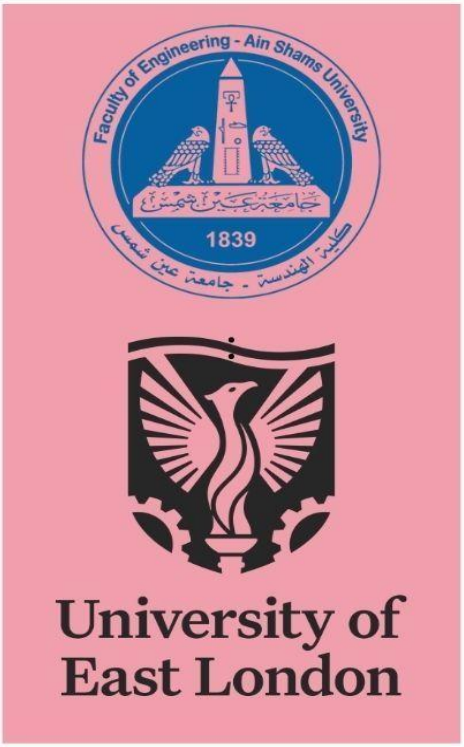
AIN SHAMS
UNIVERSITY



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

Mariam Wahdan Allam	**20P5500**
Malak Mohamed Mahfouz	**20P7813**
Mohamed Abdelnasser Mohamed	**20P9501**
Mohamed Khaled Ahmed	**20P4277**

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

List Of Figures

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.

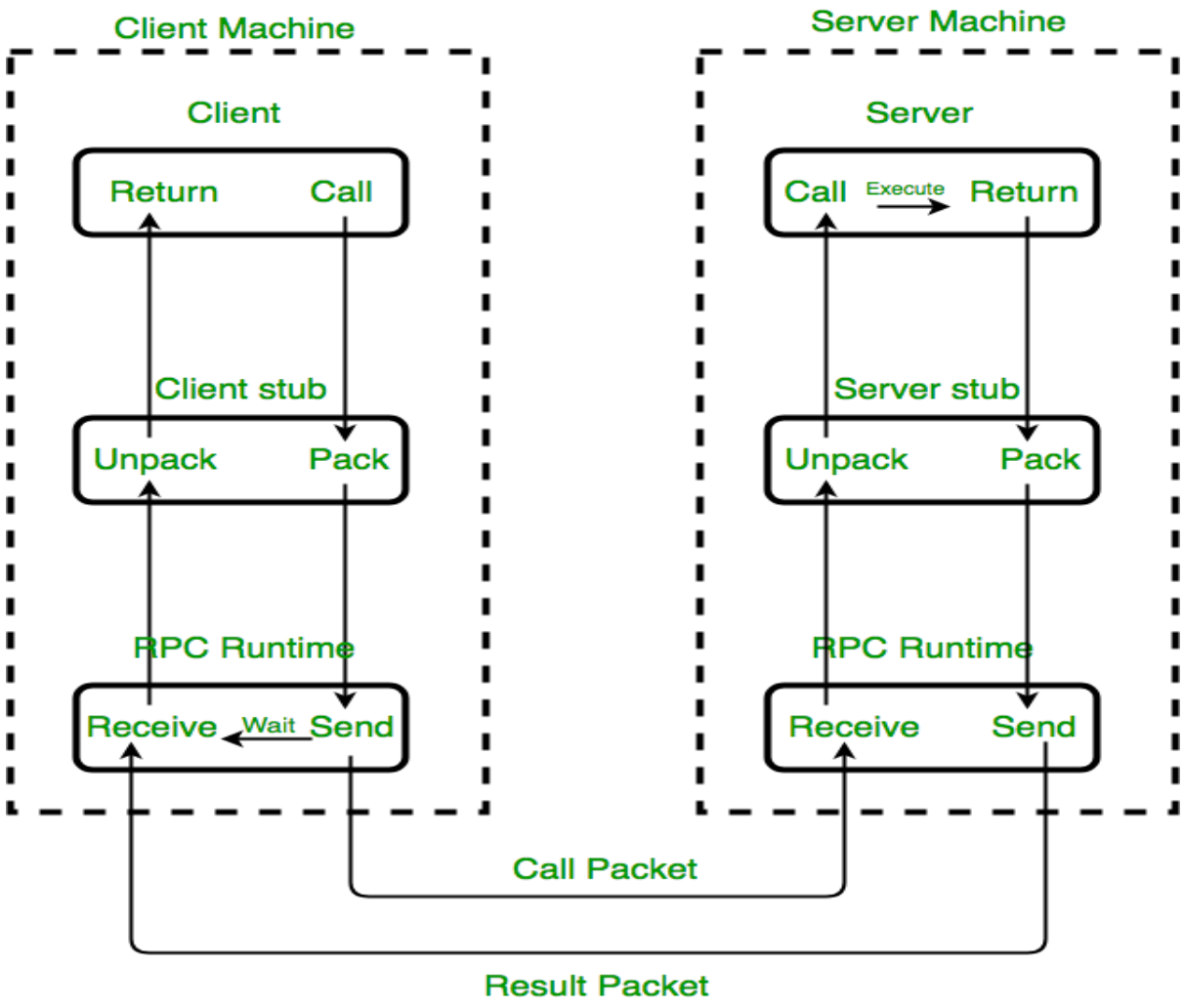


Figure 1RPC Mechanism

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:

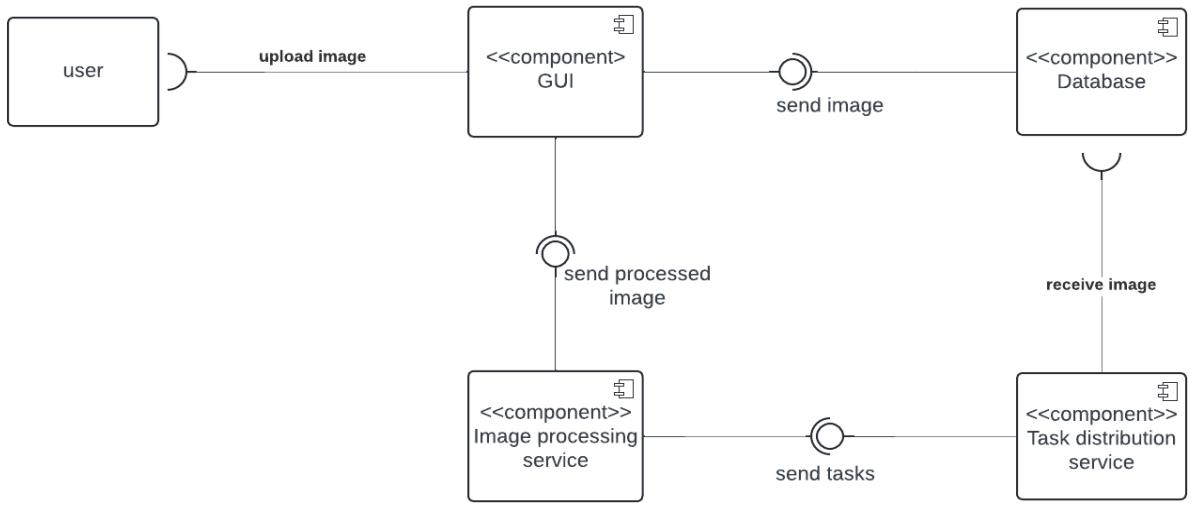


Figure 2 System Component Diagram

1.10.2 Architecture Diagram:

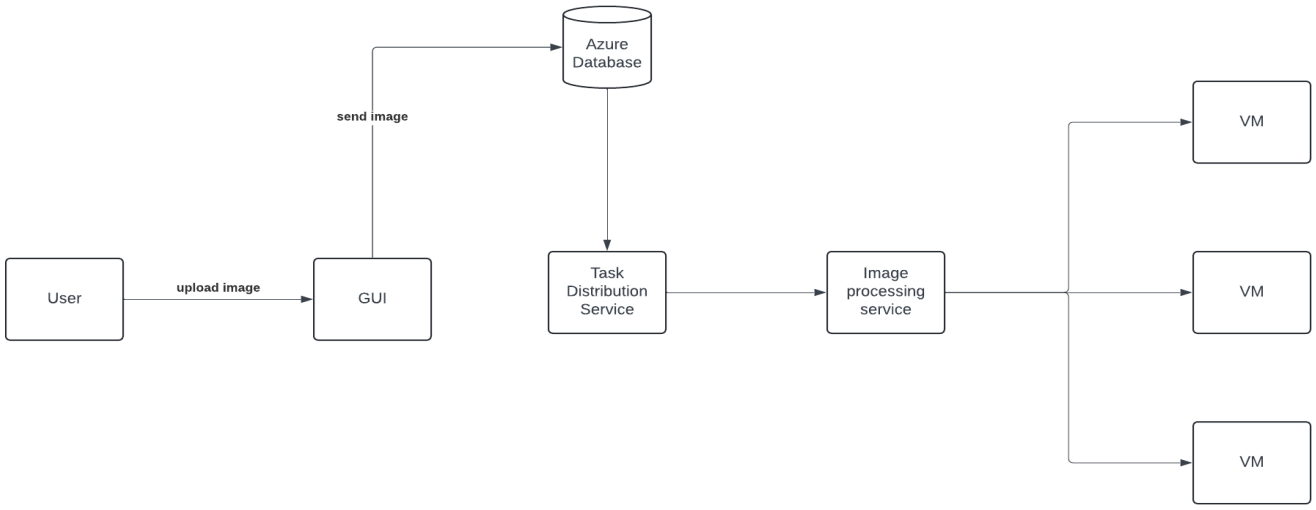


Figure 3 System Architecture Diagram

1.10.3 Sequence Diagram:

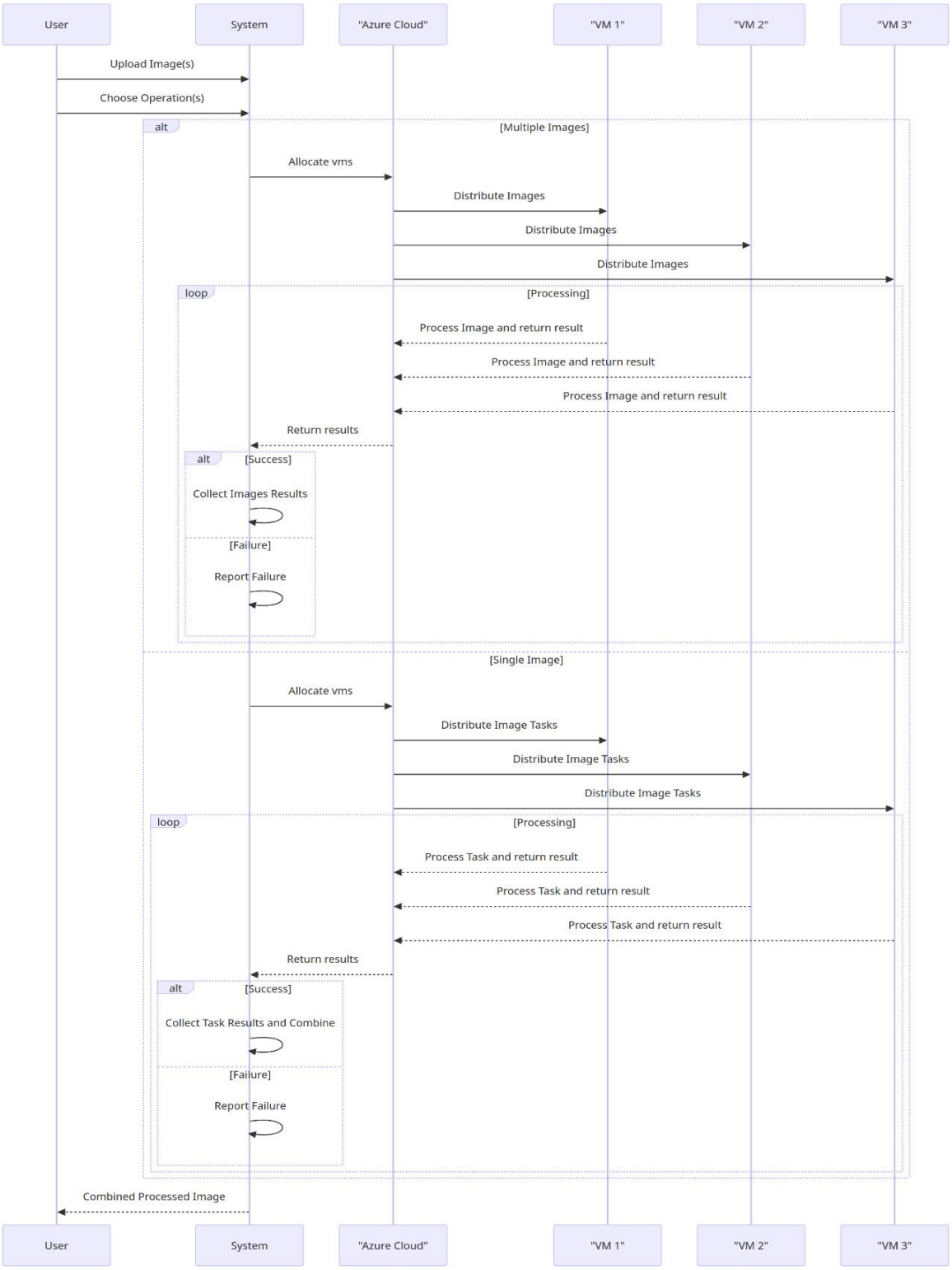


Figure 4 System Sequence Diagram

1.10.4 Network Diagram:

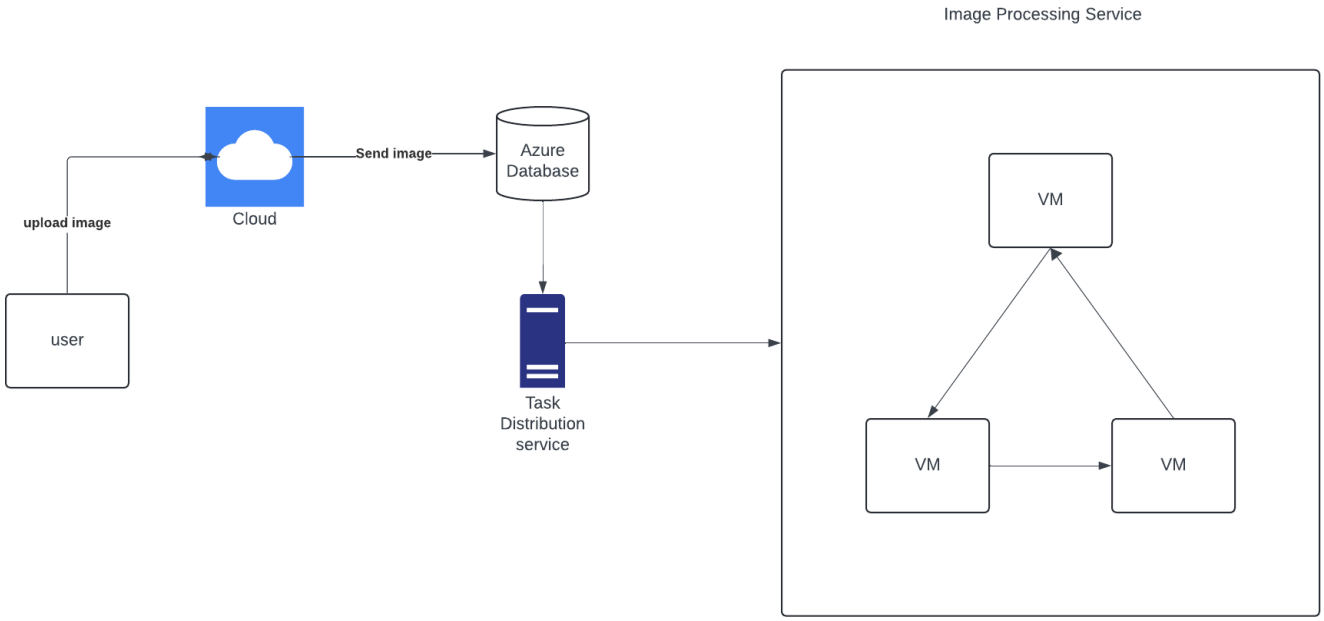


Figure 5 System Network Diagram

1.10.5 Gant Chart:

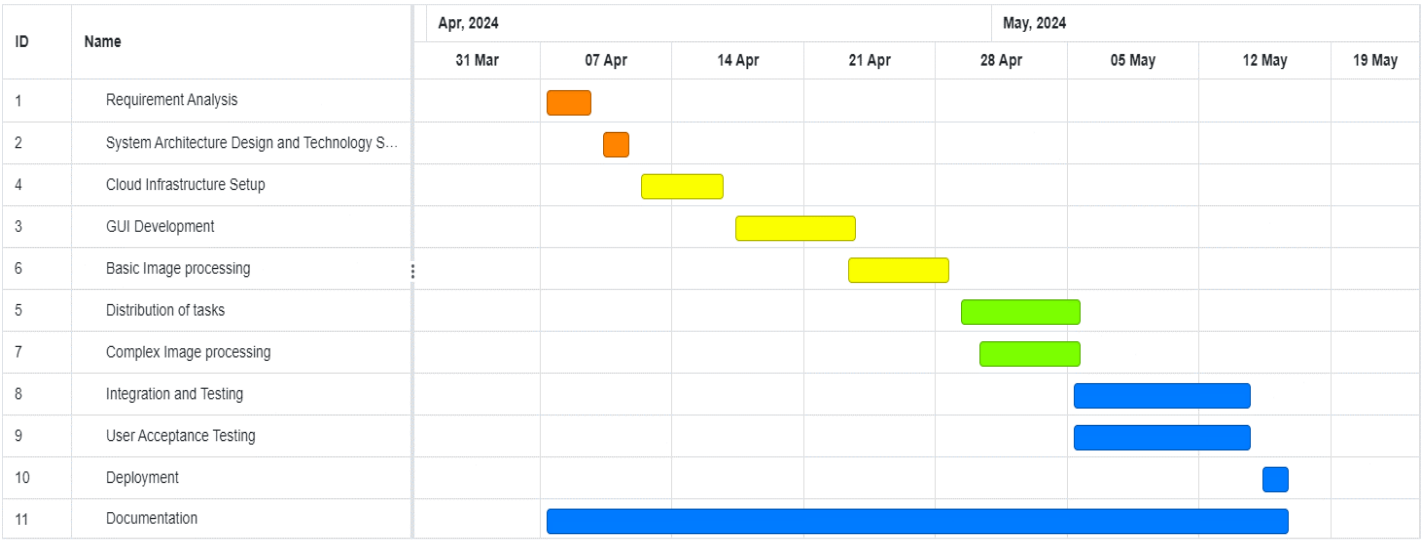


Figure 6 System Time plan

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
Total Estimated Cost		**$63,250**

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

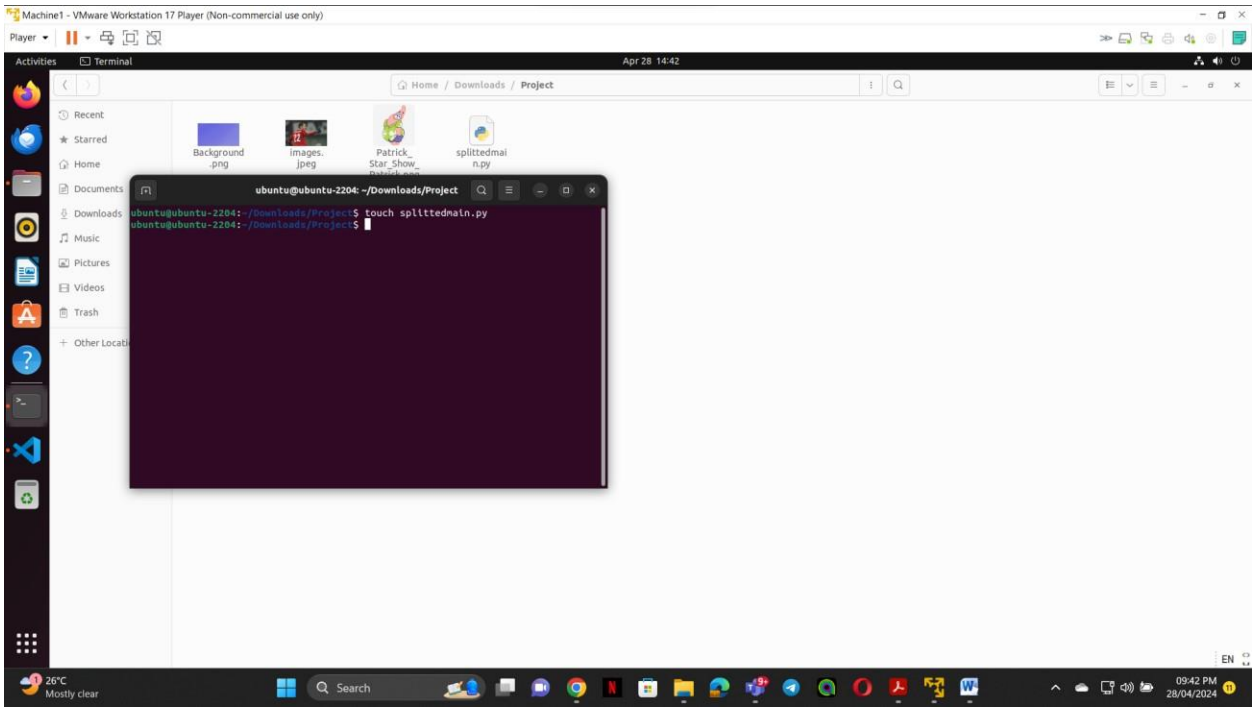


Figure 8 Code Normal Run without MPI 1) Touch

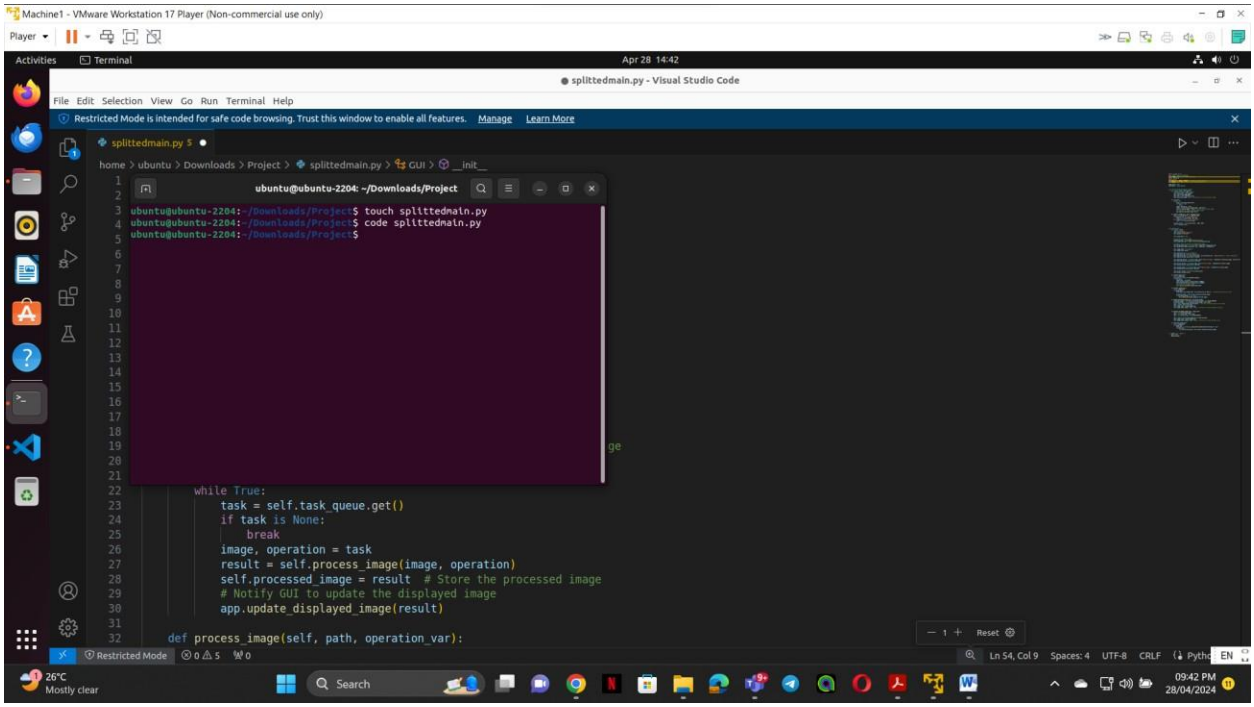


Figure 9 Code Normal Run without MPI 2) code

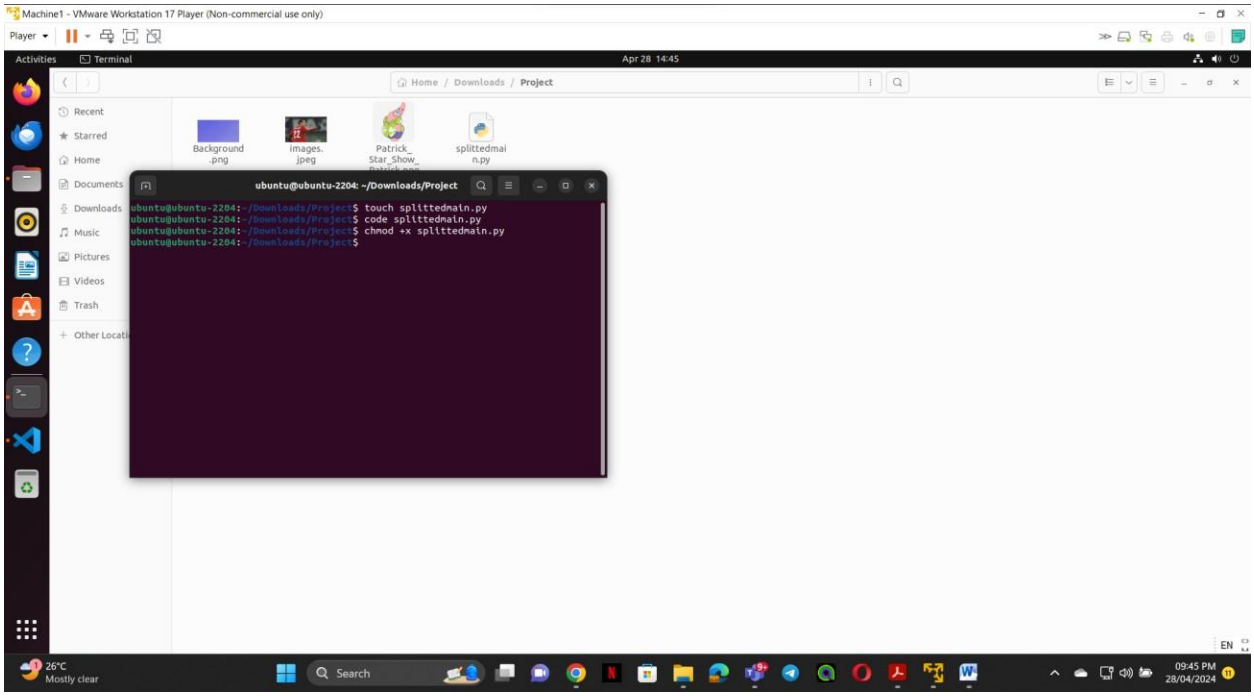


Figure 10 Code Normal Run without MPI 3) chmod

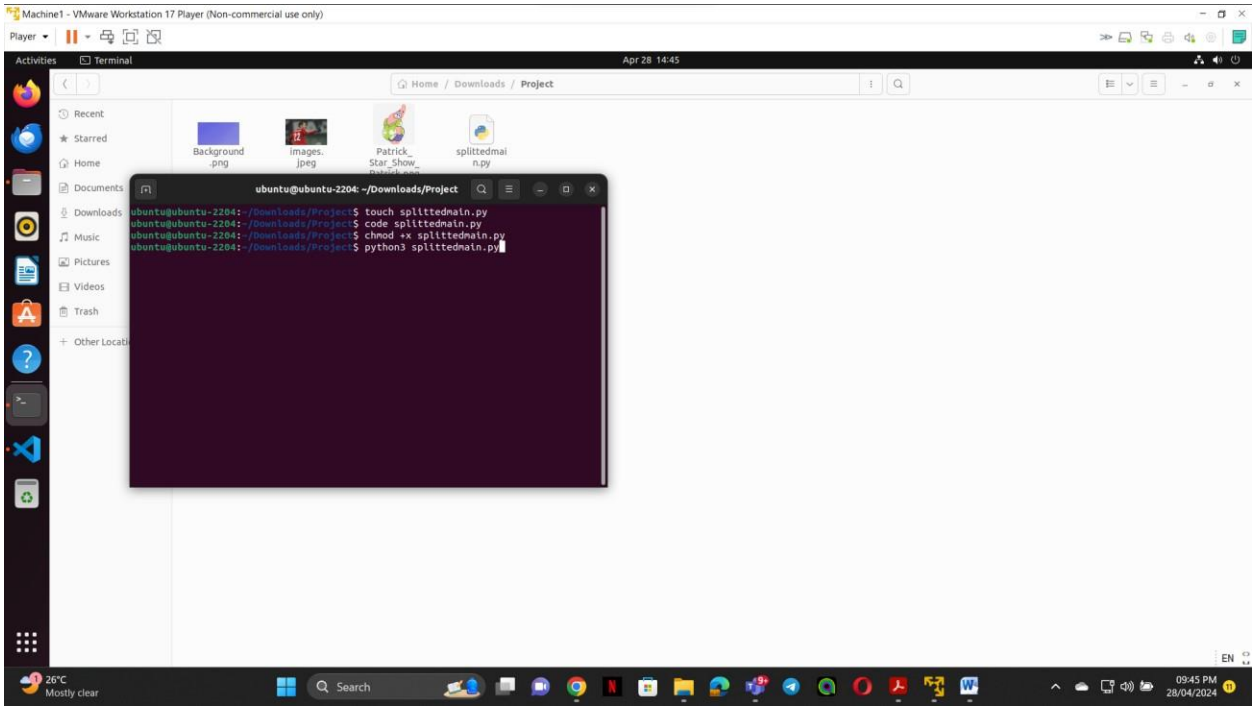


Figure 11 Normal Code Run without MPI 4) python3

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

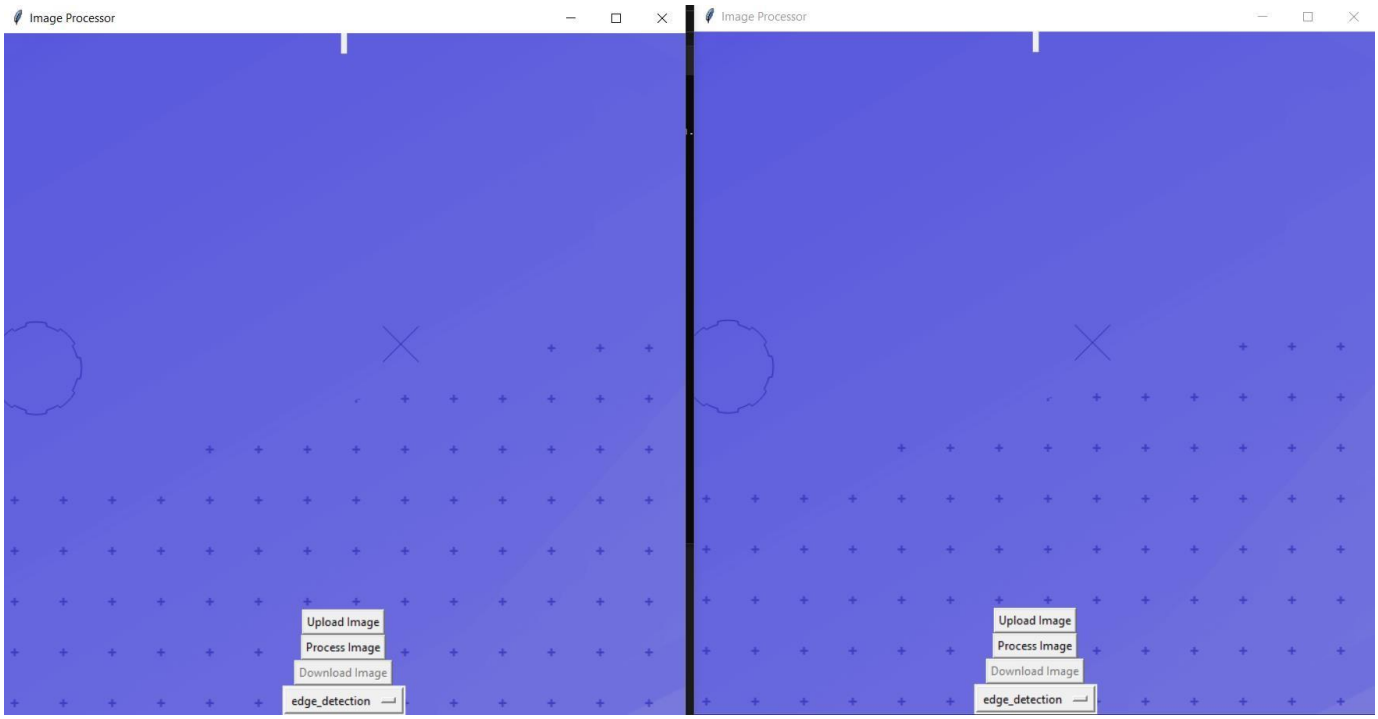


Figure 12 Output From previous run

2.2 Run

2.2.1 Home Page

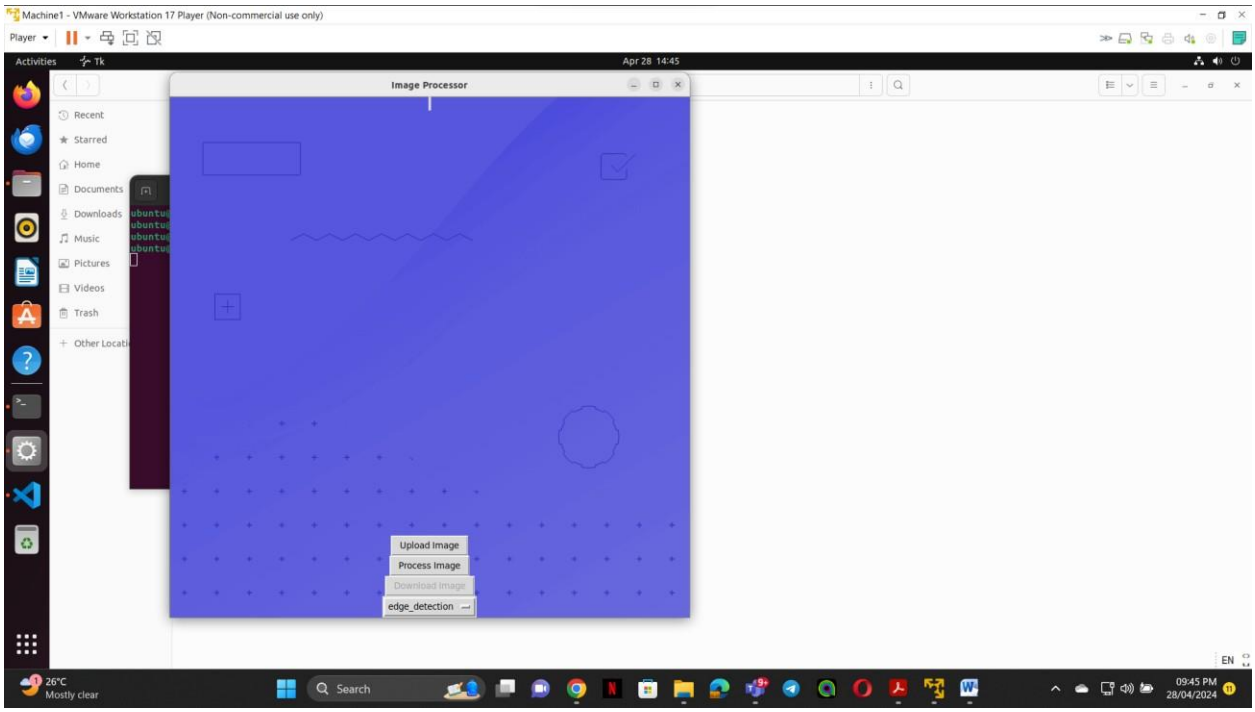


Figure 14 System Home Page

2.2.2 Click on upload button

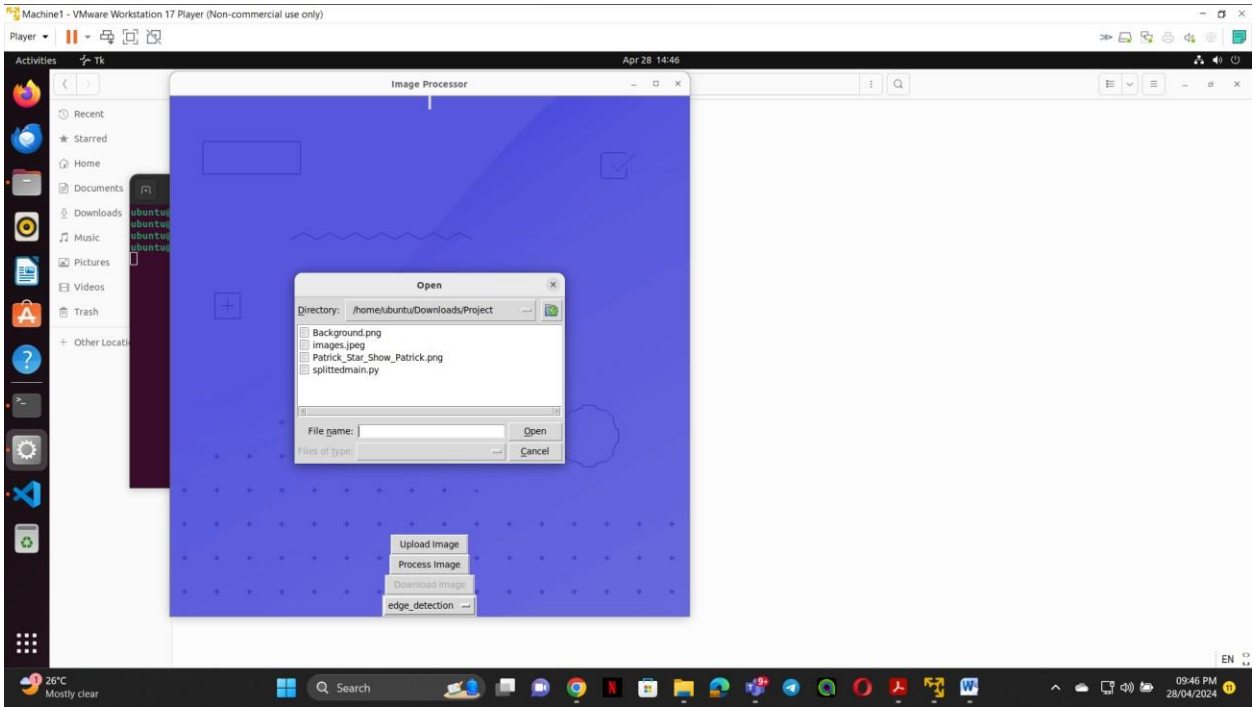


Figure 15 Uploading Image to System

2.2.3 After uploading

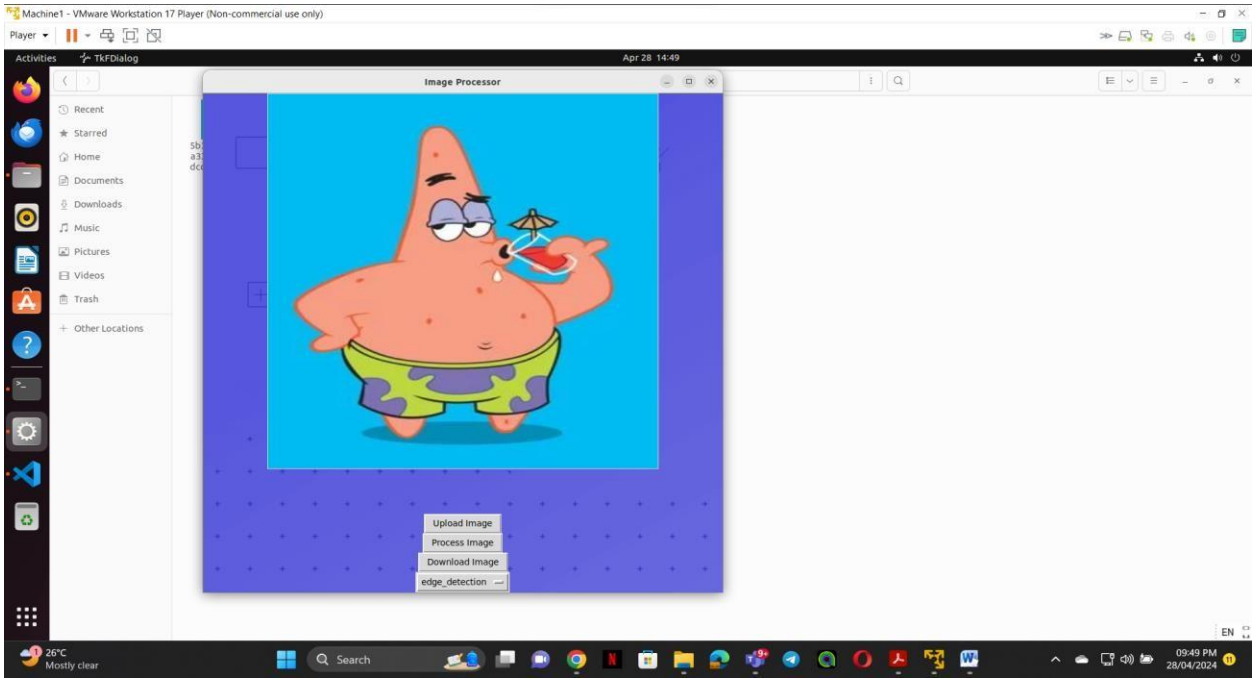


Figure 16 Image Display inside The Main window Before preprocessing

2.2.4 Choose Operation

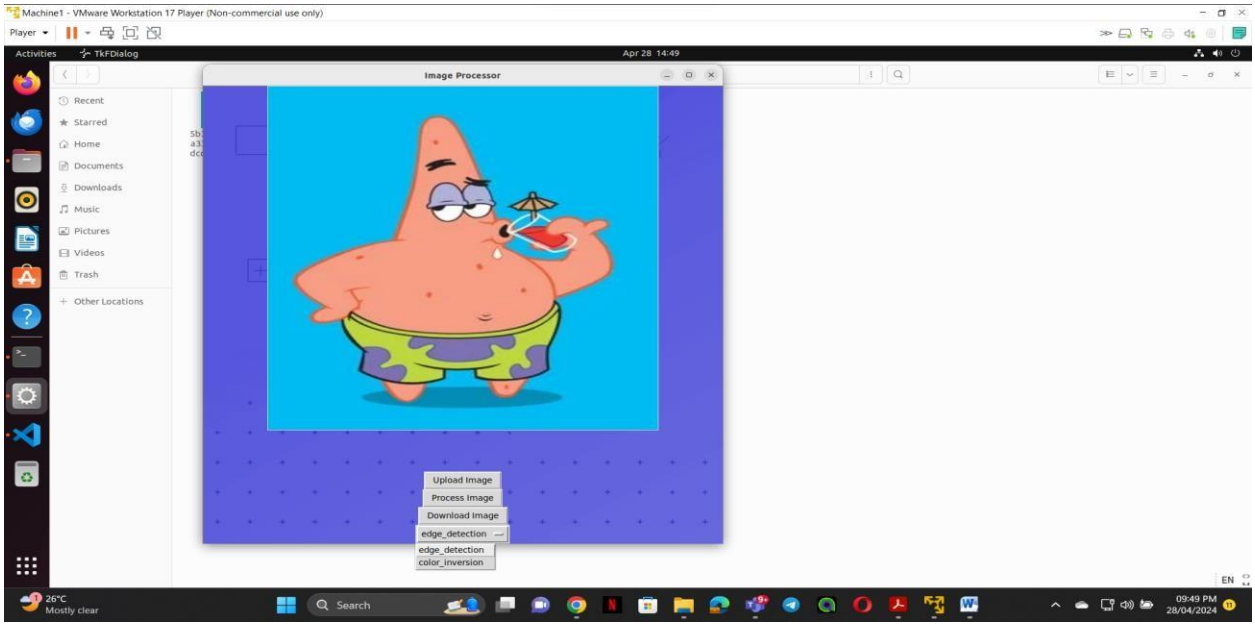


Figure 17 Choosing Operation

2.2.5 Preview Image: We used color inversion operation.

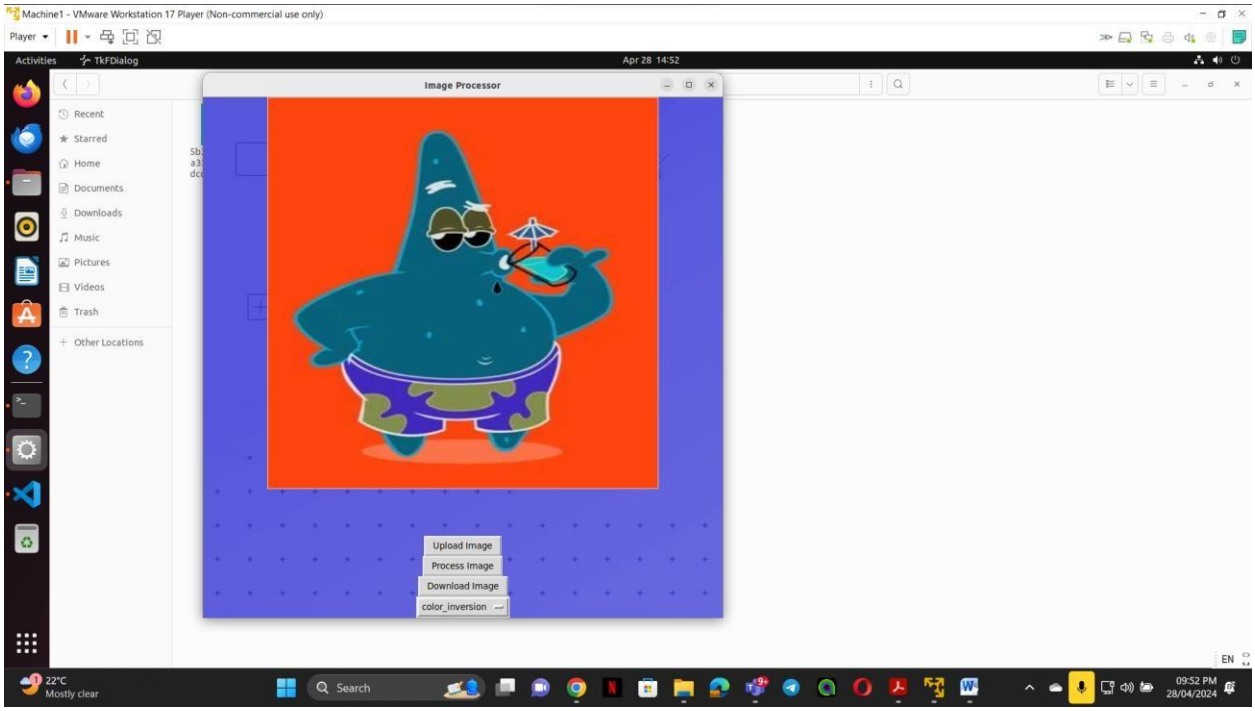


Figure 18 Output of using color inversion and Displaying new Image

2.2.7 Preview Image: We used edge detection operation.

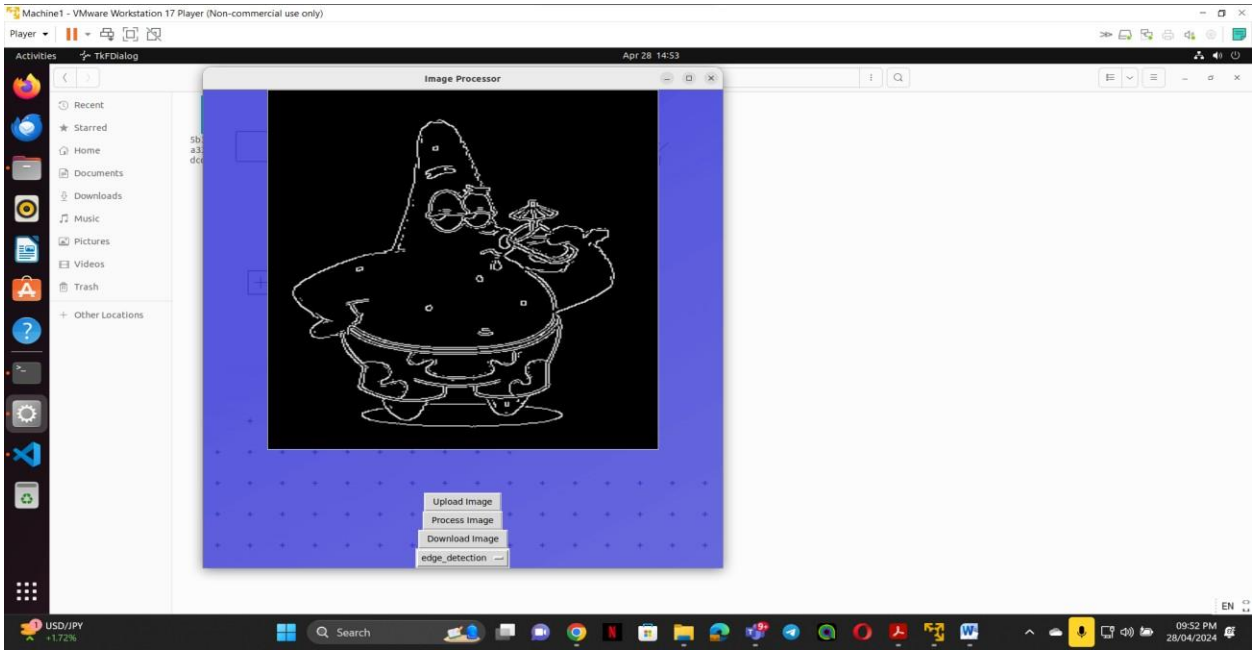


Figure 19 Output of using Edge Detection and Displaying new Image

2.2.6 Download Image

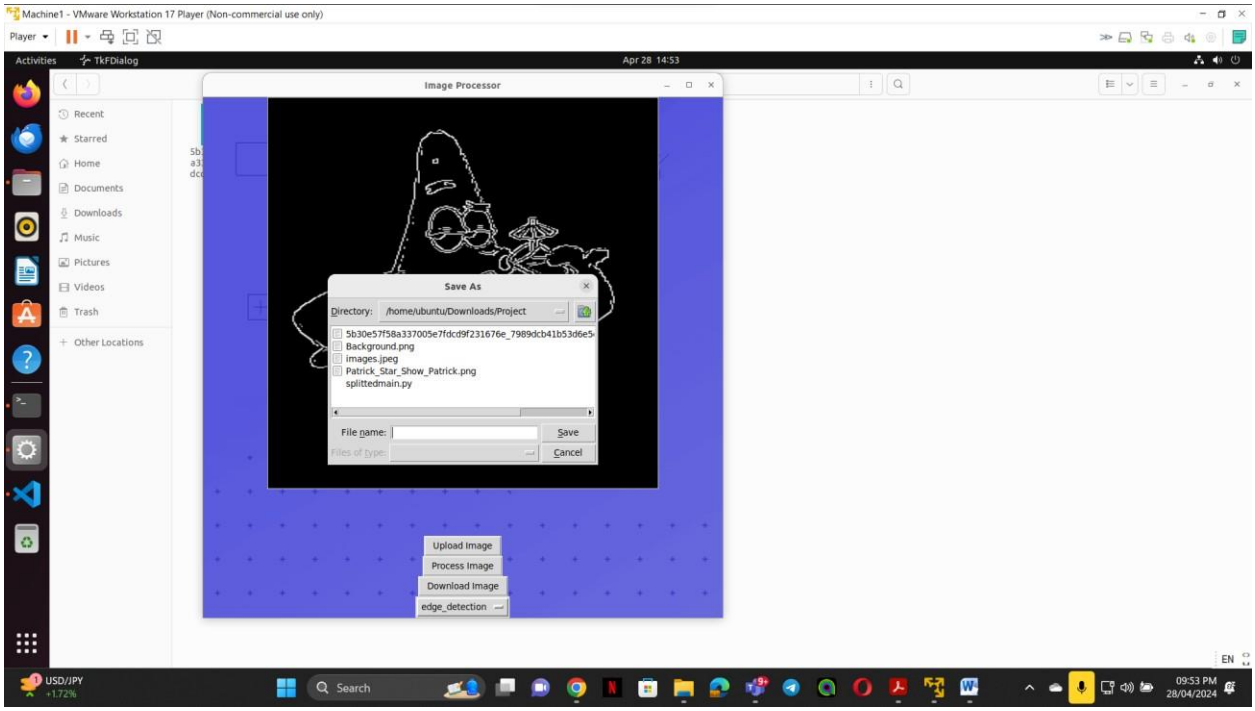
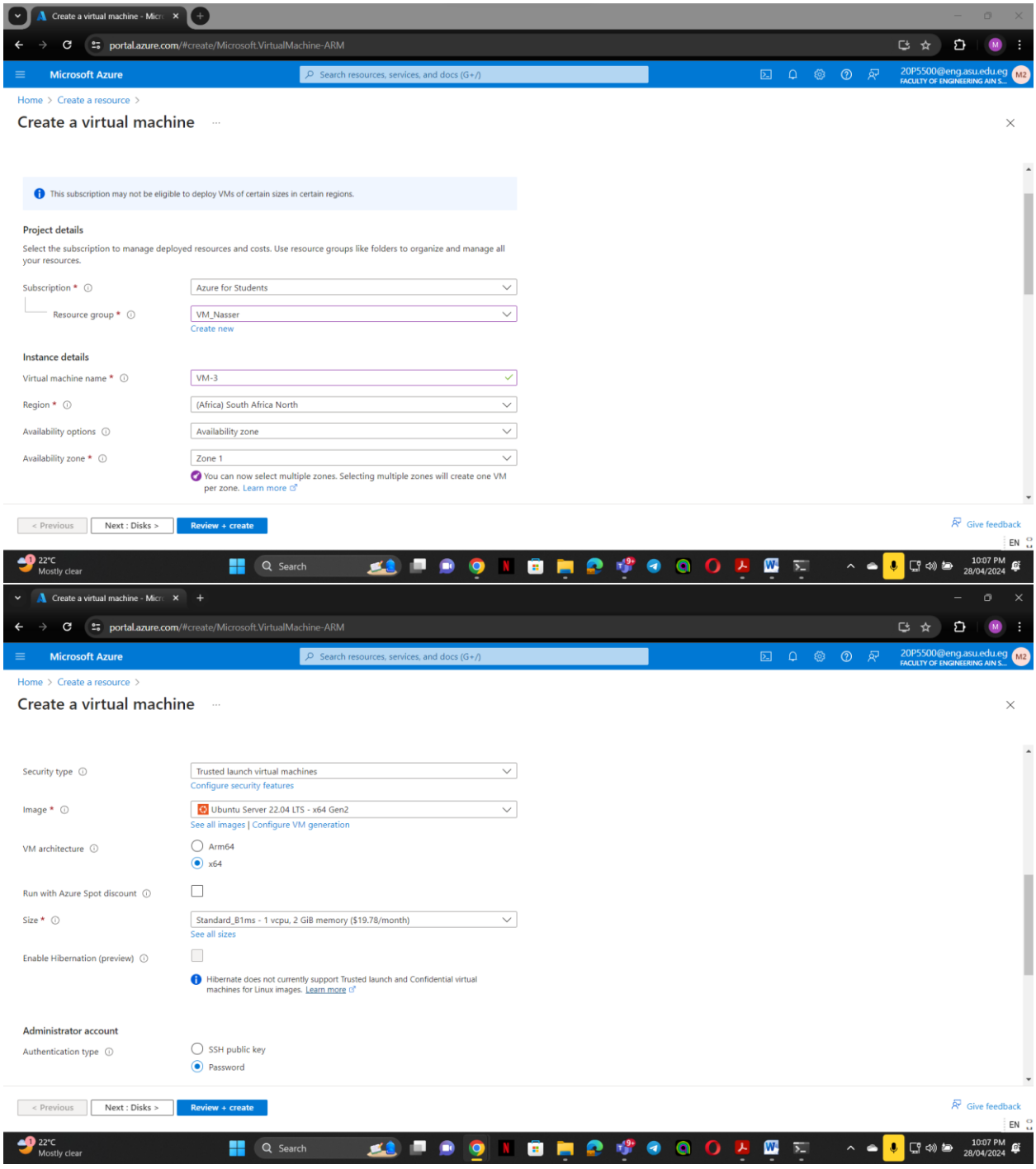
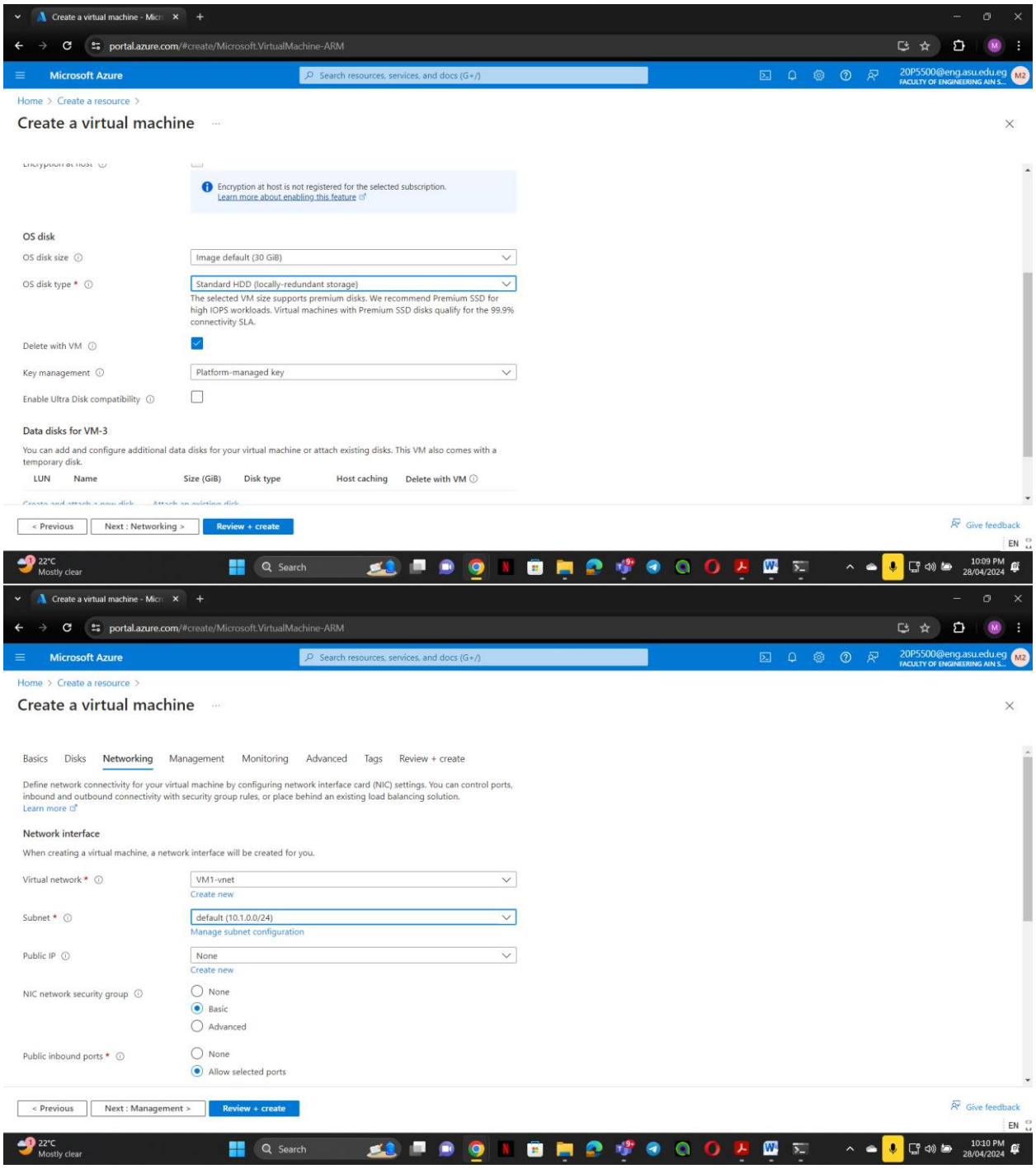


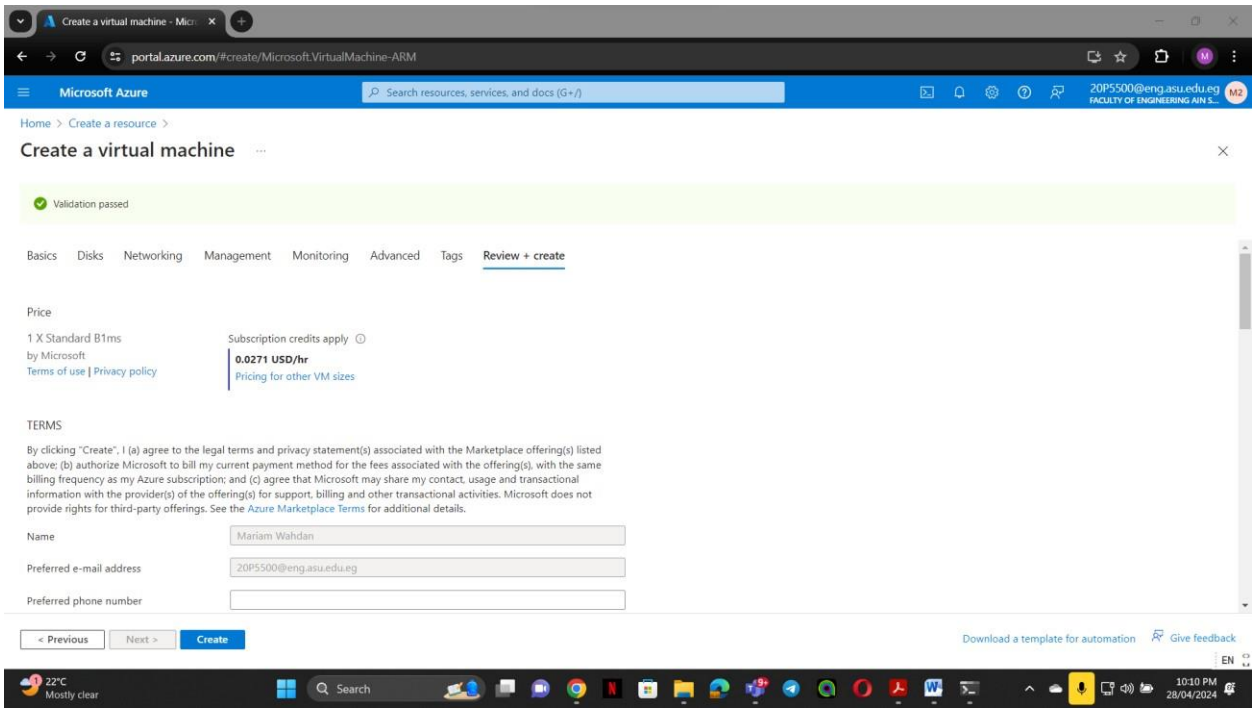
Figure 20 Downloading Image after Showing the result of image processing

2.3 Azure

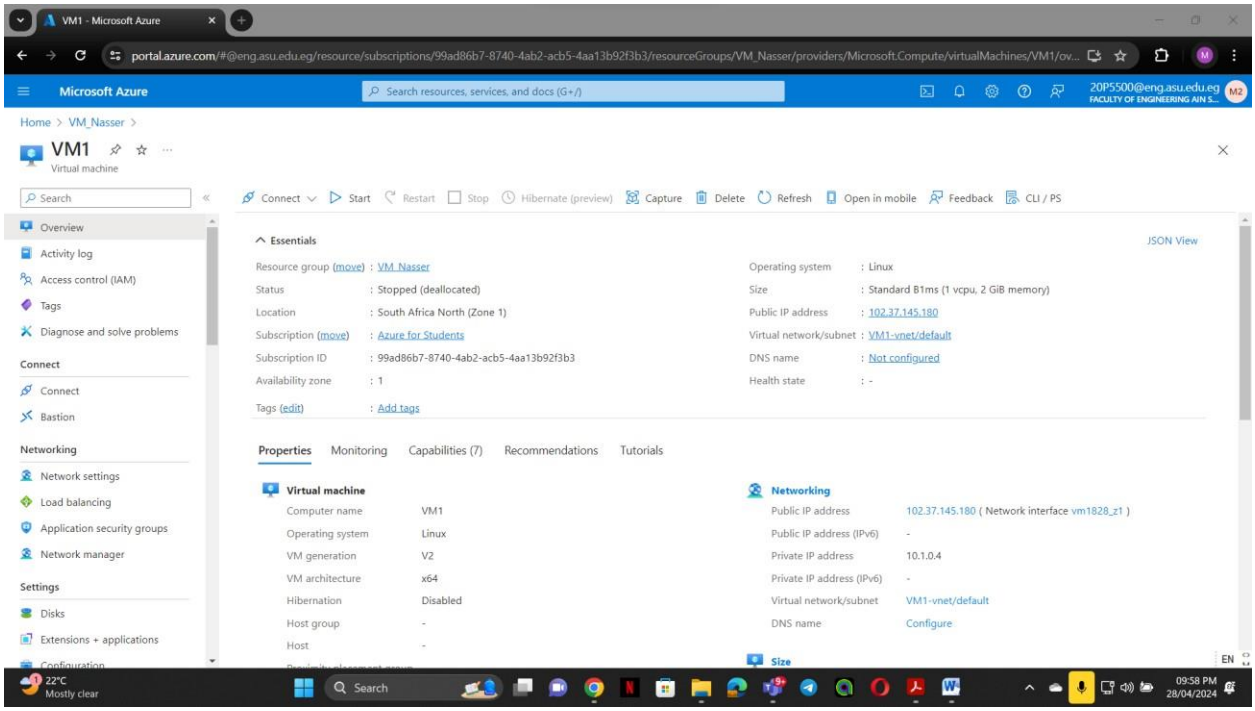
2.3.1: How to create a virtual machine

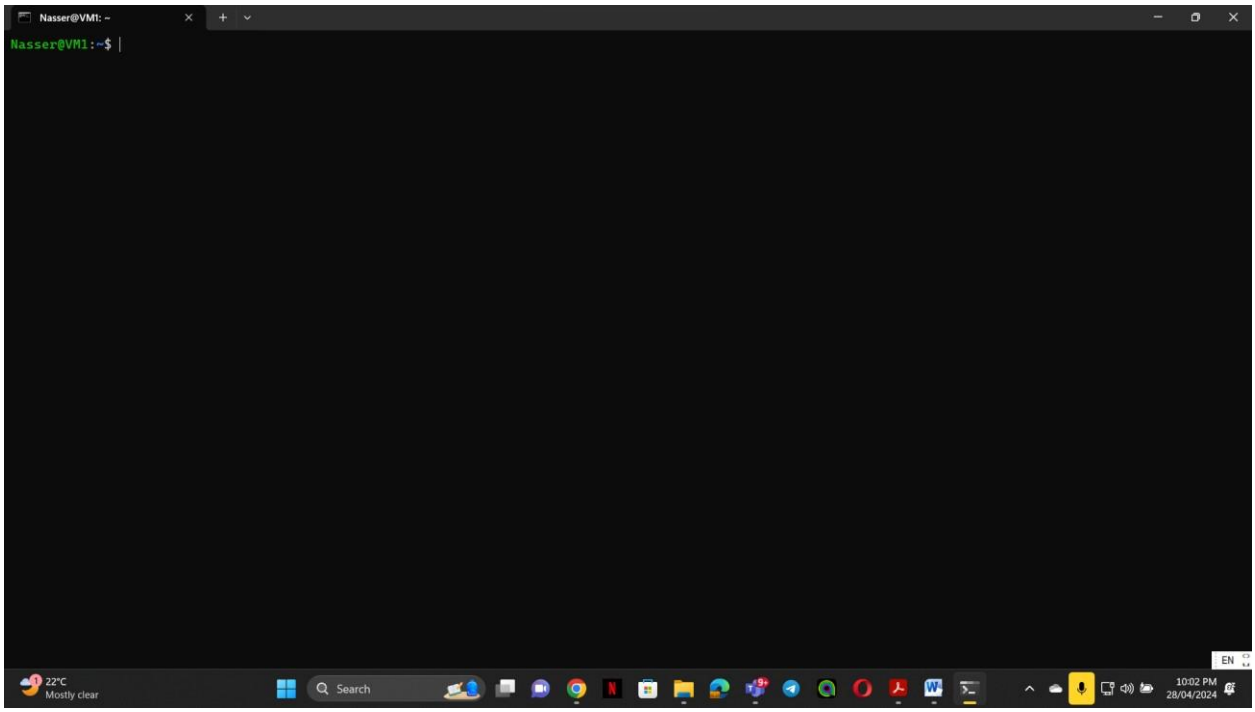



Create a virtual machine - Micr
portal.azure.com/#create/Microsoft.VirtualMachine-ARM
Microsoft Azure
Search resources, services, and docs (G+/)
20P5500@eng.asu.edu.eg
FACULTY OF ENGINEERING AIN S...
Home > Create a resource >
Create a virtual machine
Encryption at host is not registered for the selected 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
< Previous
Next : Networking >
Review + create
Give feedback
EN
22°C
Mostly clear
Search
10:09 PM
28/04/2024
Create a virtual machine - Micr
portal.azure.com/#create/Microsoft.VirtualMachine-ARM
Microsoft Azure
Search resources, services, and docs (G+/)
20P5500@eng.asu.edu.eg
FACULTY OF ENGINEERING AIN S...
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
None
Basic
Advanced
Public inbound ports *
None
Allow selected ports
< Previous
Next : Management >
Review + create
Give feedback
EN
22°C
Mostly clear
Search
10:10 PM
28/04/2024

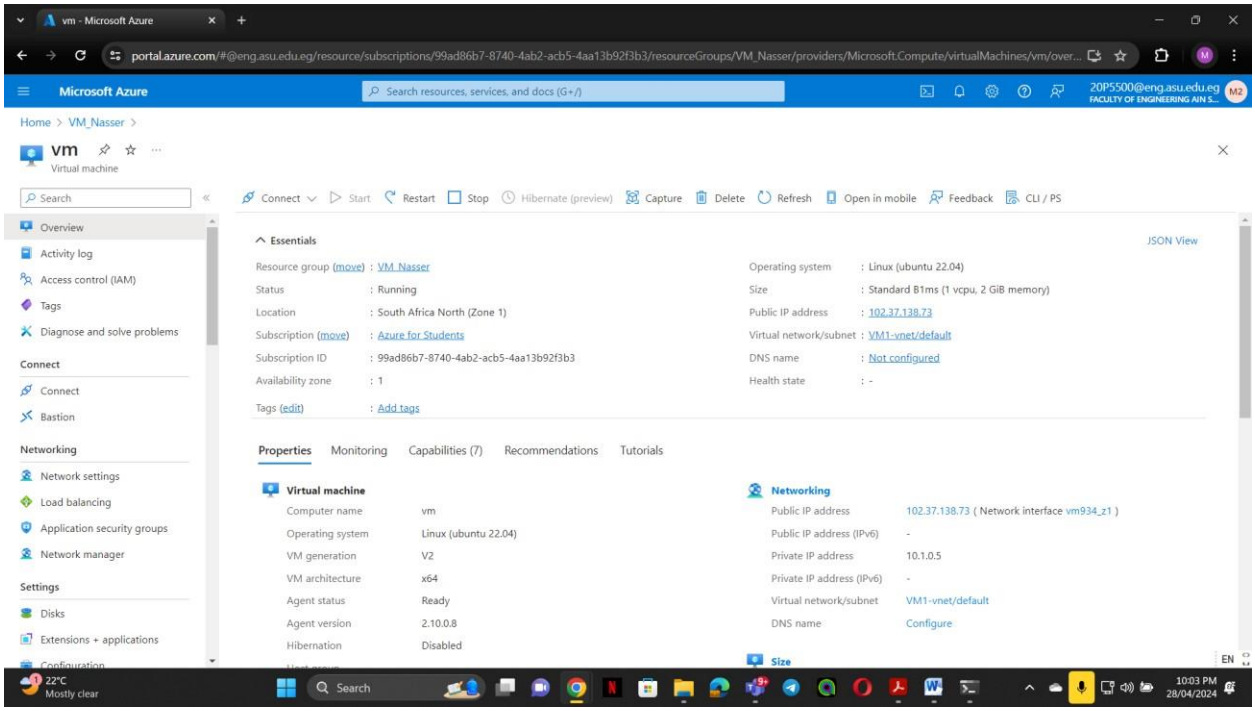


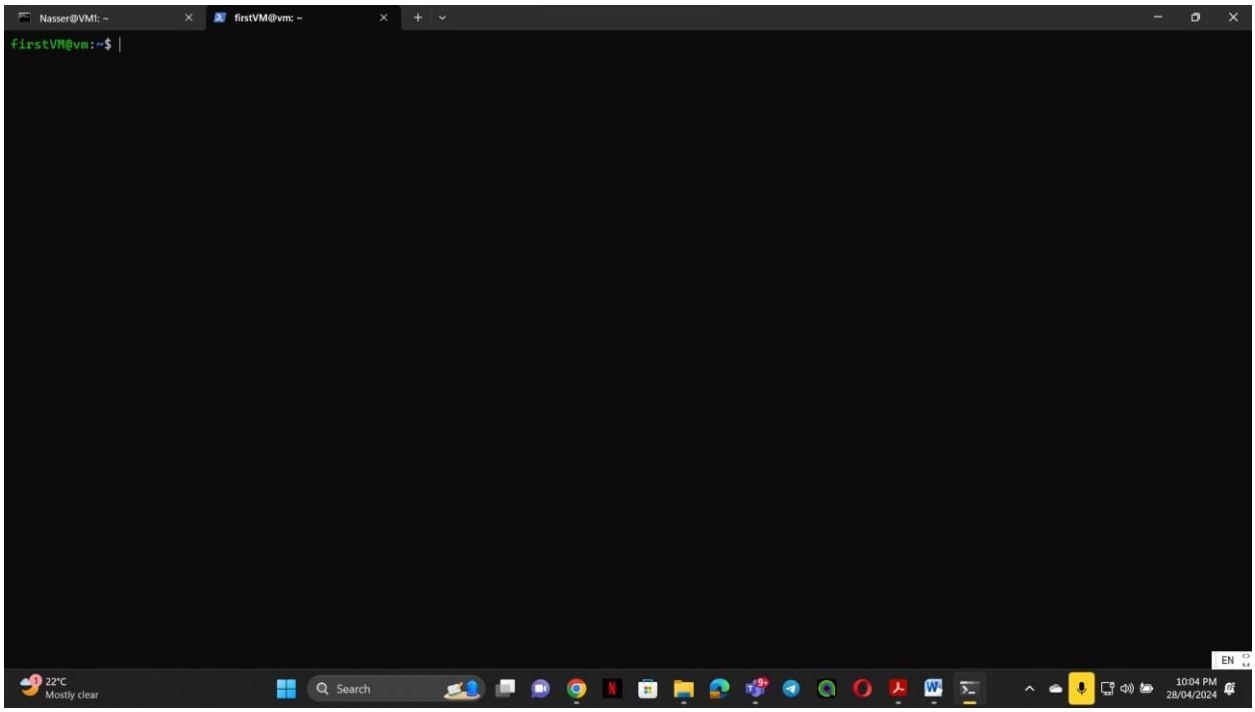
2.3.2 First VM (Master Node)



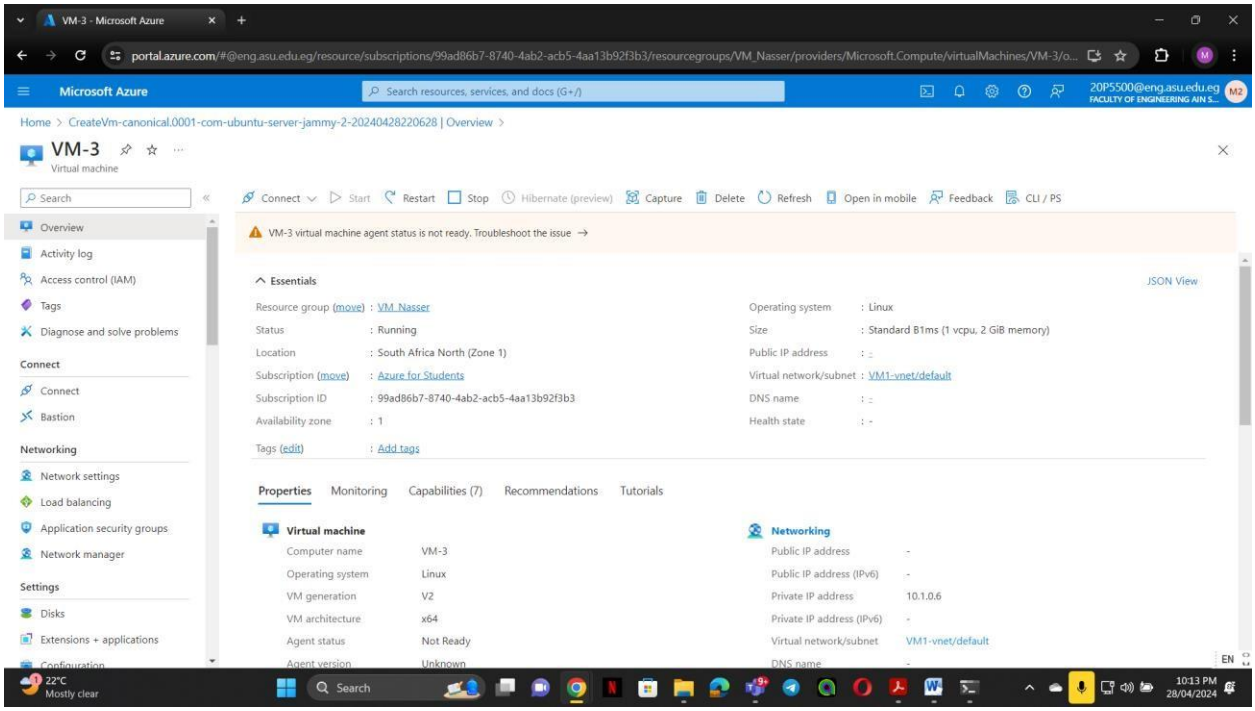


2.3.3 Second VM




Nasser@VM1: ~
firstVM@vm: ~
firstVM@vm:~$
22°C
Mostly clear
Search
EN
10:04 PM
28/04/2024

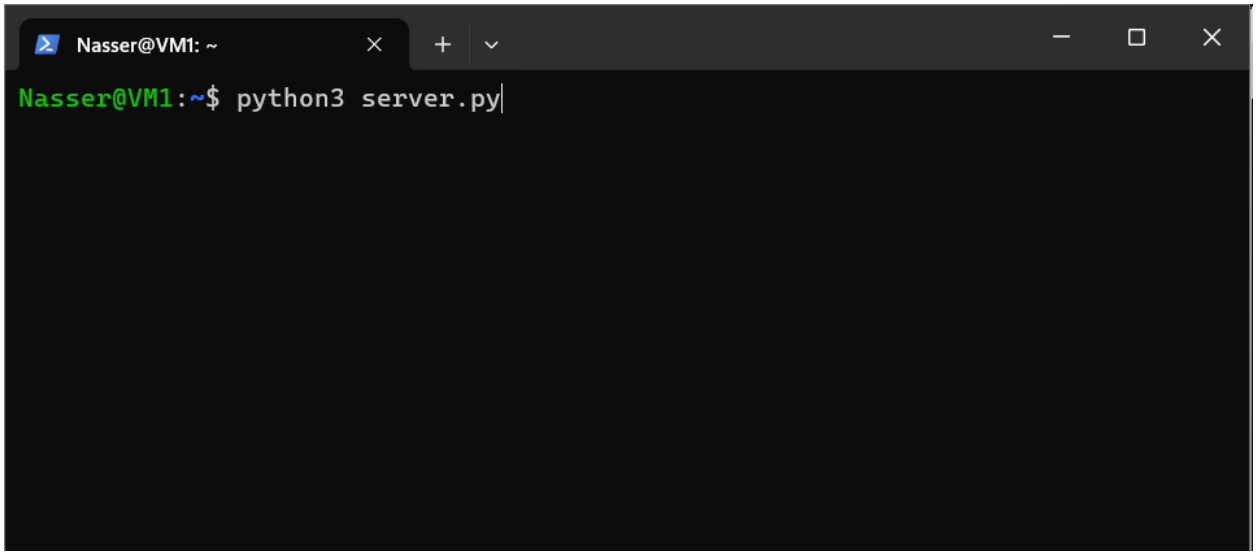
2.3.4 Third VM

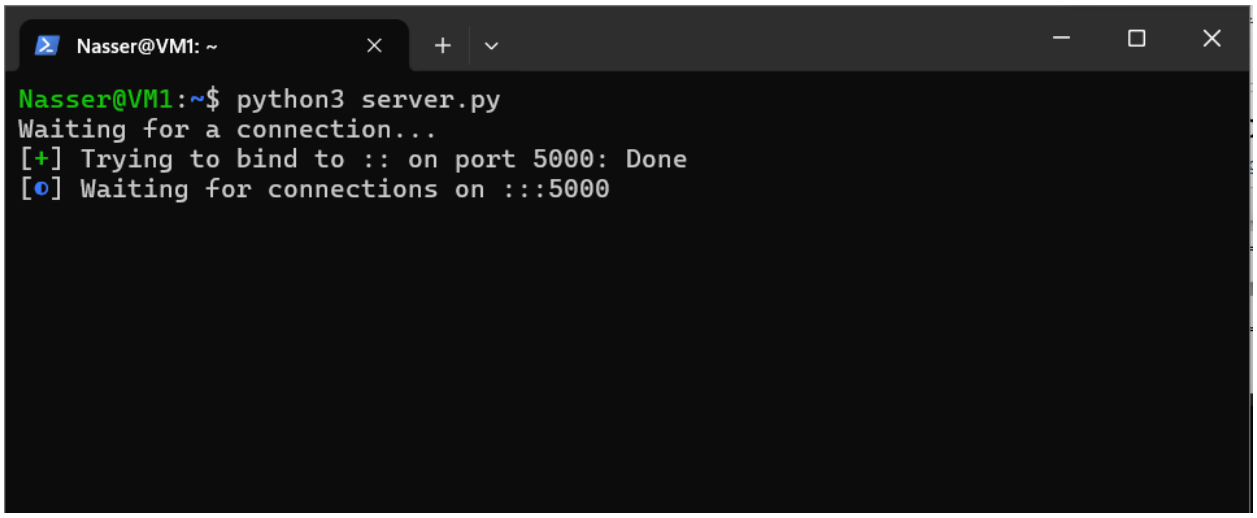

VM-3 - Microsoft Azure
portal.azure.com/#@eng.asu.edu.eg/resource/subscriptions/99ad86b7-8740-4ab2-acb5-4aa13b92f3b3/resourcegroups/VM_Nasser/providers/Microsoft.Compute/virtualMachines/VM-3/o...
Microsoft Azure
Search resources, services, and docs (G+/)
20P5500@eng.asu.edu.eg
FACULTY OF ENGINEERING AIN S...
Home > CreateVm-canonical.0001-com-ubuntu-server-jammy-2-20240428220628 | Overview >
VM-3
Virtual machine
Connect
Start
Restart
Stop
Hibernate (preview)
Capture
Delete
Refresh
Open in mobile
Feedback
CLI / PS
VM-3 virtual machine agent status is not ready. Troubleshoot the issue
Overview
Activity log
Access control (IAM)
Tags
Diagnose and solve problems
Connect
Connect
Bastion
Networking
Network settings
Load balancing
Application security groups
Network manager
Settings
Disks
Extensions + applications
Essentials
JSON View
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
Operating system : Linux
Size : Standard B1ms (1 vcpu, 2 GiB memory)
Public IP address : -
Virtual network/subnet : VM1-vnet/default
DNS name : -
Health state : -
Properties
Monitoring
Capabilities (7)
Recommendations
Tutorials
Virtual machine
Computer name VM-3
Operating system Linux
VM generation V2
VM architecture x64
Agent status Not Ready
Agent version Unknown
Networking
Public IP address -
Public IP address (IPv6) -
Private IP address 10.1.0.6
Private IP address (IPv6) -
Virtual network/subnet VM1-vnet/default
DNS name -
22°C
Mostly clear
Search
EN
10:13 PM
28/04/2024

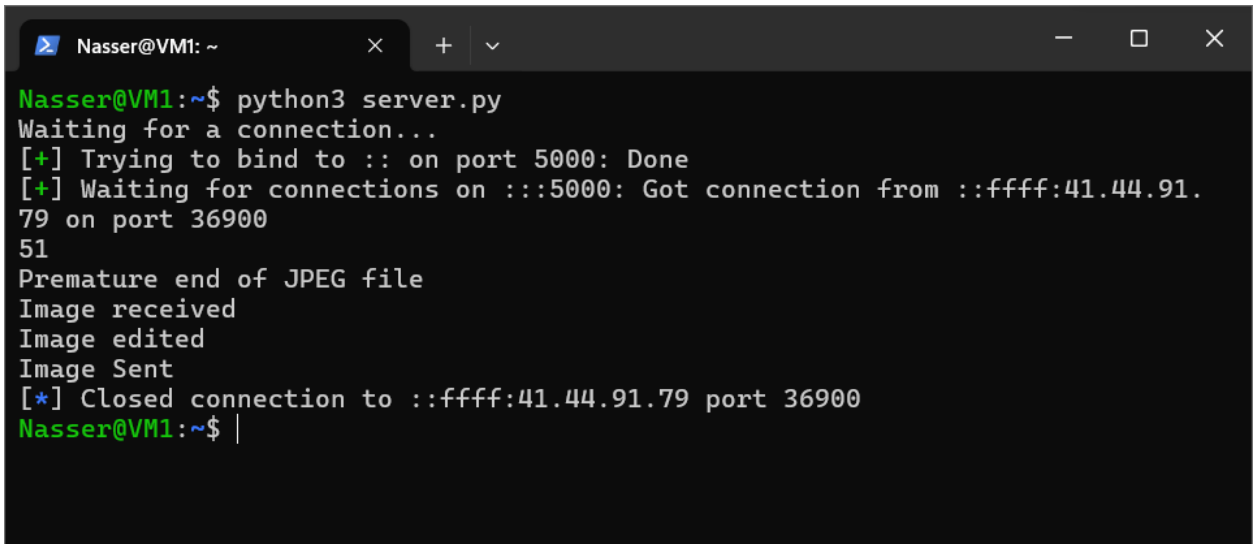
3. Phase 3

3.1 Run

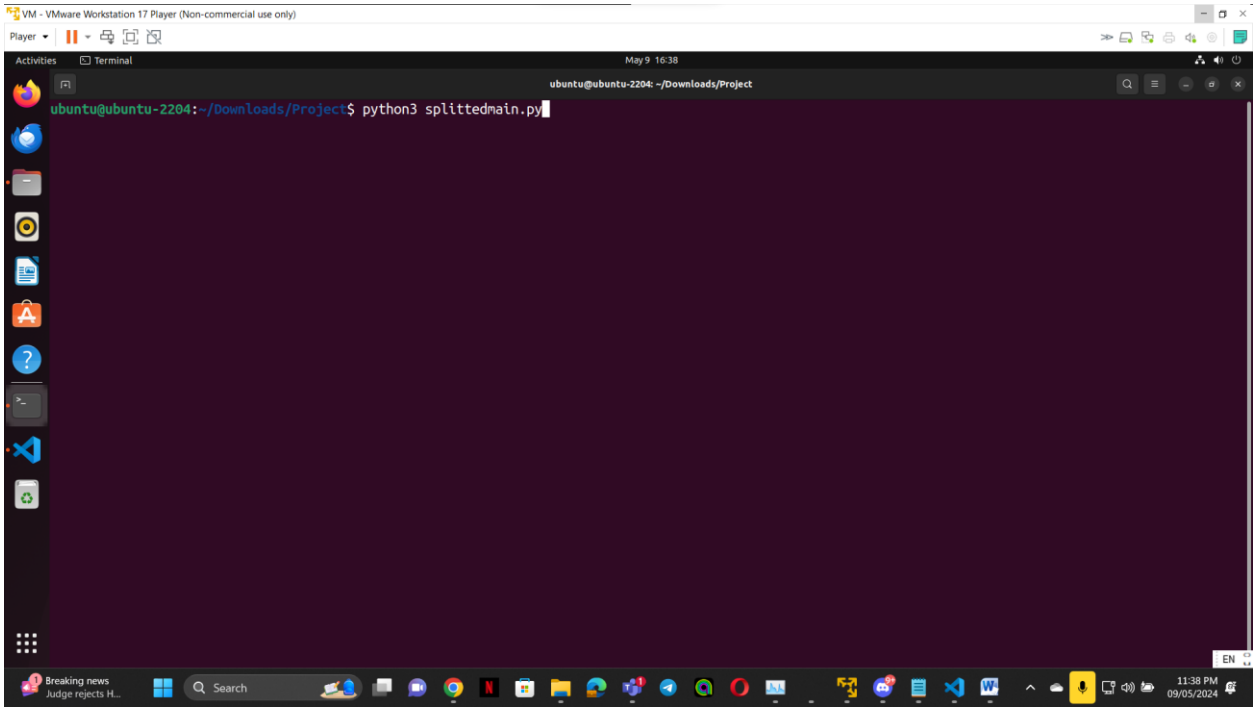
3.1.1 How to run server



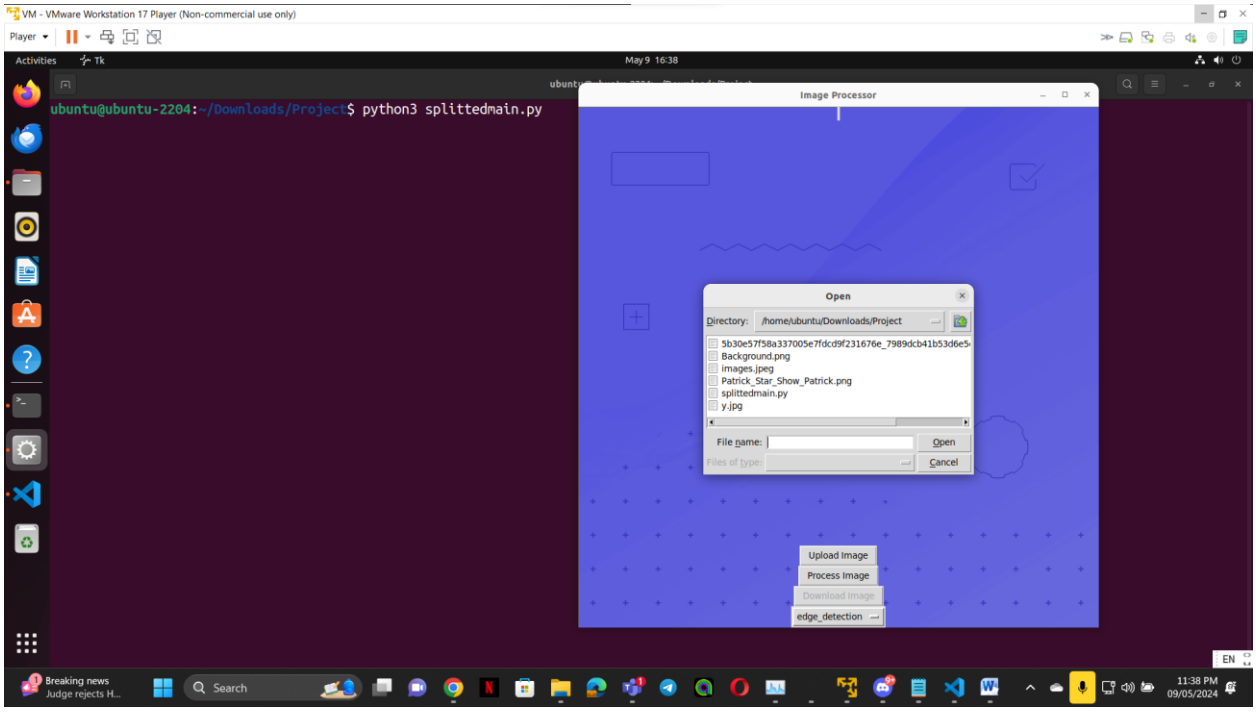


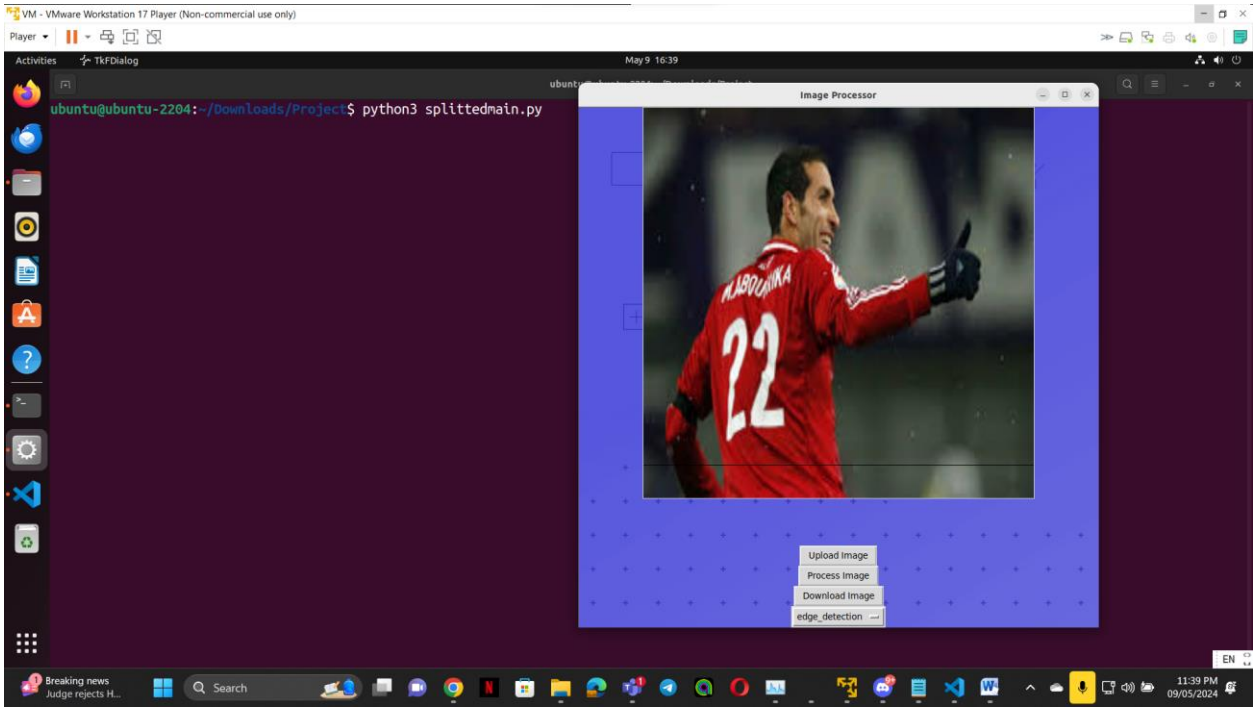


3.1.2 Run Client

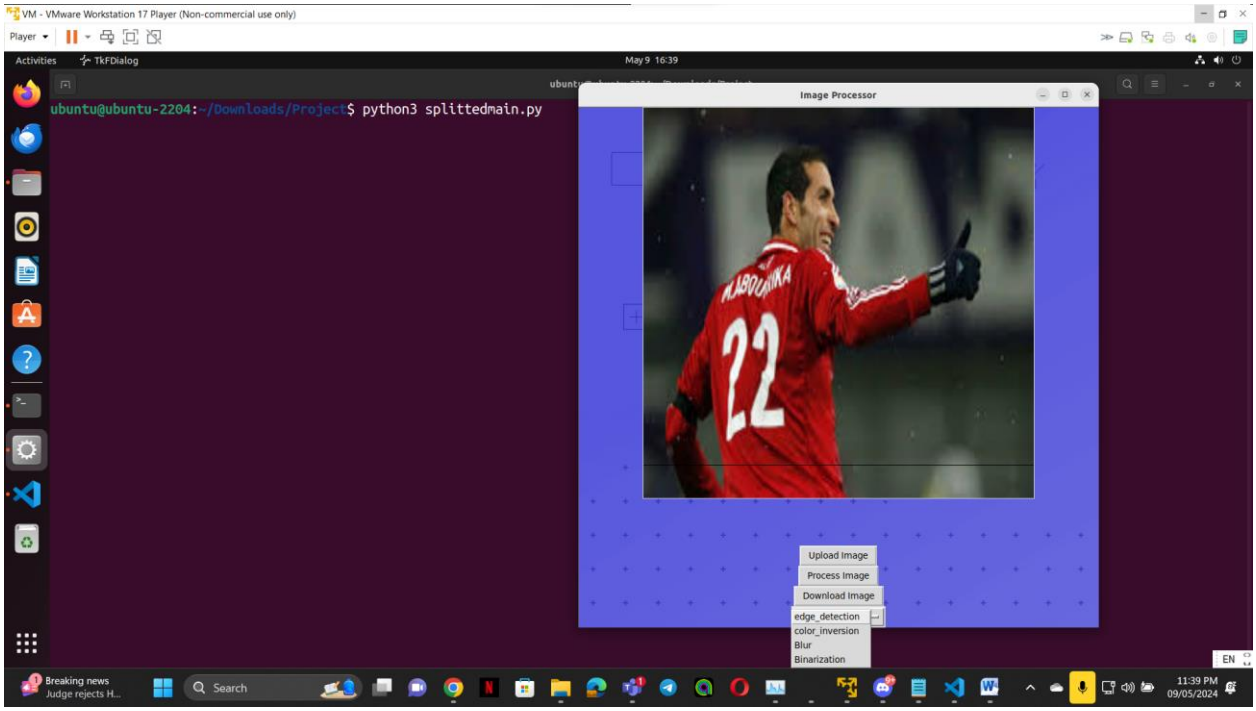


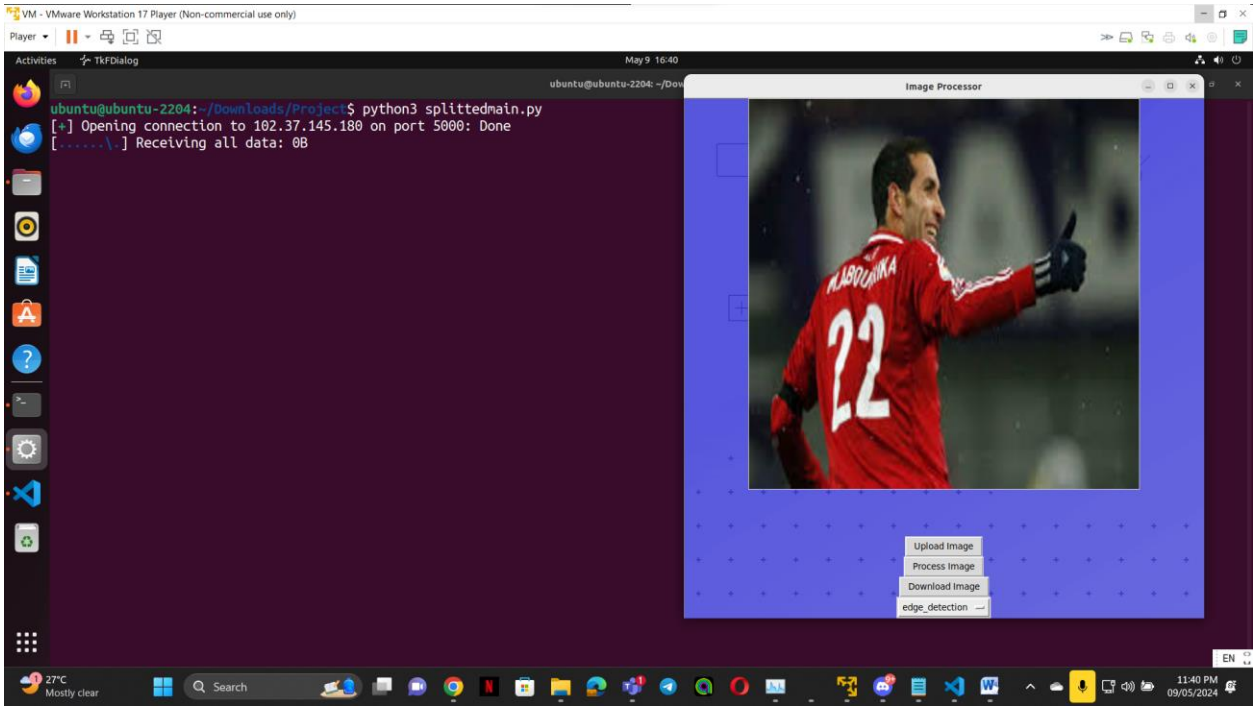
Upload Image



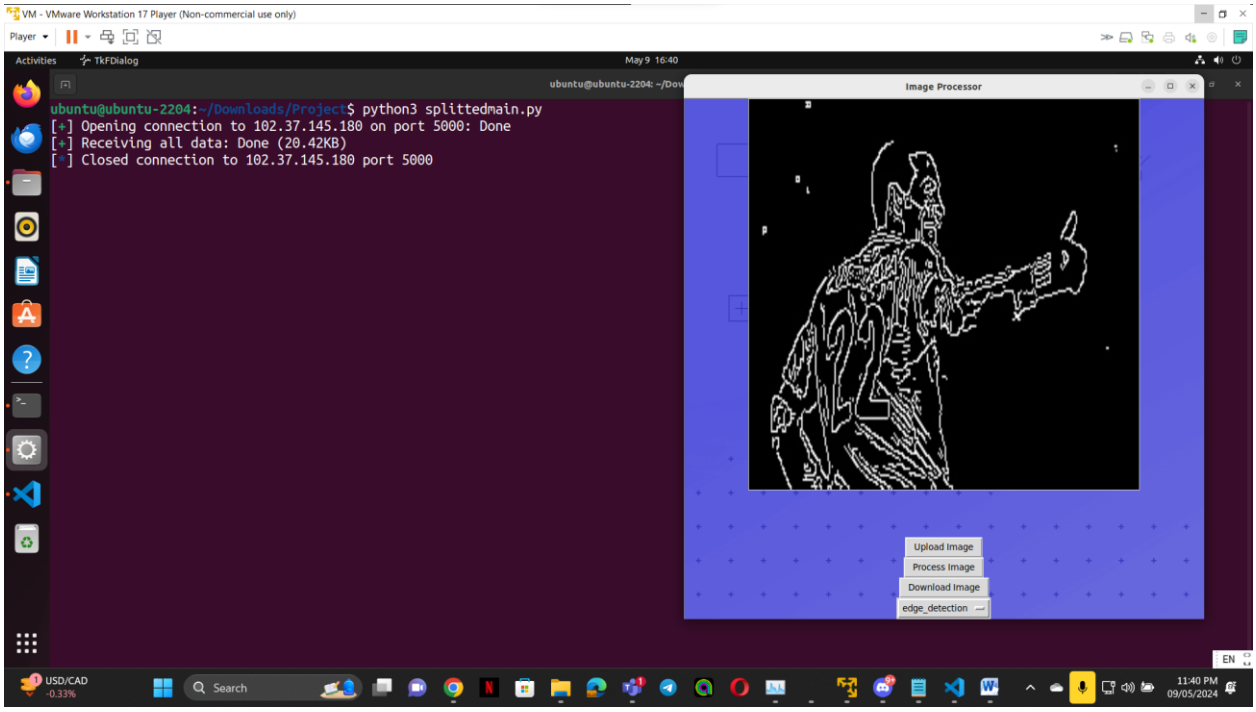


Choose Operation:

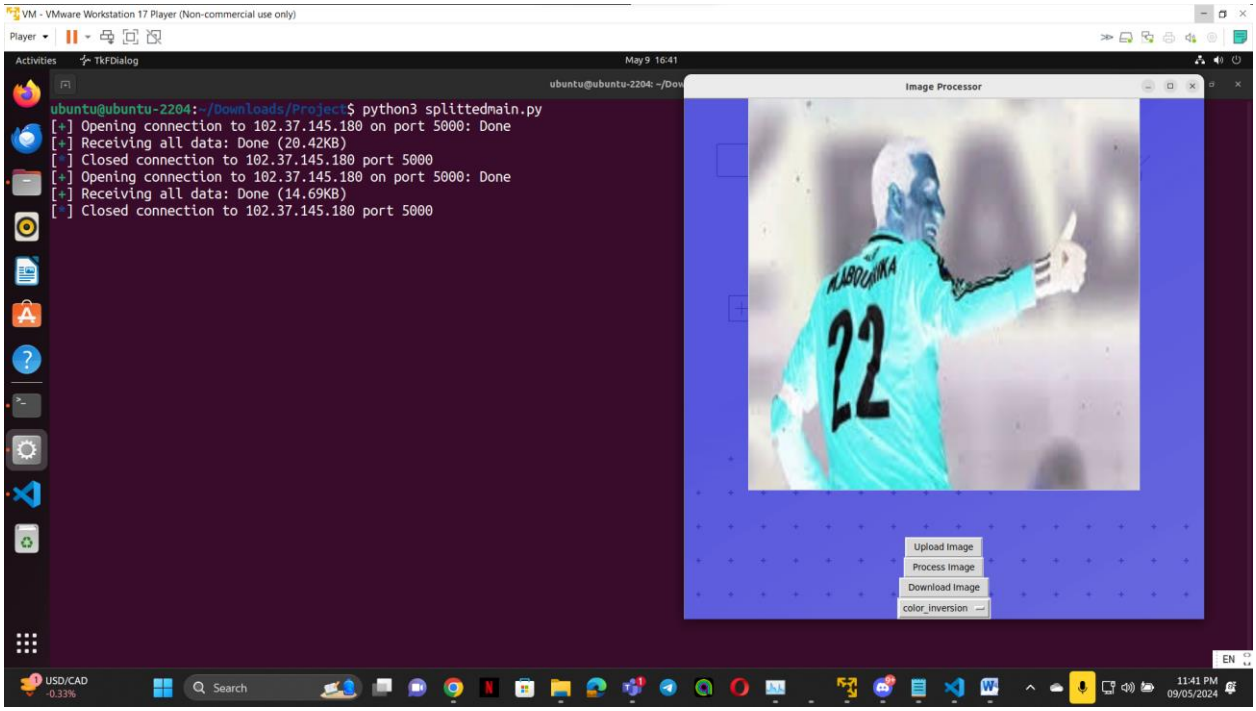




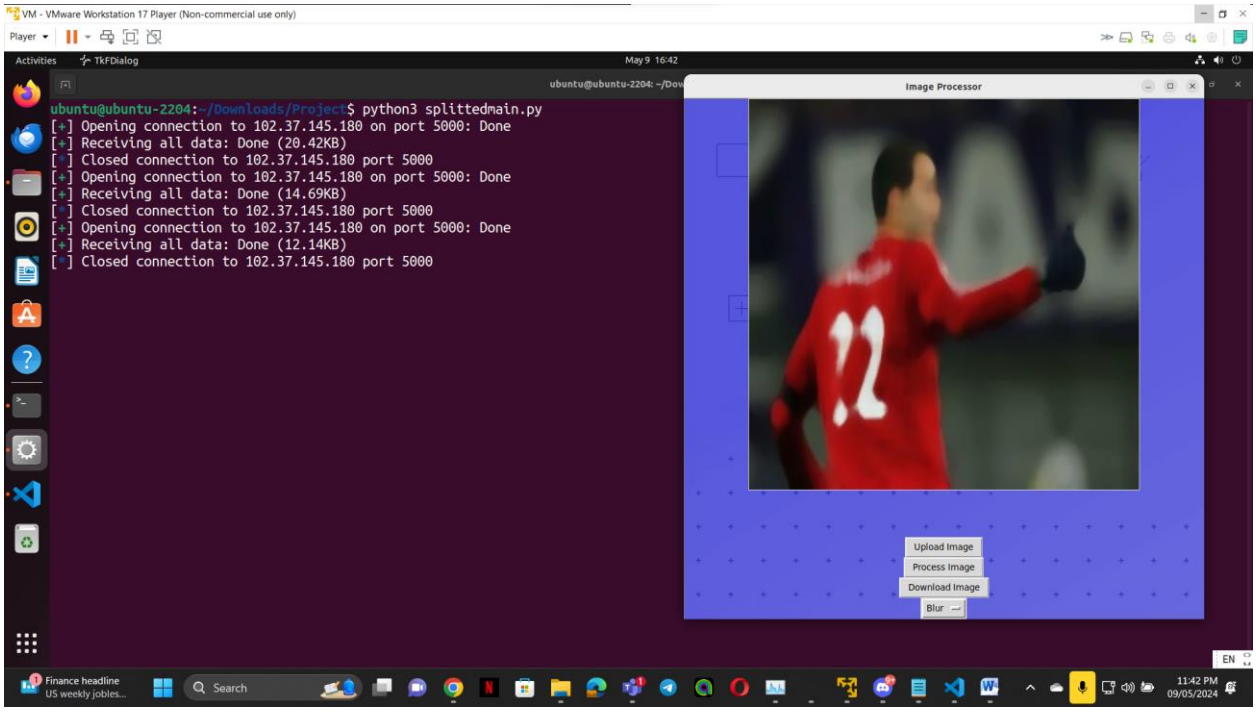
Edge detection



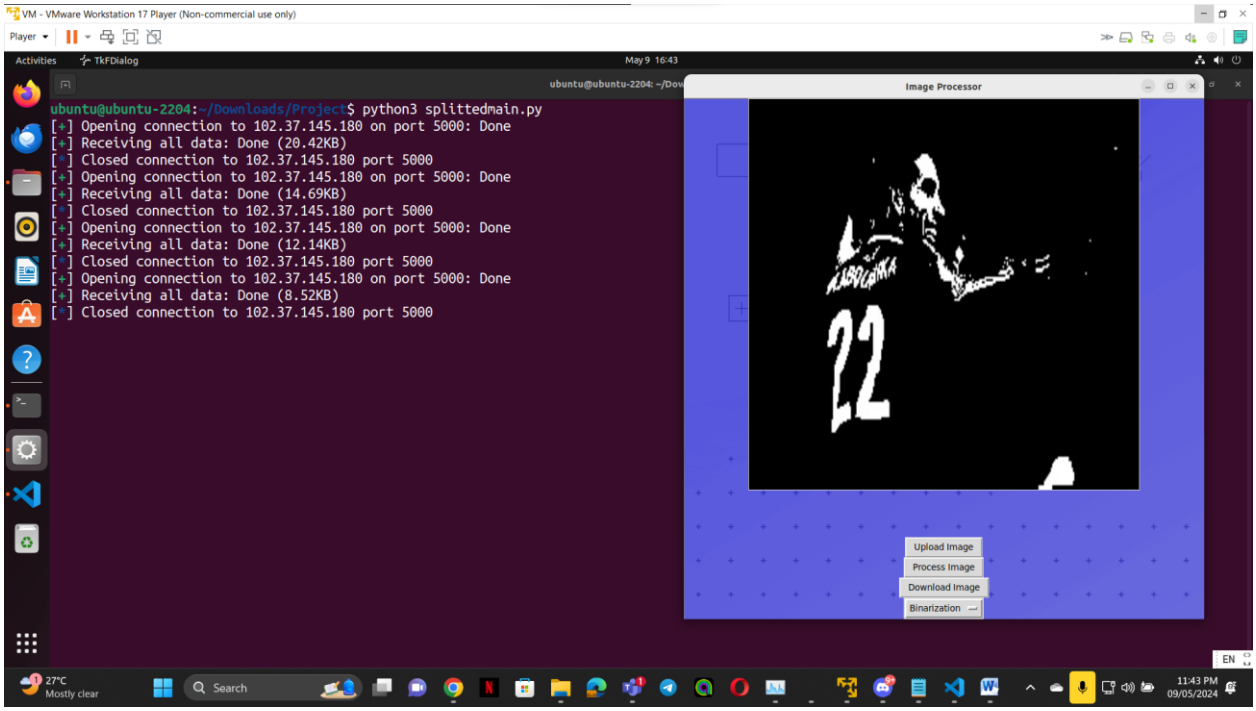
Color Inversion



Blur

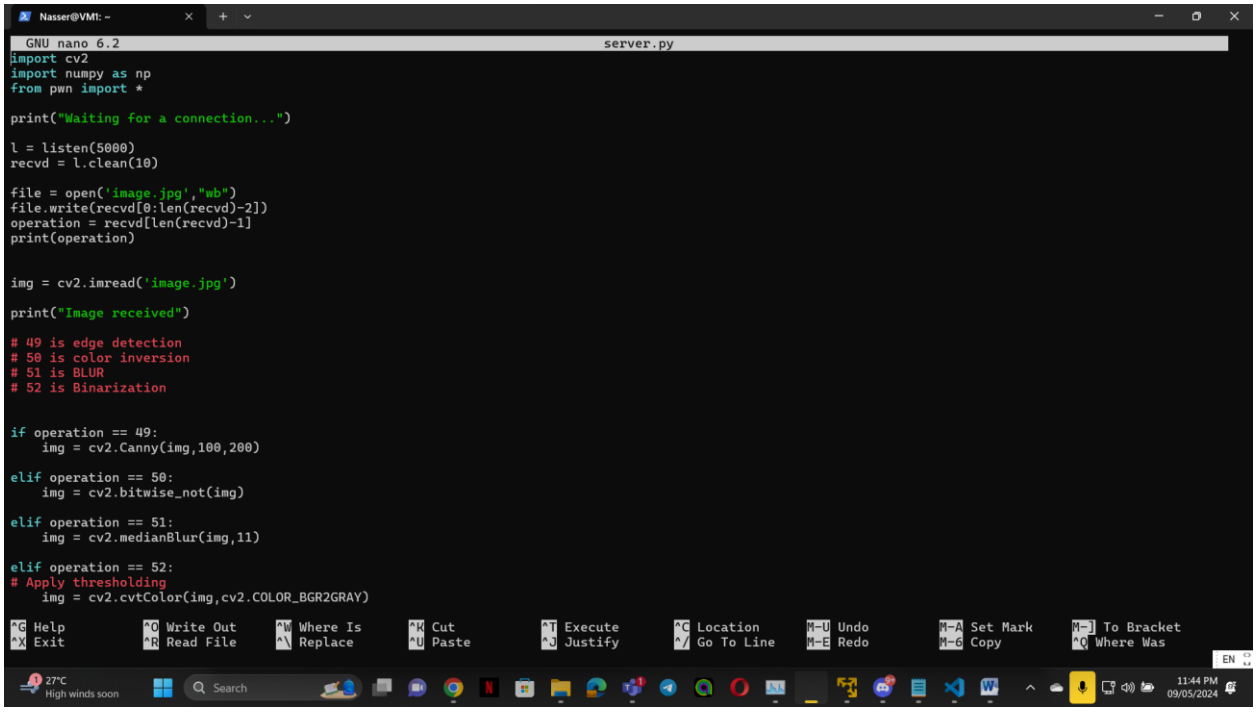


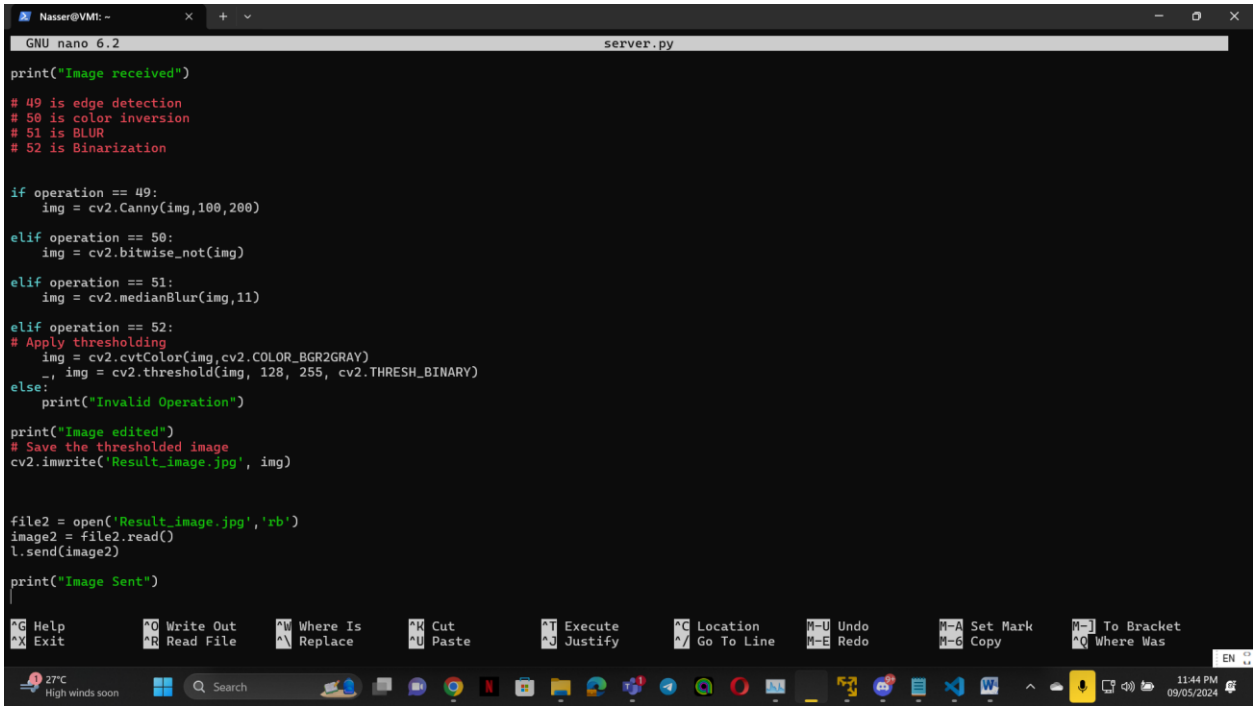
Binarization



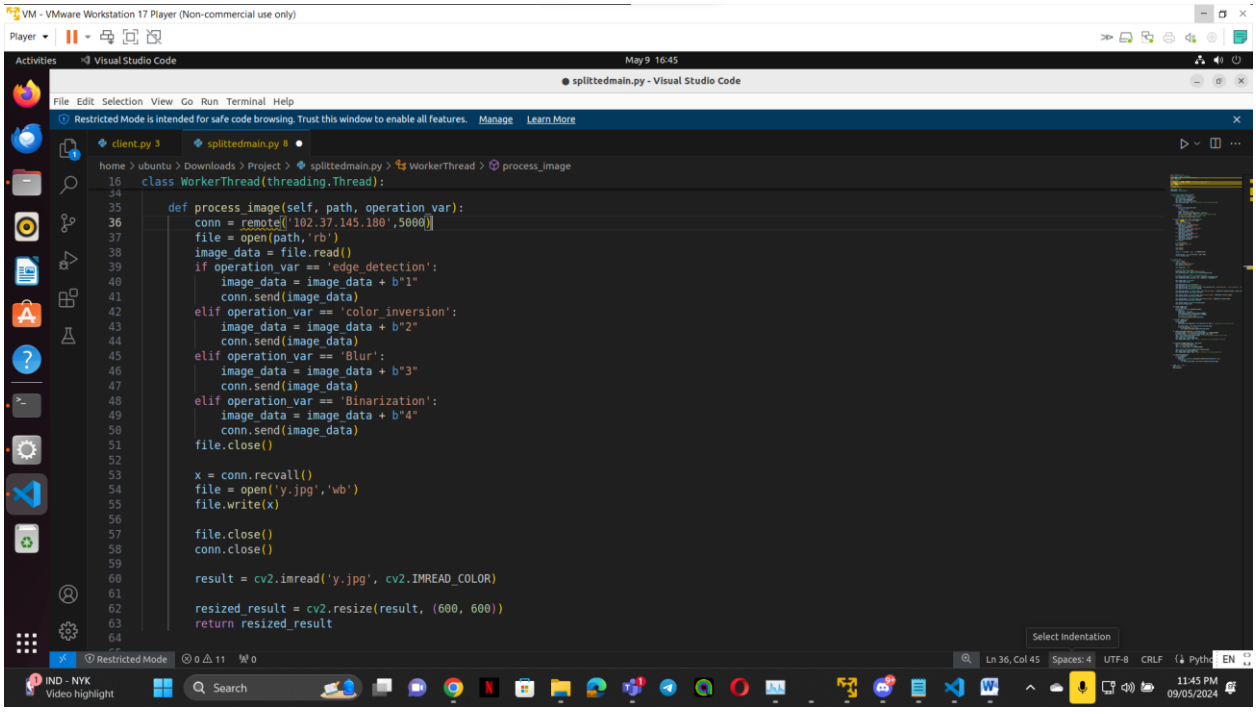
3.2 Code

3.2.1 Server




Nasser@VM1: ~
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")
^G Help ^O Write Out ^W Where Is ^K Cut ^T Execute ^C Location M-U Undo M-A Set Mark M-] To Bracket
^X Exit ^R Read File ^\ Replace ^U Paste ^J Justify ^/ Go To Line M-E Redo M-6 Copy ^Q Where Was

3.2.2 Client


splittedmain.py - Visual Studio Code
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

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:

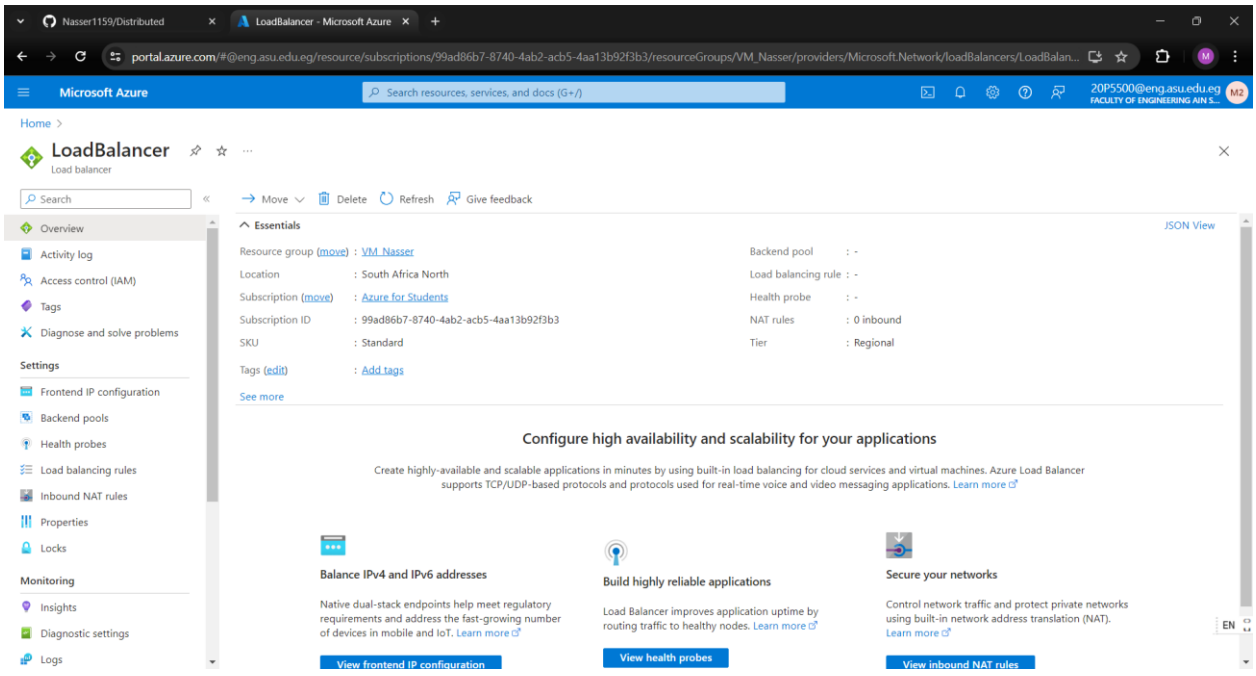
- The application is designed to run with 3 MPI processes (1 master and 2 workers).
- The master process receives the image and operation from the client, splits the image, and sends parts to the worker processes.
- Worker processes apply the selected operation and return the processed parts to the master,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

4.2 Testing

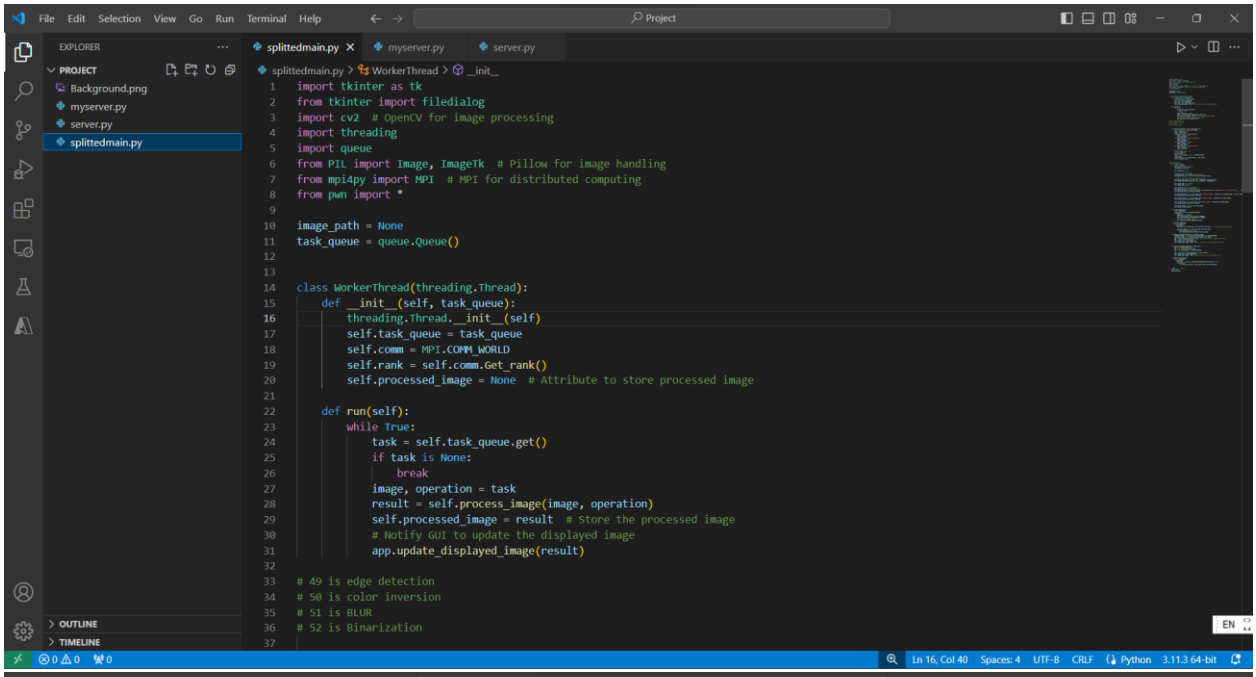
```python
class TestImageProcessor(unittest.TestCase):

    def setUp(self):
        self.app = GUI()  # Create an instance of the GUI class
        self.task_queue = queue.Queue()  # Mock the task queue

    def tearDown(self):
        self.app.destroy()  # Destroy the GUI window

    # Test cases for GUI functionality
    def test_upload_valid_image(self):
        valid_image_path = "path/to/valid_image.jpg"
        self.app.upload_image()
        self.assertIsNotNone(self.app.image_label.image)  # Check if image is displayed

    def test_upload_invalid_image(self):
        invalid_image_path = "path/to/invalid.txt"
        with self.assertRaises(Exception):  # Expect an error (modify if specific)
            self.app.upload_image()

    def test_process_without_image(self):
        with self.assertRaises(Exception):  # Expect an error (modify if specific)
            self.app.process_image()
```

```python
    # Test processing functionality - No extra argument needed for process_image
    def test_process_edge_detection(self):
        valid_image_path = "path/to/valid_image.jpg"
        self.app.upload_image()
        self.app.process_image()

        processed_image = self.app.get_processed_image()
        self.assertTrue(processed_image is not None)

    # ... (similar test cases for other processing operations)

    def test_download_without_processing(self):
        with self.assertRaises(Exception):  # Expect an error (modify if specific)
            self.app.download_image()

    def test_download_after_processing(self):
        valid_image_path = "path/to/valid_image.jpg"
        self.app.upload_image()
        self.app.process_image()
        # Simulate successful download (implementation omitted for testing purposes)
        self.app.download_image()
        # No assertions here as download functionality is not directly tested
```

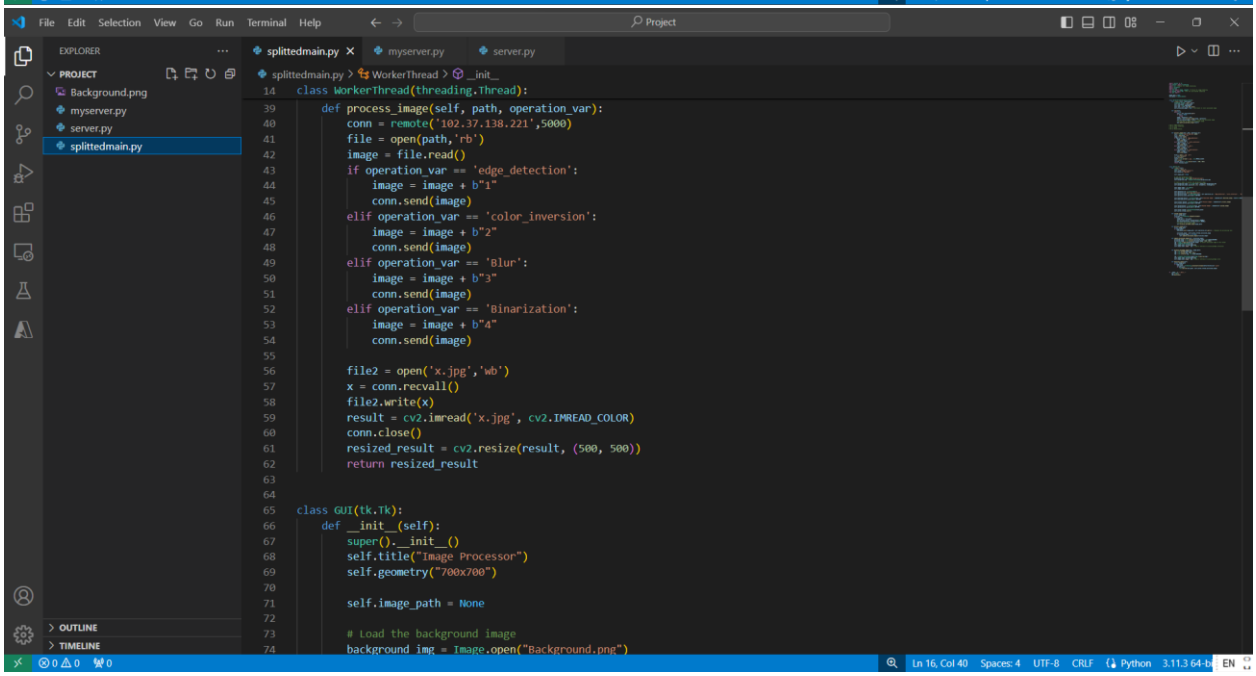
4.3 Scalability



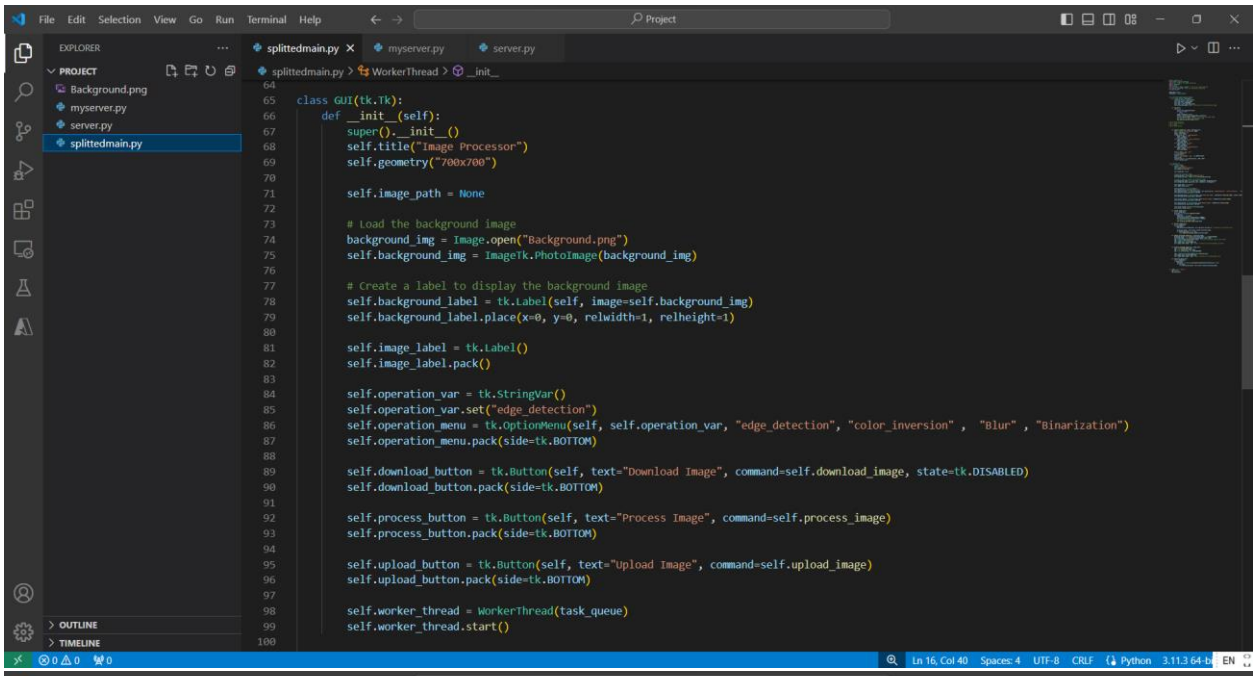
Client Code:



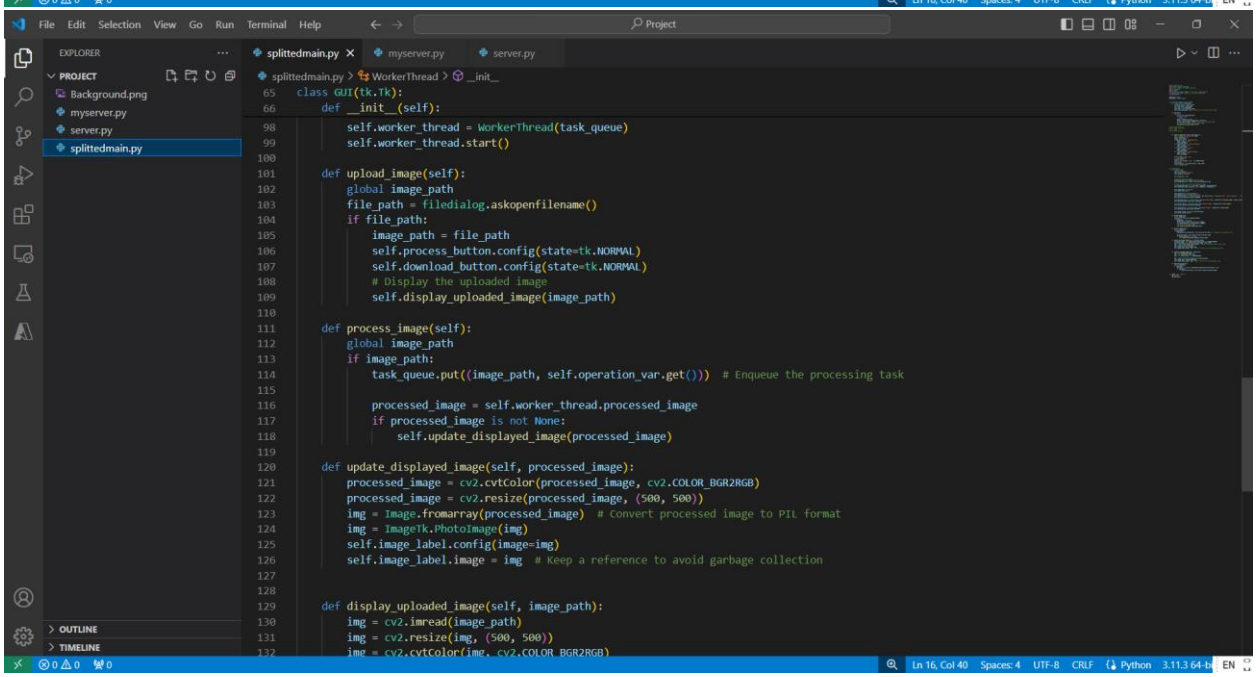
```python
import tkinter as tk
from tkinter import filedialog
import cv2  # OpenCV for image processing
import threading
import queue
from PIL import Image, ImageTk  # Pillow for image handling
from mpi4py import MPI  # MPI for distributed computing
from pwn import *

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)

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



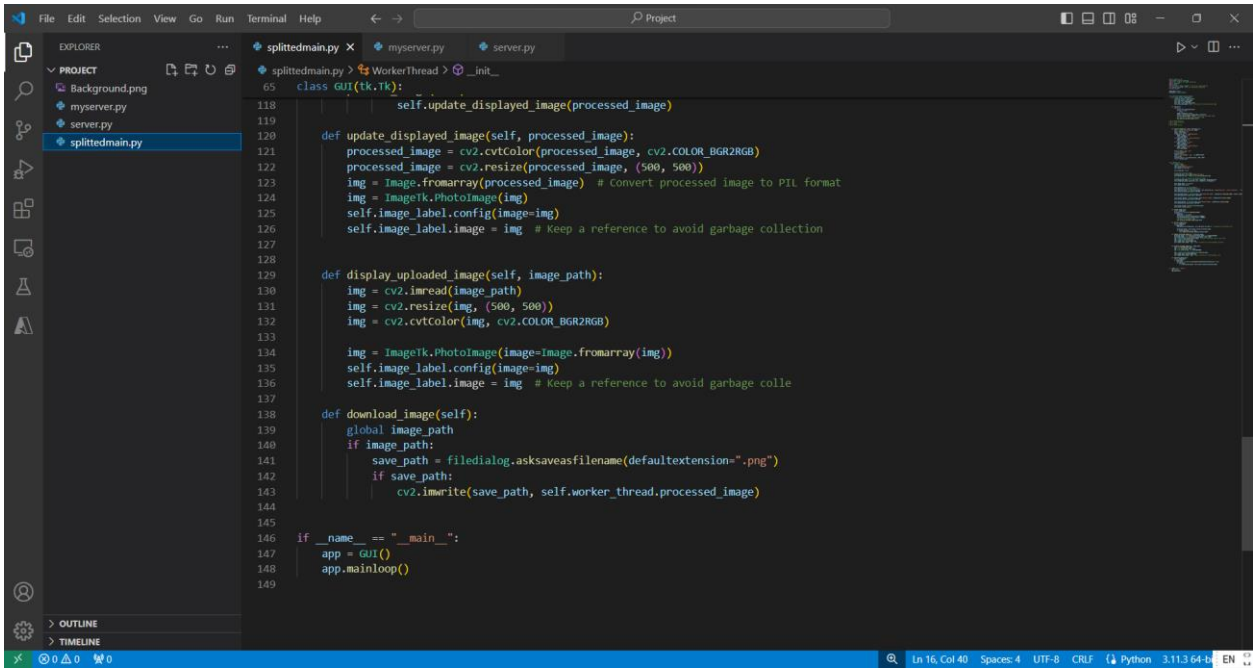
```python
class WorkerThread(threading.Thread):
    def process_image(self, path, operation_var):
        conn = remote('102.37.138.221',5000)
        file = open(path,'rb')
        image = file.read()
        if operation_var == 'edge_detection':
            image = image + b"1"
            conn.send(image)
        elif operation_var == 'color_inversion':
            image = image + b"2"
            conn.send(image)
        elif operation_var == 'Blur':
            image = image + b"3"
            conn.send(image)
        elif operation_var == 'Binarization':
            image = image + b"4"
            conn.send(image)

        file2 = open('x.jpg','wb')
        x = conn.recvall()
        file2.write(x)
        result = cv2.imread('x.jpg', cv2.IMREAD_COLOR)
        conn.close()
        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")
```



```
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" , "Blur" , "Binarization")
        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()
```

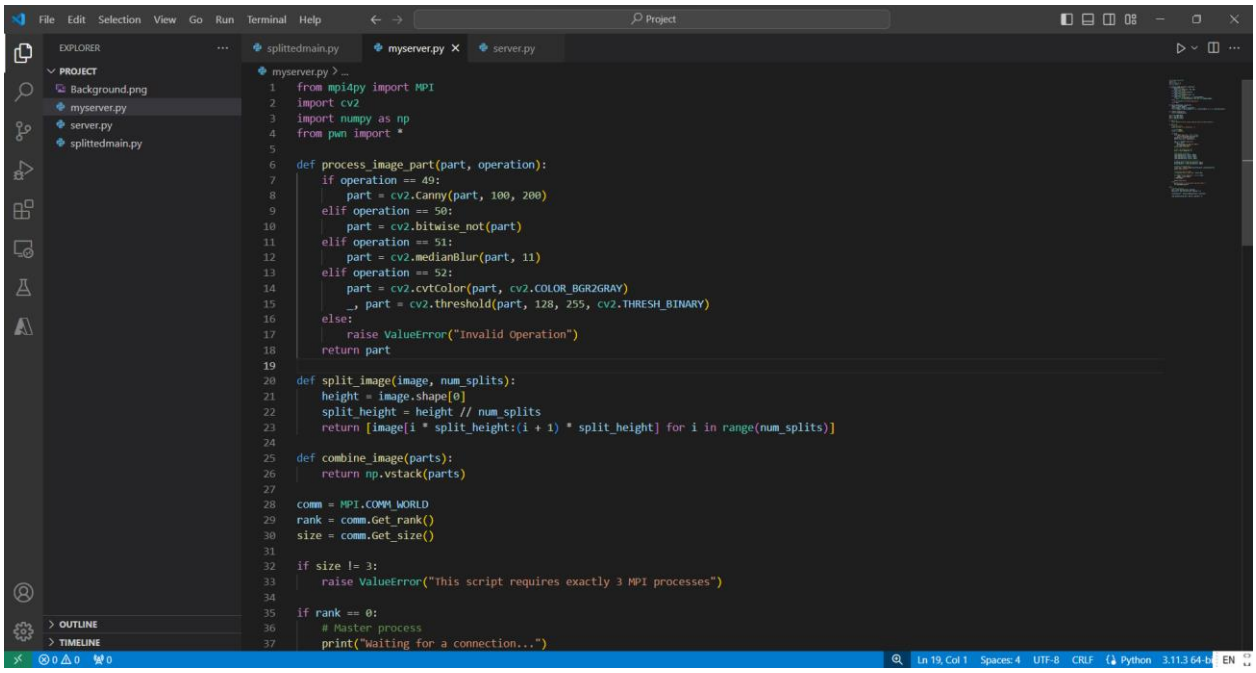


```
    def upload_image(self):
        global image_path
        file_path = filedialog.askopenfilename()
        if file_path:
            image_path = file_path
            self.process_button.config(state=tk.NORMAL)
            self.download_button.config(state=tk.NORMAL)
            # Display the 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)  # Convert processed image to PIL format
        img = ImageTk.PhotoImage(img)
        self.image_label.config(image=img)
        self.image_label.image = img  # Keep a reference to avoid garbage collection


    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)
```

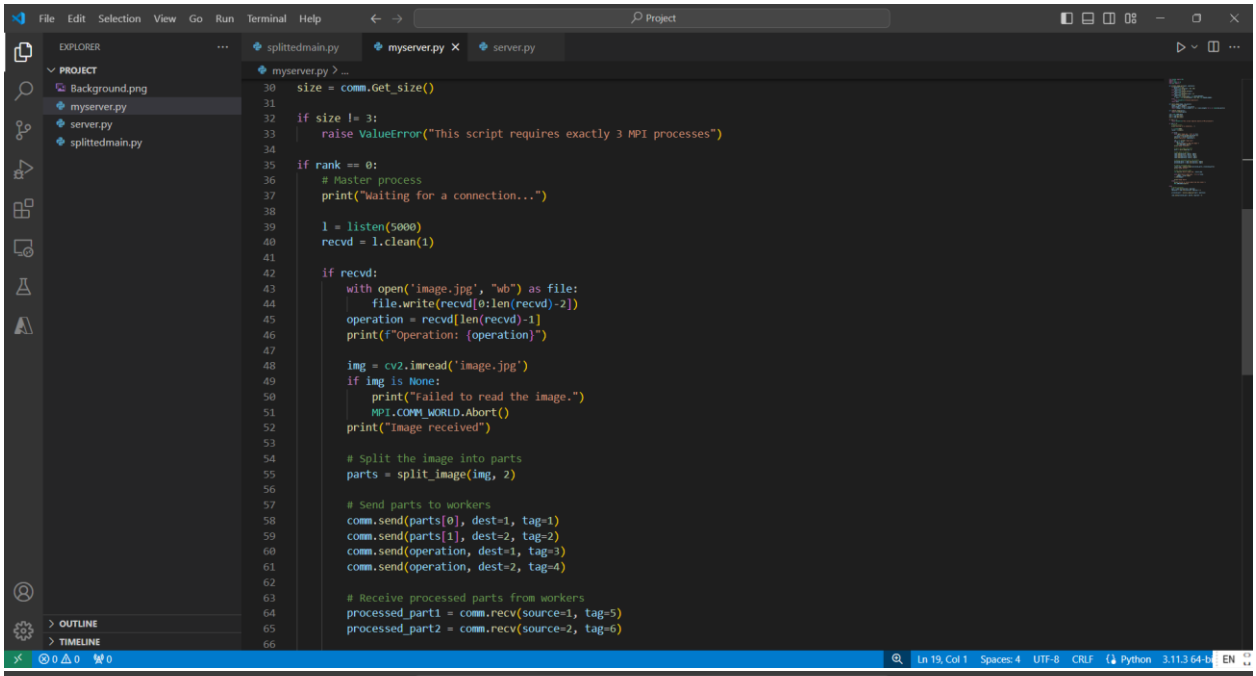


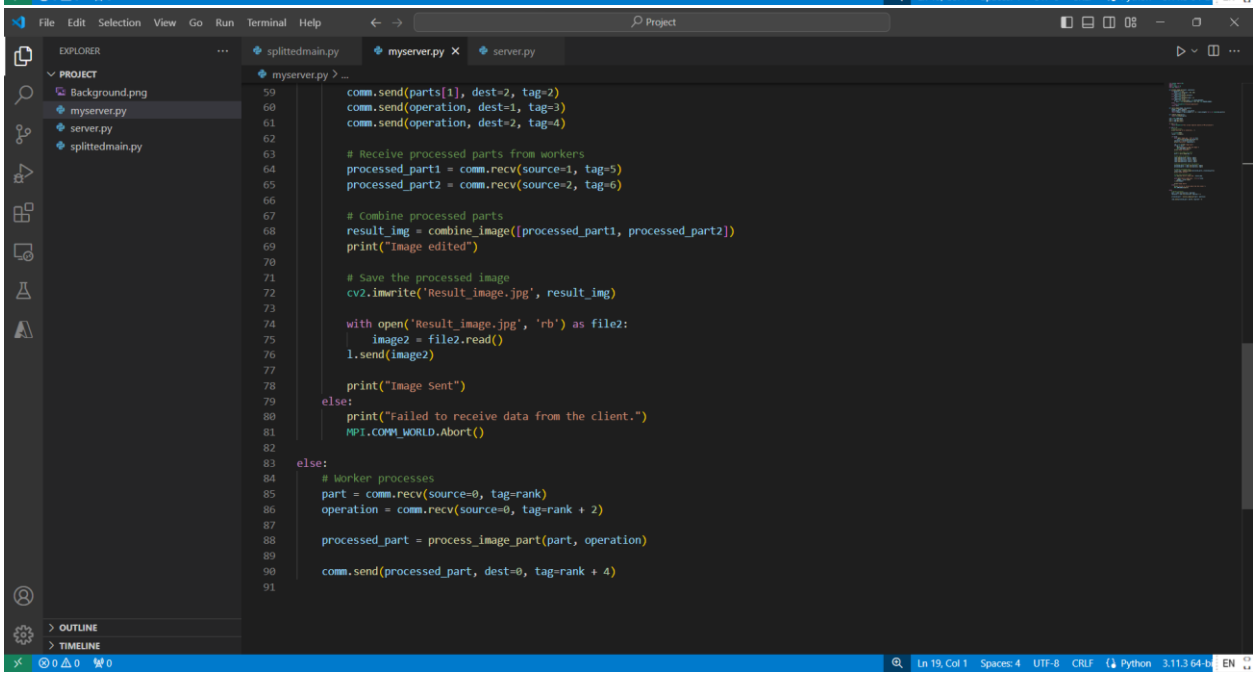
```
                self.update_displayed_image(processed_image)

    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)  # Convert processed image to PIL format
        img = ImageTk.PhotoImage(img)
        self.image_label.config(image=img)
        self.image_label.image = img  # Keep a reference to avoid garbage collection


    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()
```

Code of MPI server:

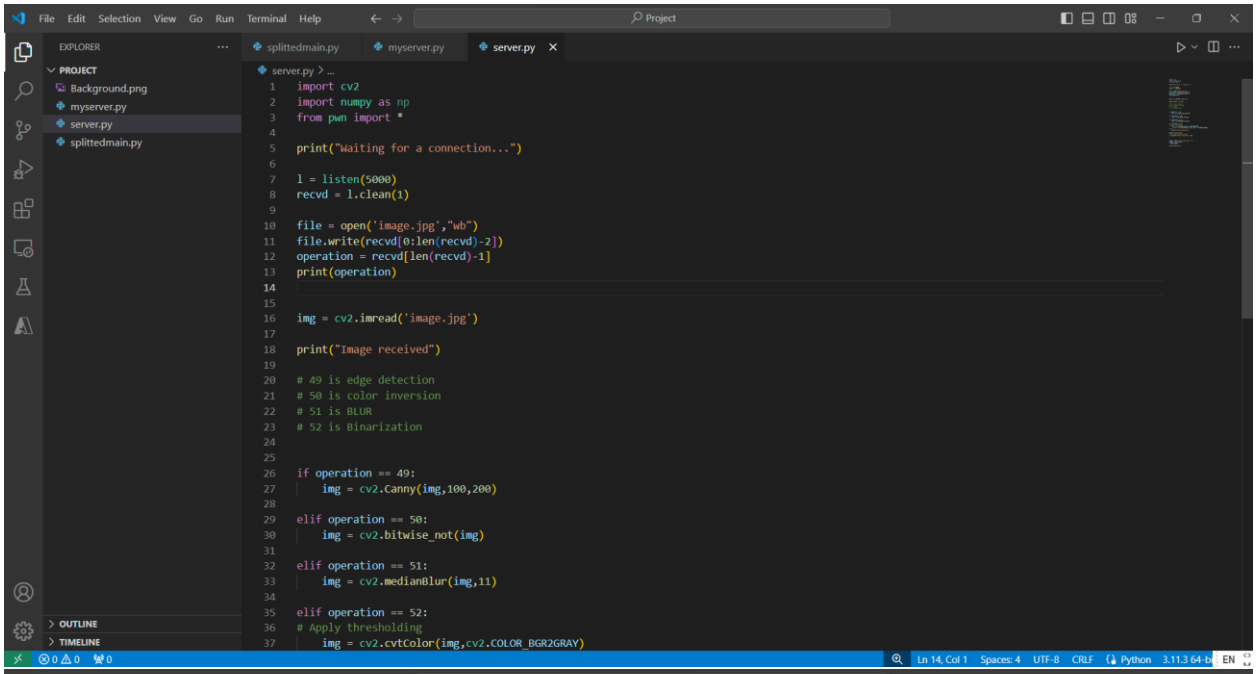


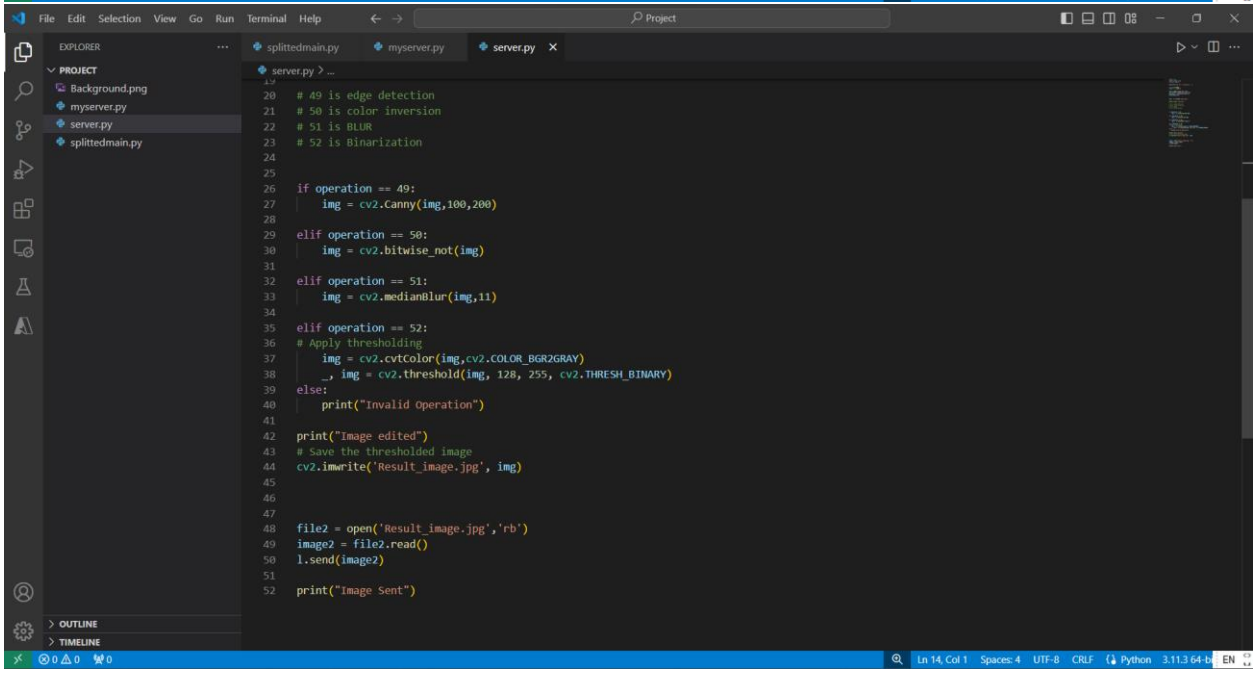
```
from mpi4py import MPI
import cv2
import numpy as np
from pwn import *

def process_image_part(part, operation):
    if operation == 49:
        part = cv2.Canny(part, 100, 200)
    elif operation == 50:
        part = cv2.bitwise_not(part)
    elif operation == 51:
        part = cv2.medianBlur(part, 11)
    elif operation == 52:
        part = cv2.cvtColor(part, cv2.COLOR_BGR2GRAY)
        _, part = cv2.threshold(part, 128, 255, cv2.THRESH_BINARY)
    else:
        raise ValueError("Invalid Operation")
    return part

def split_image(image, num_splits):
    height = image.shape[0]
    split_height = height // num_splits
    return [image[i * split_height:(i + 1) * split_height] for i in range(num_splits)]

def combine_image(parts):
    return np.vstack(parts)

comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()

if size != 3:
    raise ValueError("This script requires exactly 3 MPI processes")

if rank == 0:
    # Master process
    print("Waiting for a connection...")
```


File Edit Selection View Go Run Terminal Help
Project
EXPLORER
splittedmain.py
myserver.py
server.py
PROJECT
Background.png
myserver.py
server.py
splittedmain.py
size = comm.Get_size()
if size != 3:
raise ValueError("This script requires exactly 3 MPI processes")
if rank == 0:
# Master process
print("Waiting for a connection...")
l = listen(5000)
recvd = l.clean(1)
if recvd:
with open('image.jpg', "wb") as file:
file.write(recvd[0:len(recvd)-2])
operation = recvd[len(recvd)-1]
print(f"Operation: {operation}")
img = cv2.imread('image.jpg')
if img is None:
print("Failed to read the image.")
MPI.COMM_WORLD.Abort()
print("Image received")
# Split the image into parts
parts = split_image(img, 2)
# Send parts to workers
comm.send(parts[0], dest=1, tag=1)
comm.send(parts[1], dest=2, tag=2)
comm.send(operation, dest=1, tag=3)
comm.send(operation, dest=2, tag=4)
# Receive processed parts from workers
processed_part1 = comm.recv(source=1, tag=5)
processed_part2 = comm.recv(source=2, tag=6)
OUTLINE
TIMELINE
Ln 19, Col 1 Spaces: 4 UTF-8 CRLF Python 3.11.3 64-b EN

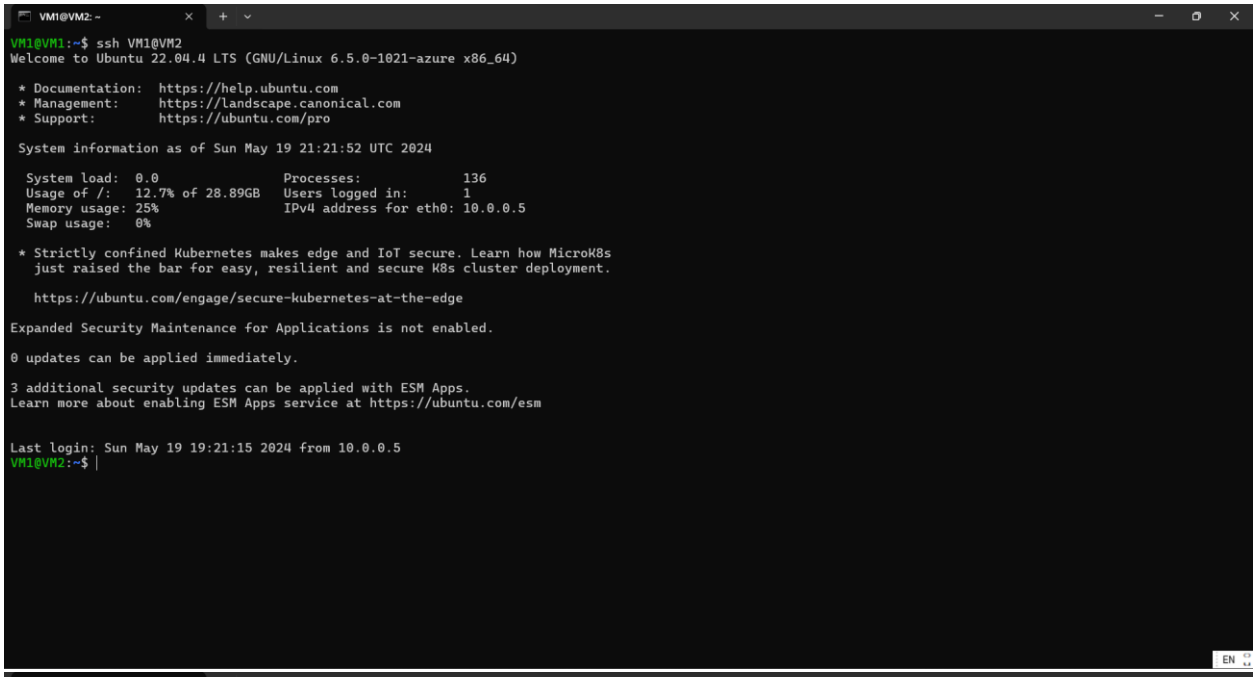

File Edit Selection View Go Run Terminal Help
Project
EXPLORER
splittedmain.py
myserver.py
server.py
PROJECT
Background.png
myserver.py
server.py
splittedmain.py
comm.send(parts[1], dest=2, tag=2)
comm.send(operation, dest=1, tag=3)
comm.send(operation, dest=2, tag=4)
# Receive processed parts from workers
processed_part1 = comm.recv(source=1, tag=5)
processed_part2 = comm.recv(source=2, tag=6)
# Combine processed parts
result_img = combine_image([processed_part1, processed_part2])
print("Image edited")
# Save the processed image
cv2.imwrite('Result_image.jpg', result_img)
with open('Result_image.jpg', 'rb') as file2:
image2 = file2.read()
l.send(image2)
print("Image Sent")
else:
print("Failed to receive data from the client.")
MPI.COMM_WORLD.Abort()
else:
# Worker processes
part = comm.recv(source=0, tag=rank)
operation = comm.recv(source=0, tag=rank + 2)
processed_part = process_image_part(part, operation)
comm.send(processed_part, dest=0, tag=rank + 4)
OUTLINE
TIMELINE
Ln 19, Col 1 Spaces: 4 UTF-8 CRLF Python 3.11.3 64-b EN

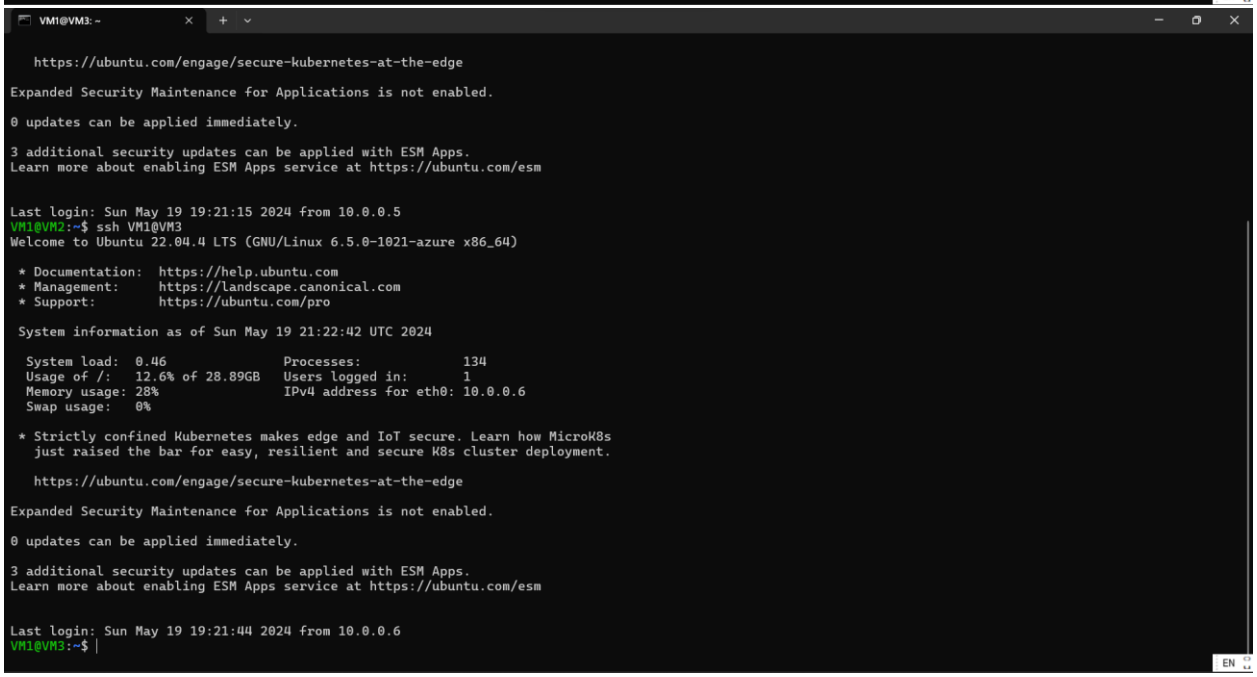
Code of Server:

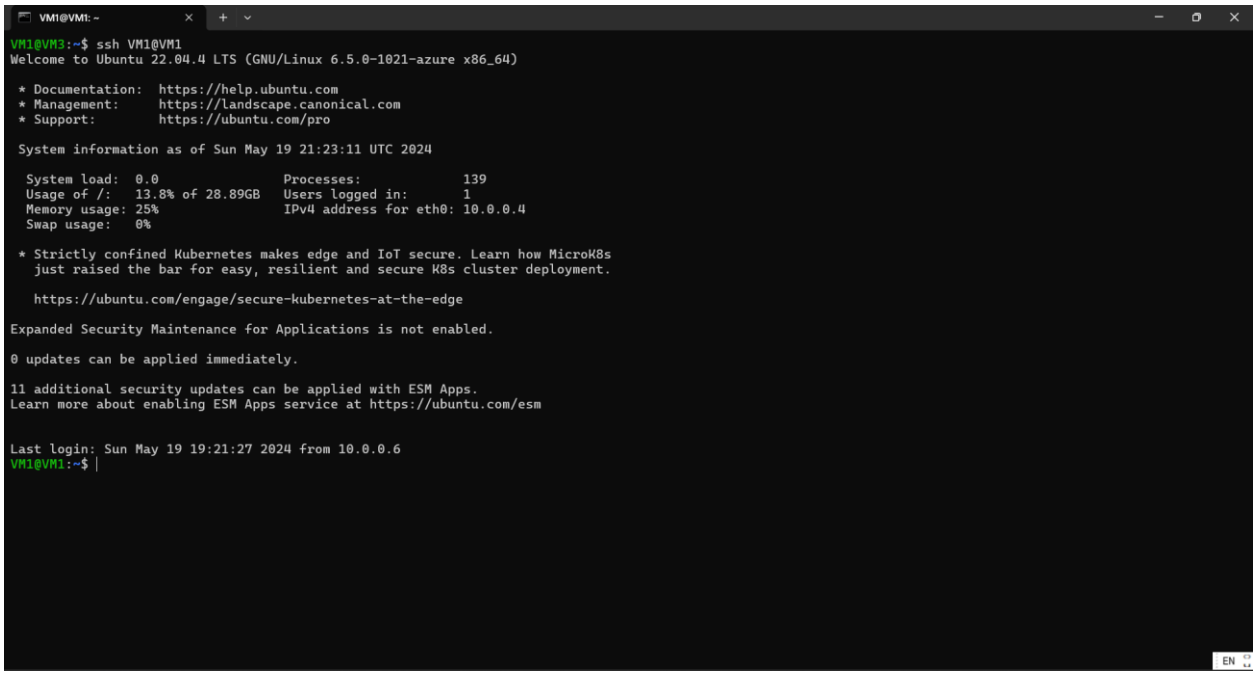




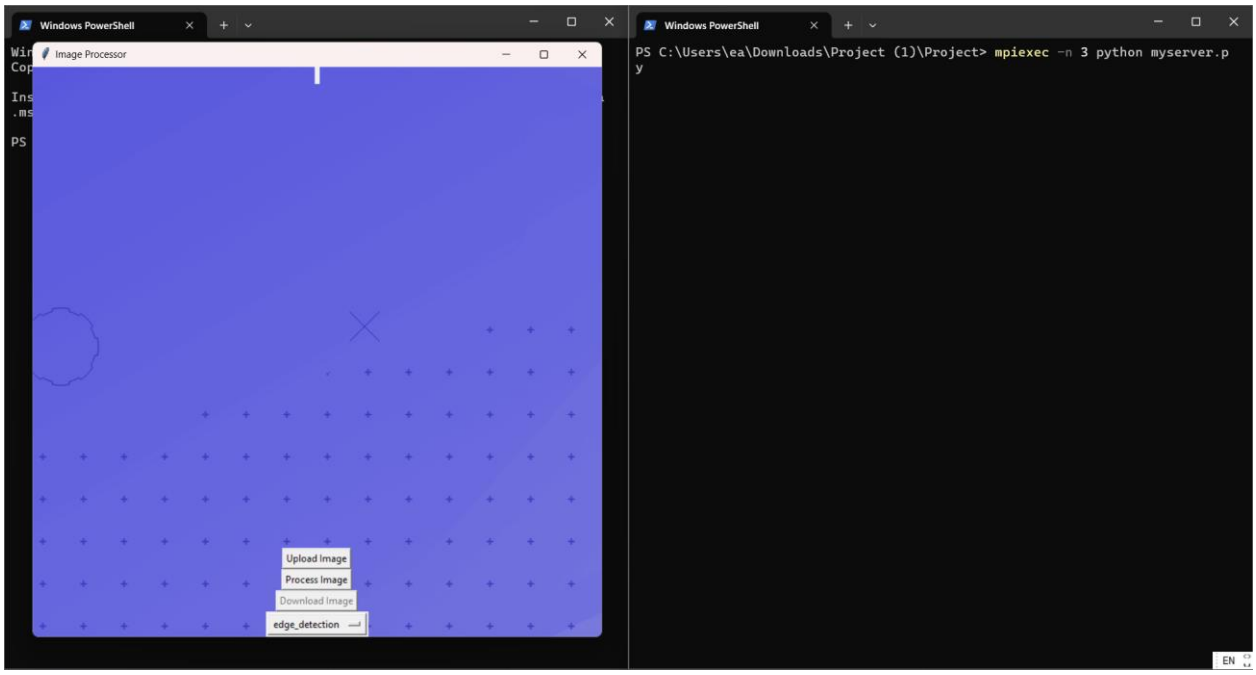
Test of MPI setup on azure virtual machines:







Run of MPI on host machine:



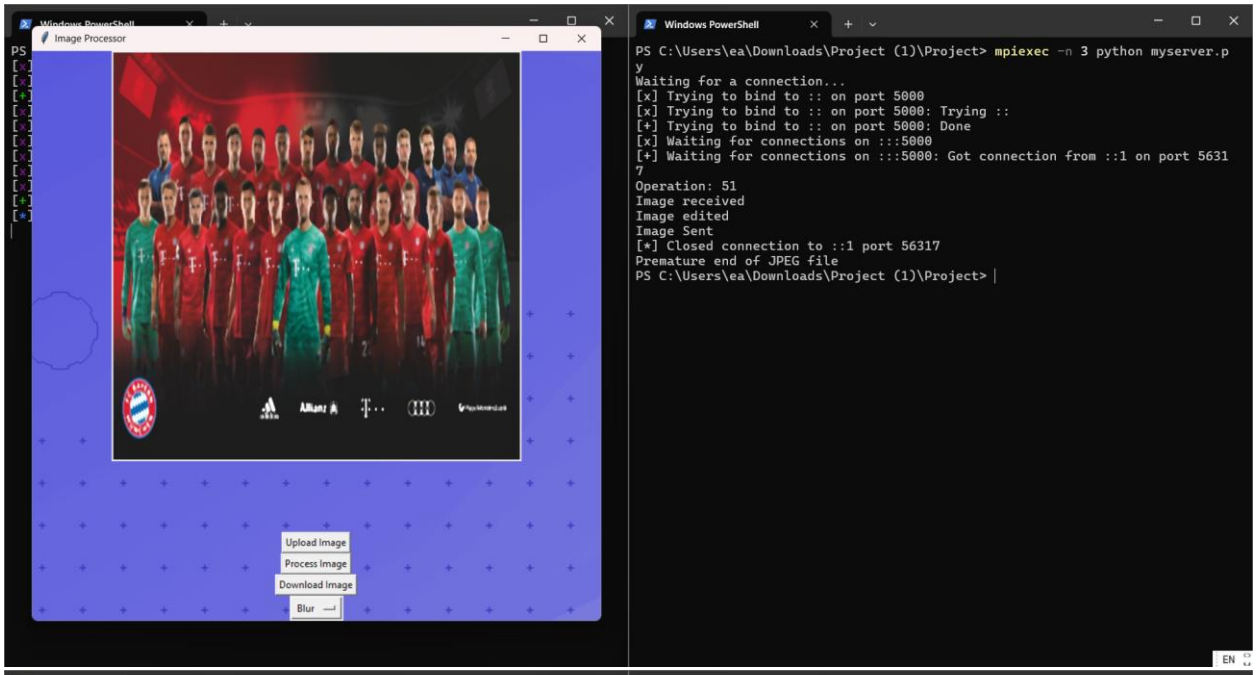


Image Processor
Upload Image
Process Image
Download Image
Blur
Windows PowerShell
PS C:\Users\ea\Downloads\Project (1)\Project> mpiexec -n 3 python myserver.p
y
Waiting for a connection...
[x] Trying to bind to :: on port 5000
[x] Trying to bind to :: on port 5000: Trying ::
[+] Trying to bind to :: on port 5000: Done
[x] Waiting for connections on :::5000
[+] Waiting for connections on :::5000: Got connection from ::1 on port 5631
7
Operation: 51
Image received
Image edited
Image Sent
[*] Closed connection to ::1 port 56317
Premature end of JPEG file
PS C:\Users\ea\Downloads\Project (1)\Project>
EN

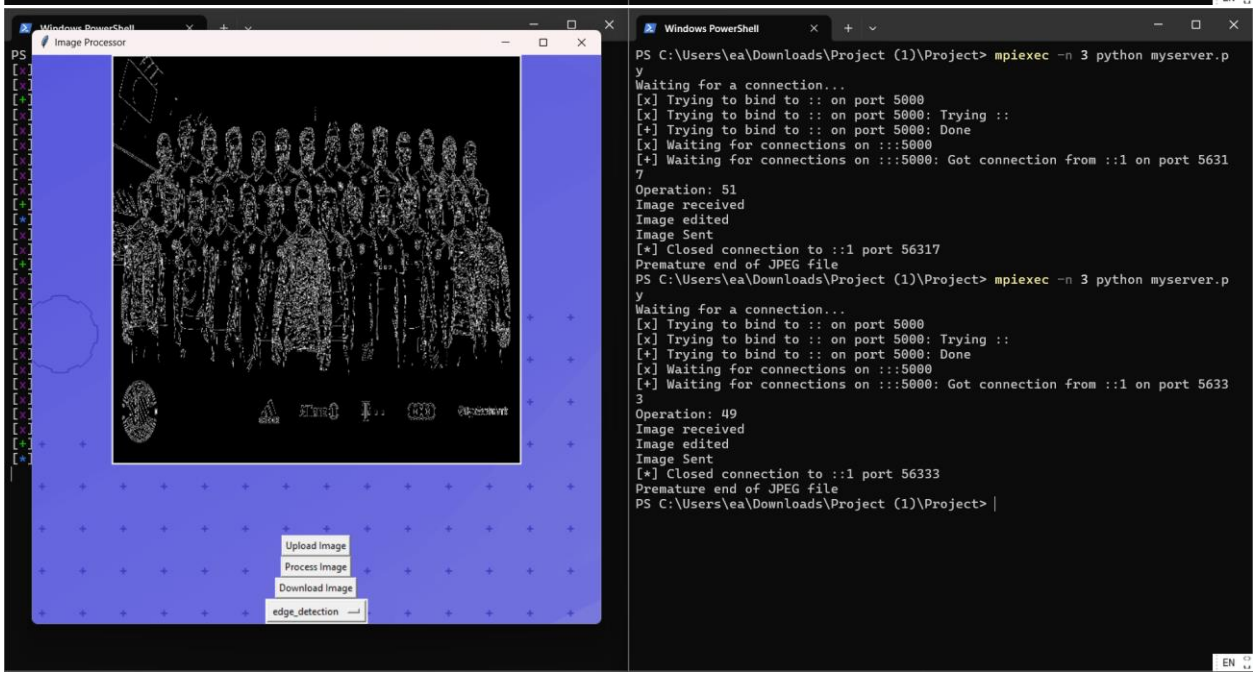


Image Processor
Upload Image
Process Image
Download Image
edge_detection
Windows PowerShell
PS C:\Users\ea\Downloads\Project (1)\Project> mpiexec -n 3 python myserver.p
y
Waiting for a connection...
[x] Trying to bind to :: on port 5000
[x] Trying to bind to :: on port 5000: Trying ::
[+] Trying to bind to :: on port 5000: Done
[x] Waiting for connections on :::5000
[+] Waiting for connections on :::5000: Got connection from ::1 on port 5631
7
Operation: 51
Image received
Image edited
Image Sent
[*] Closed connection to ::1 port 56317
Premature end of JPEG file
PS C:\Users\ea\Downloads\Project (1)\Project> mpiexec -n 3 python myserver.p
y
Waiting for a connection...
[x] Trying to bind to :: on port 5000
[x] Trying to bind to :: on port 5000: Trying ::
[+] Trying to bind to :: on port 5000: Done
[x] Waiting for connections on :::5000
[+] Waiting for connections on :::5000: Got connection from ::1 on port 5633
3
Operation: 49
Image received
Image edited
Image Sent
[*] Closed connection to ::1 port 56333
Premature end of JPEG file
PS C:\Users\ea\Downloads\Project (1)\Project>
EN


Image Processor
Upload Image
Process Image
Download Image
color_inversion
[*] Closed connection to localhost port 5000

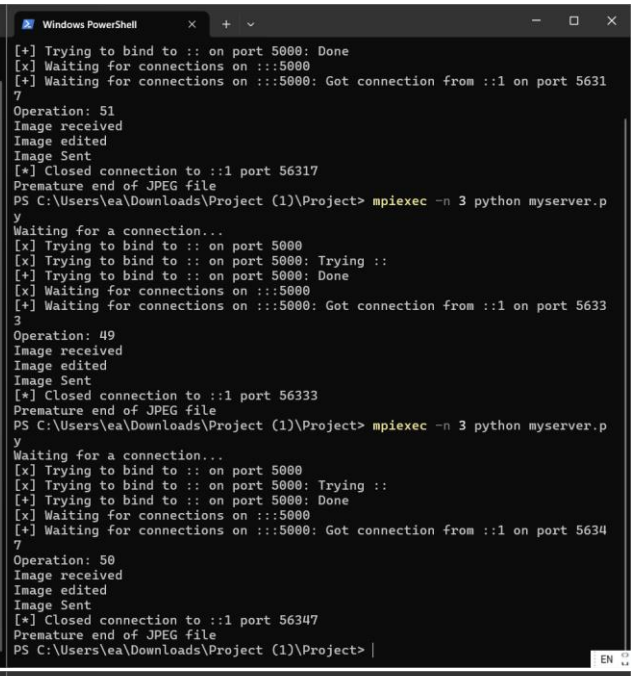
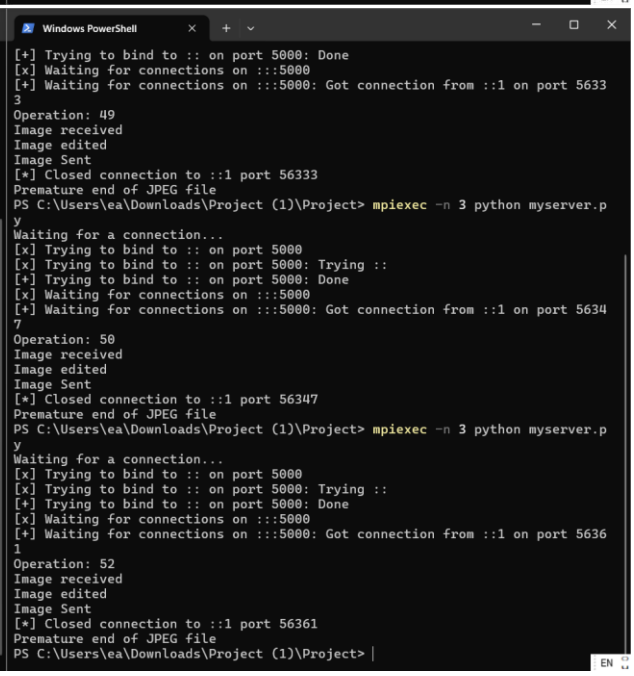
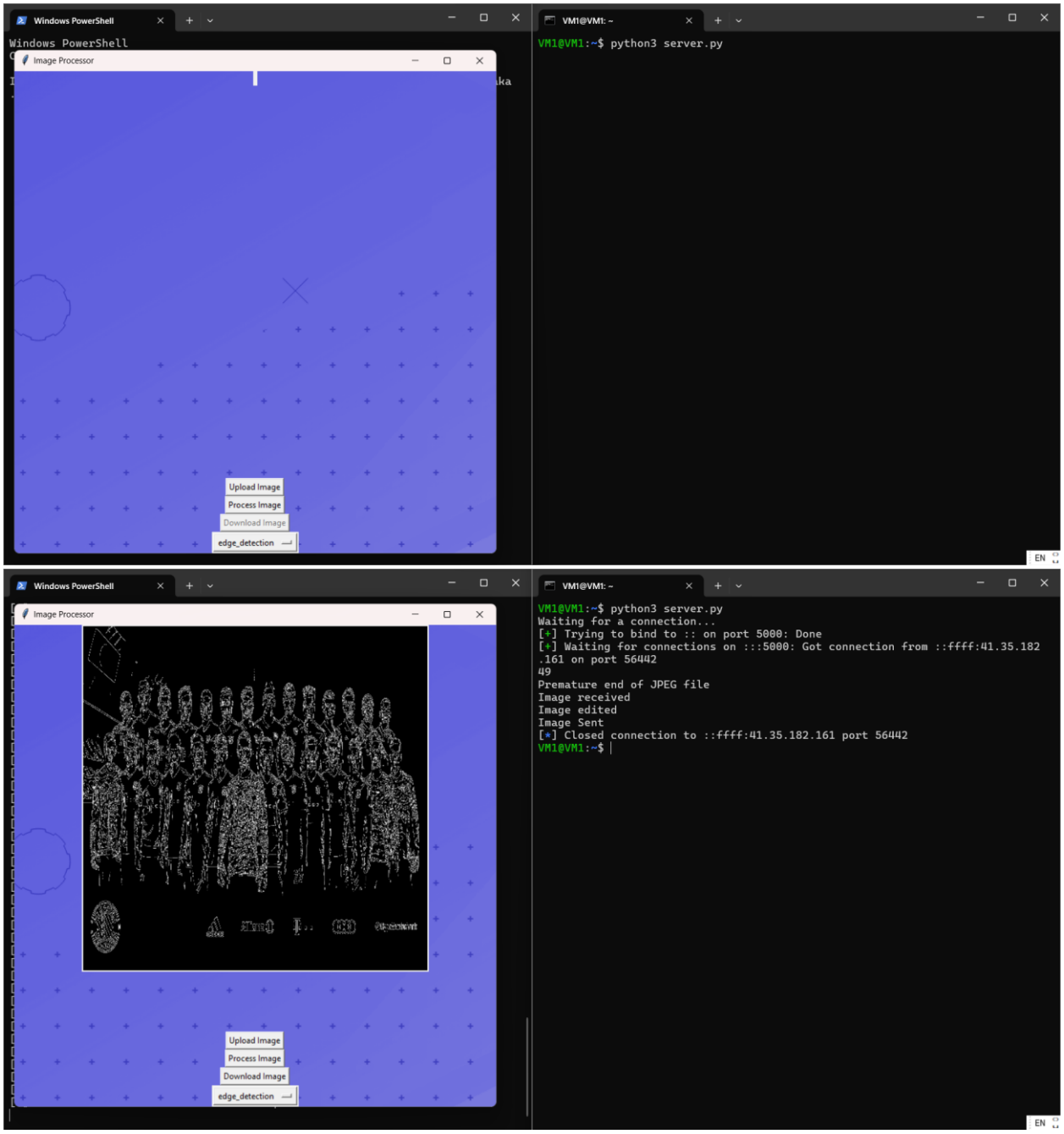
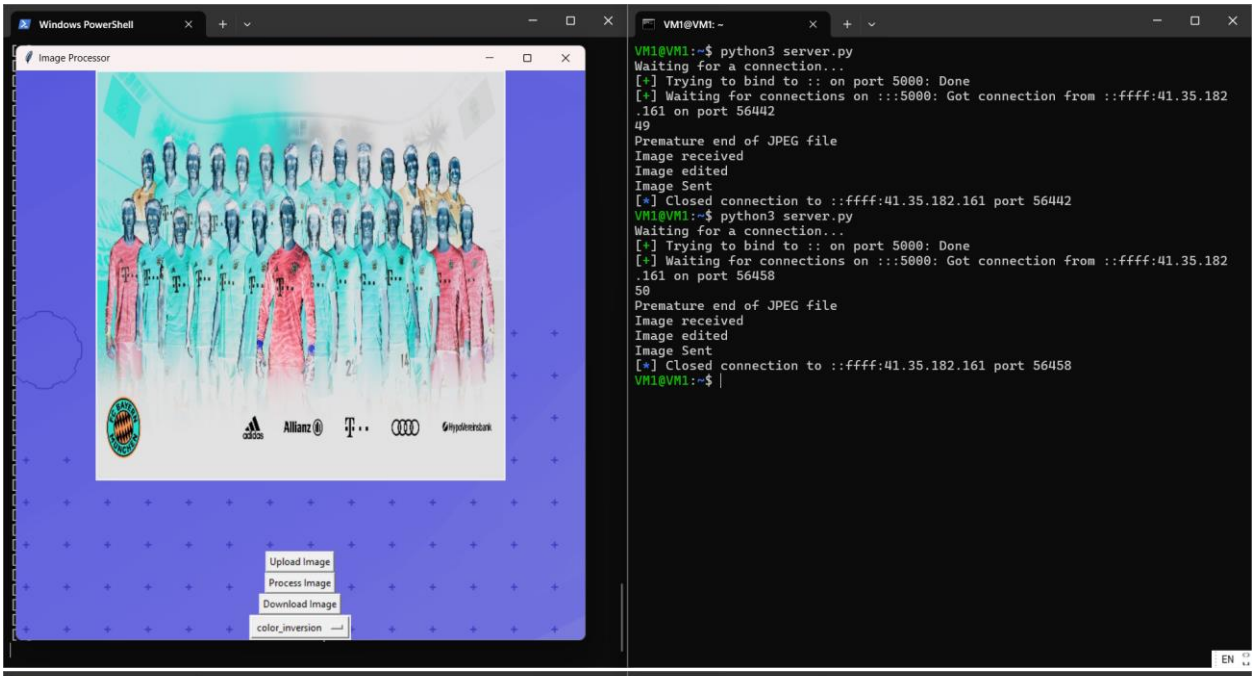

Windows PowerShell
[+] Trying to bind to :: on port 5000: Done
[x] Waiting for connections on :::5000
[+] Waiting for connections on :::5000: Got connection from ::1 on port 5631
7
Operation: 51
Image received
Image edited
Image Sent
[*] Closed connection to ::1 port 56317
Premature end of JPEG file
PS C:\Users\ea\Downloads\Project (1)\Project> mpiexec -n 3 python myserver.p
y
Waiting for a connection...
[x] Trying to bind to :: on port 5000
[x] Trying to bind to :: on port 5000: Trying ::
[+] Trying to bind to :: on port 5000: Done
[x] Waiting for connections on :::5000
[+] Waiting for connections on :::5000: Got connection from ::1 on port 5633
3
Operation: 49
Image received
Image edited
Image Sent
[*] Closed connection to ::1 port 56333
Premature end of JPEG file
PS C:\Users\ea\Downloads\Project (1)\Project> mpiexec -n 3 python myserver.p
y
Waiting for a connection...
[x] Trying to bind to :: on port 5000
[x] Trying to bind to :: on port 5000: Trying ::
[+] Trying to bind to :: on port 5000: Done
[x] Waiting for connections on :::5000
[+] Waiting for connections on :::5000: Got connection from ::1 on port 5634
7
Operation: 50
Image received
Image edited
Image Sent
[*] Closed connection to ::1 port 56347
Premature end of JPEG file
PS C:\Users\ea\Downloads\Project (1)\Project>

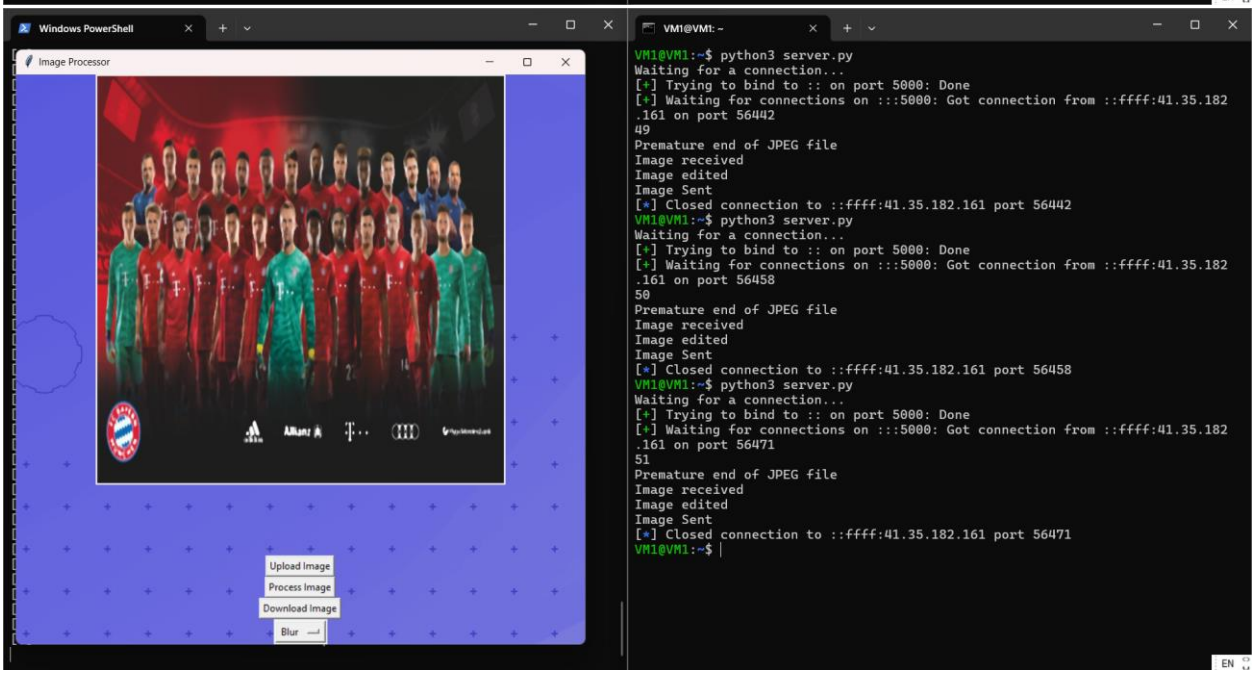

Image Processor
Upload Image
Process Image
Download Image
Binarization
[*] Closed connection to localhost port 5000

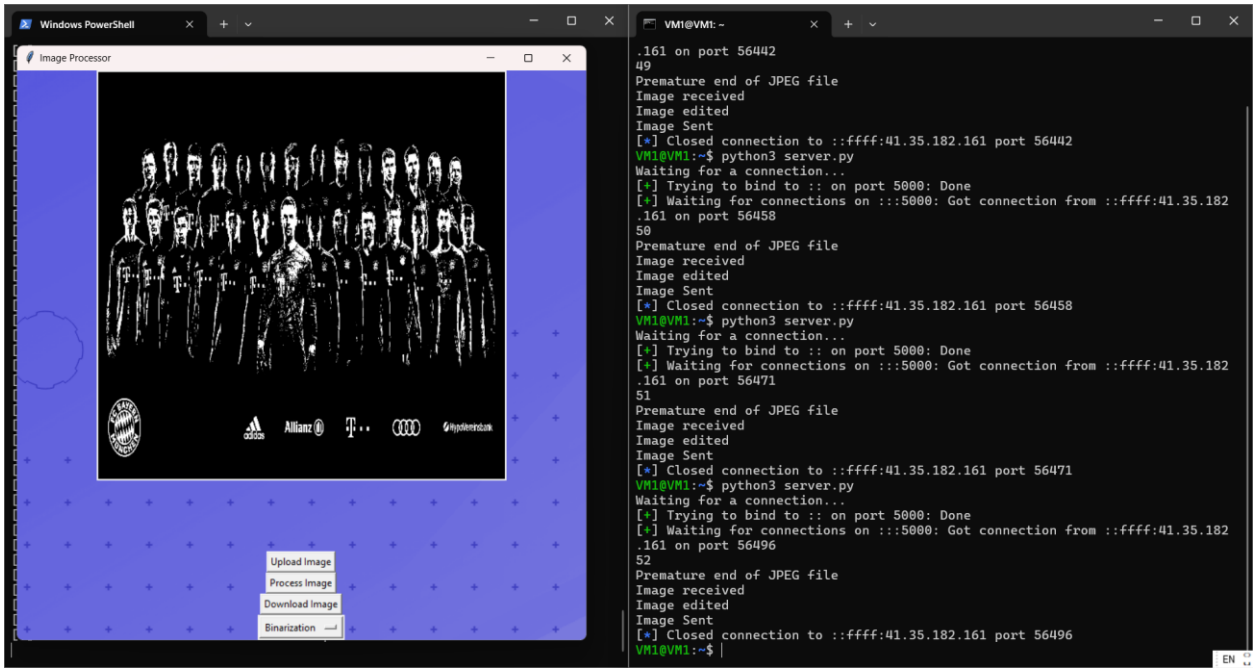

Windows PowerShell
[+] Trying to bind to :: on port 5000: Done
[x] Waiting for connections on :::5000
[+] Waiting for connections on :::5000: Got connection from ::1 on port 5633
3
Operation: 49
Image received
Image edited
Image Sent
[*] Closed connection to ::1 port 56333
Premature end of JPEG file
PS C:\Users\ea\Downloads\Project (1)\Project> mpiexec -n 3 python myserver.p
y
Waiting for a connection...
[x] Trying to bind to :: on port 5000
[x] Trying to bind to :: on port 5000: Trying ::
[+] Trying to bind to :: on port 5000: Done
[x] Waiting for connections on :::5000
[+] Waiting for connections on :::5000: Got connection from ::1 on port 5634
7
Operation: 50
Image received
Image edited
Image Sent
[*] Closed connection to ::1 port 56347
Premature end of JPEG file
PS C:\Users\ea\Downloads\Project (1)\Project> mpiexec -n 3 python myserver.p
y
Waiting for a connection...
[x] Trying to bind to :: on port 5000
[x] Trying to bind to :: on port 5000: Trying ::
[+] Trying to bind to :: on port 5000: Done
[x] Waiting for connections on :::5000
[+] Waiting for connections on :::5000: Got connection from ::1 on port 5636
1
Operation: 52
Image received
Image edited
Image Sent
[*] Closed connection to ::1 port 56361
Premature end of JPEG file
PS C:\Users\ea\Downloads\Project (1)\Project>

Run on the master node on azure:




Windows PowerShell
Image Processor
Upload Image
Process Image
Download Image
color_inversion
VM1@VM1: ~
VM1@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:41.35.182.161 on port 56442
49
Premature end of JPEG file
Image received
Image edited
Image Sent
[*] Closed connection to ::ffff:41.35.182.161 port 56442
VM1@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:41.35.182.161 on port 56458
50
Premature end of JPEG file
Image received
Image edited
Image Sent
[*] Closed connection to ::ffff:41.35.182.161 port 56458
VM1@VM1:~$
EN


Windows PowerShell
Image Processor
Upload Image
Process Image
Download Image
Blur
VM1@VM1: ~
VM1@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:41.35.182.161 on port 56442
49
Premature end of JPEG file
Image received
Image edited
Image Sent
[*] Closed connection to ::ffff:41.35.182.161 port 56442
VM1@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:41.35.182.161 on port 56458
50
Premature end of JPEG file
Image received
Image edited
Image Sent
[*] Closed connection to ::ffff:41.35.182.161 port 56458
VM1@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:41.35.182.161 on port 56471
51
Premature end of JPEG file
Image received
Image edited
Image Sent
[*] Closed connection to ::ffff:41.35.182.161 port 56471
VM1@VM1:~$
EN


Windows PowerShell
Image Processor
Upload Image
Process Image
Download Image
Binarization
VM1@VM1: ~
.161 on port 56442
49
Premature end of JPEG file
Image received
Image edited
Image Sent
[*] Closed connection to ::ffff:41.35.182.161 port 56442
VM1@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:41.35.182
.161 on port 56458
50
Premature end of JPEG file
Image received
Image edited
Image Sent
[*] Closed connection to ::ffff:41.35.182.161 port 56458
VM1@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:41.35.182
.161 on port 56471
51
Premature end of JPEG file
Image received
Image edited
Image Sent
[*] Closed connection to ::ffff:41.35.182.161 port 56471
VM1@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:41.35.182
.161 on port 56496
52
Premature end of JPEG file
Image received
Image edited
Image Sent
[*] Closed connection to ::ffff:41.35.182.161 port 56496
VM1@VM1:~$

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/1MnUPbHOxSCYaqIlHewyy4iLsjXJoyPg3/view?usp=sharing
GitHub Link: https://github.com/Nasser1159/Distributed