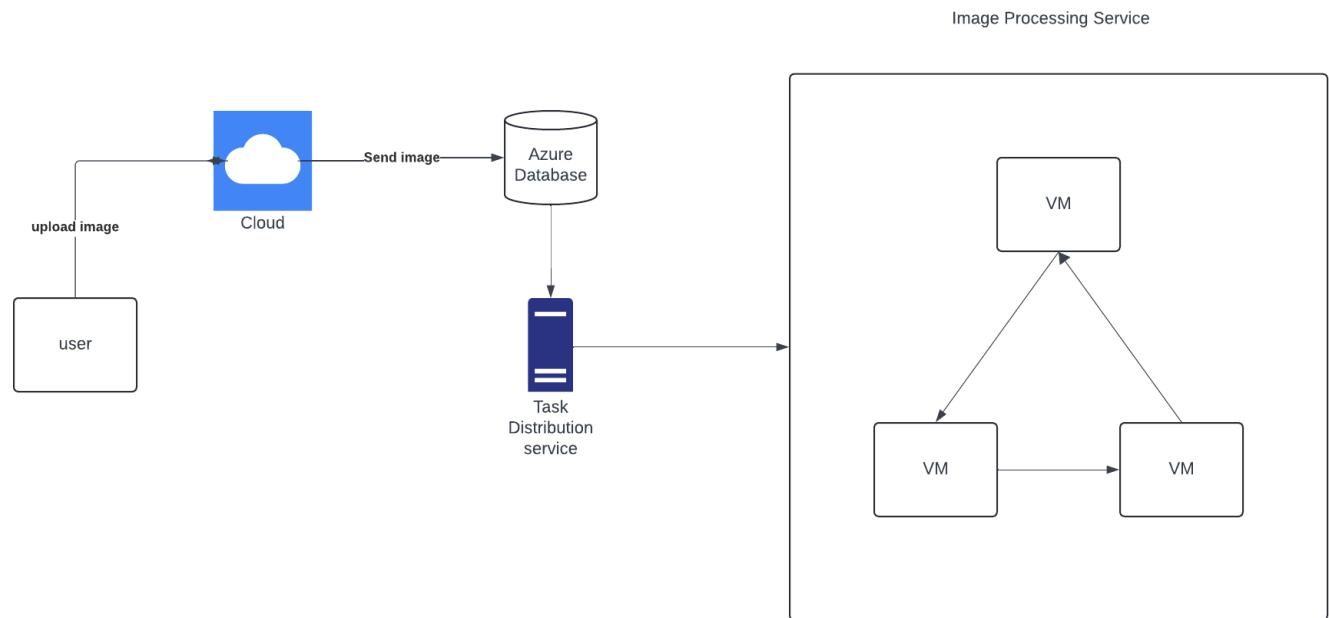


Figure 5 System Network Diagram

#### 1.10.5 Gant Chart:

ID	Name	Apr, 2024					May, 2024		
		31 Mar	07 Apr	14 Apr	21 Apr	28 Apr	05 May	12 May	19 May
1	Requirement Analysis								
2	System Architecture Design and Technology S...								
4	Cloud Infrastructure Setup								
3	GUI Development								
6	Basic Image processing								
5	Distribution of tasks								
7	Complex Image processing								
8	Integration and Testing								
9	User Acceptance Testing								
10	Deployment								
11	Documentation								

Figure 6 System Time plan

### 1.10.6 User Stories:

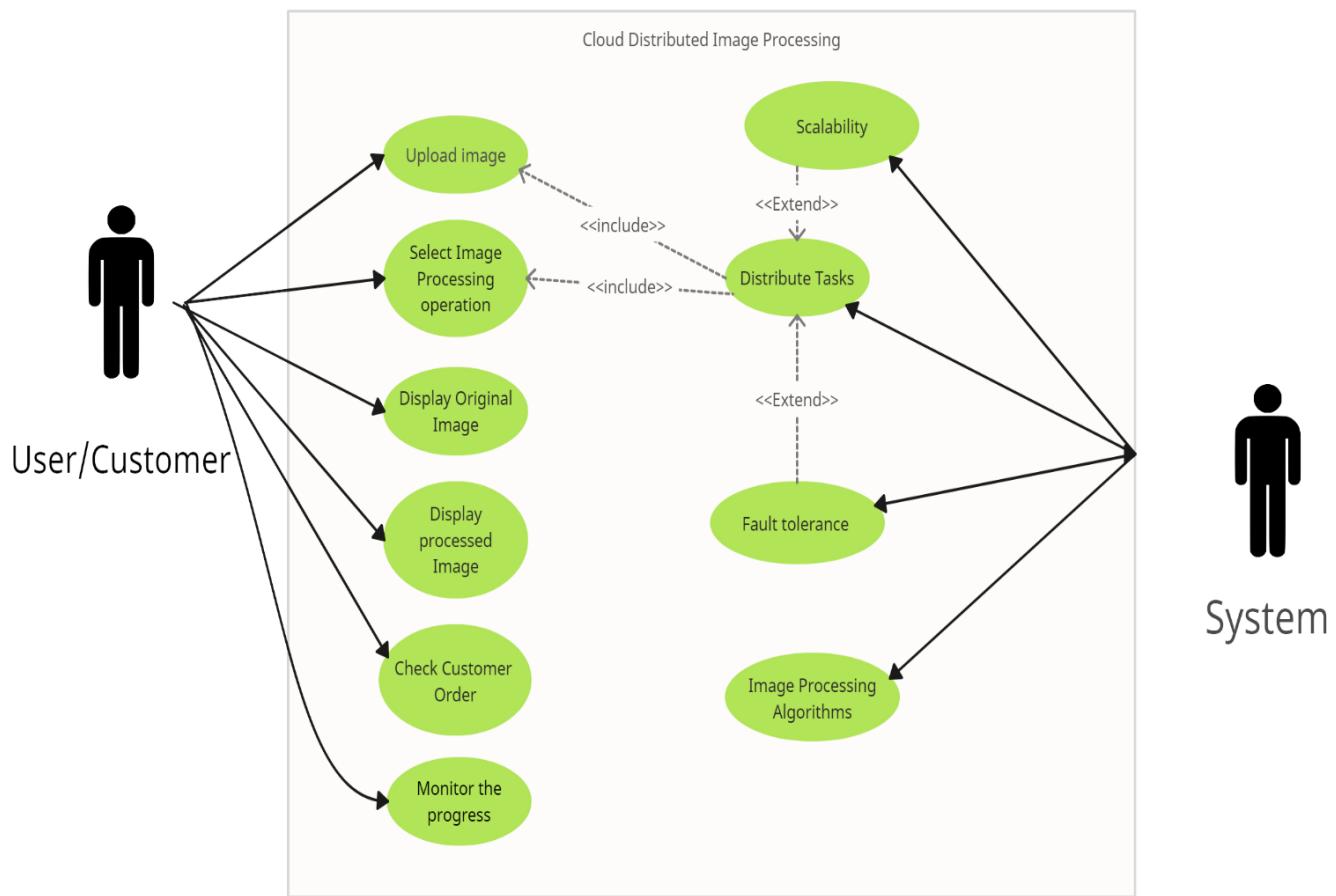


Figure 7 Use Case

## 1.11 Cost Analysis:

Cost Category	Description	Estimated Cost
Cloud Setup	Azure subscription	\$100
Virtual Machines	Renting 10 VMs for image processing (1 month)	\$2,000
Storage	100 GB storage (1 month)	\$100
Networking	Data transfer (500 GB)	\$50
Software Development	3 developers (3 months)	\$30,000
Tooling	Development tools and libraries	\$500
Maintenance	Maintenance (Semiannual)	\$600
System Maintenance	Ongoing maintenance (10% annually)	\$3,000
Cloud Service Management	Managing cloud resources (5% monthly)	\$100
User Support	2 support staff (1 year)	\$20,000
Training	Training materials and sessions	\$1,000
Contingency	Reserve for unexpected expenses (10%)	\$6,050
Licensing Fees	MPI license fee	\$500
<b>Total Estimated Cost</b>		<b>\$63,250</b>

## 1.12 Conclusion

This project aims to create an efficient and scalable distributed image processing system using cloud technologies. It employs a microservices architecture and parallel processing methods like MPI and RPC to distribute tasks across multiple VMs effectively, ensuring performance and robustness. The system will feature a user-friendly interface for easy interaction with its functionalities, such as image upload, operation selection, and downloading of processed images.

Advanced image processing algorithms will be implemented to handle tasks like filtering, edge detection, and color manipulation adeptly. Docker will be utilized to manage microservices within a cloud setting, enhancing scalability and resource management. The project plan outlines clear responsibilities and a structured timeline, ensuring a systematic approach to development.

Using Microsoft Azure as the cloud platform will provide a secure and reliable environment for processing and data storage. This project not only addresses current image processing needs but also prepares for future advancements in cloud technology.

In summary, the project is designed to deliver a comprehensive, reliable, and user-centric image processing solution, enabling users to efficiently manage various image processing tasks.

## 2. Phase 2

### 2.0 Code:

Code with GUI to upload the image and process it with one of the chosen operations, We can choose from edge detection and color inversion then download the image to the PC.

```
import tkinter as tk
from tkinter import filedialog
import cv2 # OpenCV for image processing
import threading
import queue
from PIL import Image, ImageTk # Pillow for image handling
from mpi4py import MPI # MPI for distributed computing

image_path = None
task_queue = queue.Queue()

class WorkerThread(threading.Thread):
    def __init__(self, task_queue):
        threading.Thread.__init__(self)
        self.task_queue = task_queue
        self.comm = MPI.COMM_WORLD
        self.rank = self.comm.Get_rank()
        self.processed_image = None # Attribute to store processed
image

    def run(self):
        while True:
            task = self.task_queue.get()
            if task is None:
                break
            image, operation = task
            result = self.process_image(image, operation)
            self.processed_image = result # Store the processed image
            # Notify GUI to update the displayed image
```

```

        app.update_displayed_image(result)

    def process_image(self, path, operation_var):
        img = cv2.imread(path, cv2.IMREAD_COLOR)
        if operation_var == 'edge_detection':
            result = cv2.Canny(img, 100, 200)
        elif operation_var == 'color_inversion':
            result = cv2.bitwise_not(img)

        resized_result = cv2.resize(result, (500, 500))
        return resized_result

class GUI(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("Image Processor")
        self.geometry("700x700")

        self.image_path = None

        # Load the background image
        background_img = Image.open("Background.png")
        self.background_img = ImageTk.PhotoImage(background_img)

        # Create a label to display the background image
        self.background_label = tk.Label(self,
image=self.background_img)
        self.background_label.place(x=0, y=0, relwidth=1, relheight=1)

        self.image_label = tk.Label()
        self.image_label.pack()

        self.operation_var = tk.StringVar()
        self.operation_var.set("edge_detection")

```

```

        self.operation_menu = tk.OptionMenu(self, self.operation_var,
"edge_detection", "color_inversion")
        self.operation_menu.pack(side=tk.BOTTOM)

        self.download_button = tk.Button(self, text="Download Image",
command=self.download_image, state=tk.DISABLED)
        self.download_button.pack(side=tk.BOTTOM)

        self.process_button = tk.Button(self, text="Process Image",
command=self.process_image)
        self.process_button.pack(side=tk.BOTTOM)

        self.upload_button = tk.Button(self, text="Upload Image",
command=self.upload_image)
        self.upload_button.pack(side=tk.BOTTOM)

        self.worker_thread = WorkerThread(task_queue)
        self.worker_thread.start()

def upload_image(self):
    global image_path
    file_path = filedialog.askopenfilename()
    if file_path:
        image_path = file_path
        self.process_button.config(state=tk.NORMAL)
        self.download_button.config(state=tk.NORMAL)
        # Display the uploaded image
        self.display_uploaded_image(image_path)

def process_image(self):
    global image_path
    if image_path:
        task_queue.put((image_path, self.operation_var.get())) # Enqueue the processing task

```

```

        processed_image = self.worker_thread.processed_image
        if processed_image is not None:
            self.update_displayed_image(processed_image)

    def update_displayed_image(self, processed_image):
        processed_image = cv2.cvtColor(processed_image,
cv2.COLOR_BGR2RGB)
        processed_image = cv2.resize(processed_image, (500, 500))
        img = Image.fromarray(processed_image)
        img = ImageTk.PhotoImage(img)
        self.image_label.config(image=img)
        self.image_label.image = img
    def display_uploaded_image(self, image_path):
        img = cv2.imread(image_path)
        img = cv2.resize(img, (500, 500))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        img = ImageTk.PhotoImage(image=Image.fromarray(img))
        self.image_label.config(image=img)
        self.image_label.image = img # Keep a reference to avoid
garbage colle

    def download_image(self):
        global image_path
        if image_path:
            save_path =
filedialog.asksaveasfilename(defaultextension=".png")
            if save_path:
                cv2.imwrite(save_path,
self.worker_thread.processed_image)
if __name__== "__main__":
    app = GUI()
    app.mainloop()

```

## 2.1 How To Run

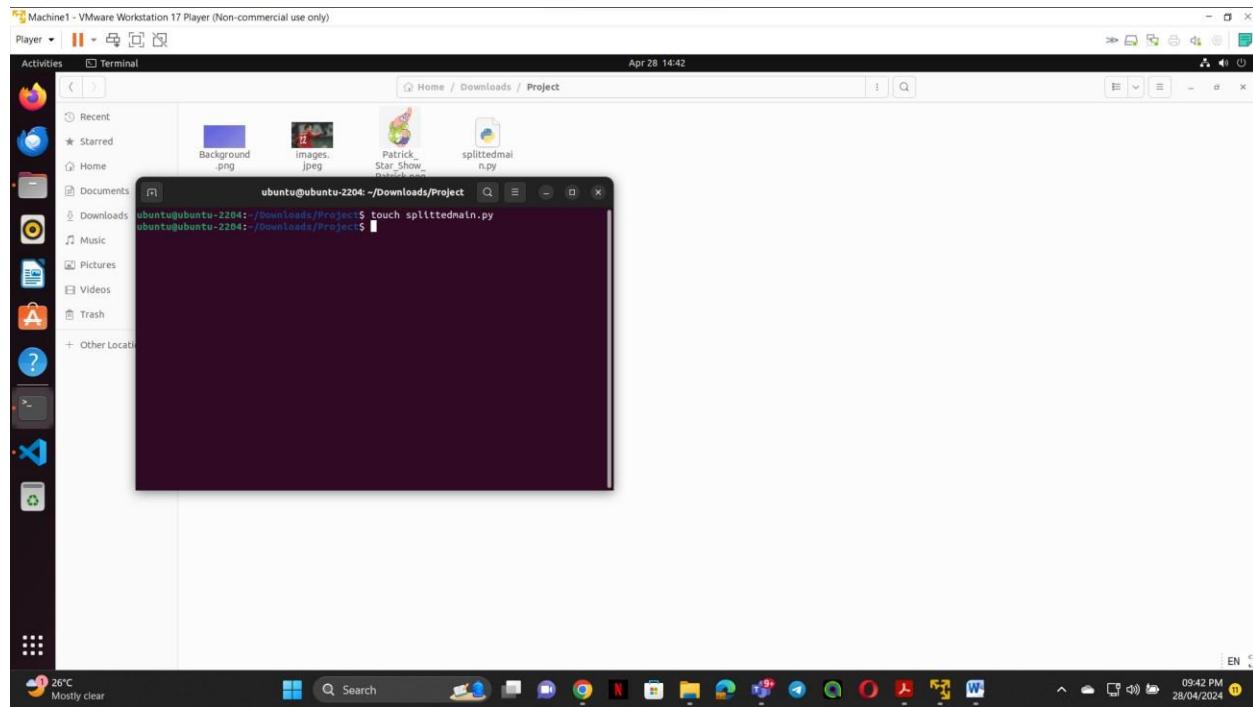


Figure 8 Code Normal Run without MPI 1) Touch

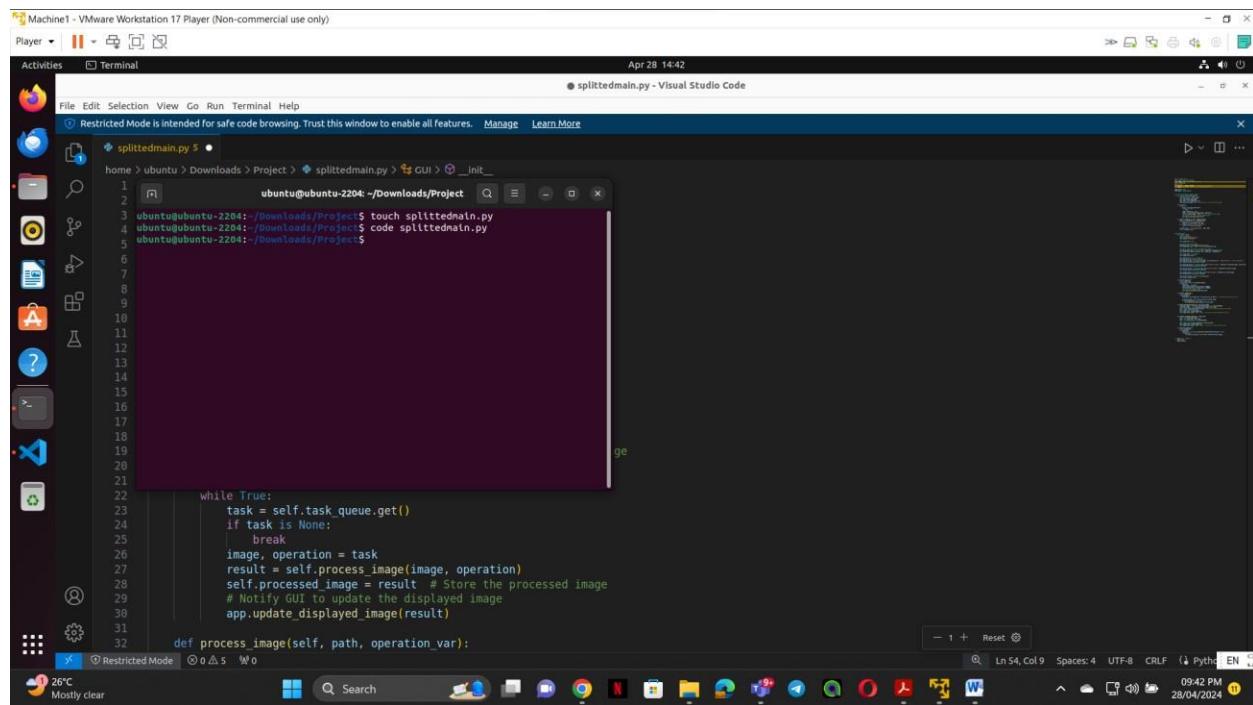


Figure 9 Code Normal Run without MPI 2) code

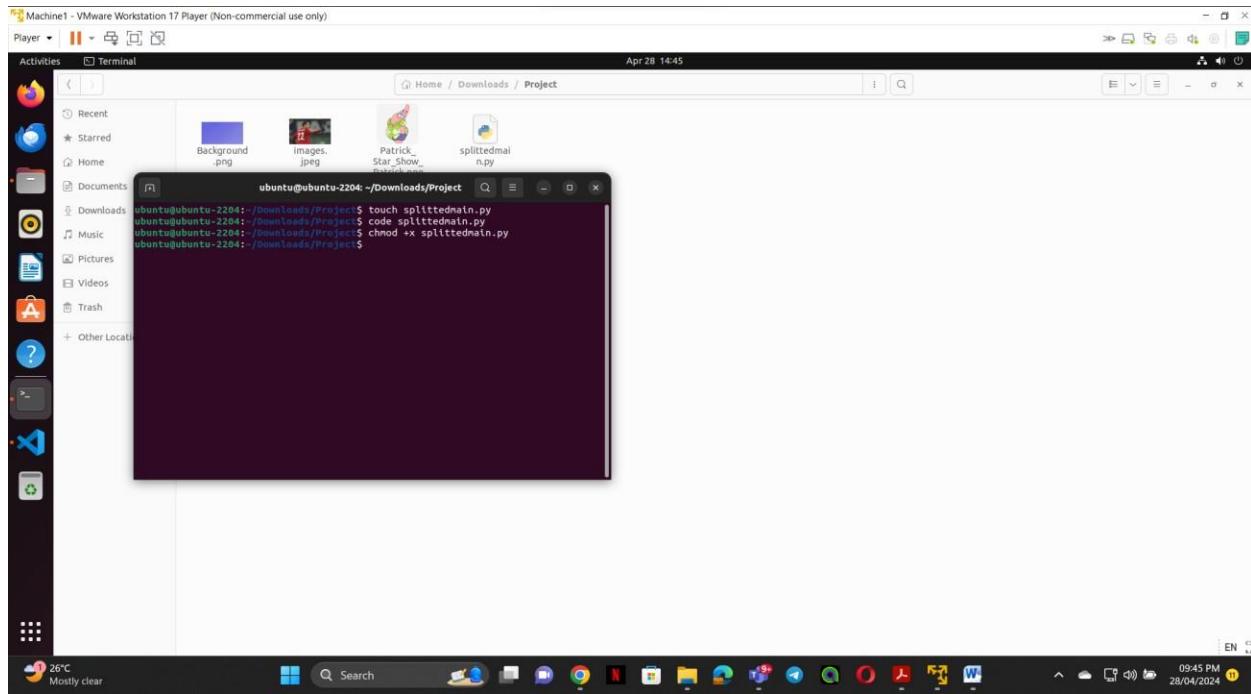


Figure 10 Code Normal Run without MPI 3) chmod

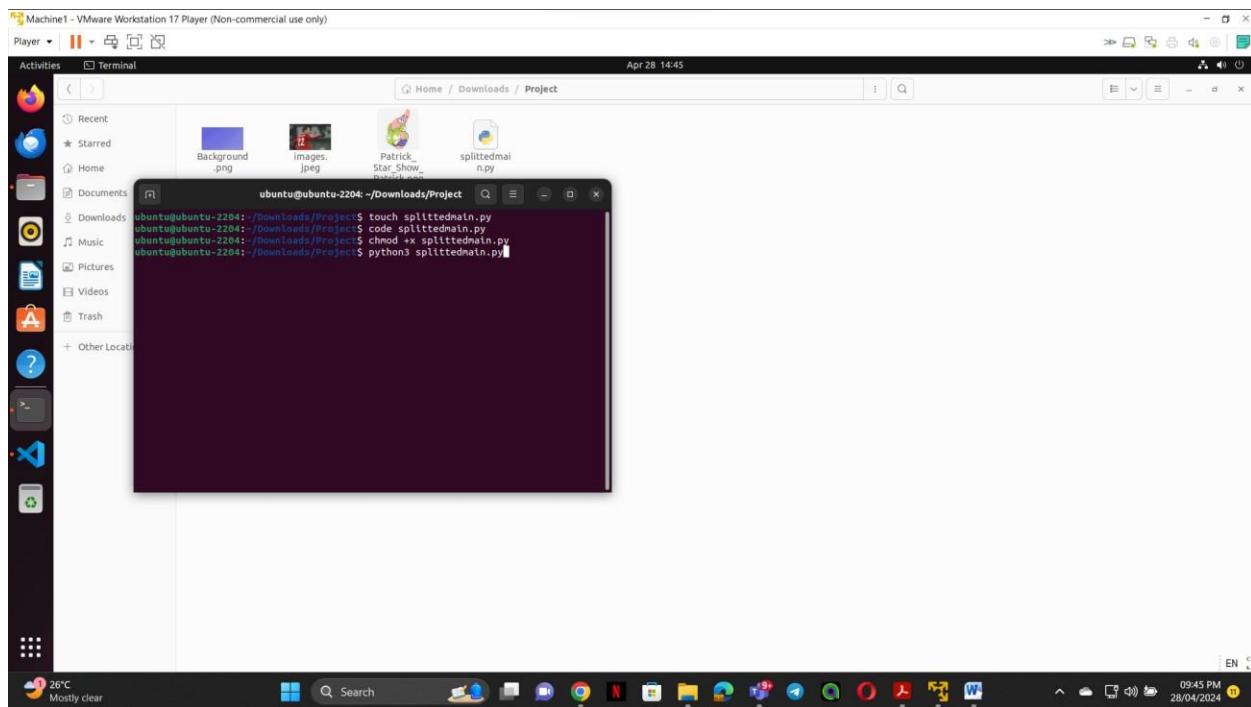


Figure 11 Normal Code Run without MPI 4) python3

```
C:\Windows\System32\cmd.exe - mpiexec -n 2 python splittedmain.py
Microsoft Windows [Version 10.0.19045.4291]
(c) Microsoft Corporation. All rights reserved.

D:\ASU\ Distributed Computing\Project>mpiexec -n 2 python splittedmain.py
```

Figure 13 Running Code Using MPI For Parallel processing Making 2 instance Of Worker Thread

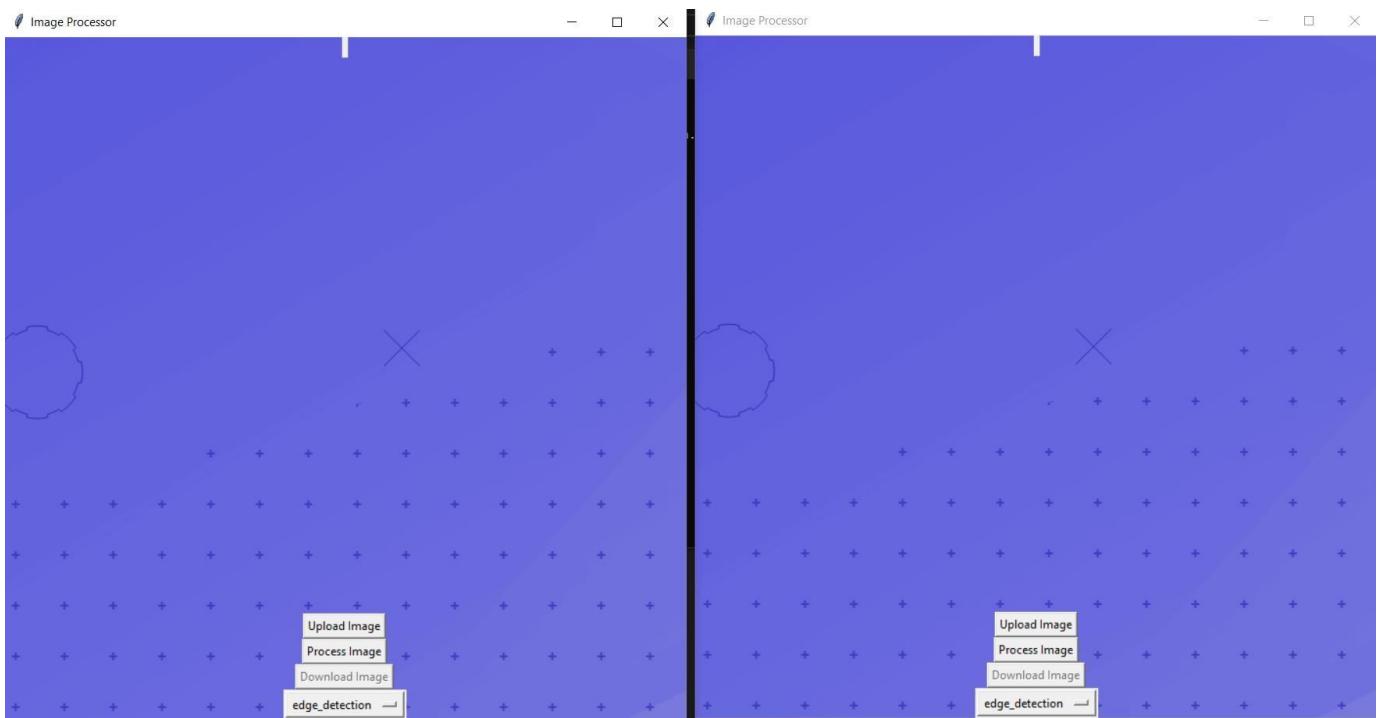


Figure 12 Output From previous run

## 2.2 Run

### 2.2.1 Home Page

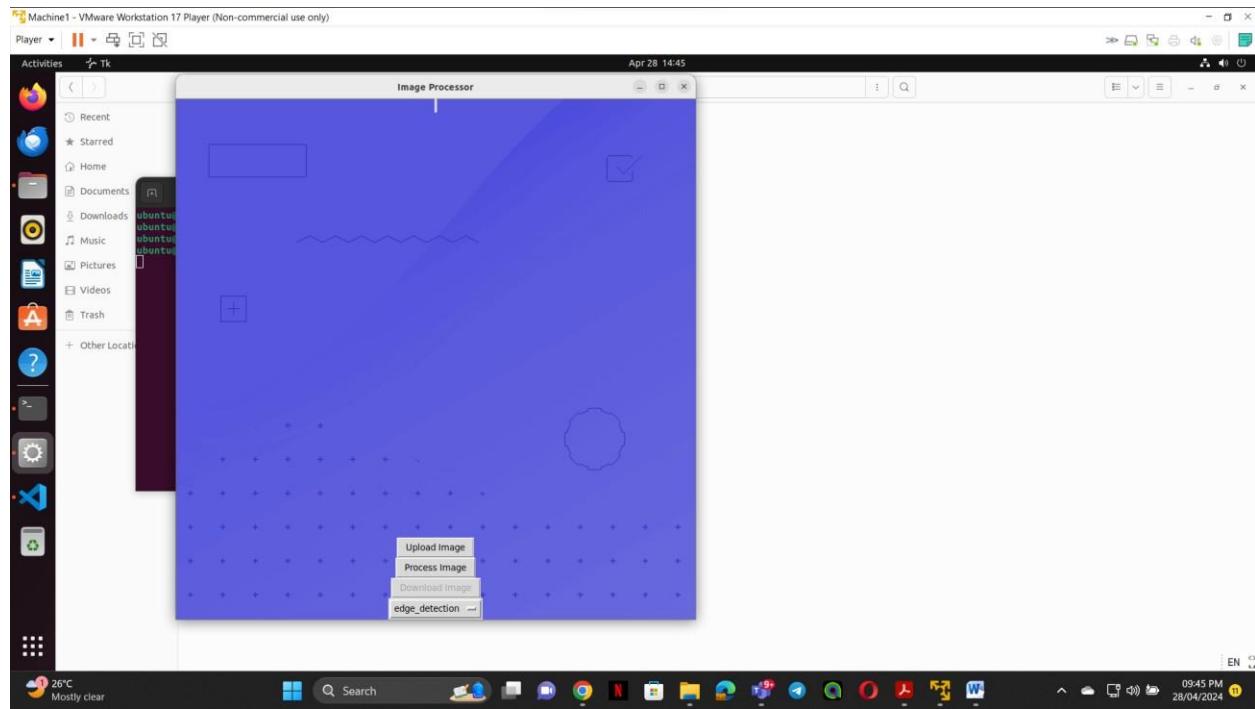


Figure 14 System Home Page

## 2.2.2 Click on upload button

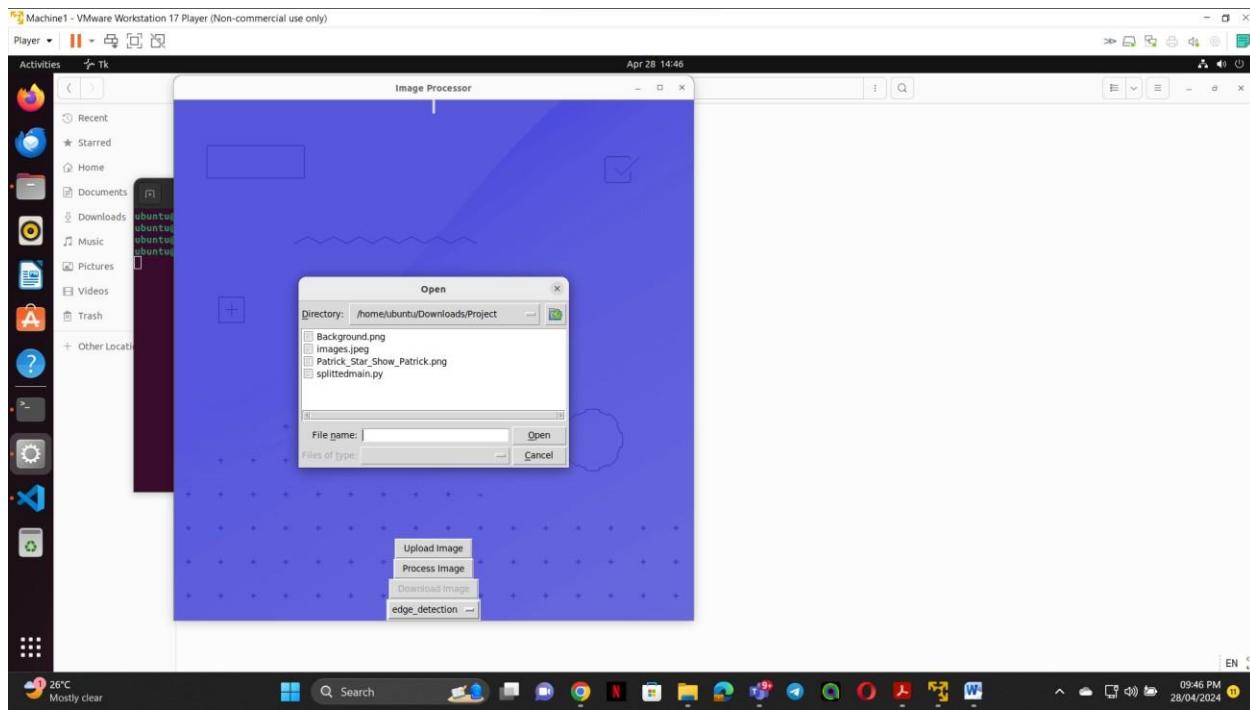


Figure 15 Uploading Image to System

### 2.2.3 After uploading

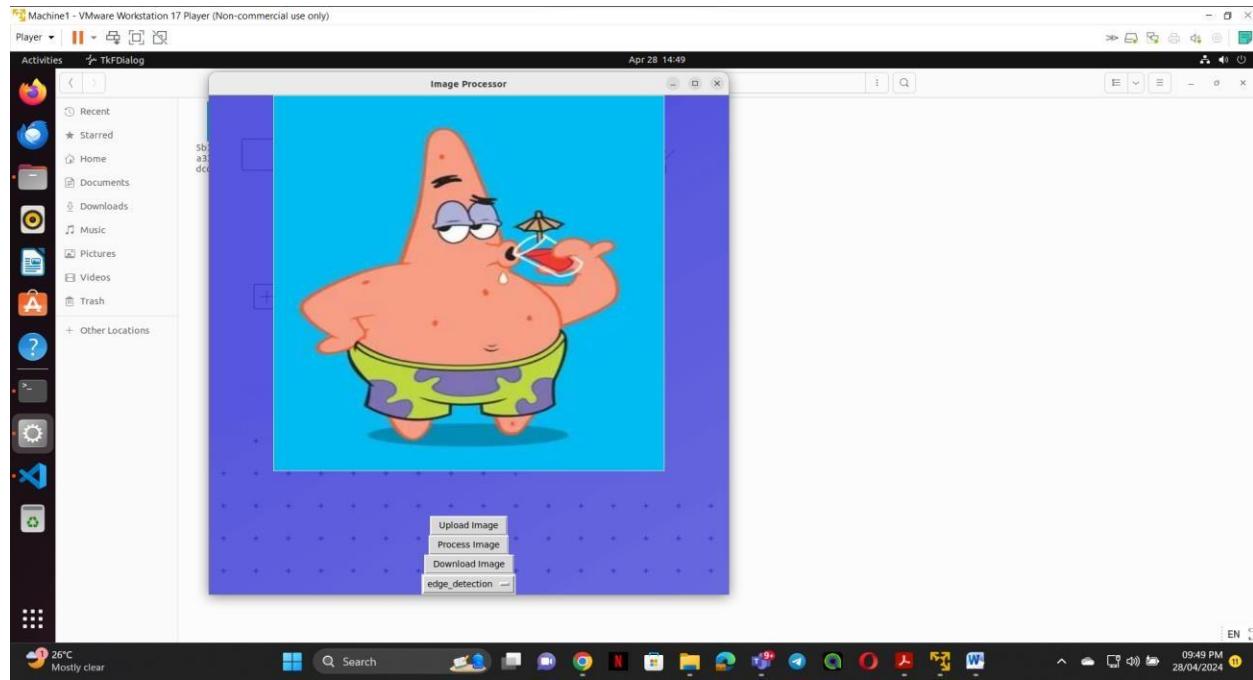


Figure 16 Image Display inside The Main window Before preprocessing

### 2.2.4 Choose Operation

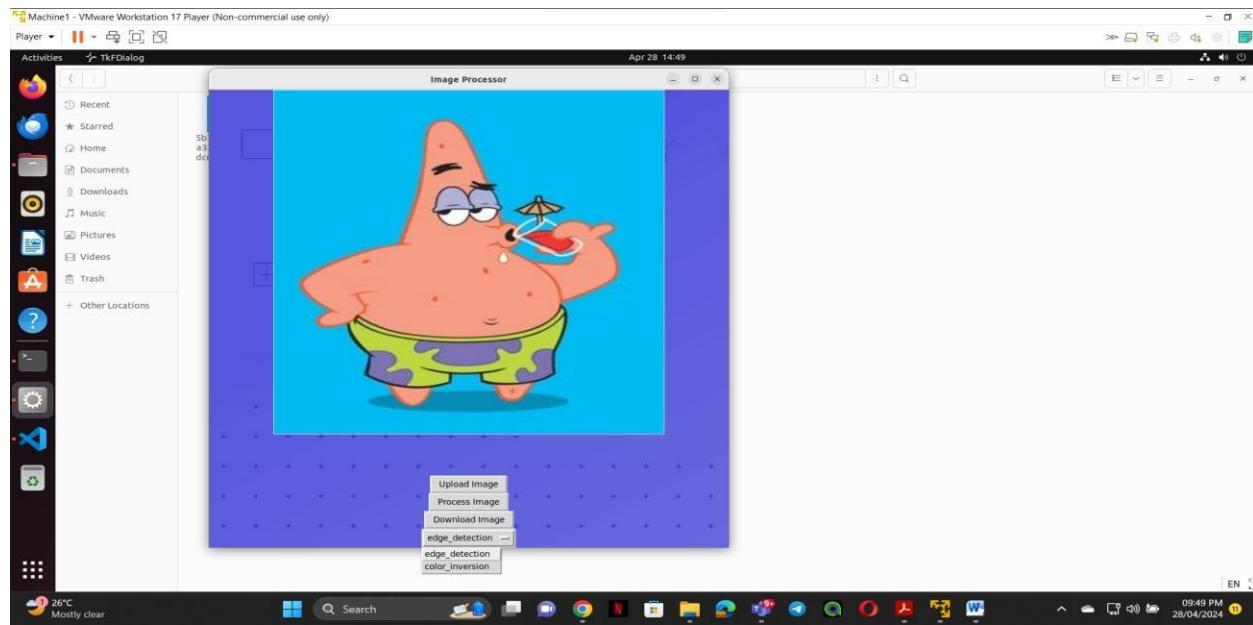


Figure 17 Choosing Operation

## 2.2.5 Preview Image: We used color inversion operation.

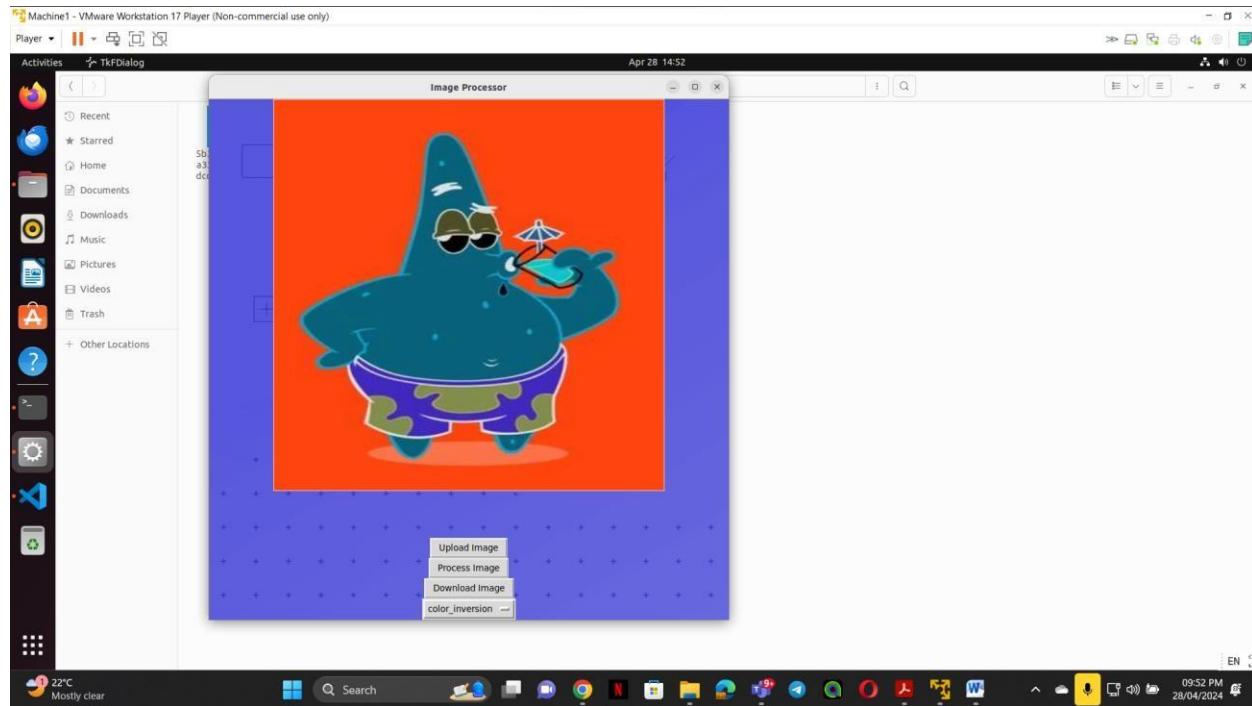


Figure 18 Output of using color inversion and Displaying new Image

## 2.2.7 Preview Image: We used edge detection operation.

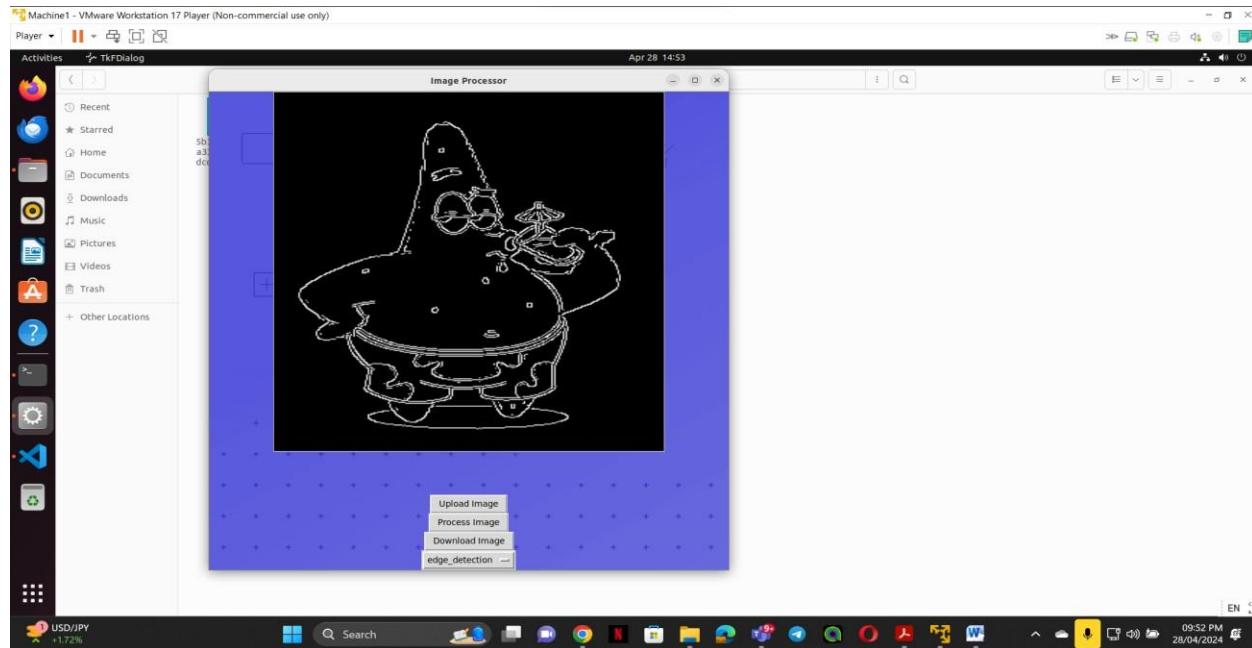


Figure 19 Output of using Edge Detection and Displaying new Image

## 2.2.6 Download Image

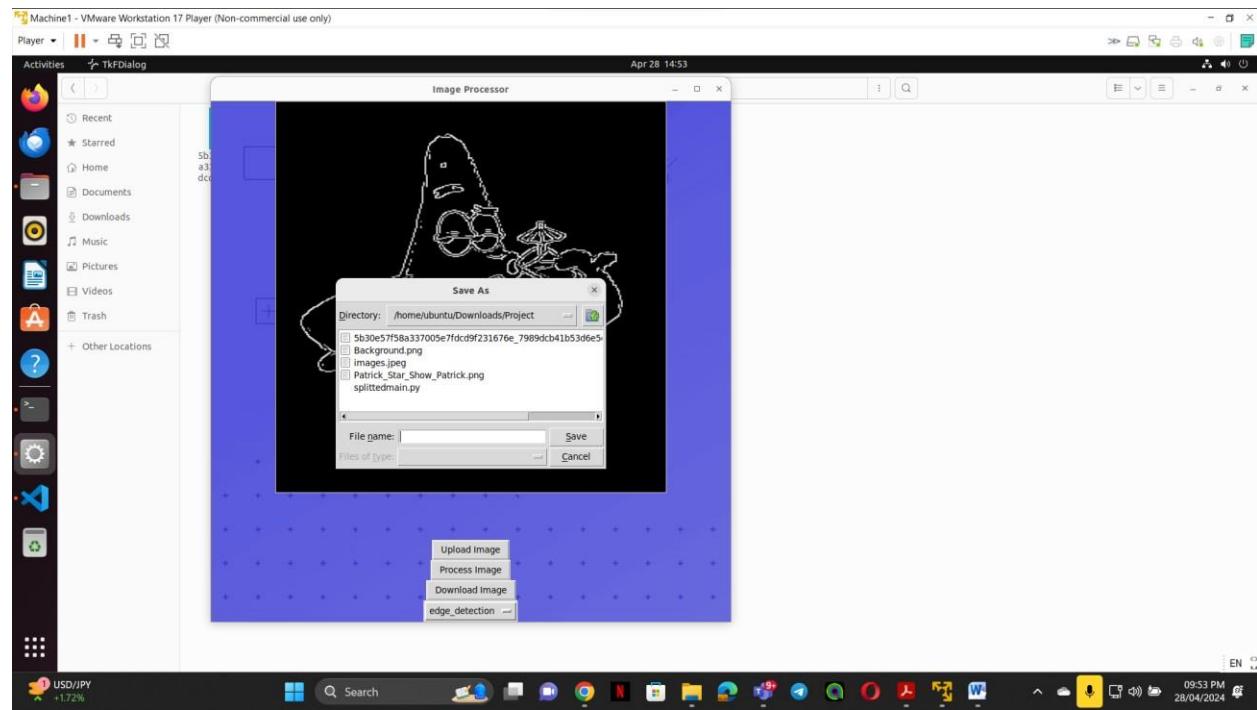


Figure 20 Downloading Image after Showing the result of image processing

## 2.3 Azure

### 2.3.1: How to create a virtual machine

This screenshot shows the Microsoft Azure portal interface for creating a new virtual machine. The process is divided into several steps:

- Project details:** Selects "Azure for Students" as the subscription and "VM\_Nasser" as the resource group.
- Instance details:** Sets the virtual machine name to "VM-3", region to "(Africa) South Africa North", and availability zone to "Zone 1". A note indicates that selecting multiple zones will create one VM per zone.
- Security type:** Set to "Trusted launch virtual machines".
- Image:** Set to "Ubuntu Server 22.04 LTS - x64 Gen2".
- VM architecture:** Set to "x64".
- Size:** Set to "Standard\_B1ms - 1 vcpu, 2 GiB memory (\$19.78/month)".
- Administrator account:** Authentication type is set to "Password".

The browser taskbar at the bottom shows the system date and time as 10:07 PM on 28/04/2024.

**Create a virtual machine - Microsoft Azure**

portal.azure.com/#create/Microsoft.VirtualMachine-ARM

Microsoft Azure

Home > Create a resource >

### Create a virtual machine

Encryption at host is not registered for the selected subscription. [Learn more about enabling this feature](#)

**OS disk**

OS disk size: Image default (30 GiB)

OS disk type: Standard HDD (locally-redundant storage) The selected VM size supports premium disks. We recommend Premium SSD for high IOPS workloads. Virtual machines with Premium SSD disks qualify for the 99.9% connectivity SLA.

Delete with VM:

Key management: Platform-managed key

Enable Ultra Disk compatibility:

**Data disks for VM-3**

You can add and configure additional data disks for your virtual machine or attach existing disks. This VM also comes with a temporary disk.

LUN	Name	Size (GiB)	Disk type	Host caching	Delete with VM
0	Temporary disk	30	Standard	None	<input type="checkbox"/>

< Previous | Next : Networking > | Review + create | Give feedback

22°C Mostly clear

Search

Microsoft Azure

Home > Create a resource >

### Create a virtual machine

Basics Disks **Networking** Management Monitoring Advanced Tags Review + create

Define network connectivity for your virtual machine by configuring network interface card (NIC) settings. You can control ports, inbound and outbound connectivity with security group rules, or place behind an existing load balancing solution. [Learn more](#)

**Network interface**

When creating a virtual machine, a network interface will be created for you.

Virtual network: VM1-vnet Create new

Subnet: default (10.1.0.0/24) Manage subnet configuration

Public IP: None Create new

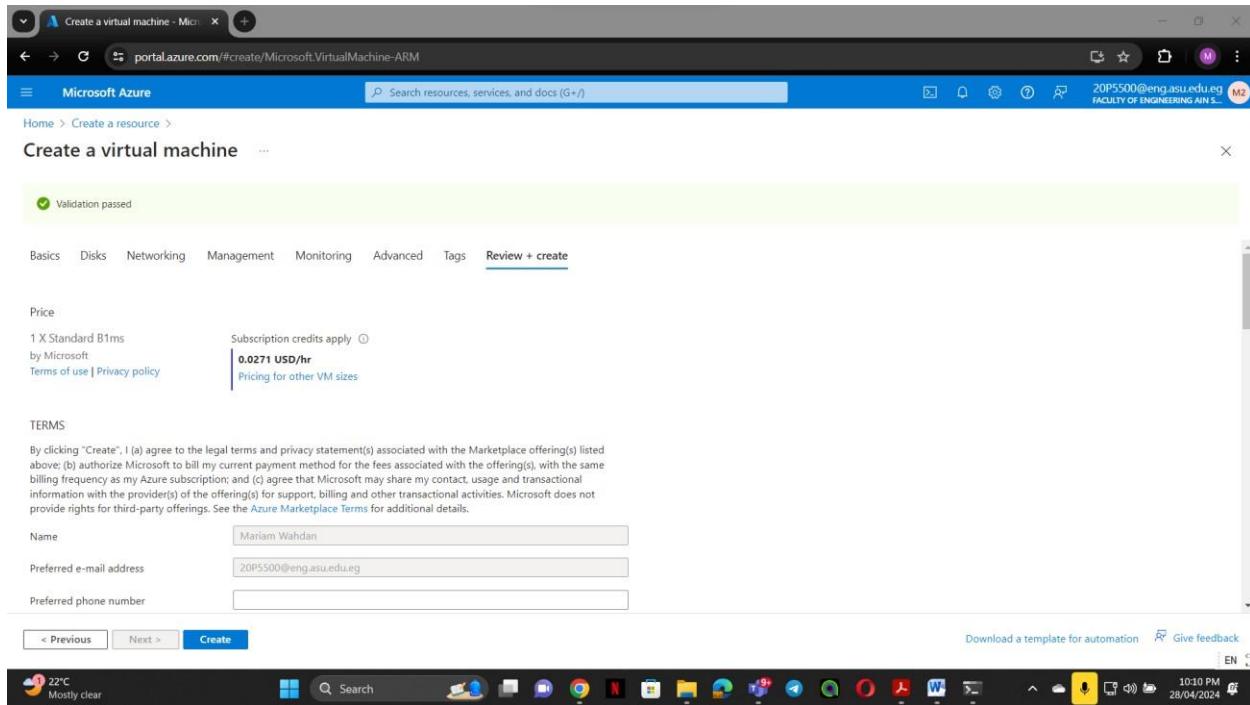
NIC network security group: Basic

Public inbound ports: Allow selected ports

< Previous | Next : Management > | Review + create | Give feedback

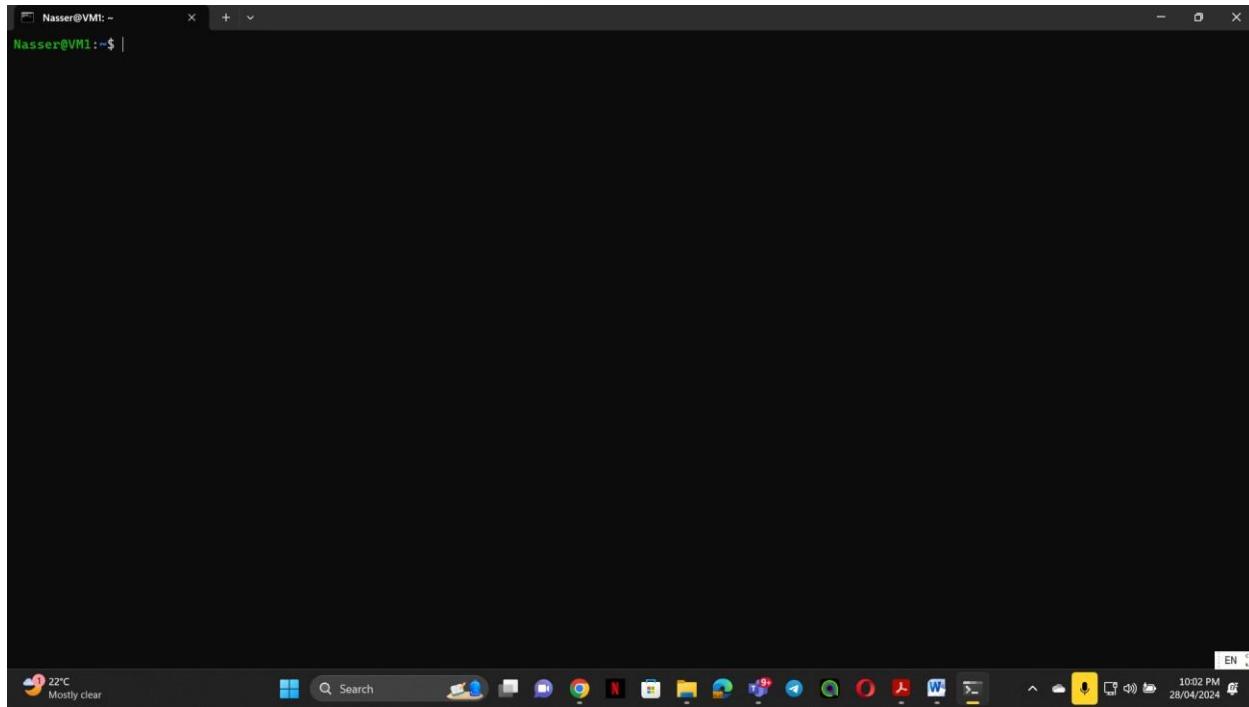
22°C Mostly clear

Search



### 2.3.2 First VM (Master Node)

Essentials	Properties	Networking
Resource group (move) : VM_Nasser	Computer name : VM1	Public IP address : 102.37.145.180 ( Network interface vm1828_z1 )
Status : Stopped (deallocated)	Operating system : Linux	Public IP address (IPv6) : -
Location : South Africa North (Zone 1)	Size : Standard B1ms (1 vcpu, 2 GiB memory)	Private IP address : 10.1.0.4
Subscription (move) : Azure for Students	Virtual network/subnet : VM1-vnet/default	Private IP address (IPv6) : -
Subscription ID : 99ad86b7-8740-4ab2-acb5-4aa13b92f3b3	DNS name : Not configured	Virtual network/subnet : VM1-vnet/default
Availability zone : 1	Health state : -	DNS name : Configure
Tags (edit) : Add tags		



### 2.3.3 Second VM

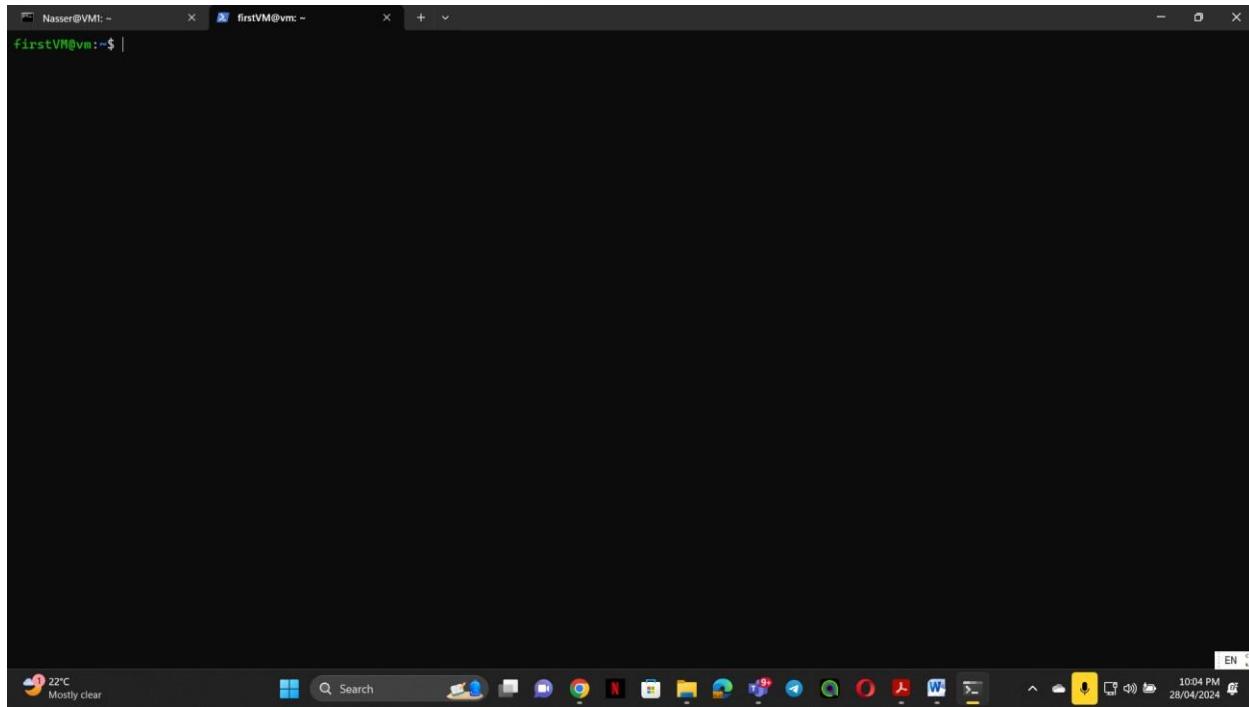
A screenshot of the Microsoft Azure portal. The URL in the address bar is 'portal.azure.com/#@eng.asu.edu.eg/resource/subscriptions/99ad86b7-8740-4ab2-acb5-4aa13b92f3b3/resourceGroups/VM\_Nasser/providers/Microsoft.Compute/virtualMachines/vm/overview'. The main content area shows the 'vm - Microsoft Azure' page for the virtual machine 'vm'.

**Essentials**

Resource group (move)	: VM_Nasser
Status	: Running
Location	: South Africa North (Zone 1)
Subscription (move)	: Azure for Students
Subscription ID	: 99ad86b7-8740-4ab2-acb5-4aa13b92f3b3
Availability zone	: 1
Tags (edit)	: Add tags

**Properties**

Virtual machine	Networking
Computer name	vm
Operating system	Linux (ubuntu 22.04)
VM generation	V2
VM architecture	x64
Agent status	Ready
Agent version	2.10.0.8
Hibernation	Disabled
Public IP address	10.23.138.73 ( Network interface vm934_z1 )
Private IP address (IPv6)	-
Private IP address (IPv6)	10.1.0.5
Virtual network/subnet	VM1-vnet/default
DNS name	Configure



### 2.3.4 Third VM

### 3. Phase 3

#### 3.1 Run

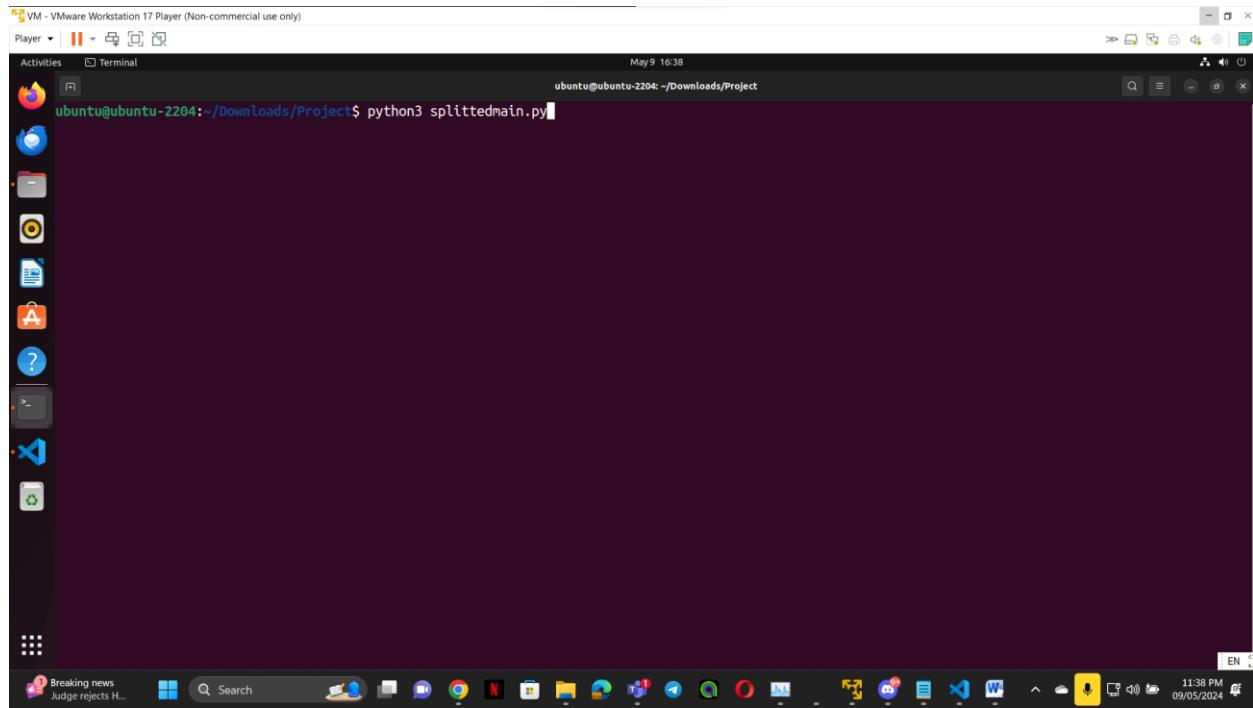
##### 3.1.1 How to run server

```
Nasser@VM1:~$ python3 server.py
```

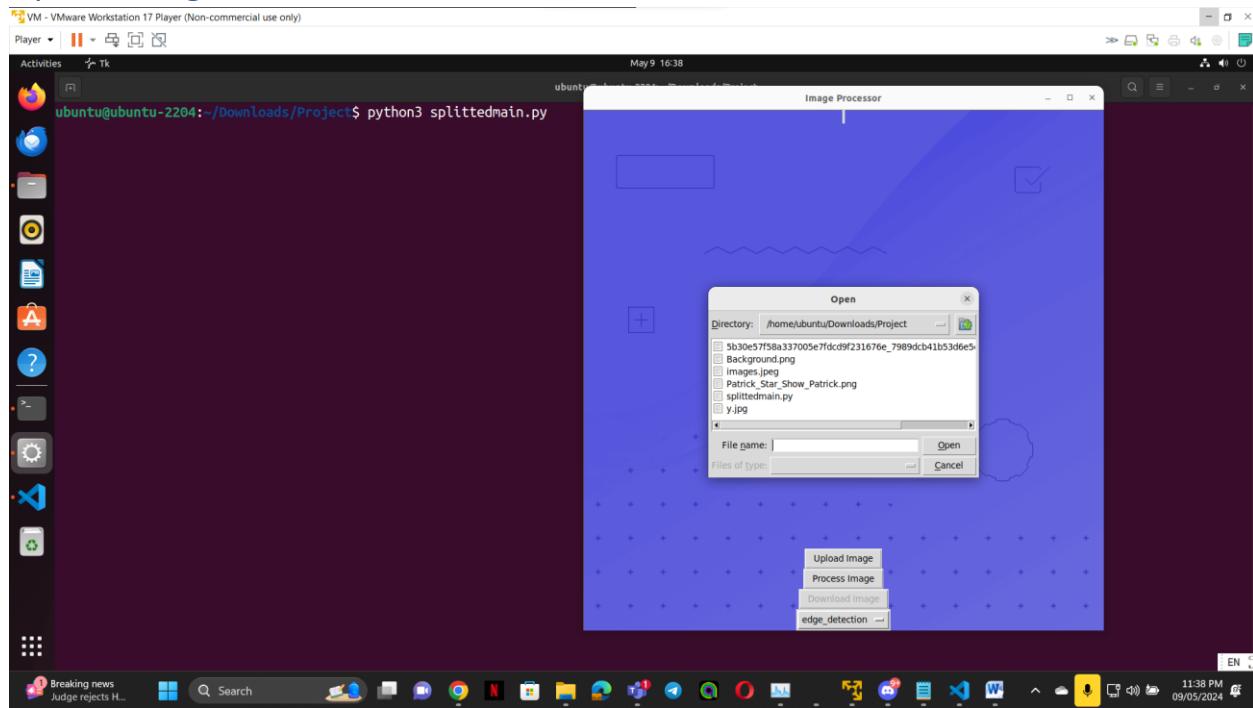
```
Nasser@VM1:~$ python3 server.py
Waiting for a connection...
[+] Trying to bind to :: on port 5000: Done
[!] Waiting for connections on :::5000
```

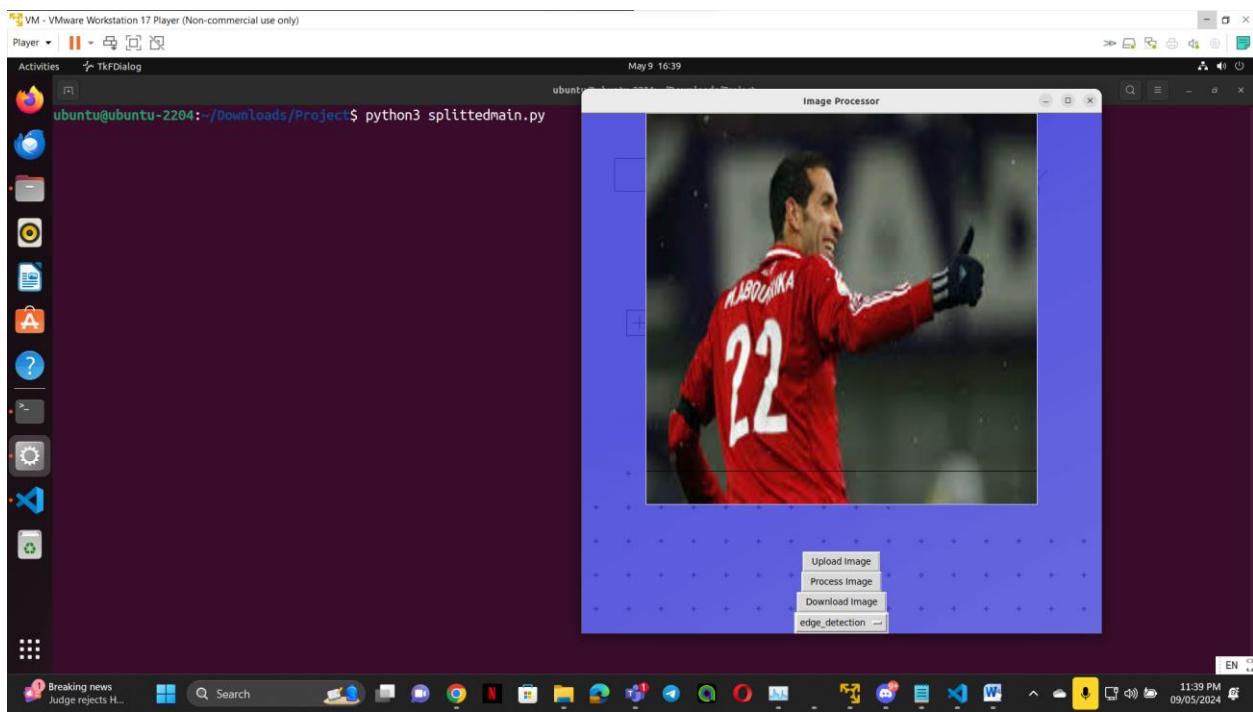
```
Nasser@VM1:~$ python3 server.py
Waiting for a connection...
[+] Trying to bind to :: on port 5000: Done
[+] Waiting for connections on :::5000: Got connection from ::ffff:41.44.91.
79 on port 36900
51
Premature end of JPEG file
Image received
Image edited
Image Sent
[*] Closed connection to ::ffff:41.44.91.79 port 36900
Nasser@VM1:~$ |
```

### 3.1.2 Run Client

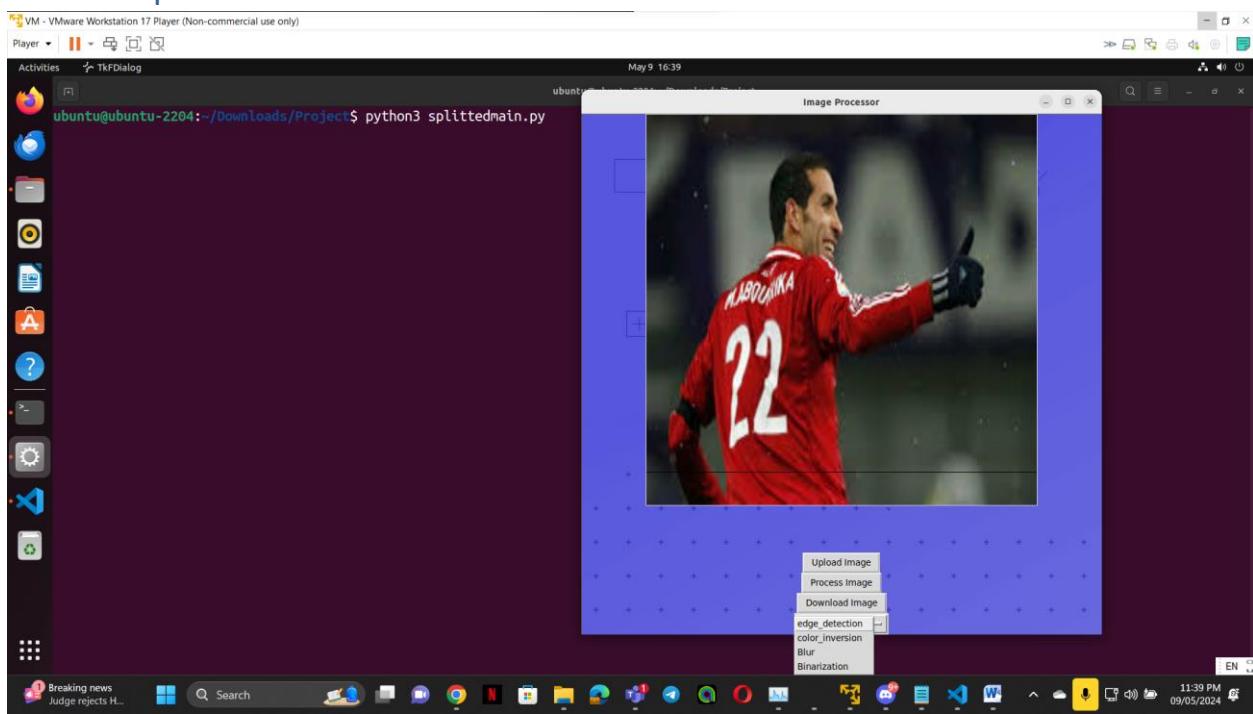


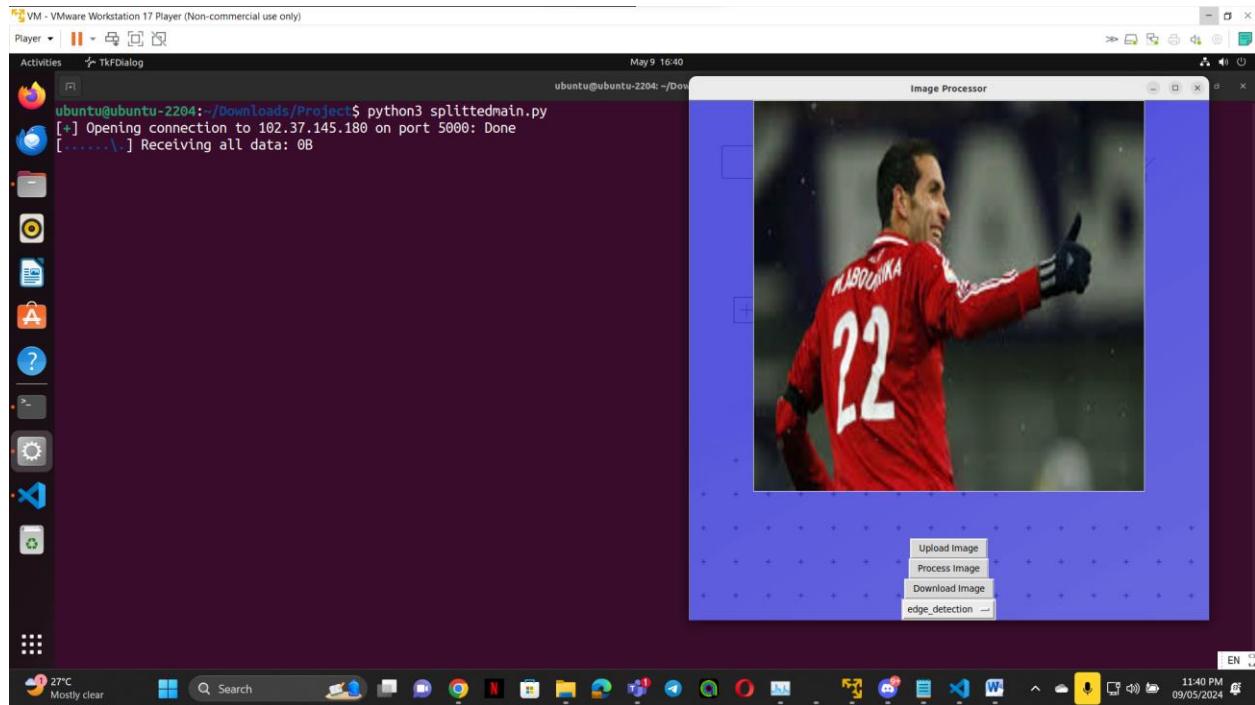
### Upload Image



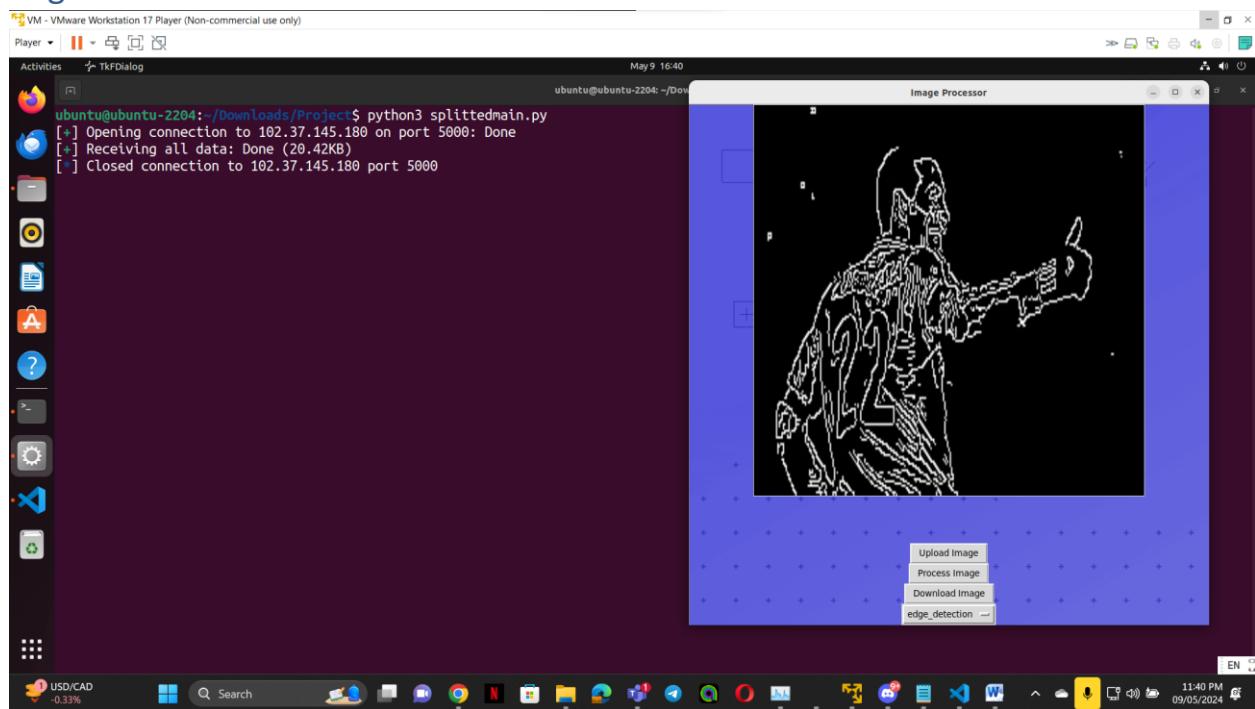


### Choose Operation:

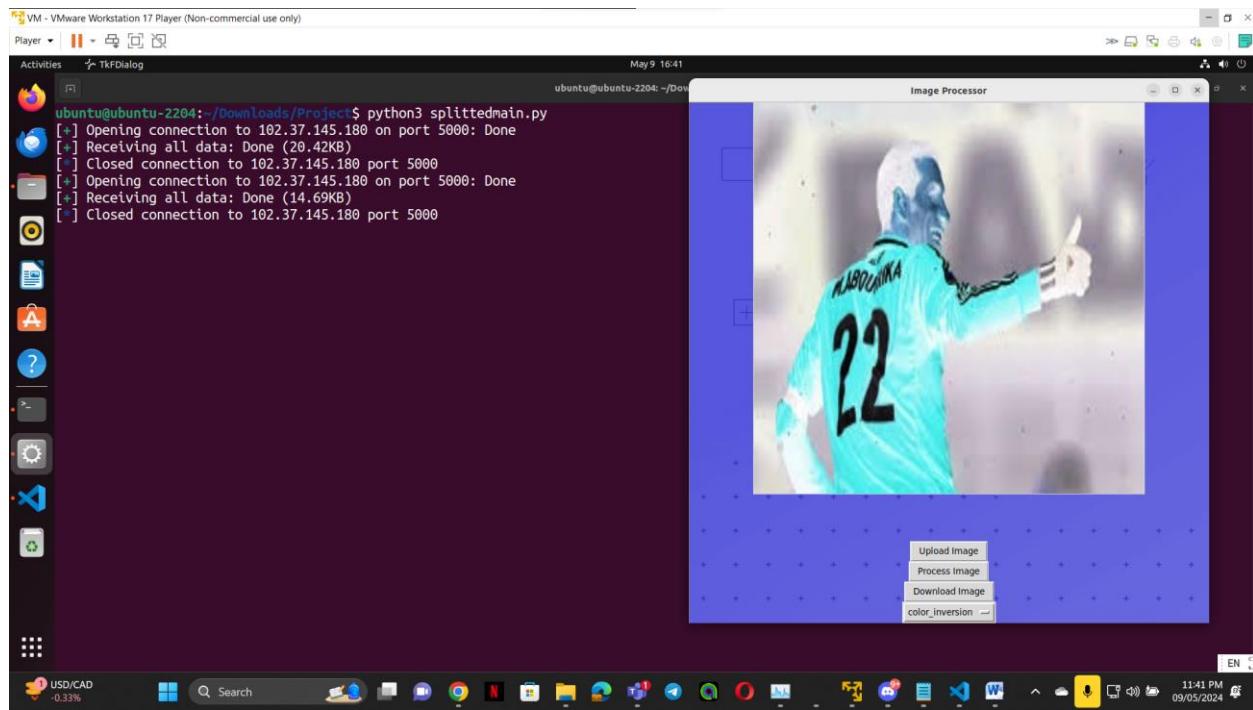




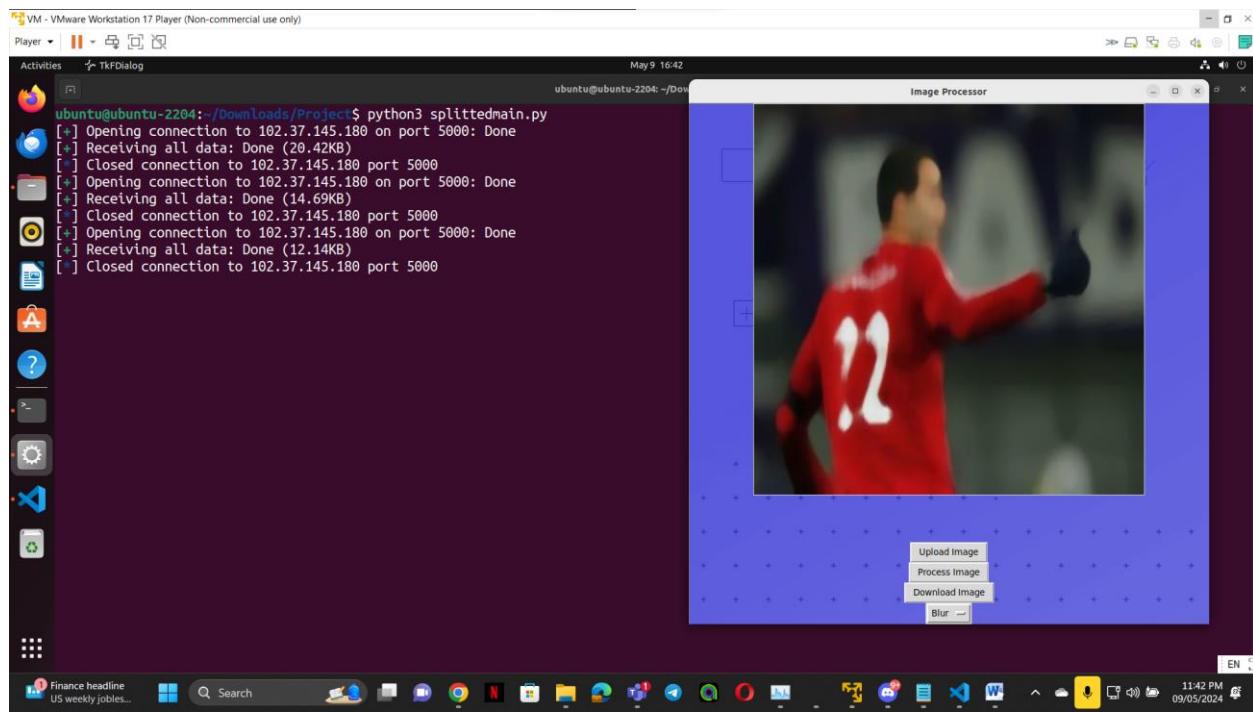
## Edge detection



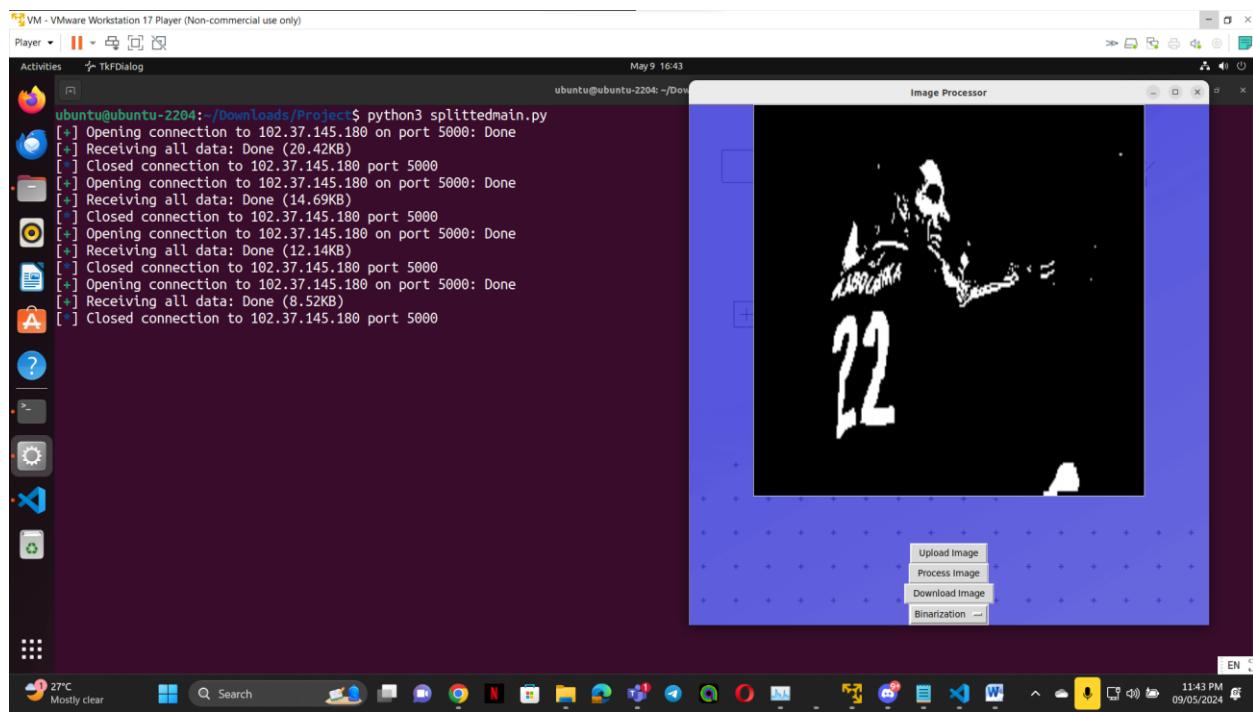
## Color Inversion



## Blur



## Binarization



## 3.2 Code

### 3.2.1 Server

```
Nasser@VM: ~          +  server.py
GNU nano 6.2
import cv2
import numpy as np
from pwn import *

print("Waiting for a connection...")
l = listen(5000)
recvfd = l.clean(10)

file = open('image.jpg','wb')
file.write(recvfd[0:len(recvfd)-2])
operation = recvfd[len(recvfd)-1]
print(operation)

img = cv2.imread('image.jpg')
print("Image received")

# 49 is edge detection
# 50 is color inversion
# 51 is BLUR
# 52 is Binarization

if operation == 49:
    img = cv2.Canny(img,100,200)
elif operation == 50:
    img = cv2.bitwise_not(img)
elif operation == 51:
    img = cv2.medianBlur(img,11)
elif operation == 52:
# Apply thresholding
    img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

```

Nasser@VMT: ~          + -
GNU nano 6.2               server.py

print("Image received")

# 49 is edge detection
# 50 is color inversion
# 51 is BLUR
# 52 is Binarization

if operation == 49:
    img = cv2.Canny(img,100,200)

elif operation == 50:
    img = cv2.bitwise_not(img)

elif operation == 51:
    img = cv2.medianBlur(img,11)

elif operation == 52:
    # Apply thresholding
    img = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    _,img = cv2.threshold(img, 128, 255, cv2.THRESH_BINARY)
else:
    print("Invalid Operation")

print("Image edited")
# Save the thresholded image
cv2.imwrite('Result_image.jpg', img)

file2 = open('Result_image.jpg', 'rb')
image2 = file2.read()
l.send(image2)

print("Image Sent")
|
```

The terminal window shows a Python script named `server.py`. The script performs image processing operations based on user input (operations 49, 50, 51, or 52) and saves the result as `Result_image.jpg`. It then reads this image from a file and sends it via a socket connection to a client. The terminal interface includes standard Linux keyboard shortcuts and a taskbar at the bottom.

### 3.2.2 Client

The screenshot shows a Visual Studio Code (VS Code) interface running in a VMware Player window. The main editor tab is `client.py`, which contains the following code:

```

File Edit Selection View Go Run Terminal Help
Restricted Mode is intended for safe code browsing. Trust this window to enable all features. Manage Learn More
client.py 3  splittedmain.py - Visual Studio Code
home > ubuntu > Downloads > Project > splittedmain.py > WorkerThread > process_image
class WorkerThread(threading.Thread):
    def process_image(self, path, operation_var):
        conn = remote(['102.37.145.180', 5000])
        file = open(path, 'rb')
        image_data = file.read()
        if operation_var == 'edge_detection':
            image_data = image_data + b"1"
            conn.send(image_data)
        elif operation_var == 'color_inversion':
            image_data = image_data + b"2"
            conn.send(image_data)
        elif operation_var == 'Blur':
            image_data = image_data + b"3"
            conn.send(image_data)
        elif operation_var == 'Binarization':
            image_data = image_data + b"4"
            conn.send(image_data)
        file.close()
        x = conn.recvall()
        file = open('y.jpg', 'wb')
        file.write(x)
        file.close()
        conn.close()
        result = cv2.imread('y.jpg', cv2.IMREAD_COLOR)
        resized_result = cv2.resize(result, (600, 600))
        return resized_result
|
```

The code implements a `WorkerThread` class that processes images using a remote connection. It handles four types of operations: edge detection, color inversion, blurring, and binarization. The processed image is then saved as `y.jpg` and its dimensions are resized to 600x600 pixels. The VS Code interface includes a sidebar with file navigation, a status bar at the bottom, and a taskbar at the very bottom.

## 4. Phase 4

### 4.1 End User Guide:

#### Overview:

This guide provides instructions for using the Image Processor Application, which allows users to upload, process, and download images using various image processing operations. The application leverages a combination of Tkinter for the graphical user interface (GUI), OpenCV for image processing, and MPI for distributed computing.

#### Installation Requirements:

Python 3.x

Required Python libraries: Tkinter, OpenCV, Pillow, mpi4py, and pwn.

#### Using the Application:

##### Step 1: Upload Image

- Click the "Upload Image" button.
- A file dialog will appear. Select the image you want to process.

##### Step 2: Select Operation

- Choose an image processing operation from the dropdown menu at the bottom. The available operations are:
  - Edge Detection: Detects the edges in the image.
  - Color Inversion: Inverts the colors of the image.
  - Blur: Applies a blur effect to the image.
  - Binarization: Converts the image to a binary image.

##### Step 3: Process Image

- Click the "Process Image" button to start the processing.
- The selected operation will be applied to the uploaded image. The processed image will be displayed in the GUI.

##### Step 4: Download Image

- Click the "Download Image" button to save the processed image to your local machine.
- A file dialog will appear. Choose the location and filename to save the processed image.

#### Technical Details:

##### GUI (Graphical User Interface):

- The GUI is built using Tkinter.
- The main components include buttons for uploading, processing, and downloading images, an option menu to select operations, and a label to display the images.

##### Worker Thread:

- The WorkerThread class handles the image processing tasks.
- It uses MPI for distributed processing. The master process sends image parts to worker processes, which then perform the specified operations.

##### Image Processing Operations:

- The operations are performed using OpenCV and sent to a remote server for processing.
- The operations are identified by specific codes (1 for Edge Detection, 2 for Color Inversion, 3 for Blur, 4 for Binarization).
- Processed images are resized to 500x500 pixels for display in the GUI.

#### Distributed Processing with MPI:

- The application is designed to run with 3 MPI processes (1 master and 2 workers).
- The master process receives the image and operation from the client, splits the image, and sends parts to the worker processes.
- Worker processes apply the selected operation and return the processed parts to the master, which combines them into the final processed image.

## 4.2 Testing

```

9 ▷ class TestImageProcessor(unittest.TestCase):
10
11 @!    def setUp(self):
12        self.app = GUI() # Create an instance of the GUI class
13        self.task_queue = queue.Queue() # Mock the task queue
14
15 @!    def tearDown(self):
16        self.app.destroy() # Destroy the GUI window
17
18    # Test cases for GUI functionality
19 ▷ def test_upload_valid_image(self):
20        valid_image_path = "path/to/valid_image.jpg"
21        self.app.upload_image()
22        self.assertIsNotNone(self.app.image_label.image) # Check if image is displayed
23
24 ▷ def test_upload_invalid_image(self):
25        invalid_image_path = "path/to/invalid.txt"
26        with self.assertRaises(Exception): # Expect an error (modify if specific)
27            self.app.upload_image()
28
29 ▷ def test_process_without_image(self):
30        with self.assertRaises(Exception): # Expect an error (modify if specific)
31            self.app.process_image()
32

```

```

3      # Test processing functionality - No extra argument needed for process_image
4  ▶ def test_process_edge_detection(self):
5      valid_image_path = "path/to/valid_image.jpg"
6      self.app.upload_image()
7      self.app.process_image()
8
9      processed_image = self.app.get_processed_image()
10     self.assertTrue(processed_image is not None)
11
12     # ... (similar test cases for other processing operations)
13
14  ▶ def test_download_without_processing(self):
15      with self.assertRaises(Exception): # Expect an error (modify if specific)
16          self.app.download_image()
17
18  ▶ def test_download_after_processing(self):
19      valid_image_path = "path/to/valid_image.jpg"
20      self.app.upload_image()
21      self.app.process_image()
22      # Simulate successful download (implementation omitted for testing purposes)
23      self.app.download_image()
24      # No assertions here as download functionality is not directly tested
25

```

## 4.3 Scalability

The screenshot shows the Microsoft Azure portal interface for managing a Load Balancer named 'LoadBalancer' under the resource group 'VM\_Nasser'. The 'Overview' tab is selected, displaying basic information such as Resource group (VM\_Nasser), Location (South Africa North), Subscription (Azure for Students), and SKU (Standard). The 'Tags' section shows 'Add tags'. On the right, there's a 'JSON View' button. Below the main details, there's a section titled 'Configure high availability and scalability for your applications' with three cards: 'Balance IPv4 and IPv6 addresses', 'Build highly reliable applications', and 'Secure your networks'. Each card has a 'View' button.

## Client Code:

The image shows two side-by-side instances of Microsoft Visual Studio Code (VS Code) running on a Windows operating system. Both windows have a dark theme and are displaying Python code related to image processing.

**Top Window (Client Code):**

- Explorer View:** Shows a project structure with files: `Background.png`, `myserver.py`, `server.py`, and `splittedmain.py`.
- Code Editor:** Displays the content of `splittedmain.py`. The code defines a `WorkerThread` class that extends `threading.Thread`. It imports `tkinter`, `cv2`, `queue`, and `pymqi`. The `\_\_init\_\_` method initializes the thread and sets up MPI communication. The `run` method processes tasks from a queue, applying operations like edge detection, color inversion, blur, or binarization to images, and then updates a GUI.
- Status Bar:** Shows "Ln 16, Col 40" and "Python 3.11.3 64-bit".

**Bottom Window (Client Code):**

- Explorer View:** Shows a project structure with files: `Background.png`, `myserver.py`, `server.py`, and `splittedmain.py`.
- Code Editor:** Displays the content of `splittedmain.py`. This version of the code uses MPI to process images. It defines a `WorkerThread` class that connects to a remote host at '102.37.138.221' port 5000. It reads images from a file, applies operations based on `operation\_var` (edge detection, color inversion, blur, binarization), and sends the processed image back to the client via a socket connection.
- Status Bar:** Shows "Ln 16, Col 40" and "Python 3.11.3 64-bit".

The image shows two side-by-side code editors, likely from a development environment like PyCharm or VS Code, displaying Python code for a graphical user interface (GUI) application.

**Top Editor:**

```

1 class GUI(tk.Tk):
2     def __init__(self):
3         super().__init__()
4         self.title("Image Processor")
5         self.geometry("700x700")
6
7         self.image_path = None
8
9         # Load the background image
10        background_img = Image.open("Background.png")
11        self.background_img = ImageTk.PhotoImage(background_img)
12
13        self.background_label = tk.Label(self, image=self.background_img)
14        self.background_label.place(x=0, y=0, relwidth=1, relheight=1)
15
16        self.image_label = tk.Label()
17        self.image_label.pack()
18
19        self.operation_var = tk.StringVar()
20        self.operation_var.set("edge_detection")
21        self.operation_menu = tk.OptionMenu(self, self.operation_var, "edge_detection", "color_inversion", "Blur", "Binarization")
22        self.operation_menu.pack(side=tk.BOTTOM)
23
24        self.download_button = tk.Button(self, text="Download Image", command=self.download_image, state=tk.DISABLED)
25        self.download_button.pack(side=tk.BOTTOM)
26
27        self.process_button = tk.Button(self, text="Process Image", command=self.process_image)
28        self.process_button.pack(side=tk.BOTTOM)
29
30        self.upload_button = tk.Button(self, text="Upload Image", command=self.upload_image)
31        self.upload_button.pack(side=tk.BOTTOM)
32
33        self.worker_thread = WorkerThread(task_queue)
34        self.worker_thread.start()

```

**Bottom Editor:**

```

1 class GUI(tk.Tk):
2     def __init__(self):
3         self.worker_thread = WorkerThread(task_queue)
4         self.worker_thread.start()
5
6         self.upload_image(self):
7             global image_path
8             file_path = filedialog.askopenfilename()
9             if file_path:
10                 image_path = file_path
11                 self.process_button.config(state=tk.NORMAL)
12                 self.download_button.config(state=tk.NORMAL)
13                 # Display the uploaded image
14                 self.display_uploaded_image(image_path)
15
16         def process_image(self):
17             global image_path
18             if image_path:
19                 task_queue.put((image_path, self.operation_var.get())) # Enqueue the processing task
20
21                 processed_image = self.worker_thread.processed_image
22                 if processed_image is not None:
23                     self.update_displayed_image(processed_image)
24
25         def update_displayed_image(self, processed_image):
26             processed_image = cv2.cvtColor(processed_image, cv2.COLOR_BGR2RGB)
27             processed_image = cv2.resize(processed_image, (500, 500))
28             img = Image.fromarray(processed_image) # Convert processed image to PIL format
29             img = ImageTk.PhotoImage(img)
30             self.image_label.config(image=img)
31             self.image_label.image = img # Keep a reference to avoid garbage collection
32
33         def display_uploaded_image(self, image_path):
34             img = cv2.imread(image_path)
35             img = cv2.resize(img, (500, 500))
36             img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

```

```

File Edit Selection View Go Run Terminal Help < > Project
EXPLORER PROJECT ...
splitmain.py myserver.py server.py
splitmain.py > _init_
65 class GUI(tk.Tk):
118     self.update_displayed_image(processed_image)
119
120     def update_displayed_image(self, processed_image):
121         processed_image = cv2.cvtColor(processed_image, cv2.COLOR_BGR2RGB)
122         processed_image = cv2.resize(processed_image, (500, 500))
123         img = Image.fromarray(processed_image) # Convert processed image to PIL format
124         img = ImageTk.PhotoImage(img)
125         self.image_label.config(image=img)
126         self.image_label.image = img # Keep a reference to avoid garbage collection
127
128     def display_uploaded_image(self, image_path):
129         img = cv2.imread(image_path)
130         img = cv2.resize(img, (500, 500))
131         img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
132
133         img = ImageTk.PhotoImage(image=Image.fromarray(img))
134         self.image_label.config(image=img)
135         self.image_label.image = img # Keep a reference to avoid garbage collection
136
137     def download_image(self):
138         global image_path
139         if image_path:
140             save_path = filedialog.asksaveasfilename(defaultextension=".png")
141             if save_path:
142                 cv2.imwrite(save_path, self.worker_thread.processed_image)
143
144
145     if __name__ == "__main__":
146         app = GUI()
147         app.mainloop()
148
149

```

> OUTLINE > TIMELINE

Ln 16, Col 40 Spaces: 4 UTF-8 CR/LF Python 3.11.3 64-bit EN

## Code of MPI server:

```

File Edit Selection View Go Run Terminal Help < > Project
EXPLORER PROJECT ...
Background.png myserver.py server.py splitmain.py
myserver.py > ...
1 from mpi4py import MPI
2 import cv2
3 import numpy as np
4 from pw import *
5
6 def process_image_part(part, operation):
7     if operation == 49:
8         part = cv2.Canny(part, 100, 200)
9     elif operation == 50:
10        part = cv2.bitwise_not(part)
11    elif operation == 51:
12        part = cv2.medianBlur(part, 11)
13    elif operation == 52:
14        part = cv2.cvtColor(part, cv2.COLOR_BGR2GRAY)
15        part = cv2.threshold(part, 128, 255, cv2.THRESH_BINARY)
16    else:
17        raise ValueError("Invalid Operation")
18    return part
19
20 def split_image(image, num_splits):
21     height = image.shape[0]
22     split_height = height // num_splits
23     return [image[i * split_height:(i + 1) * split_height] for i in range(num_splits)]
24
25 def combine_image(parts):
26     return np.vstack(parts)
27
28 COMM = MPI.COMM_WORLD
29 rank = COMM.Get_rank()
30 size = COMM.Get_size()
31
32 if size != 3:
33     raise ValueError("This script requires exactly 3 MPI processes")
34
35 if rank == 0:
36     # Master process
37     print("Waiting for a connection...")

```

> OUTLINE > TIMELINE

Ln 19, Col 1 Spaces: 4 UTF-8 CR/LF Python 3.11.3 64-bit EN

```

File Edit Selection View Go Run Terminal Help <- > Project
EXPLORER PROJECT myserver.py & myserver.py & server.py
myserver.py > ...
30     size = comm.Get_size()
31
32     if size != 3:
33         raise ValueError("This script requires exactly 3 MPI processes")
34
35     if rank == 0:
36         # Master process
37         print("Waiting for a connection...")
38
39         l = listen(5000)
40         recv = l.accept()
41
42         if recv:
43             with open('image.jpg', "wb") as file:
44                 file.write(recv[0:len(recv)-2])
45             operation = recv[0:len(recv)-1]
46             print(f"Operation: {operation}")
47
48             img = cv2.imread('image.jpg')
49             if img is None:
50                 print("Failed to read the image.")
51                 MPI.COMM_WORLD.Abort()
52             print("Image received")
53
54             # split the image into parts
55             parts = split_image(img, 2)
56
57             # send parts to workers
58             comm.send(parts[0], dest=1, tag=1)
59             comm.send(parts[1], dest=2, tag=2)
60             comm.send(operation, dest=1, tag=3)
61             comm.send(operation, dest=2, tag=4)
62
63             # Receive processed parts from workers
64             processed_part1 = comm.recv(source=1, tag=5)
65             processed_part2 = comm.recv(source=2, tag=6)
66
67             # Combine processed parts
68             result_img = combine_image([processed_part1, processed_part2])
69             print("Image edited")
70
71             # Save the processed image
72             cv2.imwrite('Result_image.jpg', result_img)
73
74             with open('Result_image.jpg', 'rb') as file2:
75                 image2 = file2.read()
76                 l.send(image2)
77
78                 print("Image Sent")
79             else:
80                 print("Failed to receive data from the client.")
81                 MPI.COMM_WORLD.Abort()
82
83         else:
84             # Worker processes
85             part = comm.recv(source=0, tag=rank)
86             operation = comm.recv(source=0, tag=rank + 2)
87
88             processed_part = process_image_part(part, operation)
89
90             comm.send(processed_part, dest=0, tag=rank + 4)
91

```

## Code of Server:

The image shows two side-by-side code editors, likely from the Visual Studio Code IDE, displaying Python scripts for a server application.

**Top Editor (server.py):**

```
File Edit Selection View Go Run Terminal Help <- > Project
PROJECT
  Background.png
  myserver.py
  server.py
  splittedmain.py
server.py > ...
1 import cv2
2 import numpy as np
3 from socket import *
4
5 print("Waiting for a connection...")
6
7 l = listen(5000)
8 recv = l.accept()
9
10 file = open('image.jpg', 'wb')
11 file.write(recv[0][len(recv)-2])
12 operation = recv[0][len(recv)-1]
13 print(operation)
14
15 img = cv2.imread('image.jpg')
16
17 print("Image received")
18
19 # 49 is edge detection
20 # 50 is color inversion
21 # 51 is BLUR
22 # 52 is binarization
23
24
25 if operation == 49:
26     img = cv2.Canny(img, 100, 200)
27
28 elif operation == 50:
29     img = cv2.bitwise_not(img)
30
31 elif operation == 51:
32     img = cv2.medianBlur(img, 11)
33
34 elif operation == 52:
35     # Apply thresholding
36     img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

**Bottom Editor (server.py):**

```
File Edit Selection View Go Run Terminal Help <- > Project
PROJECT
  Background.png
  myserver.py
  server.py
  splittedmain.py
server.py > ...
19 # 49 is edge detection
20 # 50 is color inversion
21 # 51 is BLUR
22 # 52 is binarization
23
24
25 if operation == 49:
26     img = cv2.Canny(img, 100, 200)
27
28 elif operation == 50:
29     img = cv2.bitwise_not(img)
30
31 elif operation == 51:
32     img = cv2.medianBlur(img, 11)
33
34 elif operation == 52:
35     # Apply thresholding
36     img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
37     _, img = cv2.threshold(img, 128, 255, cv2.THRESH_BINARY)
38
39 else:
40     print("Invalid operation")
41
42 print("Image edited")
43 # Save the thresholded image
44 cv2.imwrite('Result_image.jpg', img)
45
46
47 file2 = open('Result_image.jpg', 'rb')
48 image2 = file2.read()
49 l.send(image2)
50
51 print("Image Sent")
```

## Test of MPI setup on azure virtual machines:

The image shows three separate terminal windows, each titled "VM1@VM2:-". The first window displays the output of the command "ssh VM1@VM2", which logs into another Ubuntu 22.04.4 LTS (x86\_64) machine. It shows system information, including load average (0.0), memory usage (25%), swap usage (0%), and network information (IPv4 address 10.0.0.5). It also provides links for documentation, management, and support. The second window shows the result of the MPI test, indicating that 0 updates can be applied immediately. The third window shows the result of another MPI test, also indicating 0 updates can be applied immediately.

```
VM1@VM1:~$ ssh VM1@VM2
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 6.5.0-1021-azure x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/pro

System information as of Sun May 19 21:21:52 UTC 2024

System load: 0.0          Processes: 136
Usage of /: 12.7% of 28.89GB  Users logged in: 1
Memory usage: 25%          IPv4 address for eth0: 10.0.0.5
Swap usage: 0%

* Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s just raised the bar for easy, resilient and secure K8s cluster deployment.

https://ubuntu.com/engage/secure-kubernetes-at-the-edge

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

3 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

Last login: Sun May 19 19:21:15 2024 from 10.0.0.5
VM1@VM2:~$ |
```

```
VM1@VM3:~$ https://ubuntu.com/engage/secure-kubernetes-at-the-edge
Expanded Security Maintenance for Applications is not enabled.
0 updates can be applied immediately.
3 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

Last login: Sun May 19 19:21:15 2024 from 10.0.0.5
VM1@VM3:~$ ssh VM1@VM3
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 6.5.0-1021-azure x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/pro

System information as of Sun May 19 21:22:42 UTC 2024

System load: 0.46          Processes: 134
Usage of /: 12.6% of 28.89GB  Users logged in: 1
Memory usage: 28%          IPv4 address for eth0: 10.0.0.6
Swap usage: 0%

* Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s just raised the bar for easy, resilient and secure K8s cluster deployment.

https://ubuntu.com/engage/secure-kubernetes-at-the-edge

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

3 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

Last login: Sun May 19 19:21:44 2024 from 10.0.0.6
VM1@VM3:~$ |
```

```

VM1@VM1:~          + - 
VM1@VM1:~$ ssh VM1@VM1
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 6.5.0-1021-azure x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information as of Sun May 19 21:23:11 UTC 2024

System load: 0.0      Processes:           139
Usage of /: 13.8% of 28.89GB  Users logged in: 1
Memory usage: 25%      IPv4 address for eth0: 10.0.0.4
Swap usage:  0% 

* Strictly confined Kubernetes makes edge and IoT secure. Learn how MicroK8s
just raised the bar for easy, resilient and secure K8s cluster deployment.

https://ubuntu.com/engage/secure-kubernetes-at-the-edge

Expanded Security Maintenance for Applications is not enabled.

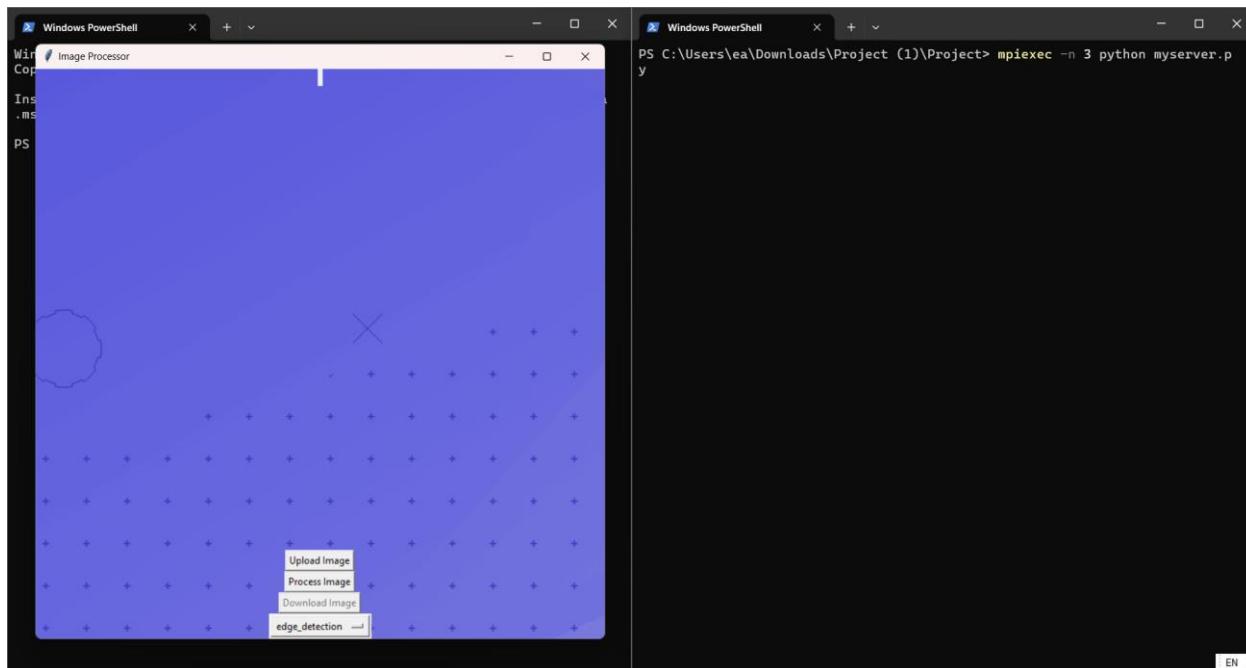
0 updates can be applied immediately.

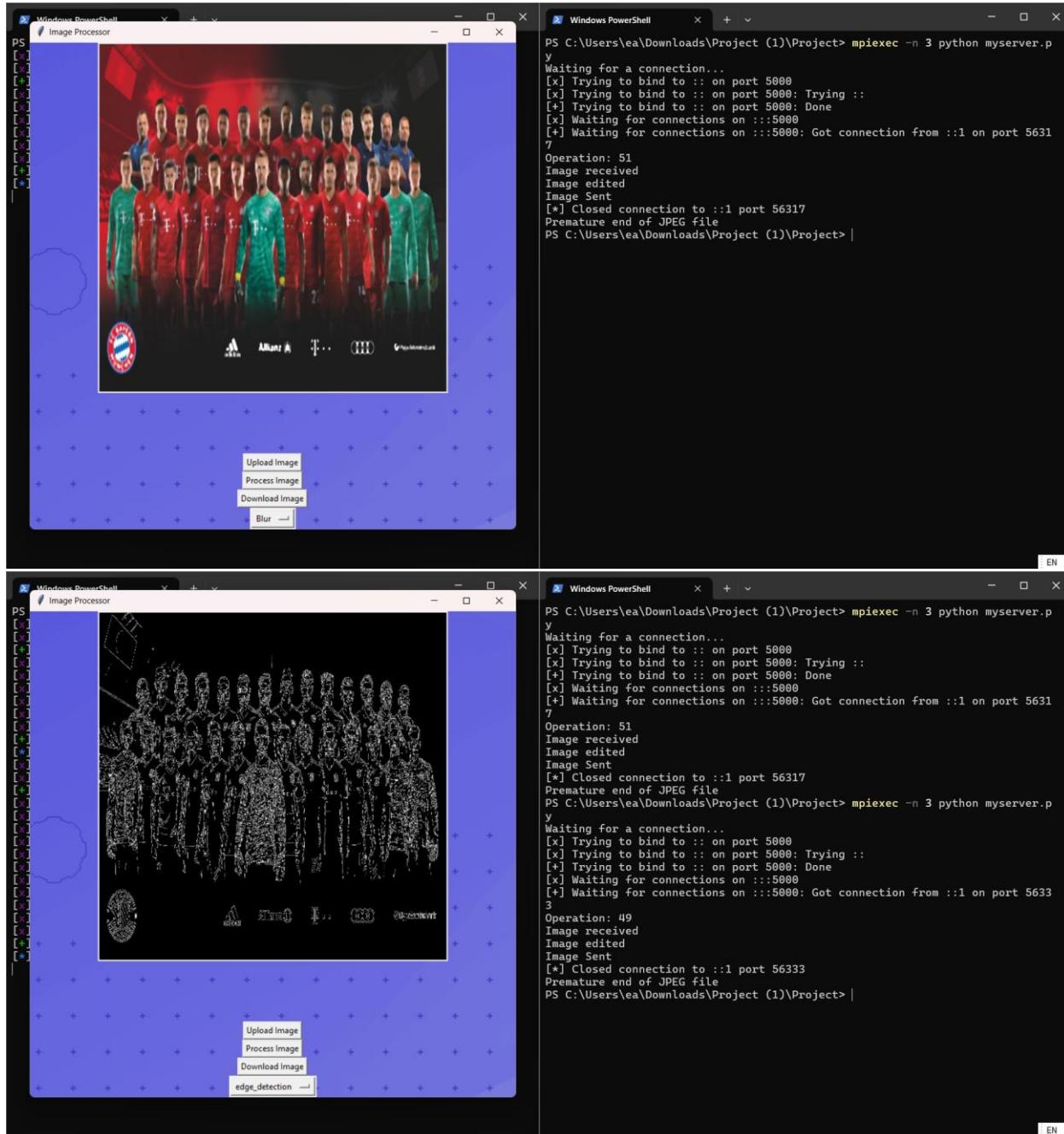
11 additional security updates can be applied with ESM Apps.
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

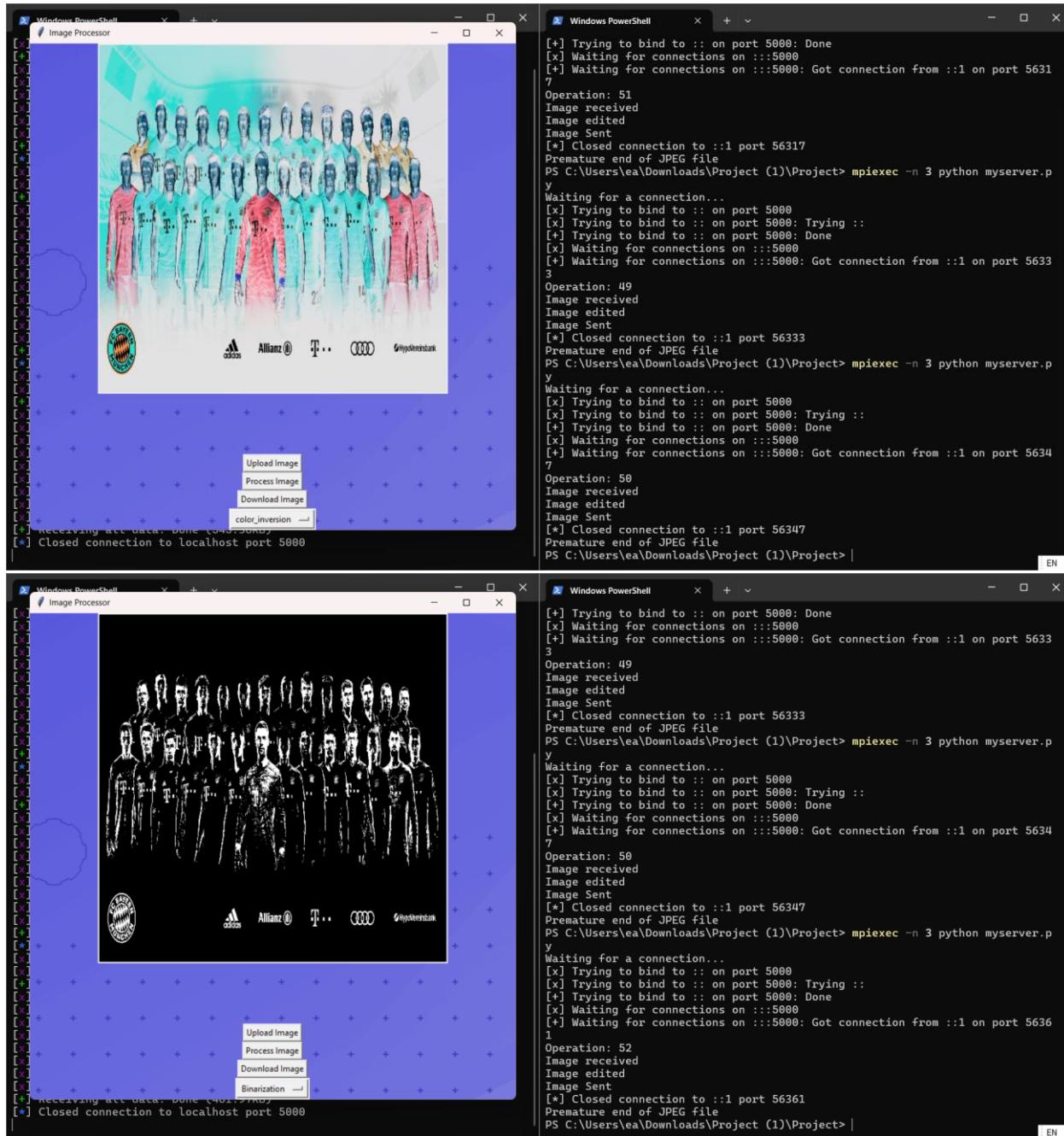
Last login: Sun May 19 19:21:27 2024 from 10.0.0.6
VM1@VM1:~$ |

```

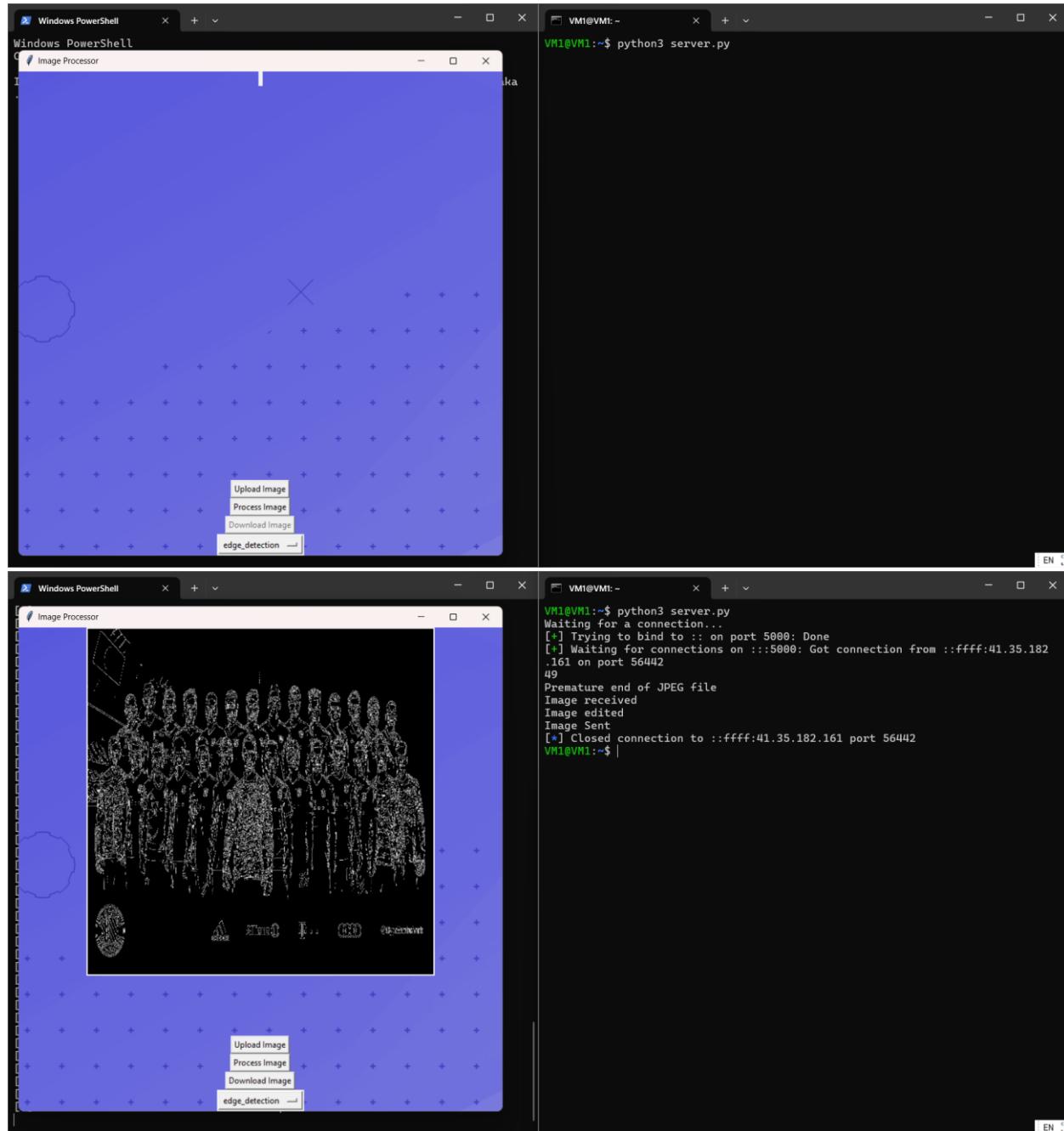
## Run of MPI on host machine:

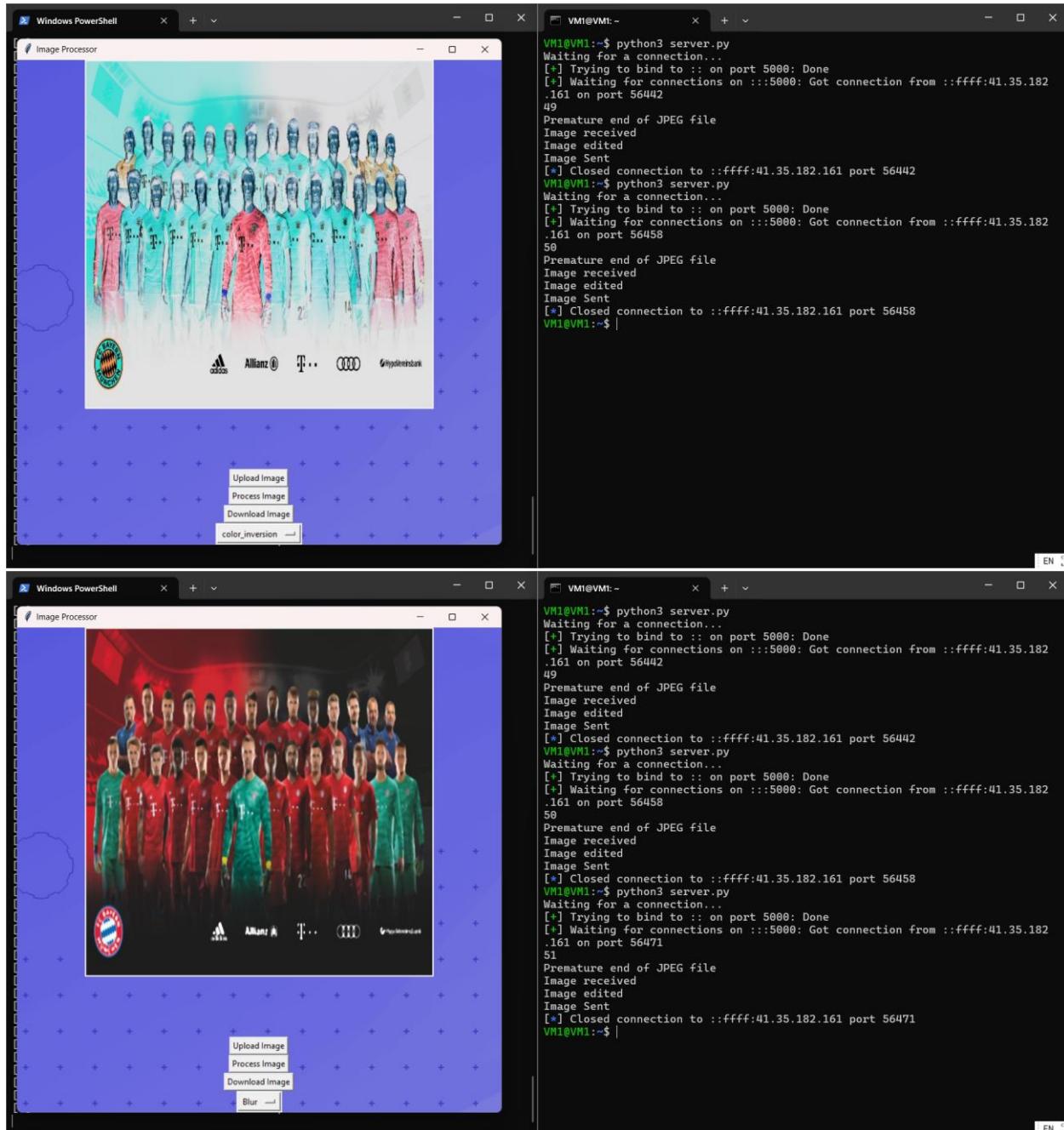


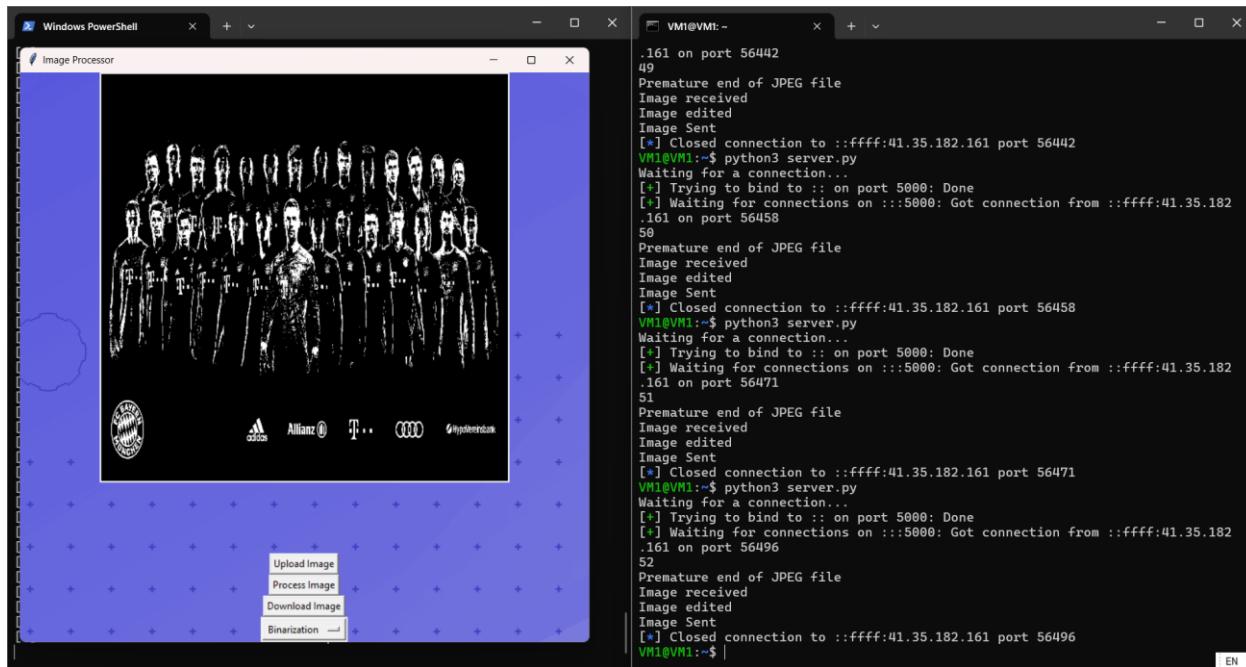




Run on the master node on azure:







#### 4.4 Conclusion

To summarize, this project has effectively developed a distributed image processing system utilizing cloud computing technologies. By employing parallel processing frameworks like MPI and OpenCL, the system efficiently distributes image processing tasks across multiple virtual machines, ensuring high performance and scalability. The implementation of sophisticated image processing algorithms, such as edge detection and color manipulation, showcases the system's ability to handle complex tasks with ease.

The system's architecture is robust, incorporating fault tolerance mechanisms to manage node failures and ensure continuous operation and reliability. Using Docker for containerization enhances the management and deployment of microservices within the cloud environment, further improving scalability and resource efficiency.

A user-friendly interface has been developed to facilitate seamless interaction, allowing users to upload images, select processing operations, and download the processed results. The detailed project plan, with clearly defined responsibilities and a structured timeline, has ensured a systematic and organized approach to development.

Leveraging Microsoft Azure as the cloud platform has provided a secure and reliable environment for processing and data storage. This choice not only addresses current image processing needs but also prepares the project to benefit from future advancements in cloud technology.

Video Link: <https://drive.google.com/file/d/1MnUPbHOxSCYaqIIHewyy4iLsjXJoyPg3/view?usp=sharing>

GitHub Link: <https://github.com/Nasser1159/Distributed>