A Project Report

on

INVOICE ASSISTANT

at

OneOnic Solution

by

Mr. Tank Viraj Rupeshbhai

EC034, 21ECUBG069

B.Tech. (EC) Semester - VIII

Faculty Supervisor	External Guide
Dr. NarendraKumar Chauhan	Mr. Umesh Ghediya

In Partial Fulfillment of Requirement of

Bachelor of Technology - Electronics & Communication

Submitted To

Department of Electronics & Communication Engineering

Faculty of Technology

Dharmsinh Desai University, Nadiad - 387001.

(March 2025)

CERTIFICATE

This is to certify that Mr. Tank Viraj Rupeshbhai (21ECUBG069) Roll number EC034 studying in B.Tech. Semester VIII (Electronics & Communication Engineering) at Faculty of Technology, Dharmsinh Desai University, Nadiad has satisfactorily delivered seminars and has satisfactorily completed term work in the subject of Industrial Training / Project for the duration 02-12-2024 to 29-03-2025.

The project title is **Invoice Assistant** .

Date:

Dr. NarendraKumar Chauhan

(Faculty Supervisor)

Dr. Purvang Dalal

(Head of Department)

Industry Certificate

Acknowledgement

I would like to express my heartfelt gratitude to everyone who has contributed to the successful completion of this project, **Invoice Assistant**.

First and foremost, I extend my sincere appreciation to **Mr. Umesh Ghediya**, my mentor at **OneOnic Solutions**, for his invaluable guidance, continuous support, and insightful suggestions throughout the development of this project. His expertise and encouragement have played a crucial role in refining the technical aspects and overall execution of this work.

I am also deeply grateful to **Dr. NarendraKumar Chauhan**, my faculty supervisor, for his academic mentorship, constructive feedback, and unwavering support. His insights and encouragement have helped me gain a deeper understanding of the concepts involved and apply them effectively in this project.

Additionally, I would like to thank **OneOnic Solutions** for providing me with the opportunity to work on this project and gain hands-on experience in real-world applications of automation and artificial intelligence. The exposure and learning from my internship have significantly contributed to the successful implementation of this project.

I also extend my gratitude to my university, **Dharmsinh Desai University**, for providing a conducive learning environment and constructive support that facilitated my project and development.

Tank Viraj

(EC-034, 21ECUBG069)

Company Profile

Founded in 2021 and headquartered in Rajkot, India, **OneOnic Solutions** is a leading full-service digital solutions provider specializing in e-commerce, web development, UI/UX design, and digital marketing. The company delivers cutting-edge technology solutions tailored to businesses of all sizes, helping them optimize workflows, enhance customer engagement, and improve operational efficiency.

The company offers a comprehensive suite of services, including:

- Enterprise & Intranet Platforms – Developing secure, cloud-based platforms for enterprise resource management and internal communications. UI/UX Design: Crafting intuitive and engaging user interfaces and experiences to enhance customer satisfaction and drive conversions.
- CRM & ERP Systems – Delivering customized Customer Relationship Management (CRM) and Enterprise Resource Planning (ERP) solutions to streamline business operations. Custom Development: Delivering bespoke web and mobile applications, enterprise platforms, CRM, and ERP systems to streamline operations and foster growth.
- E-commerce Development – Expertise in platforms like Shopify, WordPress, Magento, and Wix to provide end-to-end e-commerce solutions.

OneOnic Solutions prides itself on its collaborative approach, positioning itself as a boutique digital agency that custom-tailors impactful digital solutions aligned with industry best practices. The company boasts a technically proficient team with over 8 years of experience, comprising designers, developers, digital marketers, business analysts, and quality assurance professionals.

The agency's commitment to excellence is reflected in its portfolio, having collaborated with over 200 clients, including brands like Hiya, Snackible, and Aleena, to propel their growth through innovative e-commerce strategies and robust development.

Client testimonials highlight OneOnic's professionalism, technical expertise, and dedication to delivering projects that meet and exceed expectations.

ABSTRACT

The Invoice Assistant is an intelligent, automated solution designed to streamline invoice processing by integrating Optical Character Recognition (OCR) and artificial intelligence (AI). The system efficiently extracts and organizes invoice data from images and PDFs, reducing manual effort and eliminating errors. Through a user-friendly Telegram bot interface, it enables seamless invoice submission, real-time processing, and structured data storage. Additionally, an AI-powered query system enhances user engagement by allowing dynamic interaction with extracted invoice details.

At the core of the Invoice Assistant is the OCR and AI-driven data extraction module, which leverages state-of-the-art OCR techniques and the Google Gemini API to accurately identify key invoice details such as invoice numbers, dates, vendor information, and total amounts. The extracted data is enhanced and formatted into a structured JSON output, ensuring consistency and reliability in financial document processing.

An advanced AI-powered query system enables users to ask follow-up questions about the extracted invoice data through the Telegram bot. These queries are processed by the Gemini API, which generates accurate, context-aware responses. This feature allows for real-time clarification of invoice details, making the system highly interactive and user-friendly.

By combining OCR, AI-driven extraction, and an intelligent query system, the Invoice Assistant provides a comprehensive, automated solution that enhances accuracy, reduces processing time, and improves overall operational efficiency. This project addresses the challenges of traditional invoice processing while paving the way for future advancements in AI-driven document automation.

INDEX

List of Figures

List of Tables

Chapter 1 Introduction to Invoice Management

1.1 Project Overview

The "Invoice Assistant" project is designed to revolutionize invoice processing by integrating automation, data extraction, and artificial intelligence into a unified system. Its goal is to minimize manual intervention, reduce errors, and accelerate invoice processing through advanced technologies. By automating repetitive tasks and leveraging innovative AI techniques, the Invoice Assistant aims to significantly enhance efficiency and accuracy in invoice management.

1.2 Background and Motivation

Automation plays a critical role in modern business operations. The table below summarizes its benefits:

Benefit	Description
Efficiency and Productivity	Automated systems handle routine tasks faster, freeing up human resources for strategic work.
Error Reduction	Minimizes manual data entry, reducing the likelihood of human errors.
Scalability	Easily adapts to increasing data volumes without a proportional increase in resource requirements.
Cost Savings	Streamlined processes lower labour costs and reduce expenses associated with errors and delays.
Competitive Advantage	Enables rapid response to market changes, optimizing processes and improving customer satisfaction.

Table 1.1 Benefits of Automation

1.3 Why Automation Matters in Invoice Management ?

Imagine a small business owner drowning in stacks of invoices, manually typing data into spreadsheets for hours. A single misplaced decimal could lead to financial discrepancies, delayed payments, or even tax filing errors. This scenario is not hypothetical – it's a daily reality for millions of businesses worldwide.

1.4 Objectives

The primary objectives of the Invoice Assistant project are to:

- Ensure Accurate Data Extraction:
  Utilize Optical Character Recognition (OCR) and other techniques to reliably retrieve information from various invoice formats.
- Integrate AI Technologies:
  Implement AI models to enhance data interpretation and support interactive query handling.
- Deliver a Seamless User Experience:
  Create an intuitive interface that allows users to manage invoices effortlessly through real-time interactions.

1.5 Significance

By integrating advanced automation and AI technologies, the Invoice Assistant is set to transform invoice processing. The anticipated benefits include:

- Reduced Operational Costs:
  Lower labor expenses and fewer errors lead to significant cost savings.
- Improved Productivity:
  Accelerated processing times enable more efficient resource allocation.
- Enhanced Decision Making:
  Timely and accurate data supports better strategic and operational decisions.

Chapter 2 Foundations of Data Automation

2.1 Introduction

Data automation forms the backbone of the Invoice Assistant project. By eliminating manual data entry, the system not only reduces errors and saves time but also creates a scalable framework for processing invoice information. This chapter details two core components that establish these foundations:

- **Automated Data Entry with Google Sheets:** A Python-based solution that programmatically updates Google Sheets.
- **Automated Data Entry via Telegram Bot:** An interactive interface that allows users to submit data through a familiar messaging platform, which is then seamlessly integrated with the Google Sheets system.

Together, these components demonstrate how robust data automation can streamline and enhance invoice processing.

2.2 Automated Data Entry with Google Sheets

2.2.1 Objective and Rationale

The primary objective of this component is to streamline the process of adding data to Google Sheets by automating it with Python and Google APIs. By doing so, the solution:

- Eliminates the need for manual data entry.
- Reduces the likelihood of human errors.
- Saves time and enhances operational efficiency.
- Provides a solid framework for handling small to medium-scale datasets.

2.2.2 Scope and Tools

The solution is focused solely on automating data entry. It involves:

- Setting up API access.
- Authenticating via a service account.
- Connecting to a designated Google Sheet.
- Appending data programmatically.

Tools and Technologies Used:

Tool/Technology	Purpose
Python	Scripting and automation
gspread	Accessing and managing Google Sheets
oauth2client	OAuth2 authentication
Google Sheets API	Programmatic read/write operations
Service Account & JSON Key File	Secure authentication
Google Drive API	Managing file permissions
Google Colab	Development and testing environment

Table 2.1 Tools and Technologies used in Data Entry with Google Sheets

2.2.3 Project Workflow

The automation process follows these steps:

1. Setup in Google Cloud:
    - Create a new project and enable the Google Drive and Sheets APIs.

Step 1 :- Go to Google Cloud Console, click on New Project and provide Name of Project

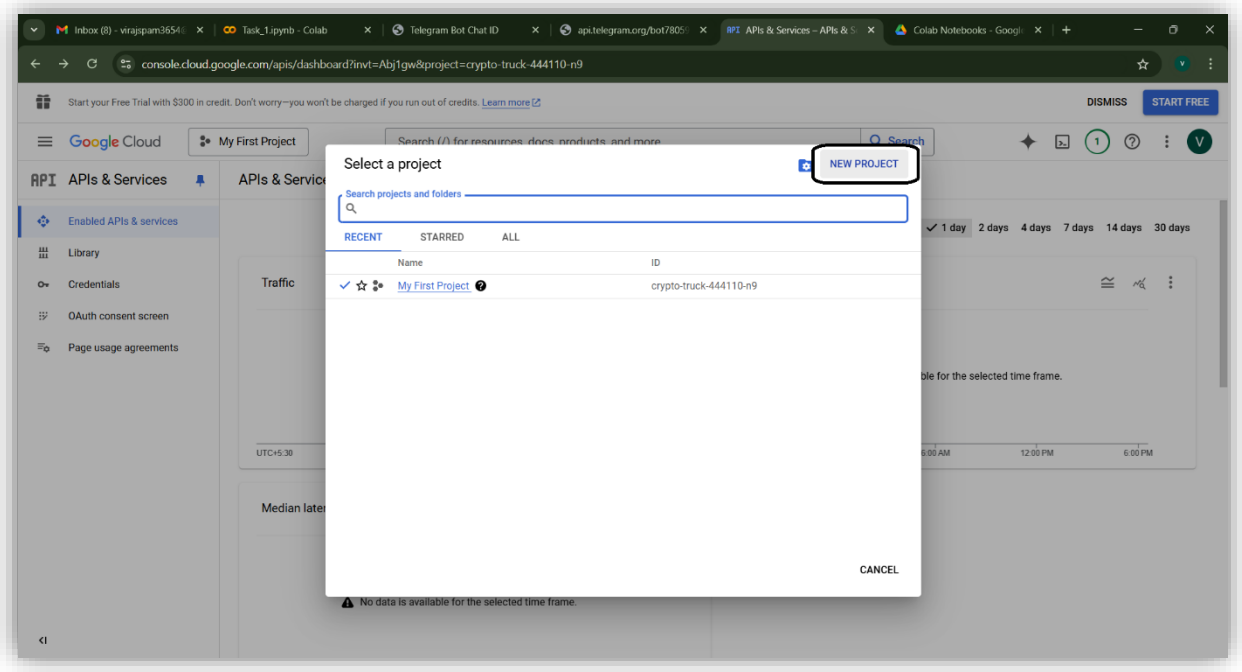


Figure 2-1 Step 1 for Setup of API

Step 2 :- Click on Enable APIs and Services

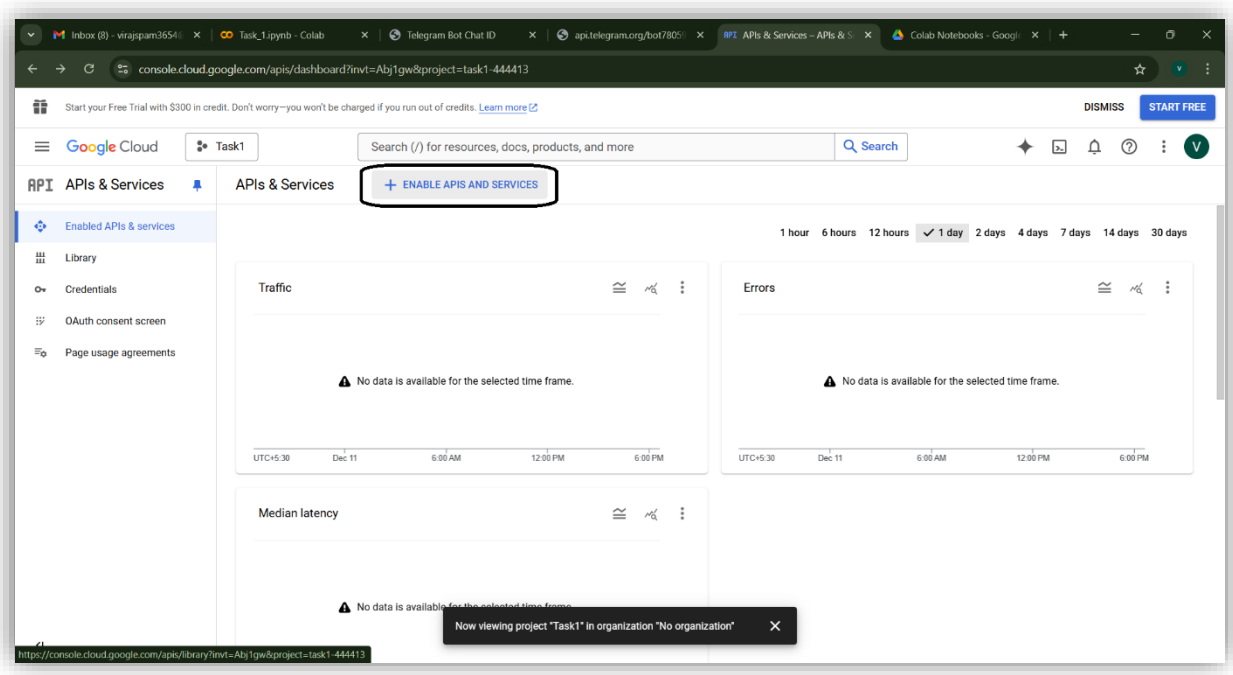


Figure 2-2 Step 2 for Setup of API

Step 3 :- Enable the Google Drive API

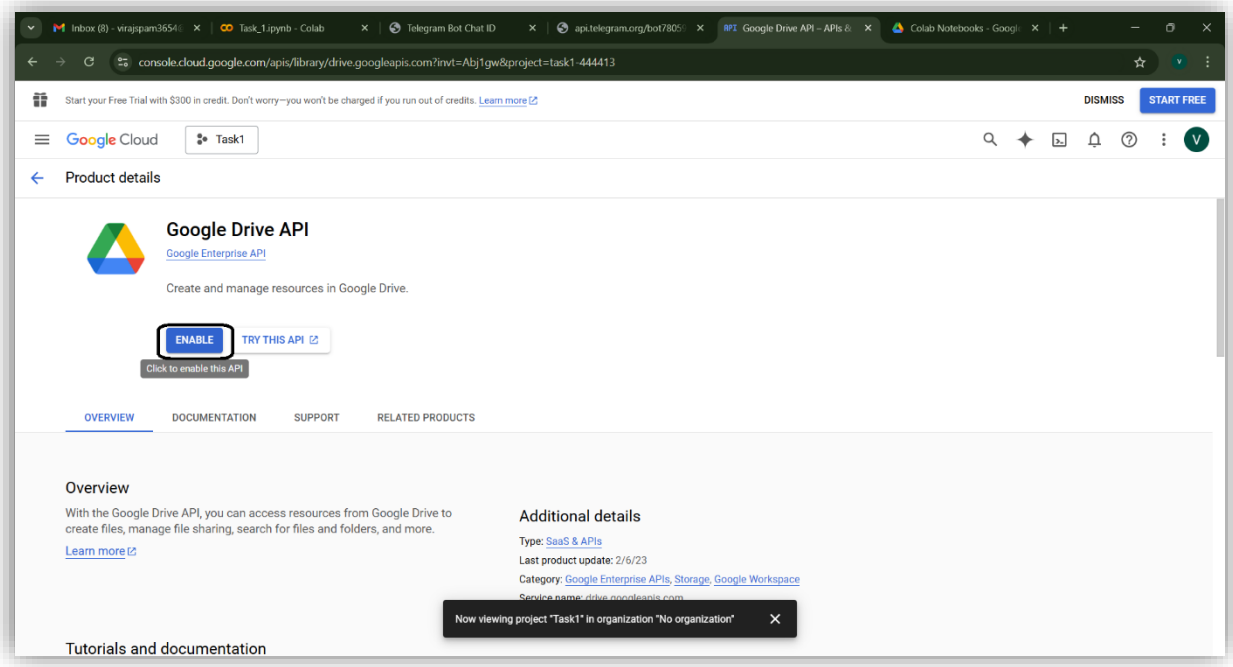


Figure 2-3 Step 3 for Setup of API

Step 4 :- Enable the Google Sheets API

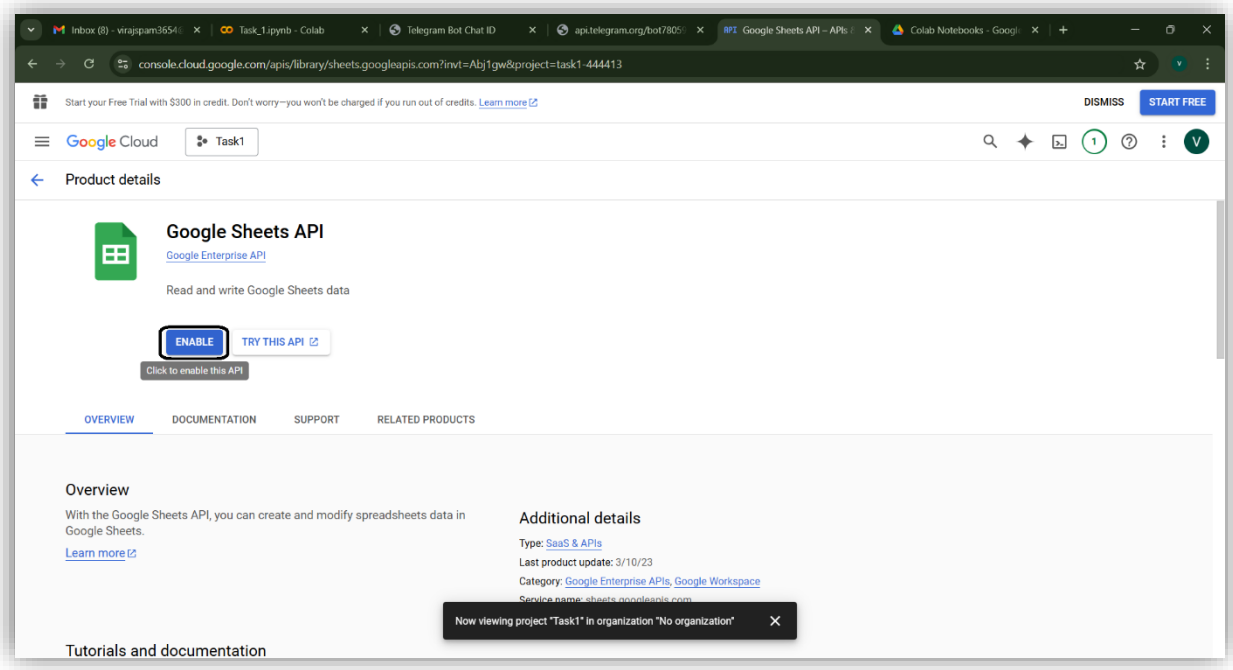


Figure 2-4 Step 4 for Setup of API

- Create a service account and download the JSON key file.

Step 5 :- Navigate to the APIs & Services > Credentials section in the Google Cloud Console.

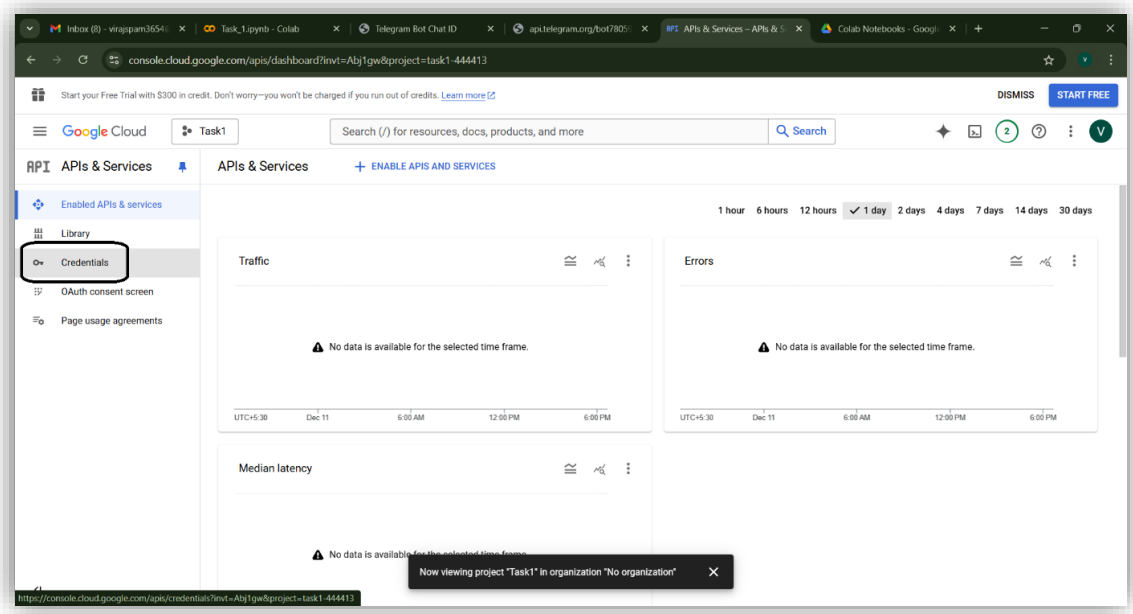


Figure 2-5 Step 5 for Setup of API

Step 6 :- Click on + Create Credentials

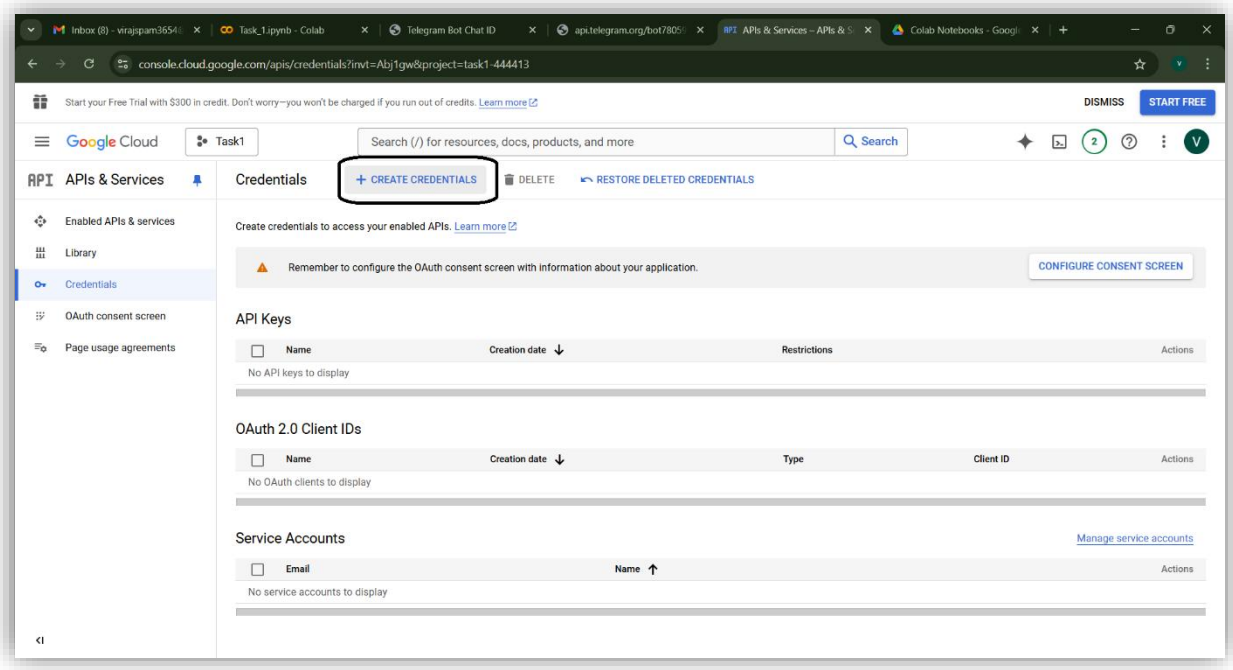


Figure 2-6 Step 6 for Setup of API

Step 7 :- And select Service Account

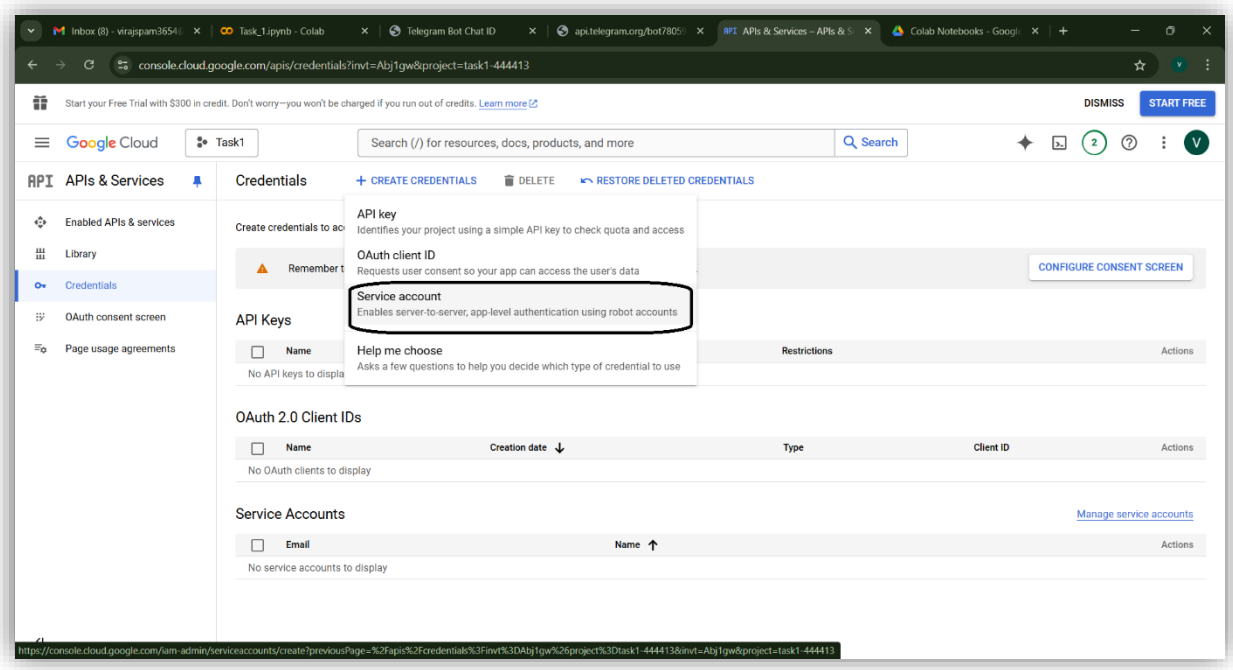


Figure 2-7 Step 7 for Setup of API

Step 8 :- Fill in the required details for the service account (name, description) and click Create.

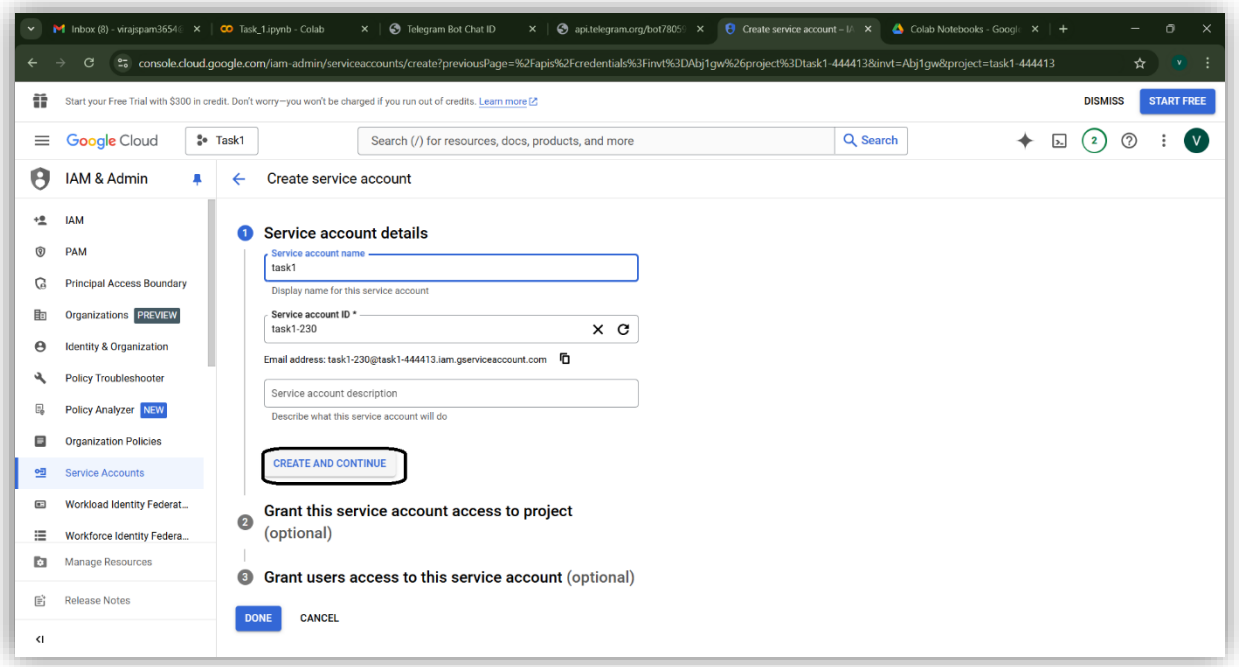


Figure 2-8 Step 8 for Setup of API.

Step 9 :- Under the "Role" section, assign the service account a role, such as Editor or Owner. Complete the process, and in the last step, click Done.

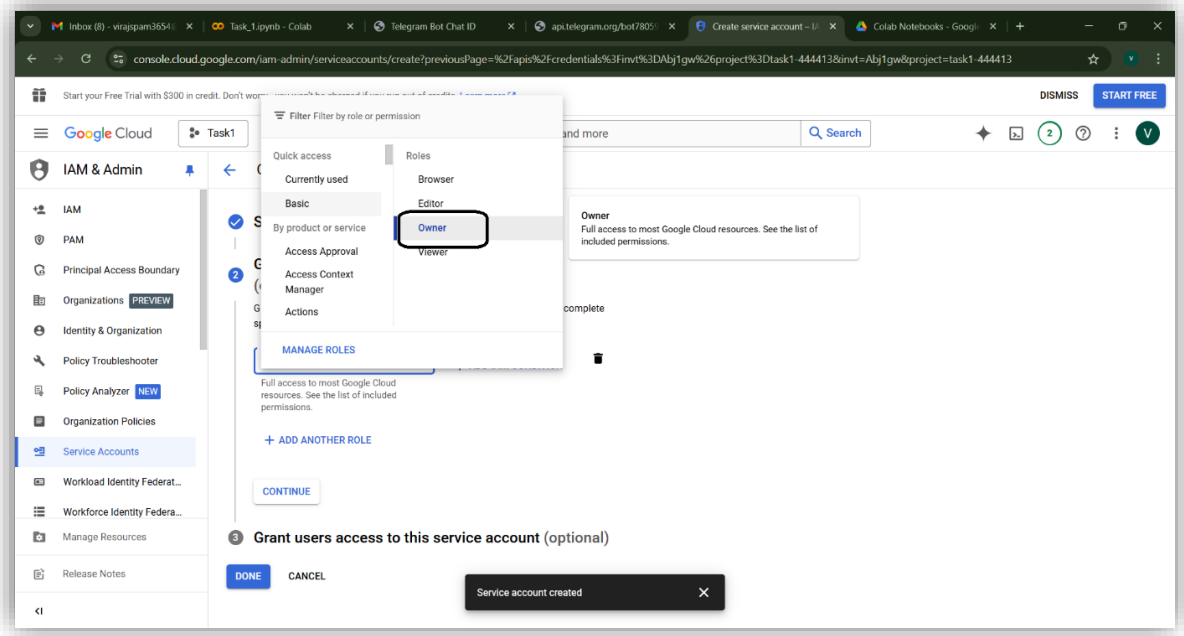


Figure 2-9 Step 9 for Setup of API

Step 10 :- After creating the service account, locate it in the "Credentials" tab and click Manage Keys.

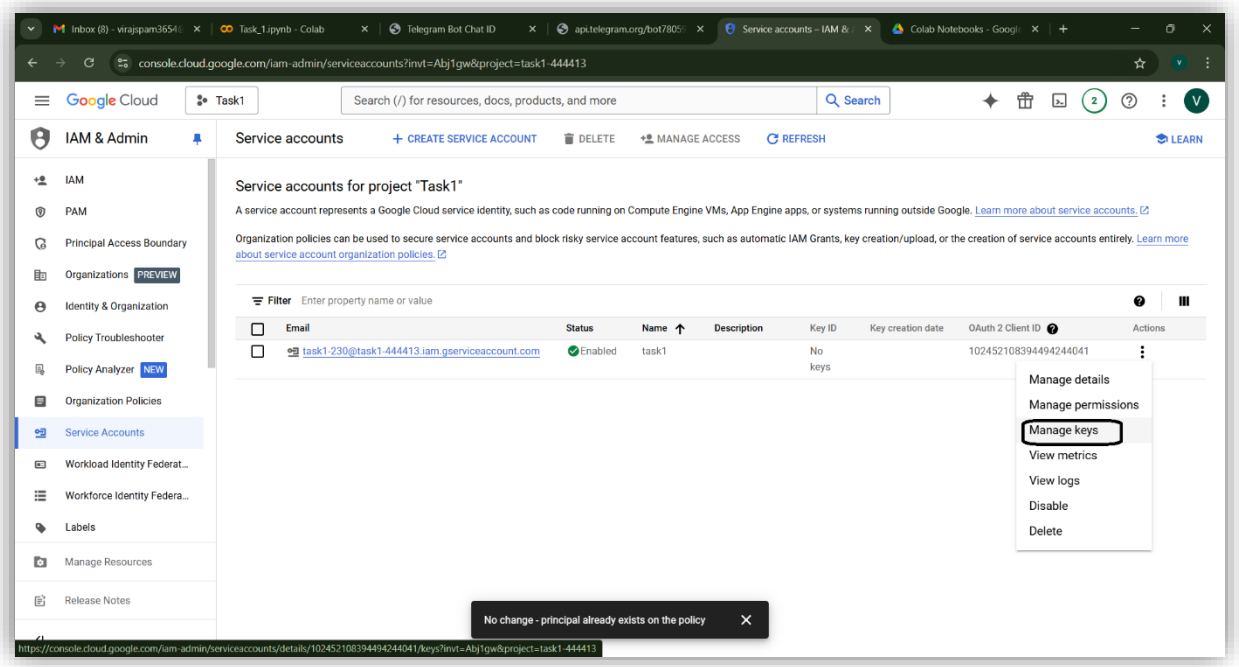


Figure 2-10 Step 10 for Setup of API

Step 11 :- Click Add Key > Create New Key, select the JSON option, and download the credentials file. And this file to Google Drive where code is present.

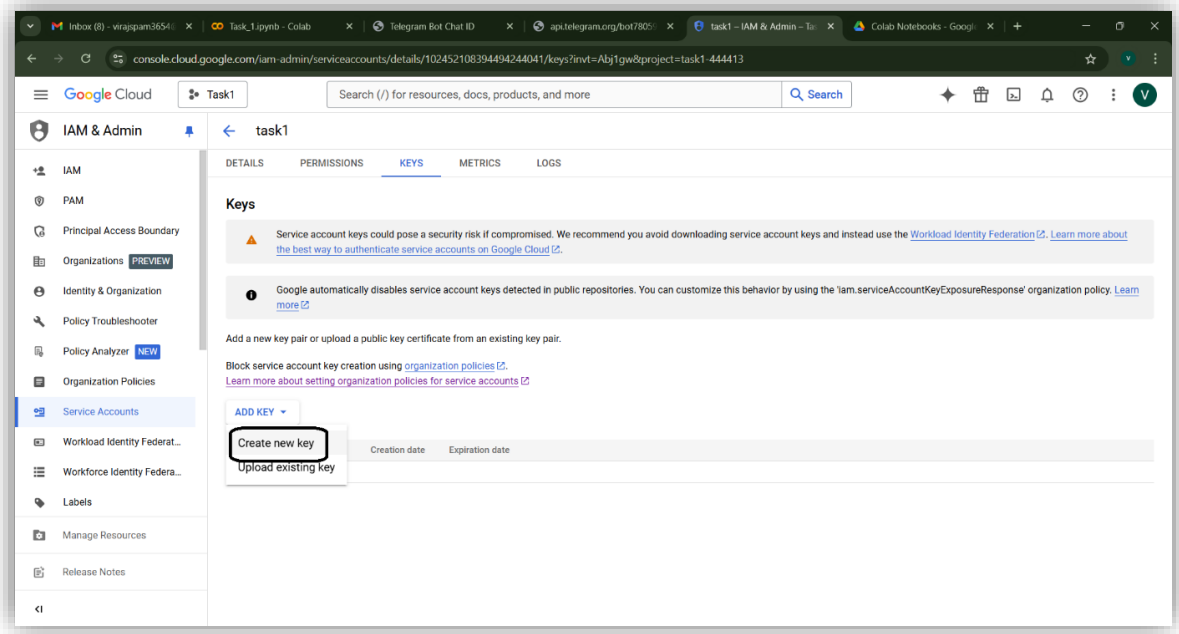


Figure 2-11 Step 11 for Setup of API (1)

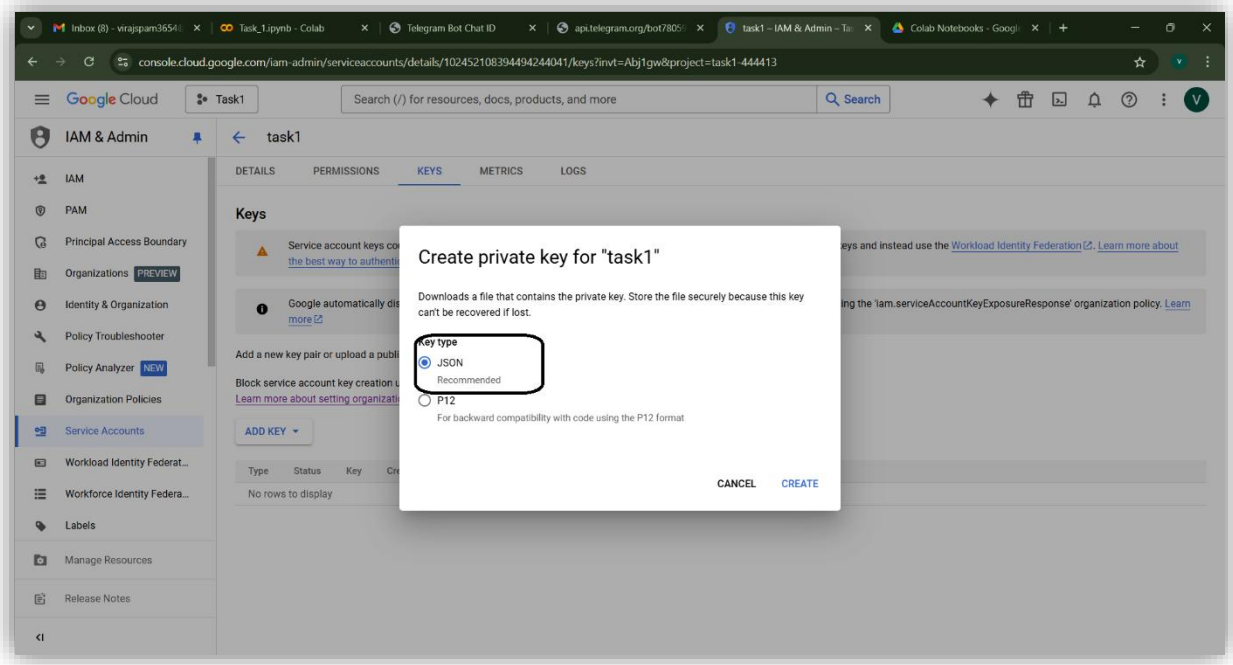


Figure 2-12 Step 11 for Setup of API (2)

- Share the target Google Sheet with the service account's email address.

Step 12 :- Open the Google Sheet you want to use or create a new sheet.And Click on Share button.

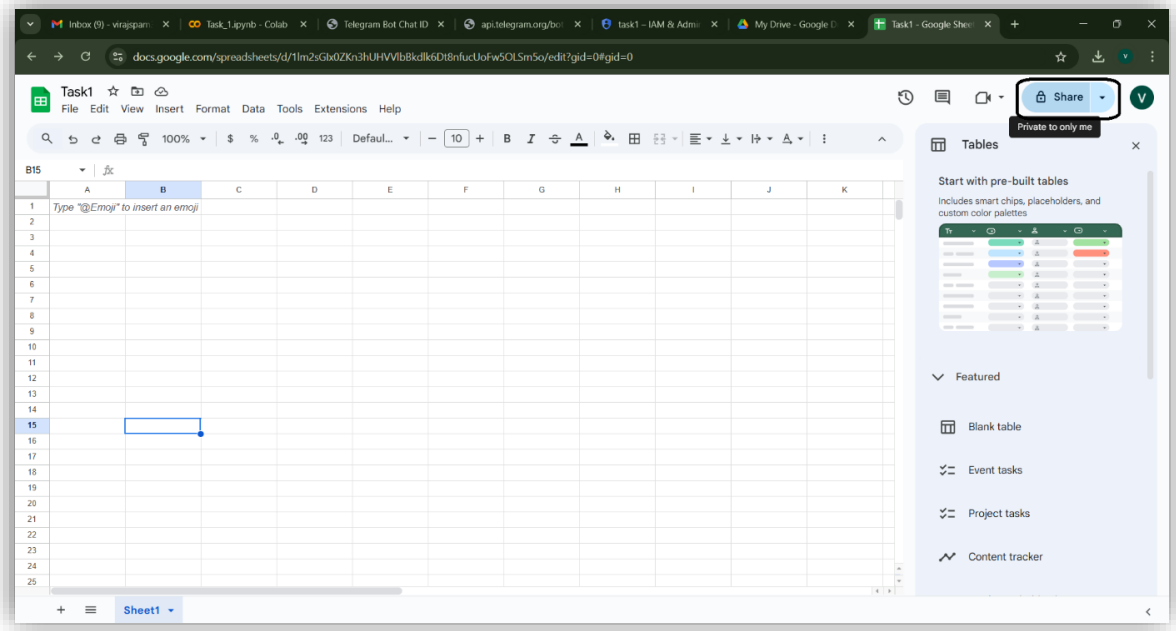


Figure 2-13 Step 12 for Setup of API.

Step 13 :- Share the Google Sheet with this email address by clicking Share in the Google Sheet and granting Editor access.

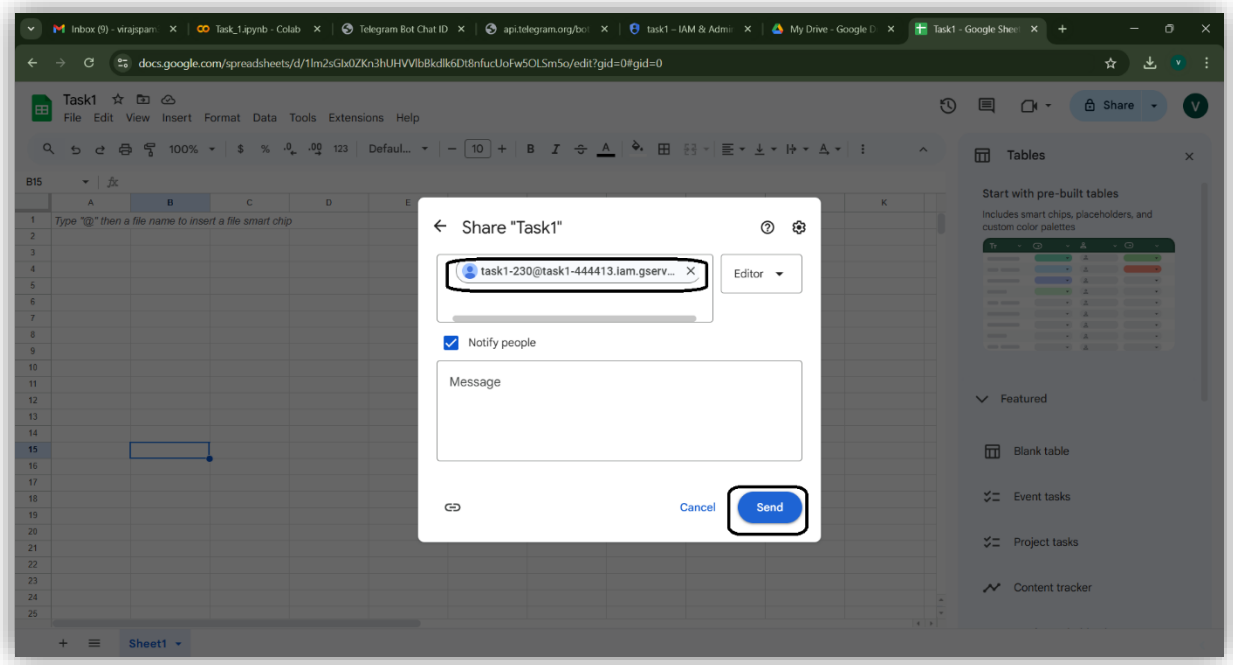


Figure 2-14 Step 13 for Setup of API

2. Script Execution:
    - The Python script authenticates using the JSON key file.
    - It connects to the specified Google Sheet using the gspread library.
    - Product data (or other structured data) is defined and appended as new rows.
3. Error Handling:
    - Robust try/except blocks manage potential errors such as API connection failures.

2.2.4 Code Explanation

The following sample code snippet illustrates the core functionality:

```
import gspread
from oauth2client.service_account import ServiceAccountCredentials

def connect_to_google_sheet(sheet_name):
    # Define the scope for the API
    scope = ["https://spreadsheets.google.com/feeds", "https://www.googleapis.com/auth/drive"]
    # Authenticate using the credentials JSON file
    credentials = ServiceAccountCredentials.from_json_keyfile_name("path/to/credentials.json", scope)
```

```
    client = gspread.authorize(credentials)
    # Open the Google Sheet
    sheet = client.open(sheet_name).sheet1
    return sheet

def add_product_data(sheet, products):
    # Add product data by appending each product as a new row
    for product, price in products.items():
        sheet.append_row([product, price])
    print("Data added successfully!")

if __name__ == "__main__":
    SHEET_NAME = "Task1"
    product_data = {
        "Laptop": 50000,
        "Smartphone": 25000,
        "Headphones": 1800,
        "Monitor": 3000,
    }
    try:
        sheet = connect_to_google_sheet(SHEET_NAME)
        add_product_data(sheet, product_data)
    except Exception as e:
        print("An error occurred:", e)
```

2.3 Result and Observation's

The implementation of automated data entry with Google Sheets has:

- Reduced Manual Effort: Data entry is fully automated, thereby reducing repetitive manual work.
- Enhanced Accuracy: Automation minimizes human error, ensuring data consistency.
- Improved Scalability: The solution can easily be extended to handle larger datasets.
- Valuable Learning: Overcoming challenges such as API configuration and authentication has provided practical insights into building robust automation systems.

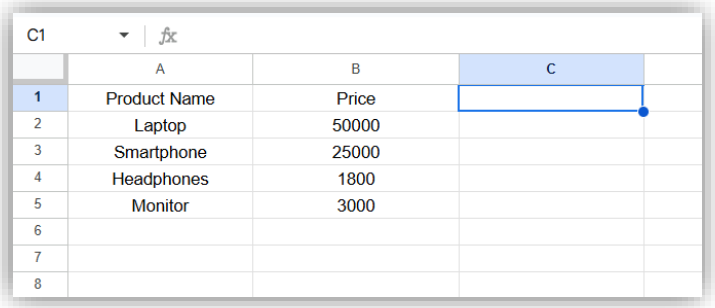


Figure 2-15 Output of Data Entry with Google Sheets

Chapter 3 Bridging Messaging Platforms and Cloud Systems

3.1 Automated Data Entry via Telegram Bot

Telegram is chosen as the bridging messaging platform because it offers a familiar, accessible interface that enables real-time communication with users. Its robust API support and asynchronous handling capabilities facilitate seamless data transfer between the user and the cloud-based backend. This integration ensures that data, such as invoice details, is promptly validated, processed, and stored, thereby enhancing overall system efficiency and user experience.

3.1.1 Objective and Rationale

The goal of integrating a Telegram bot is to provide a user-friendly, interactive interface that simplifies data entry. By enabling users to send data through a chat-based interface, the system:

- Increases accessibility by leveraging a familiar platform.
- Provides real-time feedback on data submission.
- Further reduces the dependency on manual data entry.

3.1.2 Scope and Tools

This component is designed to facilitate quick data input via Telegram, automatically forwarding the data to the Google Sheets system. It is particularly aimed at small businesses and individual users who require an efficient, error-free method of recording information.

Tools and Technologies Used:

Tool/Technology	Purpose
Python	Scripting and bot development
python-telegram-bot	Building and managing the Telegram bot
nest_asyncio	Ensuring compatibility with asynchronous tasks
gspread, oauth2client	Integration with Google Sheets automation
Telegram Bot API	Facilitating interactive user communication

Table 3.1 Tools and Technologies used in Data Entry via Telegram Bot

3.1.3 Project Workflow

The Telegram bot integration follows these steps:

1. Bot Setup:

- Create a Telegram bot using BotFather and obtain an API token.
- Configure the bot with the obtained token in the Python script.

2. User Interaction:

- The bot greets users and prompts them to submit product details in a specified format (e.g., "Product Name, Price").

3. Data Processing:

- Upon receiving the message, the bot validates and splits the data.
- It then connects to the Google Sheets automation system to append the data.

4. Feedback and Query Handling:

- The bot sends confirmation messages upon successful data entry.
- In case of input errors, it provides instructions to correct the format.

3.1.4 Code Explanation

Below is a sample code snippet demonstrating the Telegram bot integration:

```
1. import asyncio
2. import nest_asyncio
3. from telegram import Update
4. from telegram.ext import Application, CommandHandler, MessageHandler,
filters, CallbackContext
5. import gspread
6. from oauth2client.service_account import ServiceAccountCredentials
7.
8. nest_asyncio.apply()
9.
10. def connect_to_google_sheet(sheet_name):
11.                 scope    =    ["https://spreadsheets.google.com/feeds",
"https://www.googleapis.com/auth/drive"]
12.                                                 credentials          =
ServiceAccountCredentials.from_json_keyfile_name("path/to/credentials.json
", scope)
13.     client = gspread.authorize(credentials)
14.     sheet = client.open(sheet_name).sheet1
15.     return sheet
16.
17. async def start(update: Update, context: CallbackContext):
18.      await update.message.reply_text("Welcome! Send me product details
in the format: Product Name, Price")
19.
20. async def add_data(update: Update, context: CallbackContext):
21.     message = update.message.text
22.     try:
23.         product, price = map(str.strip, message.split(","))
24.         price = float(price)
```

```
25.         sheet = connect_to_google_sheet("Task1")
26.         sheet.append_row([product, price])
27.              await update.message.reply_text(f"Added: {product} - Rs
{price:.2f}")
28.     except ValueError:
29.           await update.message.reply_text("Invalid format! Please send
data in the format: Product Name, Price")
30.     except Exception as e:
31.         await update.message.reply_text(f"An error occurred: {e}")
32.
33. async def main():
34.     BOT_TOKEN = "YOUR_TELEGRAM_BOT_TOKEN"
35.     application = Application.builder().token(BOT_TOKEN).build()
36.     application.add_handler(CommandHandler("start", start))
37.                 application.add_handler(MessageHandler(filters.TEXT   &
~filters.COMMAND, add_data))
38.     print("Bot is running...")
39.     await application.run_polling()
40.
41. await main()
42.
```

3.2 Result and Observation's

The integration of the Telegram bot has:

- Enhanced User Accessibility:

  Users can easily input data through a chat interface, making the process intuitive and convenient.

- Real-Time Feedback:

  The system provides immediate confirmation or error messages, ensuring smooth interaction.

- Seamless Integration:

  Data submitted via the bot is automatically reflected in the Google Sheet, demonstrating efficient backend integration.

- Practical Insights:

  The project provided valuable experience in asynchronous programming and handling real-time interactions.

Figure 3-1 Data Entry via Telegram Bot

C11

	A	B	C
1	**Product Name**	**Price**	
2	Laptop	50000	
3	Smartphone	25000	
4	Headphones	1800	
5	Monitor	3000	
6	Mouse	1000	
7	Mouse Pad	200	
8	Desktop	3500	
9			

Figure 3-2 Output of Data Entry via Telegram Bot

3.3 Comparative Analysis and Integration

While the Google Sheets automation component handles the heavy lifting of updating and managing data, the Telegram bot offers a user-friendly interface for data submission. Together, they create a comprehensive, efficient data automation system.

Aspect	Google Sheets Automation	Telegram Bot Integration
Primary Function	Automates data entry and spreadsheet updates	Provides interactive, real-time data submission
User Interaction	Minimal; runs in the background	High; chat-based, user-friendly interface
Error Handling	Focused on API reliability and data integrity	Emphasizes validating user input and format
Scalability	Hard Coding for data entry	Dynamic Data Entry

Table 3.2 Comparative Analysis with and without Telegram Integration

Chapter 4 Optical Character Recognition (OCR) Fundamentals

4.1 Introduction to OCR

Optical Character Recognition (OCR) is a transformative technology that converts text embedded in images or scanned documents into machine-readable data. Within the context of the Invoice Assistant project, OCR is critical for extracting structured invoice information from various document formats. This chapter delves into the complete journey of OCR experiments—from foundational principles and image preprocessing to comprehensive tool comparisons and advanced analysis. By the end of this chapter, readers will gain an in-depth understanding of how OCR works, the challenges encountered, and the best practices for achieving high accuracy in text recognition.

4.2 How Text Recognition Engine Works

Text recognition is like teaching a computer to read words in pictures. Imagine you have a photo of a paper with words on it. A text recognition engine looks at the picture, figures out the words, and turns them into text that a computer and a human can understand. In this chapter, we will learn how text recognition works, step by step, so we can understand how the computer reads and understands the words from pictures.

4.2.1 Definition and Difference between Model and Engine

- **Model** :- A model is like a brain for a computer. It helps the computer learn how to do things, like reading text in pictures. When we teach a model, we show it many examples of images with text and tell it what the text says. The model then learns to recognize those patterns, so later, when it sees a new image, it can guess what the text is without being told.
- **Engine** :- An engine is like a tool or machine that does the actual work of recognizing the text in an image. It uses the model to figure out the letters and words. Think of it like a car engine: the car doesn't move by itself, but with the engine working, the car goes forward. In the same way, the engine helps the model understand the text.
- **OCR** :- OCR stands for Optical Character Recognition. It is the name of the whole process of turning text from pictures into text the computer can read. It's like a magic trick where the computer can see words in photos or scans and make them readable.

Difference between Model and Engine :-

	Model	Engine
Scope	Performs a single task.	Combines multiple tasks.
Input/Output	Inputs data; outputs predictions or results.	Takes raw input, processes it, and outputs usable results.
Role	A component of a larger system.	The complete system itself.
Example	LSTM model for recognizing characters	Tesseract OCR for end-to-end text recognition.

Table 4.1 Difference between Model and Engine

4.2.2 Steps to create own OCR system

1) Data Collection

The first step is to gather the images that contain the text you want to recognize. These could be photos of documents, receipts, books, or handwritten notes.

2) Annotation

We gather pictures of text and mark where the words are. This helps the computer learn where the words are in the pictures.

3) Preprocess the Images

Image preprocessing helps in enhancing the input images to improve text detection and recognition accuracy. Example Grayscale Conversion, Thresholding, Noise Removal and etc.

4) Text Detection

Before recognizing text, you need to detect where the text is located in the image. Use object detection models to detect text as individual regions (bounding boxes).

5) Text Recognition Model

After detecting text regions, you need a model to recognize the characters or words inside the bounding boxes. This is the core of your OCR system. For this Various Deep Learning Models are used like CNN (Convolutional Neural Networks), RNN(Recurrent Neural Networks), LSTM(Long Short-Term Memory networks). This is like teaching the computer to read the words inside the boxes by showing it lots of examples

6) Train Your Model

We train the computer to get better at reading by showing it many pictures and correcting its mistakes.

7) Post-Processing

After the computer reads the words, we fix any mistakes it made, like correcting spelling.

8) Optimization

We check how well the computer is reading and make it even better by fixing problems.

9) Deployment

We make the computer ready to read words from new pictures, like putting it into an app that can read text for you.

4.2.3 Challenges of Creating Your Own OCR Model

- **Collecting Data:** You need lots of pictures with text, and each one has to be labelled correctly, which takes a lot of time.
- **Training the Model:** Teaching the model to read text takes powerful computers and can take days or weeks because of large data set.
- **Recognizing Different Texts:** The model has to learn to read many different fonts, languages, and even handwriting, which makes it harder
- **Fixing Mistakes:** Sometimes the model makes mistakes, and you have to spend time correcting them.
- **Time:** All of this takes a lot of time and effort's making it a tough task.

4.3 Using Tesseract OCR

Tesseract OCR is an open-source engine originally developed by HP and now maintained by Google. It converts scanned images into editable text, making it a valuable tool for digitizing printed documents. While it performs best with high-quality images, proper preprocessing can enhance its accuracy on noisier inputs. Its widespread use and strong community support make it a popular choice for many OCR applications.

4.3.1 Implementation Of Tesseract OCR

```
import cv2
import pytesseract

# Function to extract text from an image
def extract_text_from_image(image_path):
    # Read the image using OpenCV
    img = cv2.imread(image_path)

    # Use pytesseract to extract text from the pre-processed image
    text = pytesseract.image_to_string(img)

    return text

# Image path
 image_path  =  '/content/drive/MyDrive/Task3/test_image/Final  Printed Text Image.jpg'

# Extract text
extracted_text = extract_text_from_image(image_path)

# Display the extracted text
print("Extracted Text:")
print(extracted_text)
```

4.3.2 Result and Observation's using Tesseract OCR

In Extracted Text we can see that we are getting '|' before 'Here' which is not initially present in the Input Image.

Input Image :-

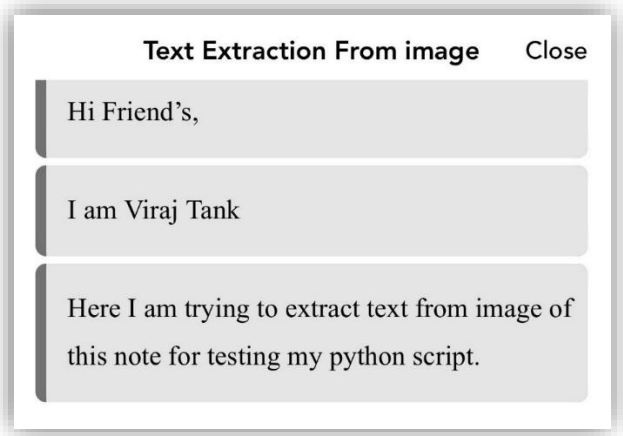


Figure 4-1 Input to Tesseract OCR

Result of Tesseract OCR :-

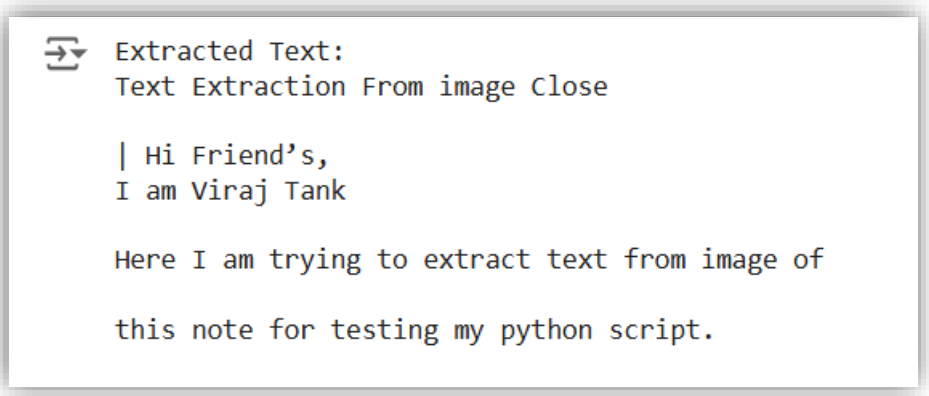


Figure 4-2 Output of Tesseract OCR

Making our own system to read text from pictures is very hard and takes a lot of time and work. We need to gather lots of pictures, teach the computer to read them, and fix any mistakes it makes. Instead of doing all this hard work, we can use a tool called Tesseract OCR, which is already good at reading text from pictures. Tesseract is easy to use, free, and can read many different types of text. We used Tesseract in a simple Python script to get text from pictures, but had some trouble getting the right result. Even with this small problem, Tesseract is still a great tool to use because it saves a lot of time and effort.

4.4 Enhancing OCR Accuracy Through Preprocessing

Image quality is paramount for effective OCR. Preprocessing techniques play a vital role in enhancing text recognition accuracy. Key methods include:

- Grayscale Conversion:
  Converting an image to grayscale simplifies the data by removing color, which reduces complexity for the OCR engine.
- Noise Reduction and Blurring:
  Techniques such as Gaussian blur help in smoothing the image, eliminating extraneous details without erasing important text features.
- Thresholding:
  This technique converts the image to a binary format (black and white), making the text stand out clearly against the background.

Now we will use Python to extract text from images in two ways: first without preprocessing and then with preprocessing steps like grayscale conversion, blurring, and thresholding. By comparing the results, we will observe how preprocessing helps the computer read text more accurately. Preprocessing is the process of cleaning and improving the image before passing it to the OCR tool. It helps make the text in the image clearer and easier for the computer to understand.

4.4.1 Workflow for Introducing Preprocessing the Image

- Input Image :-

  An image containing text is selected as the input.
- Preprocessing the Image :-

  To improve the image quality for OCR, the following preprocessing steps are applied:
  - Grayscale Conversion :- The image is converted to grayscale, removing colors. This helps reduce unnecessary information that may confuse.
  - Gaussian Blur :- A blurring filter is applied to smooth the image and reduce noise. Noise such as random pixels or distortions is minimized, making the text clearer.
  - Thresholding :- This makes the text stand out more distinctly against the background.
  - Character Filtering :- After text extraction, the output is cleaned by removing unwanted symbols and keeping only valid characters like letters and numbers.
- Text Extraction with Preprocessing :-

  The preprocessed image is passed to Tesseract OCR.
- Text Extraction Without Preprocessing :-

  The original image is directly passed to Tesseract OCR without applying any modifications or enhancements.
- Accuracy Comparison :-

  The accuracy of the text extracted is measured by comparing it with the actual text using the SequenceMatcher library.

4.4.2 Implementing Preprocessing with Tesseract

```
1. !pip install opencv-python pytesseract -q
2. !apt-get install tesseract-ocr -q
3.
4. import json
5. import cv2
```

```
import numpy as np
import pytesseract
from matplotlib import pyplot as plt
import string
from difflib import SequenceMatcher

def preprocessing(image):
    print("---------------------------------------")
    print("Original Image")
    plt.imshow(image)
    plt.show()
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray = ((gray / 255) ** 20 * 255).astype(np.uint8)
    print("---------------------------------------")
    print("Gray Scale Image")
    plt.imshow(gray)
    plt.show()
    blur = cv2.GaussianBlur(gray, (3, 3), 0)
       thresh = cv2.threshold(blur, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]
    print("---------------------------------------")
    print("Thresholded Image")
    plt.imshow(thresh)
    plt.show()
     data = pytesseract.image_to_string(thresh, lang='eng', config='--psm 6')
    allowed = string.ascii_uppercase + string.ascii_lowercase + " ?"
    lines = data.split("\n")
    res = []
    for line in lines:
        t = []
        correct = 0
        for i in line:
            if i in allowed:
                t.append(i)
                correct += 1
            elif i in "1234567890":
                t.append(i)
            elif len(t) > 0 and t[-1] != " ":
                t.append(" ")
        if correct > len(line) * 3 / 5 and correct > 2:
            res.append("".join(t))
    data = "\n".join(res)
    return data

def process(image):
    a = preprocessing(image)
    return a
# Path to image file
image_path = "/content/drive/MyDrive/Task3/test_image/test-7.jpg"
 #image_path = "/content/drive/MyDrive/Task3/test_image/ddu_3.png"   # Update this with your image's path
# Actual text for comparison
actual_text = "TO MEME OR NOT TO MEME?\n THAT IS THE QUESTION"
```

```
59.
60. # Read and process the image
61. img = cv2.imread(image_path)
62.
63. processed_output = process(img)
64.
65. # Tesseract prediction
66. tesseract_output = pytesseract.image_to_string(image_path, lang='eng',
config='--psm 6')
67. print("----------------------------------------")
68. print("Tesseract Prediction \n" + tesseract_output)
69.
70. # Custom processing and prediction
71. print("----------------------------------------")
72. print("Processed Output Text \n" + processed_output)
73.
74. # Compare the results
75.     tesseract_ratio     =     SequenceMatcher(None,     actual_text,
tesseract_output).ratio()
76.     processed_ratio     =     SequenceMatcher(None,     actual_text,
processed_output).ratio()
77.
78. print("----------------------------------------")
79. print("Tesseract Accuracy Ratio = {:.2f}".format(tesseract_ratio))
80.      print("Custom      Processing      Accuracy      Ratio      =
{:.2f}".format(processed_ratio))
```

4.4.3 Result and Observation's using Preprocessing with Tesseract

- The accuracy ratio of text extracted directly using Tesseract OCR was only 0.53.
- After applying preprocessing techniques like grayscale conversion, Gaussian blur, and thresholding, the accuracy ratio improved to 0.99.

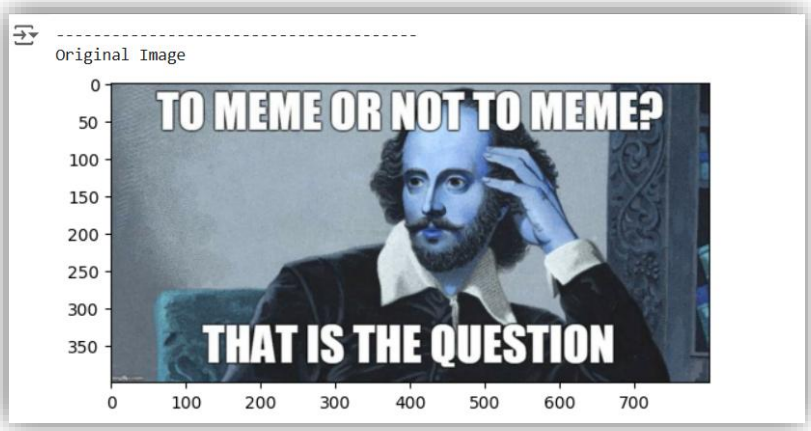


Figure 4-3 Input Image for Preprocessing

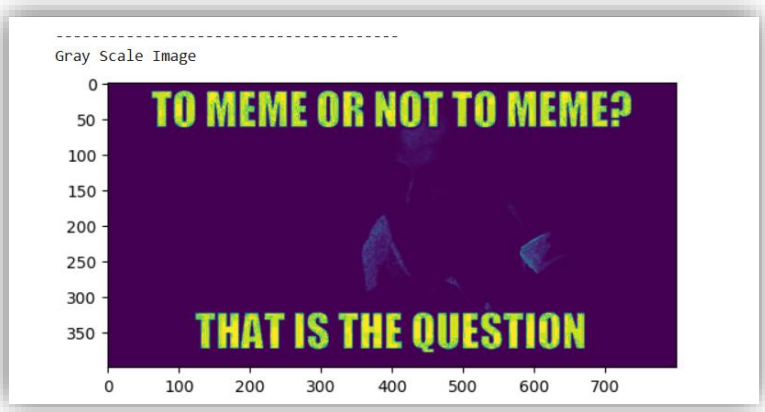


Figure 4-4 Converted Gray Scale Image

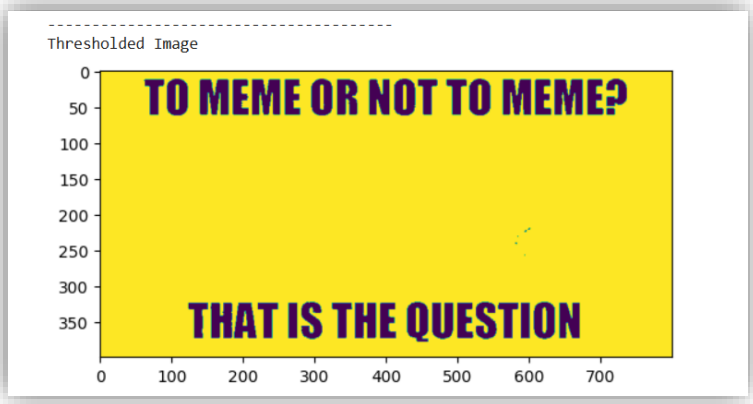


Figure 4-5 Converted Thresholded Image

```
----------------------------------------
Tesseract Prediction
__TO\MEMEOR|NOT,TO\MEME?
- rmN |
WMS si

----------------------------------------
Processed Output Text
TO MEME OR NOT TO MEME?
THAT IS THE QUESTION
```

Figure 4-6 Extracted text with and without using Preprocessing

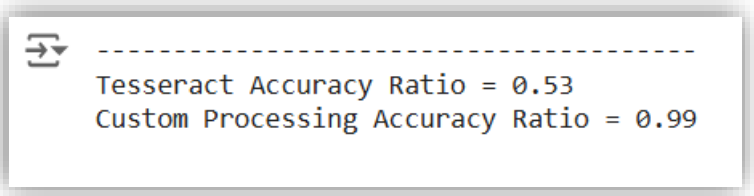


Figure 4-7 Accuracy Comparison with and without using Preprocessing

In conclusion, this test shows that cleaning up the image before using OCR makes a big difference. Without any changes to the image, the OCR got only 53% of the text right. But when we made the image clearer by changing it to black and white, smoothing it, and making the text sharper, the OCR got 99% of the text right. This proves that preparing the image before using OCR helps the computer read the text much more accurately. Because of this, Tesseract is mainly used to extract text from scanned or printed copies, where the text is clear, and not from coloured images where the text might be harder to read.

Chapter 5 Comparative Analysis of OCR Tools

5.1 Introduction to other OCR Tools

Similar to Tesseract OCR there are tools, like PyOCR, and PaddleOCR which help the computer find and understand words in images or scanned papers. Each tool works differently, and we will see which one is best for different types of text. The goal is to check how fast these tools work, how many words they get right, and how easy they are to use. This will help us choose the best tool for future tasks.

5.2 What is PyOCR and PaddleOCR

PyOCR:

PyOCR is a Python wrapper that simplifies the use of various OCR engines by providing a unified interface. It streamlines text extraction tasks with minimal setup, making it convenient for straightforward OCR applications. However, its performance may be limited when handling complex layouts or noisy images compared to more advanced alternatives.

PaddleOCR:

PaddleOCR is a deep learning–based OCR system developed by PaddlePaddle, offering robust performance across multiple languages and complex document formats. It excels at accurately detecting and recognizing text even in challenging conditions, such as low-quality images or diverse font styles. Its advanced neural network architecture and user-friendly interface make it a powerful tool for modern OCR applications.

5.2.1 Workflow for Analysis of OCR Tools

- Prepare the image:
  Load the image into the computer and change its colors to make it easier for the tools to read.
- Show the original image:
  Display the original image so you can see what the computer is trying to read.
- Use OCR tools to read the image:
  First, use Tesseract OCR to read the text and check how long it takes. Then, try PyOCR and measure the time. Finally, use PaddleOCR and note how long it takes.

- Show the text found by each OCR tool:

  For each tool, show the text it found and the time taken to extract it.

- Check the accuracy of the text:

  Compare the text each tool found with the correct text to see how accurate each tool is.

- Compare the tools:

  Look at the results and decide which tool was the most accurate and which one was the fastest.

5.2.2 Implementing Analysis of OCR Tools

```
!pip install pyocr -q
!sudo apt install tesseract-ocr -q
!pip install pytesseract -q
!pip install easyocr -q
!pip install paddlepaddle -q
!pip install paddleocr -q

# Import libraries
import PIL.Image
import pyocr
import pyocr.builders
from paddleocr import PaddleOCR
import pytesseract
import cv2
import time
from difflib import SequenceMatcher
from matplotlib import pyplot as plt

# Function to calculate accuracy
def calculate_accuracy(reference, extracted):
    """
    Calculates accuracy as the similarity ratio between reference and extracted text.
    """
    return SequenceMatcher(None, reference, extracted).ratio()


# Tesseract OCR
def extract_text_with_tesseract(image_path):
    """
    Extracts text from an image using Tesseract OCR.
    """
    try:
        image = cv2.imread(image_path)
        start_time = time.time()
        extracted_text = pytesseract.image_to_string(image)
        end_time = time.time()
```

```
        time_taken = end_time - start_time
        return extracted_text.strip(), time_taken
    except Exception as e:
        return f"Error occurred: {str(e)}", 0


# Initialize the PyOCR engine
tools = pyocr.get_available_tools()
if len(tools) == 0:
  print("No OCR tool found.")
exit(1)
ocr_tool = tools[0]


# PyOCR
def extract_text_with_pyocr(image_path):
    """
    Extracts text from an image using PyOCR.
    """
    try:
        image = PIL.Image.open(image_path)
        start_time = time.time()
                    extracted_text  =  ocr_tool.image_to_string(image,
builder=pyocr.builders.TextBuilder())
        end_time = time.time()
        time_taken = end_time - start_time
        return extracted_text.strip(), time_taken
    except Exception as e:
        return f"Error occurred: {str(e)}", 0


# Paddle OCR
def extract_text_with_paddleocr(image_path):
    """
    Extracts text from an image using PaddleOCR.
    """
    try:
        ocr = PaddleOCR(use_angle_cls=True, lang='en')
        start_time = time.time()
        results = ocr.ocr(image_path)
        end_time = time.time()
       extracted_text = "\n".join([line[1][0] for line in results[0]])
        time_taken = end_time - start_time
        return extracted_text.strip(), time_taken
    except Exception as e:
        return f"Error occurred: {str(e)}", 0

# Input Image and Reference Text
image_path = '/content/drive/MyDrive/Task5/test_image/ddu_2.jpg'
reference_text = "DHARMSINH DESAI UNIVERSITY"

# Read and process the image
img = cv2.imread(image_path)

```

```
# Convert from BGR to RGB
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
print("Original Image")
plt.imshow(img_rgb)
plt.show()


# Extract text using OCR
print("Using Tesseract OCR...")
            tesseract_text,             tesseract_time             =
extract_text_with_tesseract(image_path)
print(f"Tesseract OCR Text:\n{tesseract_text}")
print(f"Time taken by Tesseract OCR: {tesseract_time:.2f} seconds")


# Extract text using PyOCR
print("\nUsing PyOCR...")
pyocr_text, pyocr_time = extract_text_with_pyocr(image_path)
print(f"PyOCR Text:\n{pyocr_text}")
print(f"Time taken by PyOCR: {pyocr_time:.2f} seconds")


# PaddleOCR
print("\nUsing PaddleOCR...")
            paddleocr_text,             paddleocr_time             =
extract_text_with_paddleocr(image_path)
print(f"PaddleOCR Text:\n{paddleocr_text}")
print(f"Time taken by PaddleOCR: {paddleocr_time:.2f} seconds")


# Accuracy Comparison
print("\nComparison:")
 print(f"Tesseract OCR Accuracy: {calculate_accuracy(reference_text,
tesseract_text):.2f}")
    print(f"PyOCR    Accuracy:    {calculate_accuracy(reference_text,
pyocr_text):.2f}")
  print(f"Paddle  OCR  Accuracy:  {calculate_accuracy(reference_text,
paddleocr_text):.2f}")
```

5.2.3 Result and Observation's for Analysis of OCR Tools

- Tesseract OCR's accuracy was 0% and it took 0.31sec.
- PyOCR's accuracy was also 0% and it took 0.17sec.
- PaddleOCR's accuracy was 100% and it took 0.31sec.

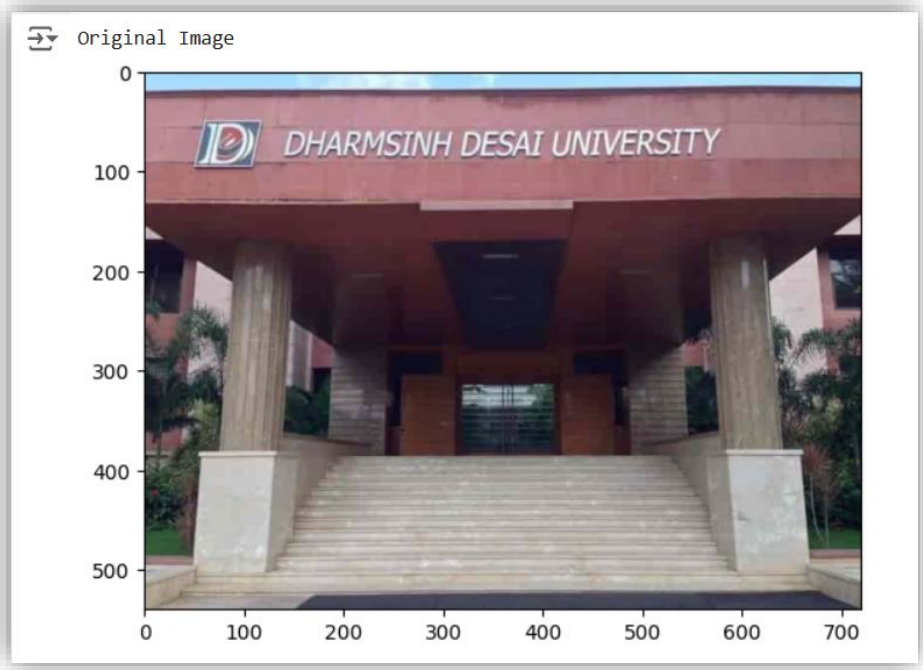


Figure 5-1 Input Image for Analysis of OCR Tools

```
Using Tesseract OCR...
Tesseract OCR Text:

Time taken by Tesseract OCR: 0.31 seconds
```

Figure 5-2 Output Of Tesseract

```
Using PyOCR...
PyOCR Text:

Time taken by PyOCR: 0.17 seconds
```

Figure 5-3 Output Of PyOCR

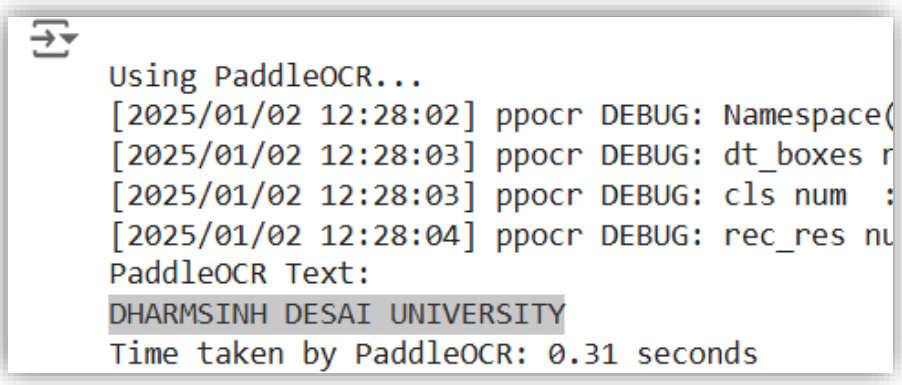


Figure 5-4 Output of PaddleOCR

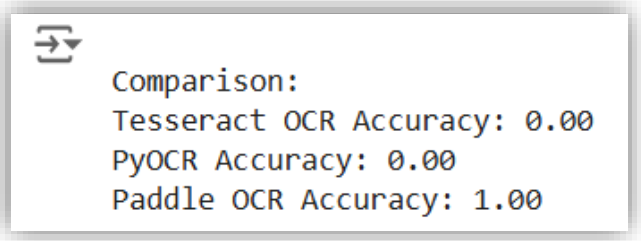


Figure 5-5 Comparison Of Accuracy of each OCR

From the tests, we found that Tesseract OCR and PyOCR didn't extract any text from the image. However, PaddleOCR worked perfectly and got all the text with 100% accuracy. Tesseract and PaddleOCR took the same time of 0.31 seconds, while PyOCR was a little faster at 0.17 seconds.

In this case, PaddleOCR was the best because it read the text correctly and was just as fast as Tesseract. But if the image is a little tougher or has more complicated text, these all tools might not work well. In such cases, we would need to use more advanced OCR tools like KerasOCR or EasyOCR to get better results.

5.3 Advanced OCR Tools

Now we will explore how computers read text from images using OCR tools like Tesseract OCR, EasyOCR, and Keras-OCR. Previously PaddleOCR worked perfectly, while Tesseract and PyOCR failed, showing that some tools work better for specific tasks. Now, we'll compare these three tools to see which is fastest, most accurate, and easiest to use. The goal is to find the best tool for tasks like digitizing documents or reading text from photos, especially for complex images where advanced tools might be needed. By testing these tools, we aim to

understand their strengths and weaknesses, helping us choose the right OCR tool for different real-world applications. This comparison will also give us insights into how well these tools handle challenges like noisy backgrounds or handwritten text.

5.4 What is EasyOCR and KerasOCR

EasyOCR:

EasyOCR is a deep learning–based OCR tool that supports multiple languages and delivers robust text detection and recognition, even in challenging conditions. Its user-friendly API simplifies the integration of OCR capabilities into applications. Additionally, EasyOCR provides confidence scores and bounding boxes, making it valuable for validating and fine-tuning extracted text.

KerasOCR:

KerasOCR leverages the Keras framework to provide an end-to-end OCR pipeline that combines text detection and recognition. It is particularly adept at handling complex layouts, offering detailed bounding box annotations alongside recognized text. Its modular design allows for customization and fine-tuning, making it a strong choice for advanced OCR applications.

5.4.1 Workflow for Comparison between EasyOCR and KerasOCR

- Prepare the image:
  Load the image into the computer and change its colors to make it easier for the tools to read.
- Show the original image:
  Display the original image so you can see what the computer is trying to read.
- Extract Text Using Tesseract OCR:
  Use Tesseract OCR to extract text from both images. Measure the time it takes for each image and save the results.
- Extract Text Using EasyOCR:
  Use EasyOCR to extract text and confidence scores from both images. Measure the time it takes and store the results in a table for better organization.
- Extract Text Using KerasOCR:

Use KerasOCR to extract text and bounding boxes from both images. Measure the time and save the results in a table for comparison.

- Display OCR Results:
  For each image, show the text extracted by Tesseract, EasyOCR, and KerasOCR. Include the time each tool took to process the image.

- Visualize Results:
  Compare the bounding boxes and text visually for EasyOCR and KerasOCR. Use side-by-side plots to see how well each tool detected text in the images.

5.4.2 Implementing Comparison between EasyOCR and KerasOCR

```
!pip install keras-ocr --upgrade --force-reinstall -q
!pip install --force-reinstall -v "tensorflow==2.15.1" -q

!pip install easyocr -q
!pip install pytesseract -q
!apt-get install tesseract-ocr -q

# Import libraries
import easyocr
import keras_ocr
import pytesseract
from pytesseract import Output
import cv2
import time
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt

# Tesseract OCR
def extract_text_with_tesseract(image_path):
    """
    Extracts text from an image using Tesseract OCR.
    """
    try:
        # Read the image
        image = cv2.imread(image_path)

        # Perform OCR using Tesseract
        start_time = time.time()
        extracted_text = pytesseract.image_to_string(image)
        end_time = time.time()

        time_taken = end_time - start_time
        return extracted_text, time_taken

    except Exception as e:
        return f"Error occurred: {str(e)}", 0
```

```
39.
40. # Easy OCR
41. def extract_with_easyocr(image_path):
42.     """
43.     Extracts text and bounding boxes from an image using EasyOCR and
measures time.
44.     """
45.     reader = easyocr.Reader(['en'], gpu=True)
46.     start_time = time.time()
47.     results = reader.readtext(image_path)
48.     end_time = time.time()
49.     df = pd.DataFrame(results, columns=['bbox', 'text', 'conf'])
50.     time_taken = end_time - start_time
51.     return df, time_taken
52.
53. # Keras OCR
54. def extract_with_kerasocr(image_path):
55.     """
56.     Extracts text and bounding boxes from an image using KerasOCR and
measures time.
57.     """
58.     pipeline = keras_ocr.pipeline.Pipeline()
59.     start_time = time.time()
60.     results = pipeline.recognize([image_path])
61.     end_time = time.time()
62.     df = pd.DataFrame(results[0], columns=['text', 'bbox'])
63.     time_taken = end_time - start_time
64.     return df, time_taken
65.
66. # Visualization for comparison
67. def plot_compare(img_fn, easyocr_df, kerasocr_df):
68.     """
69.     Plots EasyOCR and KerasOCR results side by side for a single image.
70.     """
71.     fig, axs = plt.subplots(1, 2, figsize=(15, 10))
72.
73.     # EasyOCR results
74.     easy_results = easyocr_df[['text', 'bbox']].values.tolist()
75.     easy_results = [(x[0], np.array(x[1])) for x in easy_results]
76.     keras_ocr.tools.drawAnnotations(plt.imread(img_fn), easy_results,
ax=axs[0])
77.     axs[0].set_title('EasyOCR Results', fontsize=24)
78.
79.     # KerasOCR results
80.     keras_results = kerasocr_df[['text', 'bbox']].values.tolist()
81.     keras_results = [(x[0], np.array(x[1])) for x in keras_results]
82.     keras_ocr.tools.drawAnnotations(plt.imread(img_fn), keras_results,
ax=axs[1])
83.     axs[1].set_title('KerasOCR Results', fontsize=24)
84.
85.     plt.show()
86.
87. # Path to the input image
88. image_path_1 = "/content/drive/MyDrive/Task5/test_image/ddu_2.jpg"
```

```
image_path_2 = "/content/drive/MyDrive/Task5/test_image/ddu.jpg"

# Process image
img_1 = cv2.imread(image_path_1)
img_rgb_1 = cv2.cvtColor(img_1, cv2.COLOR_BGR2RGB)
print("Test Image 1")
plt.imshow(img_rgb_1)
plt.show()
img_2 = cv2.imread(image_path_2)
img_rgb_2 = cv2.cvtColor(img_2, cv2.COLOR_BGR2RGB)
print("\nTest Image 2")
plt.imshow(img_rgb_2)
plt.show()

# Tesseract OCR
print("\nUsing Tesseract OCR...")
            tesseract_text,              tesseract_time              =
extract_text_with_tesseract(image_path_1)
print(f"\nTesseract Results For Test Image 1:\n{tesseract_text}")
print(f"Time taken: {tesseract_time:.2f} seconds")
            tesseract_text,              tesseract_time              =
extract_text_with_tesseract(image_path_2)
print(f"\nTesseract Results For Test Image 2:\n{tesseract_text}")
print(f"Time taken: {tesseract_time:.2f} seconds")

# EasyOCR
print("\nUsing EasyOCR...")
easyocr_df_1, easyocr_time_1 = extract_with_easyocr(image_path_1)
print(f"\n\nEasyOCR Results For Test Image 1:\n{easyocr_df_1[['text',
'conf']]}")
print(f"Time taken: {easyocr_time_1:.2f} seconds")
easyocr_df_2, easyocr_time_2 = extract_with_easyocr(image_path_2)
print(f"\n\nEasyOCR Results For Test Image 2:\n{easyocr_df_2[['text',
'conf']]}")
print(f"Time taken: {easyocr_time_2:.2f} seconds")

# KerasOCR
print("\nUsing KerasOCR...")
kerasocr_df_1, kerasocr_time_1 = extract_with_kerasocr(image_path_1)
      print(f"\n\nKerasOCR      Results      For      Test      Image
1:\n{kerasocr_df_1[['text']]}")
print(f"Time taken: {kerasocr_time_1:.2f} seconds")
kerasocr_df_2, kerasocr_time_2 = extract_with_kerasocr(image_path_2)
      print(f"\n\nKerasOCR      Results      For      Test      Image
2:\n{kerasocr_df_2[['text']]}")
print(f"Time taken: {kerasocr_time_2:.2f} seconds")

# Visualization
print("\nVisualizing OCR Results...")
print("\n\n")
plot_compare(image_path_1, easyocr_df_1, kerasocr_df_1)
print("\n\n")
plot_compare(image_path_2, easyocr_df_2, kerasocr_df_2)
```

5.4.3 Result and Observation's for Comparison between EasyOCR and KerasOCR

- Easy OCR took 8.64sec for image 1 and 6sec for image 2.
- Keras OCR took 45.17sec for image 1 and 27.85sec for image 2.
- According to accuracy KerasOCR out performs EasyOCR and According to time taken EasyOCR out performs KerasOCR.

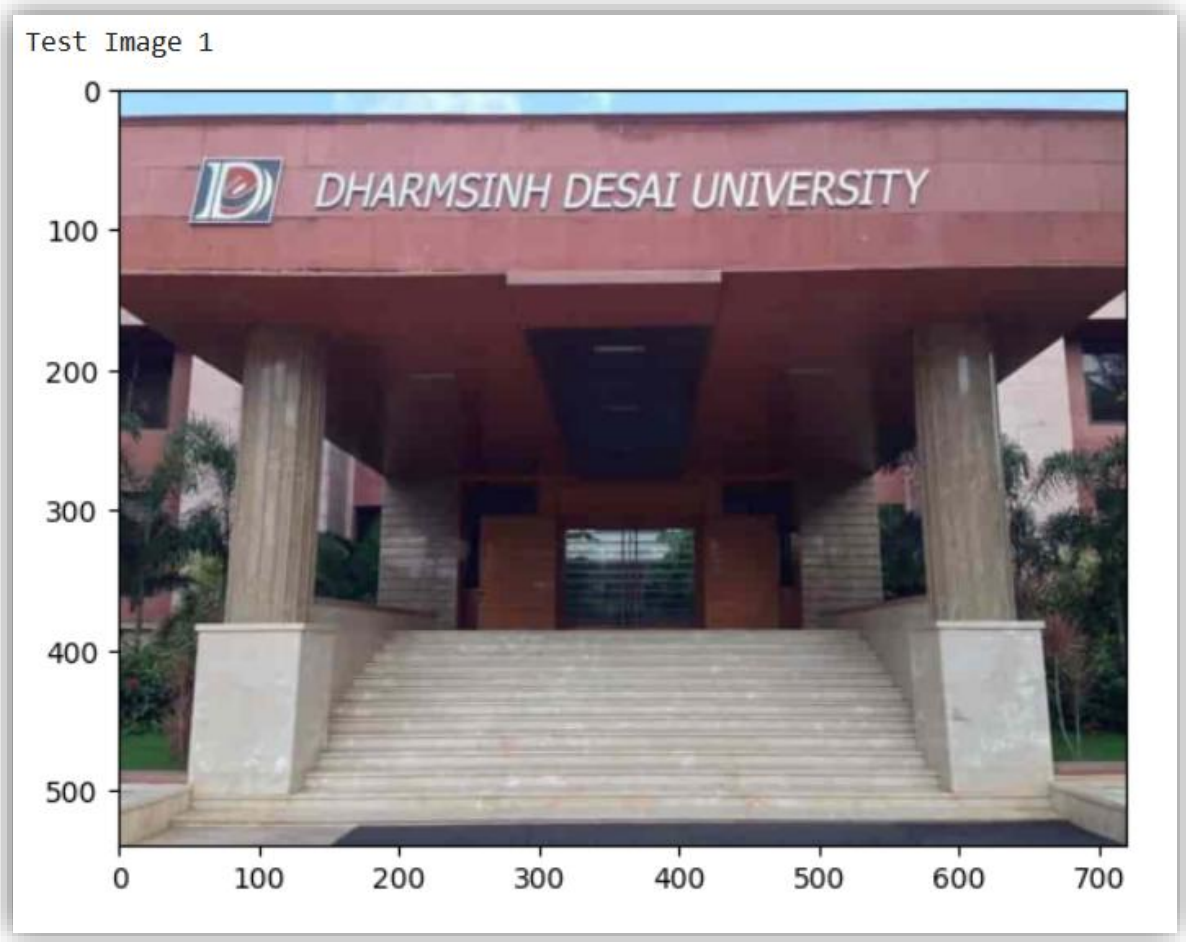


Figure 5-6 Input Image 1 Comparison between EasyOCR and KerasOCR

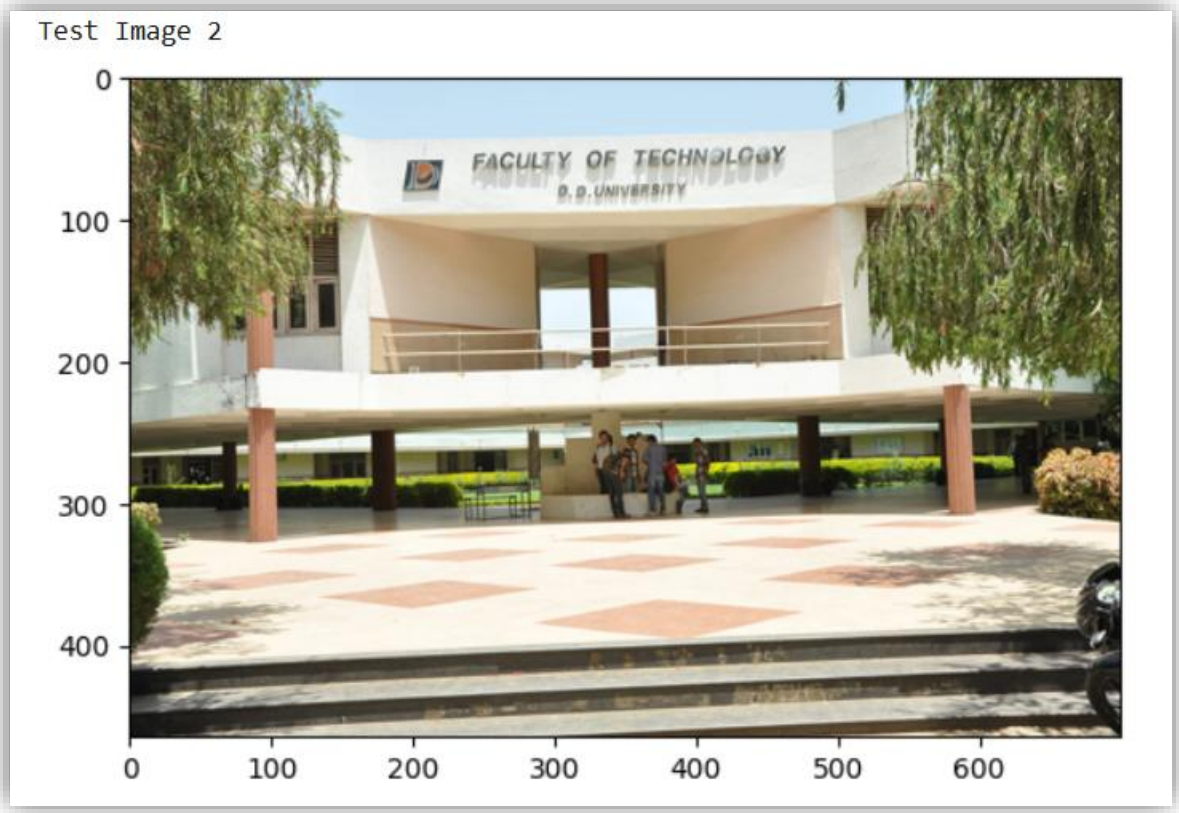


Figure 5-7 Input Image 2 for Comparison between EasyOCR and KerasOCR

```
Using Tesseract OCR...

Tesseract Results For Test Image 1:


Time taken: 0.52 seconds

Tesseract Results For Test Image 2:



GULTY OF TEGHN@L@eY
2,2, UNIVERSITY

Time taken: 0.75 seconds
```

Figure 5-8 Output Of Tesseract

```
WARNING:easyocr.easyocr:Neither CUDA nor MPS

Using EasyOCR...
WARNING:easyocr.easyocr:Neither CUDA nor MPS


EasyOCR Results For Test Image 1:
                          text      conf
0  DHARMSINH DESAI UNIVERSITY  0.885446
Time taken: 8.64 seconds


EasyOCR Results For Test Image 2:
         text      conf
0     FACULTY  0.982230
1          OF  0.778774
2  TECHNOLOuY  0.525393
3  university  0.658975
Time taken: 6.00 seconds
```

Figure 5-9 Output Of EasyOCR

```
Using KerasOCR...
Looking for /root/.keras-ocr/craft_mlt_25k.h5
Looking for /root/.keras-ocr/crnn_kurapan.h5
1/1 [==============================] - 29s 29s/step
1/1 [==============================] - 3s 3s/step


KerasOCR Results For Test Image 1:
         text
0  university
1   dharmsinh
2       desai
Time taken: 45.17 seconds
Looking for /root/.keras-ocr/craft_mlt_25k.h5
Looking for /root/.keras-ocr/crnn_kurapan.h5
1/1 [==============================] - 21s 21s/step
1/1 [==============================] - 6s 6s/step


KerasOCR Results For Test Image 2:
         text
0  technology
1     faculty
2          of
3   univesity
4           d
5          di
Time taken: 27.85 seconds
```

Figure 5-10 Output Of Keras OCR

Figure 5-11 Visual Comparison between EasyOCR and KerasOCR (1)

Figure 5-12 Visual Comparison between EasyOCR and KerasOCR (2)

Figure 5-13 Visual Comparison between EasyOCR and KerasOCR (3)

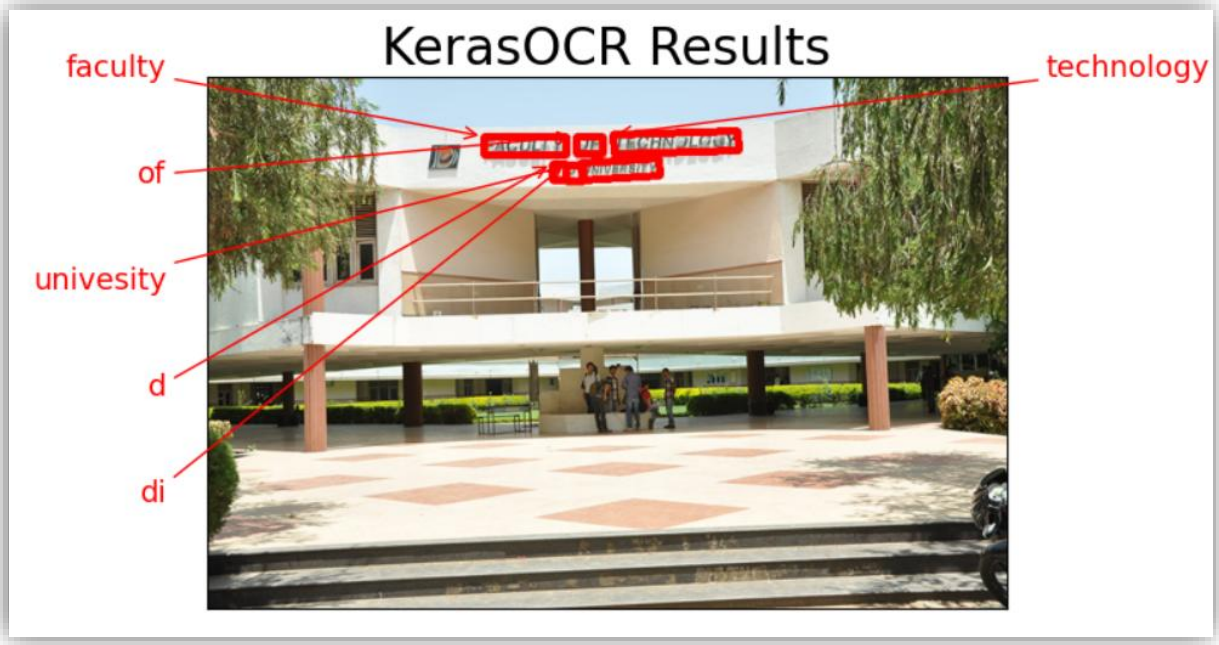


Figure 5-14 Visual Comparison between EasyOCR and KerasOCR (4)

From the tests, we observe that EasyOCR is much faster, taking 8.64 seconds for Image 1 and 6 seconds for Image 2. In comparison, KerasOCR took longer, with 45.17 seconds for Image 1 and 27.85 seconds for Image 2.

However, when it comes to accuracy, KerasOCR performed better than EasyOCR, making it more reliable for extracting text correctly.

If speed is the priority, EasyOCR is the better choice. But if accuracy is more important, KerasOCR is the preferred tool despite its longer processing time.

5.5 Comparison of all OCR Tools

A variety of OCR tools have been explored to determine the best fit. Here, we compare several popular OCR engines based on key criteria:

OCR Tool	Accuracy	Speed	Notable Feature
Tesseract	Moderate	Fast	Widely used; best with clean, high-quality images.
PyOCR	Low to Moderate	Very Fast	Lightweight; struggles with complex document layouts.
PaddleOCR	High	Moderate	Excellent with varied fonts and noisy backgrounds.
EasyOCR	High	Slow	Supports multiple languages; provides confidence scores.
KerasOCR	Very High	Very Slow	Deep learning–based; effective with complex layouts.

Table 5.1 Comparison of all OCR Tools

- Tesseract OCR has been a long-standing choice for many OCR applications; however, its performance may vary with image quality.
- PyOCR offers simplicity and speed but often falls short in handling images with cluttered backgrounds.
- PaddleOCR and EasyOCR represent modern approaches that incorporate deep learning, yielding high accuracy across a range of conditions.
- KerasOCR is particularly promising for advanced applications where complex document layouts need to be processed.

Chapter 6 Advanced Data Extraction Techniques

6.1 Introduction to Regex

After converting invoice images to raw text through OCR, the next crucial step is to extract meaningful, structured information. This chapter explains how advanced data extraction techniques, particularly regular expressions (regex) and additional text processing methods, are used to identify key fields—such as invoice numbers, dates, totals, and vendor details—from the unstructured OCR output.

6.1.1 What is Regex

1) **Regex** :- Regex (Regular Expressions) is a powerful tool used to find specific patterns in text. It works like a search query but is much more flexible and precise. For example, if you want to extract an invoice number or an address from a block of text, regex can help you find exactly what you're looking for by defining a pattern. It's especially useful for pulling out structured information (like dates, amounts, or IDs) from unstructured text. To get a proper idea how regex works see below example's.

   - **r"Invoice Number\s*:\s*(\S+)"**
     This pattern looks for the phrase "Invoice Number", followed by a colon and optional spaces (\s*).
     (\S+): This part captures the actual invoice number, which is a sequence of non-whitespace characters (like INV-1234).
     If the text is "Invoice Number: INV-1234", the regex will extract INV-1234 as the invoice number as sequence of non-whitespace characters (like INV-1234).
   - **r"Billing Address\s*:\s*([\s\S]*?)\n\n"**
     This pattern looks for the phrase "Billing Address", followed by a colon and optional spaces (\s*).
     ([\s\S]*?): This part captures everything (including spaces and newlines) after the colon until it finds two newlines (\n\n), which usually marks the end of the address.

6.2 Introduction to JSON

JSON (JavaScript Object Notation) is a way to store information in a neat and organized format. Think of it like a labelled box where you can put things. We use JSON because it's easy to read

and write, and it organizes data into clear key-value pairs. Unlike a plain text file, JSON tells you exactly what each piece of information means. It's also lightweight and works with almost every programming language, making it perfect for sharing data between computers or over the internet. For example, if you save an invoice number and date in a plain text file, it might look messy, but in JSON, it's clear and organized. This makes JSON a better choice for storing and sharing data.

6.3 Extracting Structured Data from Raw OCR Output

The OCR process provides a text output that often contains noise, misrecognized characters, and inconsistent formatting. Advanced data extraction techniques address these challenges by:

- **Parsing and Structuring Data:** Identifying patterns within the text to extract fields like invoice number, date, and total amount.
- **Error Correction:** Applying regex and post-processing methods to correct common OCR mistakes.
- **Data Validation:** Ensuring that the extracted information conforms to expected formats (e.g., dates, currency).

6.4 Techniques and Tools

Regular expressions are the primary tool used for scanning through the raw text to find patterns that match invoice components. Python's re module offers powerful functionality for:

- **Pattern Matching:** Creating regex patterns to detect dates, numbers, and alphanumeric sequences.
- **Substitution and Cleaning:** Removing unwanted characters or correcting misrecognized symbols.
- **Aggregation:** Organizing extracted data into a structured format like dictionaries or dataframes for further processing.

6.5 Implementation of Regex

```
1. !pip install pytesseract -q
2. !pip install pillow -q
3. !pip install paddleocr -q
4. !pip install paddlepaddle -q
5. !sudo apt install tesseract-ocr -q
6.
7. # Import Libraries
```

```
import pytesseract
from PIL import Image
import re
import json
from paddleocr import PaddleOCR

# Function to extract text using Tesseract OCR
def extract_text_tesseract(image):
    return pytesseract.image_to_string(image)

# Function to extract text using PaddleOCR
def extract_text_paddleocr(image_path):
    ocr = PaddleOCR(use_angle_cls=True, lang='en')
    result = ocr.ocr(image_path)
    return "\n".join([line[1][0] for line in result[0]])

# Function to extract data using regex
def extract_data_using_regex(text, patterns):
    extracted_data = {}
    for field, pattern in patterns.items():
        match = re.search(pattern, text)
          extracted_data[field] = match.group(1).strip() if match else None
    return extracted_data

# Function to save the extracted data to JSON and display it on the console
def save_to_json(data, file_path):
    with open(file_path, "w") as json_file:
        json.dump(data, json_file, indent=4)
    print("Data saved to json file")
    print(json.dumps(data, indent=4))

# Path to the image on Google Drive
 image_path = "/content/drive/MyDrive/Task3/invoices/invoice_pages-to-jpg-0001.jpg"
output_file_path = "/content/drive/MyDrive/Colab Notebooks/task8.json"

# Load the image
image = Image.open(image_path)

# Extract text using Tesseract OCR
extracted_text_tesseract = extract_text_tesseract(image)

# Extract text using PaddleOCR
extracted_text_paddle = extract_text_paddleocr(image_path)

# Define regex patterns for fields
patterns = {
    "invoice_number": r"Invoice Number\s*:\s*(\S+)",
    "billing_address": r"Billing Address\s*:\s*([\s\S]*?)\n\n",
    "shipping_address": r"Shipping Address\s*:\s*([\s\S]*?)\n\n",
    "order_number": r"Order Number\s*:\s*(\S+)",
    "order_date": r"Order Date\s*:\s*(\S+)",
```

```
    "invoice_date": r"Invoice Date\s*:\s*(\S+)",
    "pan_no": r"PAN No\s*:\s*(\S+)",
    "gst_no": r"GST Registration No\s*:\s*(\S+)",
    "total_amount": r"Total Amount\s*:\s*₹([\d,]+(?:\.\d{2})?)",
}

# Organize data using regex from Tesseract OCR
extracted_data_tesseract = extract_data_using_regex(extracted_text_tesseract, patterns)

# Organize data using regex from PaddleOCR
extracted_data_paddle = extract_data_using_regex(extracted_text_paddle, patterns)


# Combine results from both OCRs
comparison_data = {
    "Tesseract OCR": extracted_data_tesseract,
    "PaddleOCR": extracted_data_paddle
}

# Save the extracted to .json file
save_to_json(comparison_data, output_file_path)
```

6.5.1 Result and Observation's using Regex

- For fields like billing address and shipping address, Tesseract OCR does not provide the desired output. The extracted text is also containing the text from its side column's which is not desired.
- In the case of billing address and shipping address, PaddleOCR returns null values, meaning because of two columns it fails to extract any data for these fields.

Figure 6-1 Output on Console using Regex

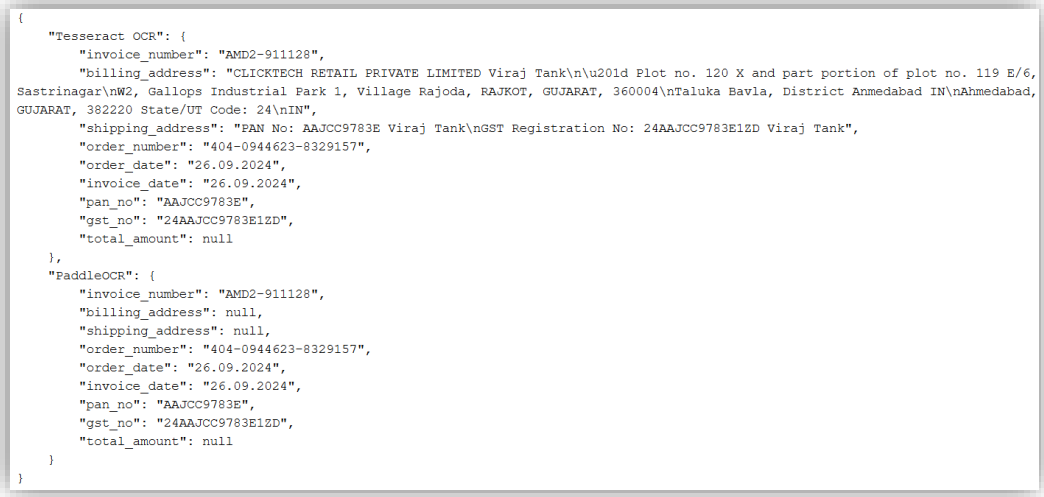


Figure 6-2 Generated JSON File

In conclusion, the system works well for extracting simple fields like invoice numbers and dates, but it struggles with multi-line content such as billing and shipping addresses. This is because the data in these fields is spread across multiple lines and sometimes arranged in different columns, making it harder for the OCR tools to extract accurately. While we can modify the code to handle these fields better, it would only work for invoices with a similar layout, like the Amazon invoice used in this case. This means the code would lose its flexibility to process different types of invoices. For now, the system is effective for structured data but requires improvements to handle more complex or varied invoice formats.

Chapter 7 Integration of AI in Invoice Processing

7.1 Introduction to Gemini-1.5-flash

Now, we will use a powerful AI model, to extract information from invoices. Unlike the previous task, where we could only work with one type of invoice (like Amazon invoices), Gemini-1.5-flash can handle any kind of invoice—whether it's from a restaurant, an online store, or a utility bill. This makes it much more flexible and useful. Gemini-1.5-flash can understand complex layouts, including multi-line text and tables, and extract details like supplier information, addresses, item lists, payment details, and taxes.

7.2 Overview of Gemini Variants

Model variant	Input(s)	Output	Optimized for
gemini-2.0-flash	Audio, images, videos, and text	Text, images (coming soon), and audio (coming soon)	Next generation features, speed, and multimodal generation for a diverse variety of tasks
gemini-2.0-flash-lite	Audio, images, videos, and text	Text	A Gemini 2.0 Flash model optimized for cost efficiency and low latency
gemini-1.5-flash	Audio, images, videos, and text	Text	Fast and versatile performance across a diverse variety of tasks
gemini-1.5-flash-8b	Audio, images, videos, and text	Text	High volume and lower intelligence tasks
gemini-1.5-pro	Audio, images, videos, and text	Text	Complex reasoning tasks requiring more intelligence
text-embedding-004	Text	Text embeddings	Measuring the relatedness of text strings

Table 7.1 Overview of Gemini Variants

We used gemini-1.5-flash because it delivers fast and versatile performance, making it ideal for real-time text extraction from invoices. Its optimization for a wide range of tasks ensures that it efficiently processes multimodal input like images and text while maintaining the speed needed for interactive applications. This balance of speed and versatility is essential for our Invoice Assistant, which requires rapid, accurate extraction of structured data without the additional overhead of more complex models.

7.3 Why Gemini 1.5 Flash

Gemini-1.5-flash is a fast and versatile model optimized for handling a variety of inputs including audio, images, videos, and text. It features an impressive input token limit of 1,048,576 and an output token limit of 8,192, ensuring it can process extensive data efficiently. Additionally, it supports up to 3,600 images per prompt, 1 hour of video, and approximately 9.5 hours of audio. This robust specification makes it ideal for real-time applications, delivering the speed and reliability needed for our Invoice Assistant's advanced data extraction tasks.

7.4 Configuration Process for Gemini API Key

- Visit to "https://aistudio.google.com".
- Or else you can do same from google cloud console.
- Click on Get API Key.

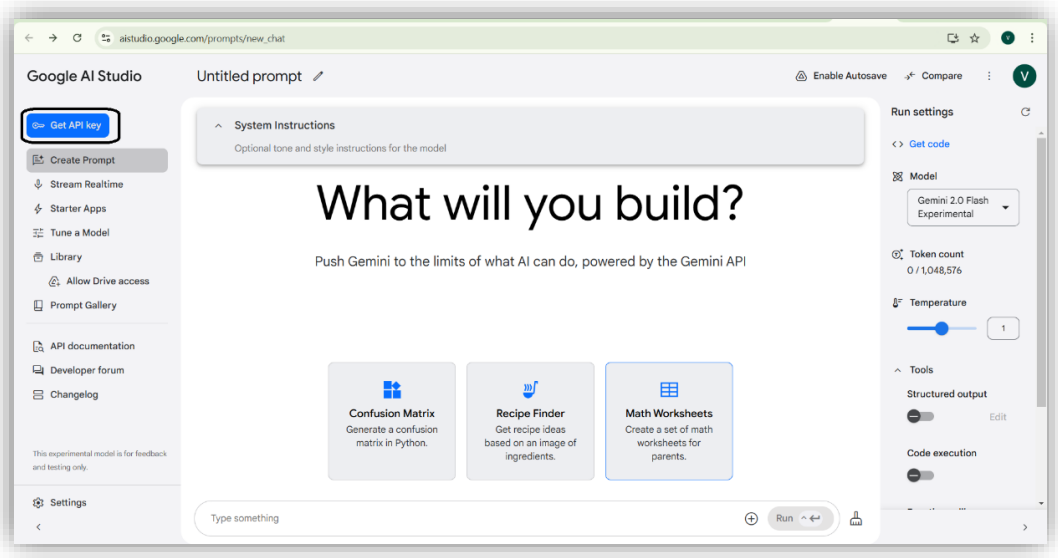


Figure 7-1 Get Gemini-1.5-flash API Key

- Now Create API Key.

Figure 7-2 Create Gemini-1.5-flash API Key

- Now copy the API Key for further use.

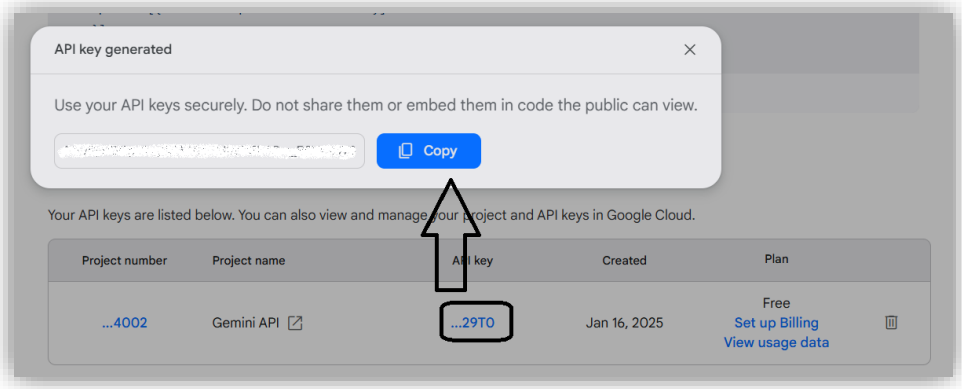


Figure 7-3 Copy Gemini-1.5-flash API Key

7.5 Workflow for Integrating AI in Invoice Processing

- Install Required Libraries:
  Install necessary libraries like google.generativeai, json, and matplotlib to process invoices and visualize results.

- Set Up Google Gemini:
  Configure the Google Gemini API using the provided API key to enable communication with the AI model.

- Load the Invoice Image:
  Load the invoice image from the specified file path.
- Prepare Image Data:
  Convert the invoice image into a format that can be processed by Google Gemini.
- Extract Invoice Fields:
  Use the "gemini-1.5-flash" model to extract detailed information from the invoice, such as supplier details, billing and shipping addresses, item lists, payment details, and tax calculations.
- Clean the Extracted Data:
  Remove unnecessary delimiters or formatting from the extracted text to prepare it for JSON conversion.
- Convert Data to JSON:
  Convert the cleaned text into a structured JSON format for easy access and further use.
- Save Data to JSON File:
  Save the extracted and organized data into a JSON file for easy access and sharing.
- Display the Results:
  Print the extracted data on the console and display the input invoice image for verification.

7.6 Implementation of Integrating AI in Invoice Processing

```
import json
import google.generativeai as genai
import os
import cv2
import matplotlib.pyplot as plt

# Set the API key directly
from google.colab import userdata
genai.configure(api_key=userdata.get('GOOGLE_API_KEY'))

# Function to Prepare Image For Google Gemini
def prepare_image_data(file_path):
    if not os.path.exists(file_path):
        raise FileNotFoundError(f"File not found: {file_path}")

    with open(file_path, "rb") as file:
        bytes_data = file.read()
```

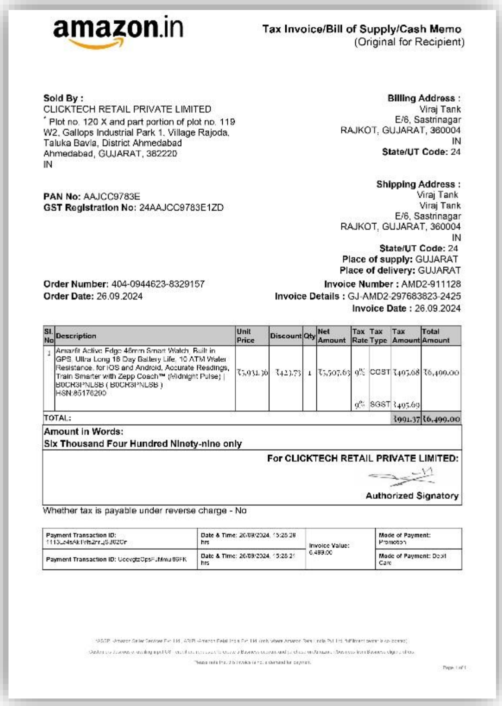
```
18.
19.     image_data = [
20.         {
21.             "mime_type": "image/jpeg",
22.             "data": bytes_data
23.         }
24.     ]
25.     return image_data
26.
27. # Function for Initialization of Google Gemini
28. def extract_invoice_fields(image_data):
29.     input_prompt ="""
30.            # Complete prompt is written in collab file please refer
that.
31.                """
32.
33.     model = genai.GenerativeModel('gemini-1.5-flash')
34.
35.     response = model.generate_content([input_prompt, image_data[0]])
36.     return response.text
37.
38. # Function to Save data to Json
39. def save_to_json(data, output_path):
40.     with open(output_path, "w") as json_file:
41.         json.dump(data, json_file, indent=4)
42.
43.
44. # Main Function
45. def main():
46.
47.     # Input and Output File Paths
48.                                                 #input_file_path         =
"/content/drive/MyDrive/Task3/invoices/invoice_pages-to-jpg-0001.jpg"
49.      input_file_path = "/content/drive/MyDrive/Task3/invoices/our-food-
bill-gives-an.jpg"
50.                   output_file_path    =    "/content/drive/MyDrive/Colab
Notebooks/task9.json"
51.
52.     # Read and Print the Image
53.     img = cv2.imread(input_file_path)
54.     print("Input Invoice Image")
55.     plt.imshow(img)
56.     plt.axis('off')
57.     plt.show()
58.
59.     try:
60.         # Prepare image data
61.         image_data = prepare_image_data(input_file_path)
62.
63.         # Extract fields from invoice using Gemini
64.         response_text = extract_invoice_fields(image_data)
65.
```

```python
        # Cleaning the initial_response to remove delimiters and language specifier
        json_content = response_text.strip('```').replace('json\n', '', 1).strip()

        # Convert response text to JSON
        extracted_data = json.loads(json_content)

        # Save extracted data to a JSON file
        save_to_json(extracted_data, output_file_path)

        print("\nFields extracted successfully!")
        print(json.dumps(extracted_data, indent=4))
    except Exception as e:
        print(f"An error occurred: {e}")

if __name__ == "__main__":
    main()
```

7.7 Result and Observation's using Gemini-1.5-flash

- Now, we are getting all the details properly, including the billing and shipping addresses.
- Some fields that are not present in the invoice are marked as null.

Test Image 1



amazon.in

Tax Invoice/Bill of Supply/Cash Memo
(Original for Recipient)

Sold By :
CLICKTECH RETAIL PRIVATE LIMITED
* Plot no. 120 X and part portion of plot no. 119
W2, Gallops Industrial Park 1, Village Rajoda,
Taluka Bavla, District Ahmedabad
Ahmedabad, GUJARAT, 382220
IN

Billing Address :
Viraj Tank
E/6, Sastrinagar
RAJKOT, GUJARAT, 360004
IN
State/UT Code: 24

PAN No: AAJCC9783E
GST Registration No: 24AAJCC9783E1ZD

Shipping Address :
Viraj Tank
Viraj Tank
E/6, Sastrinagar
RAJKOT, GUJARAT, 360004
IN
State/UT Code: 24
Place of supply: GUJARAT
Place of delivery: GUJARAT

Order Number: 404-0944623-8329157
Order Date: 26.09.2024

Invoice Number : AMD2-911128
Invoice Details : GJ-AMD2-297683823-2425
Invoice Date : 26.09.2024

Sl. No	Description	Unit Price	Discount	Qty	Net Amount	Tax Rate	Tax Type	Tax Amount	Total Amount
1	Amazfit Active Edge 46mm Smart Watch, Built in GPS, Ultra Long 16 Day Battery Life, 10 ATM Water Resistance, for iOS and Android, Accurate Readings, Train Smarter with Zepp Coach™ (Midnight Pulse) | B0CR3PNLSB (B0CR3PNLSB) HSN:85176290	₹5,931.36	₹423.73	1	₹5,507.63	9%	CGST	₹495.68	₹6,499.00
						9%	SGST	₹495.69	
TOTAL:								₹991.37	₹6,499.00

Amount in Words:
Six Thousand Four Hundred Ninety-nine only

For CLICKTECH RETAIL PRIVATE LIMITED:

Authorized Signatory

Whether tax is payable under reverse charge - No

Payment Transaction ID: [illegible]	**Date & Time:** 26/09/2024, 15:28:28 hrs	**Invoice Value:** 6,499.00	**Mode of Payment:** Promotion
Payment Transaction ID: [illegible]	**Date & Time:** 26/09/2024, 15:28:21 hrs		**Mode of Payment:** Debit Card

Page 1 of 1

Figure 7-4 Input Image 1 using Gemini

Output of Image 1


```
Fields extracted successfully!
{
    "Supplier": {
        "Name": "CLICKTECH RETAIL PRIVATE LIMITED",
        "Address": "Plot no. 120 X and part portion of plot no. 119 W2, Gallops Industrial
Park 1, Village Rajoda, Taluka Bavla, District Ahmedabad Ahmedabad, GUJARAT, 382220 IN",
        "PAN Number": "AAJCC9783E",
        "GST Number": "24AAJCC9783E1ZD",
        "Contact": null
    },
    "ReceiptDetails": {
        "Billing Address": "Viraj Tank E/6, Sastrinagar RAJKOT, GUJARAT, 360004 IN",
        "Shipping Address": "Viraj Tank E/6, Sastrinagar RAJKOT, GUJARAT, 360004 IN",
        "Invoice Number": "AMD2-911128",
        "Date": "2024-09-26",
        "Time": "15:28:29"
    },
    "Items": [
        {
            "ItemName": "Amazfit Active Edge 46mm Smart Watch, Built in GPS, Ultra-Long 16-Day
Battery Life, 10 ATM Water Resistance, for iOS and Android, Accurate Readings, Train Smarter
with Zepp Coach \u2122 (Midnight Pulse) | B0CR3PNLSB (B0CR3PNLSB)",
            "Quantity": 1,
            "Unit Price": "5507.63",
            "TotalPrice": "6499.00"
        }
    ],
    "Payment Details": {
        "Payment Method": "Promotion/Debit Card",
        "Currency": "Indian Rupees",
        "Total Amount": "6499.00",
        "Taxes": "991.37",
        "Discounts": "423.73"
    },
    "Tax calculation": {
        "CGST": "495.68",
        "SGST": "495.69",
        "Total Tax": "991.37"
    }
}
```


Test Image 2

Figure 7-5 Input Image 2 using Gemini

Output of Image 2


```
Fields extracted successfully!
{
    "Supplier": {
        "Name": "HOTEL AMER PALACE RESTAURANT",
        "Address": "Hoshangabad Road, Ratanpur, Bhopal-462046, Madhya Pradesh",
        "GSTIN": "23AABFH6030L1ZN",
        "TIN": "23894007031",
        "Contact": null
    },
    "ReceiptDetails": {
        "Billing Address": null,
        "Shipping Address": null,
        "Invoice Number": "GRT1715",
        "Date": "2021-07-24",
        "Time": "15:14"
    },
    "Items": [
        {
            "ItemName": "VEG MANCHAW SO",
            "Quantity": 1,
            "Unit Price": "119.00",
            "TotalPrice": "119.00"
        },
        {
            "ItemName": "VEG SWEET CORN",
            "Quantity": 1,
```

```
            "Unit Price": "119.00",
            "TotalPrice": "119.00"
        },
        {
            "ItemName": "DAL TADKA",
            "Quantity": 1,
            "Unit Price": "215.00",
            "TotalPrice": "215.00"
        },
        {
            "ItemName": "JEERA RICE",
            "Quantity": 1,
            "Unit Price": "145.00",
            "TotalPrice": "145.00"
        },
        {
            "ItemName": "PLAIN PAPAD",
            "Quantity": 2,
            "Unit Price": "40.00",
            "TotalPrice": "80.00"
        },
        {
            "ItemName": "BAKED VEG WITH",
            "Quantity": 1,
            "Unit Price": "270.00",
            "TotalPrice": "270.00"
        },
        {
            "ItemName": "MUSHROOM MATTA",
            "Quantity": 1,
            "Unit Price": "265.00",
            "TotalPrice": "265.00"
        },
        {
            "ItemName": "BUTTER TANDOOR",
            "Quantity": 1,
            "Unit Price": "27.00",
            "TotalPrice": "27.00"
        },
        {
            "ItemName": "MISSI ROTI",
            "Quantity": 1,
            "Unit Price": "30.00",
            "TotalPrice": "30.00"
        },
        {
            "ItemName": "GARLIC NAAN",
            "Quantity": 1,
            "Unit Price": "45.00",
            "TotalPrice": "45.00"
        }
    ],
    "Payment Details": {
        "Payment Method": null,
        "Currency": "Indian Rupee",
        "Total Amount": "1381.00",
        "Taxes": "65.76",
        "Discounts": null
    },
    "Tax Calculation": {
        "CGST": "32.88",
        "SGST": "32.88",
        "Total Tax": "65.76"
    },
    "Subtotal": 1315.24
}
```

Here, we successfully used Google Gemini to extract detailed information from invoices. Unlike the previous task, which was limited to specific invoice types (like Amazon invoices), this system can now handle any kind of invoice, whether it's from a restaurant, e-commerce platform, or utility bill. All fields, including billing and shipping addresses, are extracted correctly, and the data is saved in a JSON file for easy access and further use. This makes the system highly flexible and useful for various applications.

However, there is one drawback: the free version of Gemini allows only 15 requests per minute. This limitation needs to be considered when processing multiple invoices.

Chapter 8 Interactive Invoice Query System

8.1 Introduction to Query System

Here, we will create a system where users can ask questions about an invoice and get instant answers. Using Google Gemini, a smart AI model, the system can read and understand the content of an invoice. Unlike earlier tasks, this system is interactive, meaning users can ask multiple questions—like "What is the invoice number?" or "What is the total amount?"—and get quick, conversational answers. This makes it a simple and user-friendly tool for understanding invoice details without any hassle.

8.2 Workflow for Interactive Query System

- Install Required Libraries:
  Install necessary libraries like google.generativeai, json, and matplotlib to process invoices and visualize results.
- Set Up Google Gemini:
  Configure the Google Gemini API using the provided API key to enable communication with the AI model.
- Load the Invoice Image:
  Load the invoice image from the specified file path.
- Prepare Image Data:
  Convert the invoice image into a format that can be processed by Google Gemini.
- Ask User Questions:
  Prompt the user to ask questions about the invoice, such as the invoice number, billing address, or total amount.
- Get Gemini's Response:
  Use the "gemini-1.5-flash" model to generate answers to the user's questions based on the invoice content.
- Display the Response:
  Show the response to the user in a conversational format.

- Continue or Exit:

  Ask the user if they want to ask another question. If yes, repeat the process; if no, end the session.

8.3 Implementation of Interactive Query System

```
import os
from PIL import Image
import google.generativeai as genai

# Set the API key directly
from google.colab import userdata
genai.configure(api_key=userdata.get('GOOGLE_API_KEY'))

# Function to load the Gemini model and get a response
def get_gemini_response(input_prompt, image, question):
    model = genai.GenerativeModel('gemini-1.5-flash')
        response = model.generate_content([input_prompt, image[0], question])
    return response.text

# Function to load and process the image
def input_image_setup(image_path):
    if not os.path.exists(image_path):
        raise FileNotFoundError(f"The file does not exist.")

    # Open the image and convert to bytes
    with open(image_path, "rb") as file:
        bytes_data = file.read()

    image_parts = [
        {
            "mime_type": "image/jpeg",
            "data": bytes_data
        }
    ]
    return image_parts

# Main Function
def main():
    image_path = "/content/drive/MyDrive/Task3/invoices/invoice_pages-to-jpg-0001.jpg"

    try:
        image_data = input_image_setup(image_path)
    except FileNotFoundError as e:
        print(e)
        return

    input_prompt = """
        You are an expert in understanding invoices.
        You will receive input images as invoices &
```

```
45.          You are given with the data and user query.
46.          You just need to answer based on the information provided.
47.          Answer in a conversational manner, as if talking to a human.
48.           Any questions outside the information in the invoice will be
ignored and not answered.
49.          Thank you!
50.      """
51.
52.      while True:
53.          # Ask the user for a question
54.          print("\nAsk a question about the invoice:")
55.          question = input()
56.
57.          # Get the response from the Gemini model
58.               response  =  get_gemini_response(input_prompt,  image_data,
question)
59.
60.          # Display the response
61.          print(response)
62.
63.          # Ask if the user wants to ask another question
64.          continue_prompt = input("\nDo you want to ask another question?
(yes/no): ").strip().lower()
65.          if continue_prompt not in ("yes", "y"):
66.              print("\nGoodbye!")
67.              break
68.
69.
70. # Run the program
71. if __name__ == "__main__":
72.      main()
```

8.4 Result and Observation's for Interactive Query System

- We can ask multiple Questions about the invoice and the system answer's perfectly.
- The system works well for real-time queries, making it a practical tool for quickly extracting and understanding invoice details.
- It provides accurate responses to questions like order date, delivery address, and other invoice details.

```
Ask a question about the invoice:
What was the order date?
The order date was 26.09.2024.


Do you want to ask another question? (yes/no): yes

Ask a question about the invoice:
What was the delivery address?
The delivery address was Viraj Tank, E/6, Sastrinagar, RAJKOT, GUJARAT, 360004, IN.


Do you want to ask another question? (yes/no): no

Goodbye!
```

Figure 8-1 Interactive Query and Answer's

Here, we created an interactive system that allows users to ask questions about an invoice and get instant, accurate answers. Using Google Gemini, the system can understand and respond to queries like order date, delivery address, and total amount in a conversational manner. Unlike previous tasks, this system just focuses on a particular query asked by the user. So the system will only provide the data which user ask's and not all the data which might be a case for some user's.

Next, we will combine both functionality to create a complete and proper project. This integrated system will automate the entire process—from extracting and organizing invoice data to allowing users to interactively query and understand the details—making it a powerful tool for businesses and individuals.

Chapter 9 Development of Invoice Assistant

9.1 Introduction of Invoice Assistant

In today's digital world, managing invoices manually can be time-consuming and error-prone. The Invoice Assistant is designed to automate the extraction and analysis of invoice data using artificial intelligence. This project integrates a Telegram bot that allows users to upload invoices in PDF or image format, which are then processed using Google Gemini to extract key details such as supplier information, invoice numbers, itemized lists, and payment details.

9.2 System Flowchart of Invoice Assistant

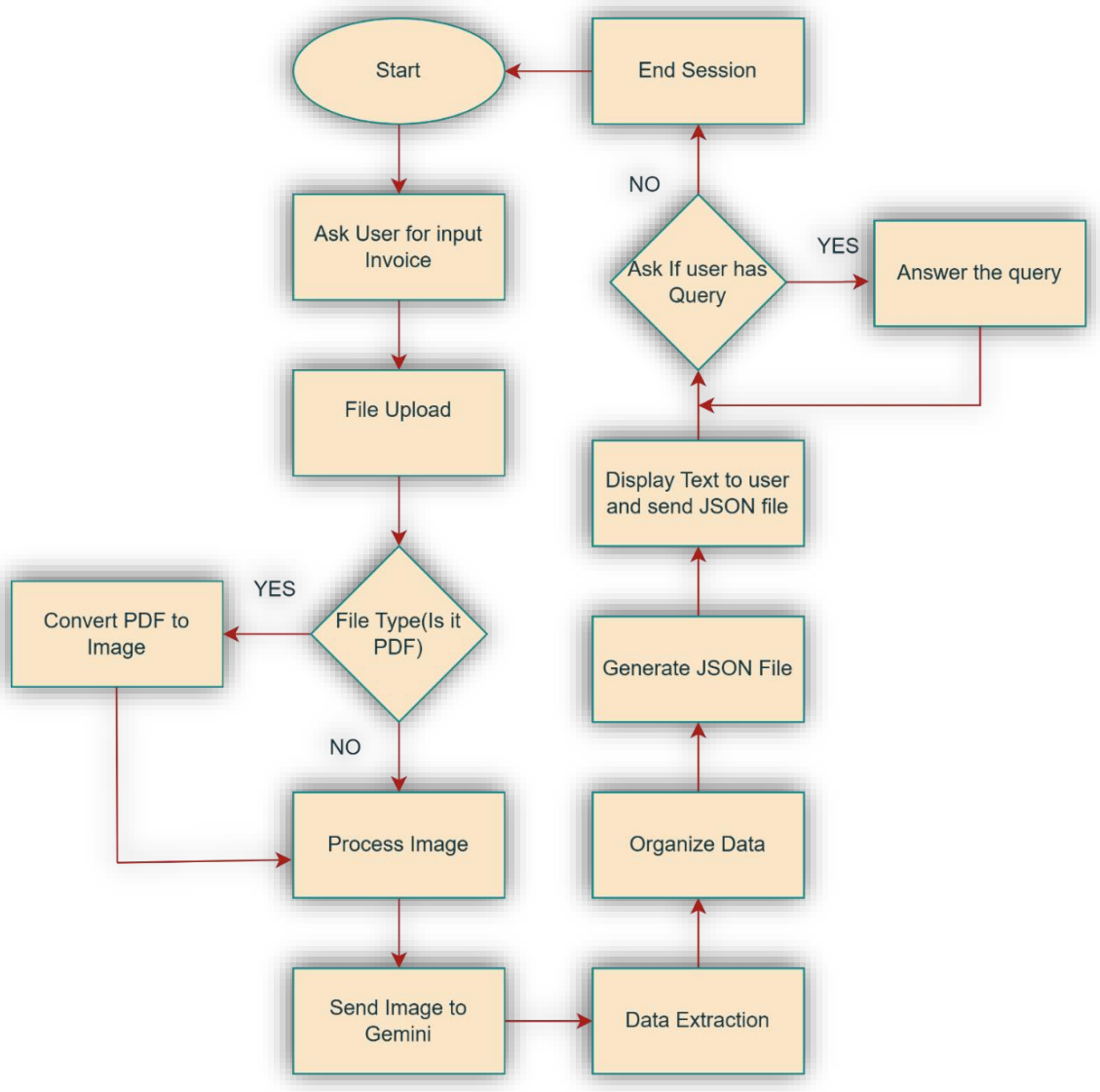


Figure 9-1 System Flowchart of Invoice Assistant

9.3 Detailed Workflow of Invoice Assistant

- System Startup:

  The start function responds to the /start command by greeting the user and instructing them to upload an invoice file. It runs asynchronously to ensure smooth interaction without delays.
- Check File Type:

  Then it manages file uploads from users while handling connection errors. It first checks if the uploaded file is a supported format (PDF or image).
- Convert PDF to Image:

  If the file is a PDF, it is converted into an image for further processing. Then it creates a user specific directory and downloads the file.
- Data Extraction:

  For next step now it sends the invoice image to Google Gemini for text extraction and processes the extracted data.
- Data Cleaning:

  The extracted text is cleaned. That is it removes extra characters with might hinder us while converting it to JSON file.
- Convert to JSON:

  The extracted text is cleaned and converted into JSON format. If valid, the data is saved as a JSON file.
- Send JSON File:

  The data is saved as a JSON file in the user's directory and sent to the user. If the response is not valid JSON, the raw text is shared instead.
- Ask Query:

  Now in next step it allows users to ask questions about their uploaded invoice. It first checks if the user has uploaded an invoice; if not, it prompts them to do so.
- Response to User:

  The user's question is formatted into a structured prompt for Google Gemini, which extracts relevant details from the invoice and responds back to user with an answer.
- End Session:

  Lastly it clears the user's session by removing their data, ensuring a fresh start for the next interaction. It then sends a message confirming that the session has ended and

instructs the user to restart with the /start command if they want to process another invoice.

9.4 Implementation of Invoice Assistant

Block 1

```
import os
import asyncio
import time
from dotenv import load_dotenv
from telegram import Update
from telegram.ext import Application, CommandHandler, MessageHandler, filters, ContextTypes
from pdf2image import convert_from_path
import google.generativeai as genai
import json
import logging
from tenacity import retry, stop_after_attempt, wait_exponential, retry_if_exception_type
from telegram.error import NetworkError, TimedOut

```

Explanation

This block imports all the necessary libraries to set up the Telegram bot, process invoices, and handle errors. It includes modules for handling operating system tasks, asynchronous execution, and loading environment variables securely. The Telegram bot framework allows interaction with users, while pdf2image helps convert PDFs into images for text extraction. Google Gemini is integrated for extracting structured data from invoices. Logging ensures proper tracking of activities, and tenacity provides a retry mechanism to handle network failures, ensuring smooth bot operation. These imports lay the foundation for the entire project.

Block 2

```
# Load environment variables from .env file
load_dotenv()
genai.configure(api_key=os.getenv("GOOGLE_API_KEY"))

# Global configurations
USER_SESSIONS = {}
QUESTION_PROMPT_TEMPLATE = """
        You are an expert in understanding invoices.
        You will receive input images as invoices &
        You are given with the data and user query.
        You just need to answer based on the information provided.
```

```
        Answer in a conversational manner, as if talking to a human.
         Any questions outside the information in the invoice will be ignored and not answered.
        Thank you!
           Answer this question based strictly on the invoice content:
                Question: {question}

                Follow these guidelines:
                 1. Be specific and use exact values from the invoice when possible
               2. If information is missing/unclear, state "Not clearly specified in the invoice"
                3. Format currency values with their symbols (e.g., ₹, $, €)
               4. Never guess or assume values not shown in the invoice
 """
EXTRACTION_PROMPT = """
                    You are an expert in understanding and extracting detailed information from invoices of any kind.
                 You will receive various invoice images as input and need to provide accurate and comprehensive answers based on the extracted details.
                The extracted information should cover a wide range of invoices, including food invoices, e-commerce invoices, utility bills, and any other type of invoice.
                Dont use "\n" in extracted json format anywhere.
                  Below is an example structure, but please adapt and modify the structure as needed for different invoice types:

                {
                    "Supplier": {
                        "Name": "Supplier Name",
                        "Address": "123 Supplier St, City, ZIP",
                        "PAN Number": "PAN123456789",
                        "GST Number": "GST0000",
                        "Contact": "xxxxxxxxxxx"
                    },
                    "ReceiptDetails": {
                        "Billing Address": "Customer Billing Address",
                       "Shipping Address": "Customer Shipping Address",
                        "Invoice Number": "INV123456",
                        "Date": "YYYY-MM-DD",
                        "Time": "HH:MM:SS",
                    },
                    "Items": [
                        {
                            "ItemName": "Product Name or Service",
                            "Quantity": 2,
                            "Unit Price": "50.00",
                            "Total Price": "100.00"
                        },
                        {
                            "ItemName": "Another Product",
                            "Quantity": 3,
```

```
                            "Unit Price": "30.00",
                            "Total Price": "90.00"
                        }
                    ],

                    "Payment Details" :{
                          "Payment Method": "Credit Card / Cash / Bank Transfer / Other",
                        "Currency": "USD",
                        "Total Amount": "250.00",
                        "Taxes": "25.00",
                        "Discounts": "5.00"
                    }
                    "Tax calculation" :{
                     ......
                    }
                }

                Instructions for Processing Invoices:

                1. Tax Calculations:
                      Sum all taxes (e.g., CGST, SGST, and any others) and ensure the total tax amount is accurate.
                     If tax percentages are listed, verify they sum up correctly to the total tax amount.
                      Ensure that the taxable amount plus total taxes equals the final amount after applying any discounts.

                2. Item Extraction:
                     Split item details correctly even if item names or descriptions span multiple rows.
                    Treat every line within the same column as part of a single item.
                     If quantity is more than one then unit price and total price can not be same
                    If unit price is not given of an item then divide amount by quantity to get Unit price

                3. Invoice Generation Time:
                    Look for common formats such as "am/PM" to identify when the invoice was generated or paid.

                4. Data Verification:
                      Cross-verify extracted data to ensure it matches the expected invoice totals:
                     Verify if the total amount equals the sum of item prices plus taxes, minus any discounts.
                      Check if the total quantity of items matches the sum of quantities for each item if provided.

                5. Subtotal Calculation:
                   Ignore extracting 'Subtotal' directly from the image.
                     The actual 'Subtotal' should be calculated as:
                    Subtotal = TotalAmount - TaxAmount + DiscountAmount`
```

```
                    Ensure the validation that:
                    FinalTotal = Subtotal + TaxAmount - DiscountAmount`

                6. Currency Extraction:
                   Don't just use the currency sumbol in json as it is. Find out the name of that currency adn then add that name in currency.
                """


# Configure logging
logging.basicConfig(
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    level=logging.INFO)
logger = logging.getLogger(__name__)


# Retry decorator for network operations
network_retry = retry(
    stop=stop_after_attempt(3),
    wait=wait_exponential(multiplier=1, min=2, max=10),
          retry=retry_if_exception_type((NetworkError,  TimeoutError, ConnectionError)),
      before_sleep=lambda _: logger.warning("Retrying due to network error..."),
    reraise=True
)

```

Explanation

The load_dotenv() function loads environment variables into the program, while genai.configure(api_key=os.getenv("GOOGLE_API_KEY")) retrieves the Google Gemini API key. This keeps sensitive information out of the code, making the project safer and easier to manage.

The USER_SESSIONS dictionary helps keep track of active users and their uploaded invoices. The QUESTION_PROMPT_TEMPLATE ensures that Gemini only answers based on the invoice content, providing accurate responses. The EXTRACTION_PROMPT defines how invoice details should be extracted, including supplier information, payment details, and tax calculations. It also includes specific rules to improve accuracy, such as calculating subtotals correctly and verifying extracted amounts.

Logging is used to monitor system events, track errors, and help with debugging. This makes it easier to find and fix issues if something goes wrong. The network_retry decorator helps handle network-related failures by retrying failed operations up to three times. It uses an

exponential backoff strategy to avoid overwhelming the system and logs warnings before each retry, making the bot more reliable.

Block 3

```
1. # Handles the /start command.
2. async def start(update: Update, context: ContextTypes.DEFAULT_TYPE):
3.     await update.message.reply_text(
4.         "Hi! I'm your Invoice Assistant. Please send me an invoice file
(image or PDF) to get started."
5. )
6.
```

Explanation

The start function responds to the /start command by greeting the user and instructing them to upload an invoice file. It runs asynchronously to ensure smooth interaction without delays.

Block 4

```
 1. # Handles file uploads with connection error handling
 2.     async     def     handle_file(update:     Update,     context:
ContextTypes.DEFAULT_TYPE):
 3.     user = update.message.from_user
 4.     try:
 5.         document = update.message.document
 6.
 7.         # Check if the file format is supported
 8.           if not document.file_name.lower().endswith(('.pdf', '.jpg',
'.jpeg', '.png')):
 9.             await update.message.reply_text("⚠️ Unsupported file format!
Please upload a PDF or image.")
10.              return
11.
12.         # Create user directory
13.         user_dir = f"user_{user.id}"
14.         os.makedirs(user_dir, exist_ok=True)
15.
16.         # Retryable download operation
17.         @network_retry
18.         async def download_with_retry():
19.             file = await document.get_file()
20.             file_path = os.path.join(user_dir, document.file_name)
21.             await file.download_to_drive(file_path)
22.             return file_path
23.
24.       await update.message.reply_text("🔄 Processing your document...")
```

```
25.         file_path = await download_with_retry()
26.
27.         # Convert PDF files to images
28.         if document.file_name.lower().endswith('.pdf'):
29.             file_path = await convert_pdf_to_image(file_path, user_dir)
30.             if not file_path:
31.                 await update.message.reply_text("❌ Failed to process
PDF.")
32.                 return
33.
34.         # Store file path and process
35.         USER_SESSIONS[user.id] = {"image_path": file_path}
36.         await process_invoice_image(update, file_path, user_id=user.id)
37.
38.     except Exception as e:
39.         logger.error(f"File handling error: {str(e)}")
40.         await update.message.reply_text("⚠️ Connection issue detected.
Please try again later.")
41.
```

Explanation

The handle_file function manages file uploads from users while handling connection errors. It first checks if the uploaded file is a supported format (PDF or image). If valid, it creates a user-specific directory and downloads the file using a retry mechanism to handle network failures. If the file is a PDF, it is converted into an image for further processing. The file path is then stored in USER_SESSIONS, and the invoice is processed. If any error occurs during this process, an error message is logged, and the user is notified.

Block 5

```
1. # Converts PDF to JPEG image using threads.
2. async def convert_pdf_to_image(pdf_path: str, output_dir: str):
3.     try:
4.         images = await asyncio.to_thread(convert_from_path, pdf_path)
5.         image_path = f"{output_dir}/invoice.jpg"
6.         await asyncio.to_thread(images[0].save, image_path, "JPEG")
7.         return image_path
8.     except Exception as e:
9.         print(f"PDF conversion error: {e}")
10.         return None
11.
```

Explanation

The convert_pdf_to_image function converts a PDF into a JPEG using threading to keep the process efficient. It saves the first page as invoice.jpg in the output directory. If an error occurs, it logs the issue and returns None.

Block 6

```
# Processes an invoice image, extracts data using Gemini, and sends the result to the user.
async def process_invoice_image(update: Update, image_path: str, user_id: int):
    try:
        # Retryable Gemini API call
        @network_retry
        async def get_gemini_response():
            return await asyncio.to_thread(
                                        genai.GenerativeModel('gemini-1.5-flash').generate_content,
                [EXTRACTION_PROMPT, {"mime_type": "image/jpeg", "data": open(image_path, "rb").read()}]
            )

        response = await get_gemini_response()
        extracted_text = response.text

        # Clean and validate JSON response
        cleaned_json = extracted_text.strip('```').replace('json\n', '', 1).strip()

        try:
            json_data = json.loads(cleaned_json)
            user_dir = f"user_{user_id}"
          base_name = os.path.splitext(os.path.basename(image_path))[0]
            json_filename = f"{base_name}_data.json"
            json_path = os.path.join(user_dir, json_filename)

            with open(json_path, 'w') as json_file:
                json.dump(json_data, json_file, indent=4)

            # Retryable send operations
            @network_retry
            async def send_responses():
               await update.message.reply_text("✅ Extraction complete! Here's the JSON data:")
                await update.message.reply_text(f"{extracted_text}")
                  await update.message.reply_text("✅ Here's the JSON File:")
                                                                    await update.message.reply_document(document=open(json_path,                 'rb'), filename=json_filename)
```

```
35.                     await update.message.reply_text("💡 You can now ask
questions using /ask")
36.                 await send_responses()
37.
38.             except json.JSONDecodeError:
39.                   await update.message.reply_text("⚠️ The response wasn't
valid JSON. Here's the raw text:")
40.                 await update.message.reply_text(extracted_text)
41.
42.         except Exception as e:
43.             logger.error(f"Processing error: {str(e)}")
44.             await update.message.reply_text("⚠️ Service unavailable. Please
try again later.")
45.
```

Explanation

The process_invoice_image function sends the invoice image to Google Gemini for text extraction and processes the extracted data. It uses a retry mechanism to handle network failures when calling Gemini. The extracted text is cleaned and converted into JSON format. If valid, the data is saved as a JSON file in the user's directory and sent to the user. If the response is not valid JSON, the raw text is shared instead. Any errors during processing are logged, and the user is informed if the service is unavailable.

Block 7

```
 1. # Handles user questions about the uploaded invoice
 2.     async     def     ask_question(update:     Update,     context:
ContextTypes.DEFAULT_TYPE):
 3.     user = update.message.from_user
 4.     try:
 5.         if not USER_SESSIONS.get(user.id, {}).get("image_path"):
 6.           await update.message.reply_text("⚠️ Please upload an invoice
first!")
 7.             return
 8.
 9.         question = " ".join(context.args)
10.        full_prompt = QUESTION_PROMPT_TEMPLATE.format(question=question)
11.
12.         # Retryable Gemini API call
13.         @network_retry
14.         async def get_answer():
15.             return await asyncio.to_thread(
16.                                     genai.GenerativeModel('gemini-1.5-
flash').generate_content,
17.                 [full_prompt, {"mime_type": "image/jpeg",
18.                                                         "data":
open(USER_SESSIONS[user.id]["image_path"], "rb").read()}]
```

```
19.             )
20.
21.         response = await get_answer()
22.                                 await    update.message.reply_text(f"📝
Response:\n\n{response.text}")
23.           await update.message.reply_text("💡 Ask another question or
/end to finish")
24.
25.     except Exception as e:
26.         logger.error(f"Question error: {str(e)}")
27.         await update.message.reply_text("⚠️ Service unavailable. Please
try your question again.")
28.
```

Explanation

The ask_question function allows users to ask questions about their uploaded invoice. It first checks if the user has uploaded an invoice; if not, it prompts them to do so. The user's question is formatted into a structured prompt for Google Gemini, which extracts relevant details from the invoice. The function uses a retry mechanism to handle network failures. The bot then sends Gemini's response back to the user and invites them to ask more questions or end the session. If an error occurs, it is logged, and the user is notified.

Block 8

```
1. # Cleans up conversation resources.
2.     async     def     end_conversation(update:     Update,     context:
ContextTypes.DEFAULT_TYPE):
3.     user = update.message.from_user
4.     USER_SESSIONS.pop(user.id, None)
5.     await update.message.reply_text("💬 Session ended. Start again with
/start.")
6.
```

Explanation

The end_conversation function clears the user's session by removing their data from USER_SESSIONS, ensuring a fresh start for the next interaction. It then sends a message confirming that the session has ended and instructs the user to restart with the /start command if they want to process another invoice.

Block 9

```
1. # Main application setup. Initializes the bot and registers handlers.
```

```
def main():
    application = Application.builder().token(os.getenv("TELEGRAM_BOT_TOKEN")).read_timeout(30).write_timeout(30).build()

    # Register handlers
    handlers = [
        CommandHandler('start', start),
        CommandHandler('ask', ask_question),
        CommandHandler('end', end_conversation),
        MessageHandler(filters.Document.ALL, handle_file)
    ]

    for handler in handlers:
        application.add_handler(handler)

    # Start the bot in polling mode
    application.run_polling()


if __name__ == "__main__":
    main()

```

Explanation

The main function sets up and starts the Telegram bot. It initializes the bot with the API token stored in environment variables and configures timeout settings. Command handlers for /start, /ask, and /end are registered, along with a message handler to process uploaded files. The bot runs in polling mode, continuously checking for new messages and responding to user interactions.

9.5 Project Demonstration

To experience the Invoice Assistant in action, scan the QR code below. This will open the Telegram bot, where you can upload an invoice and ask questions about it.

How to Use the Bot:

1. Scan the QR code to open the bot in Telegram.
2. Click Start or type /start to begin.
3. Upload an invoice as a PDF or image (JPG, PNG).
4. Wait for the bot to process and extract details.
5. Ask any questions about the invoice using /ask <your question>.

6. Type /end to close the session.

Scan the QR Code Below to Start the Demo

Figure 9-2 Telegram Bot's QR Code

9.6 Result and Observation's of Invoice Assistant

- The bot successfully extracts and organizes invoice details into a structured JSON format.
- Google Gemini provides accurate results, even for different invoice layouts.
- When providing multiple invoices at a time it extracts data from all invoices but answers only to the last invoice uploaded.
- Sometimes when there is connection issue we have retry mechanism which ensures smooth operation.
- Logging helps in debugging issues and tracking user interactions efficiently.
- Some invoices with unclear text or poor image quality may result in incomplete or inaccurate extraction.
- In this we can still optimise the input prompts for further accuracy for complex layouts.

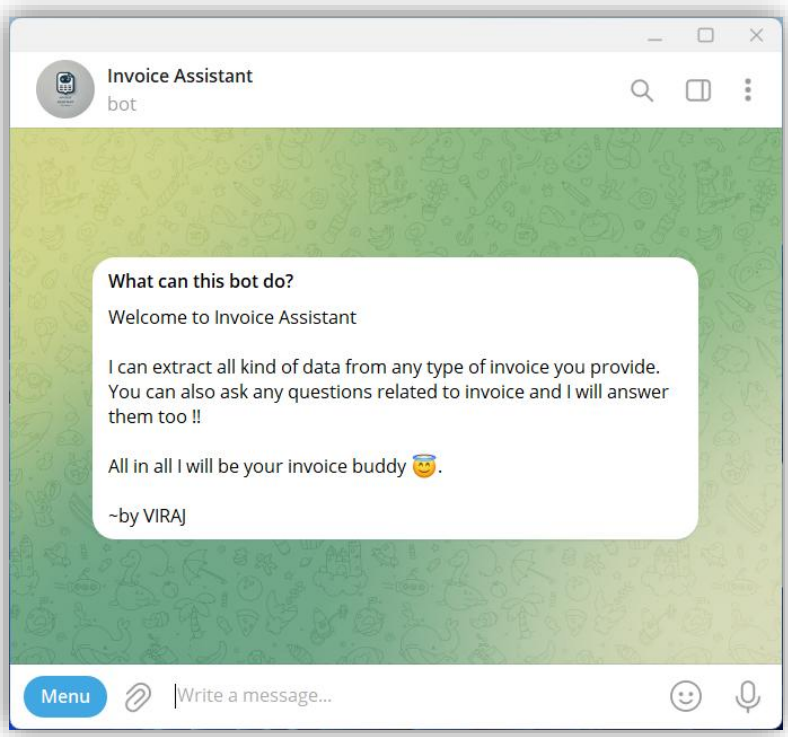


Figure 9-3 Telegram Bot Screenshot 1

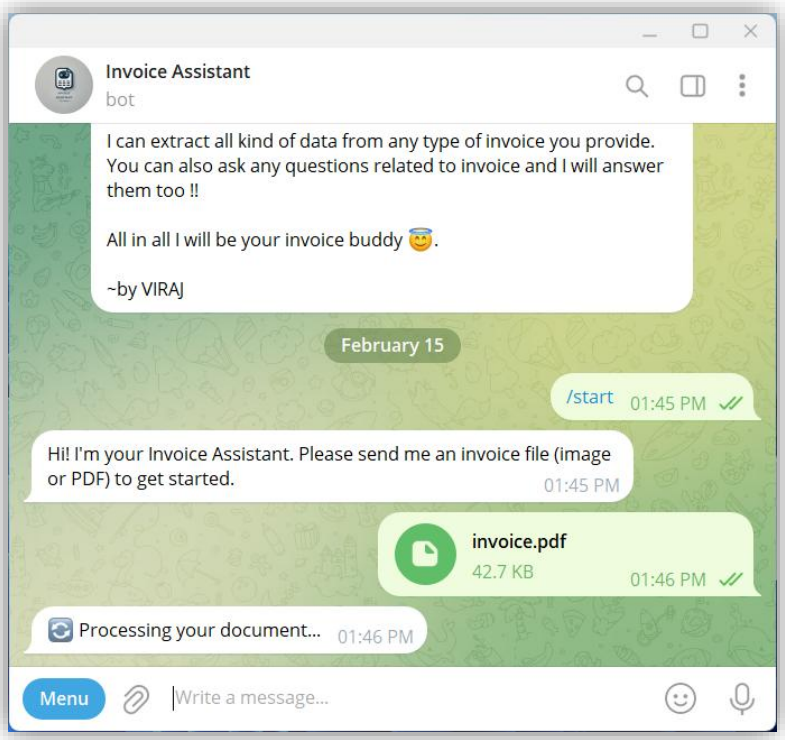


Figure 9-4 Telegram Bot Screenshot 2

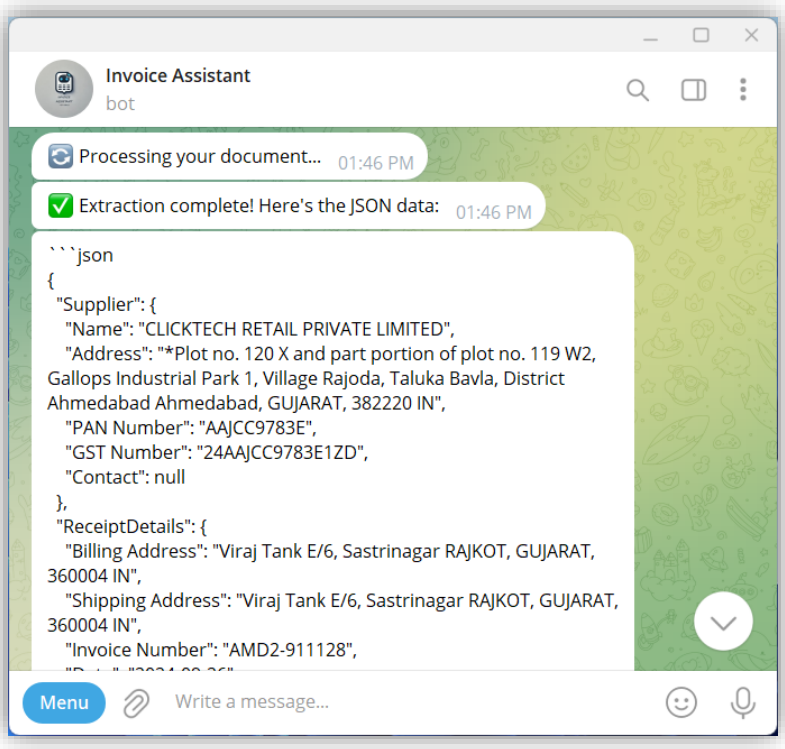


Figure 9-5 Telegram Bot Screenshot 3

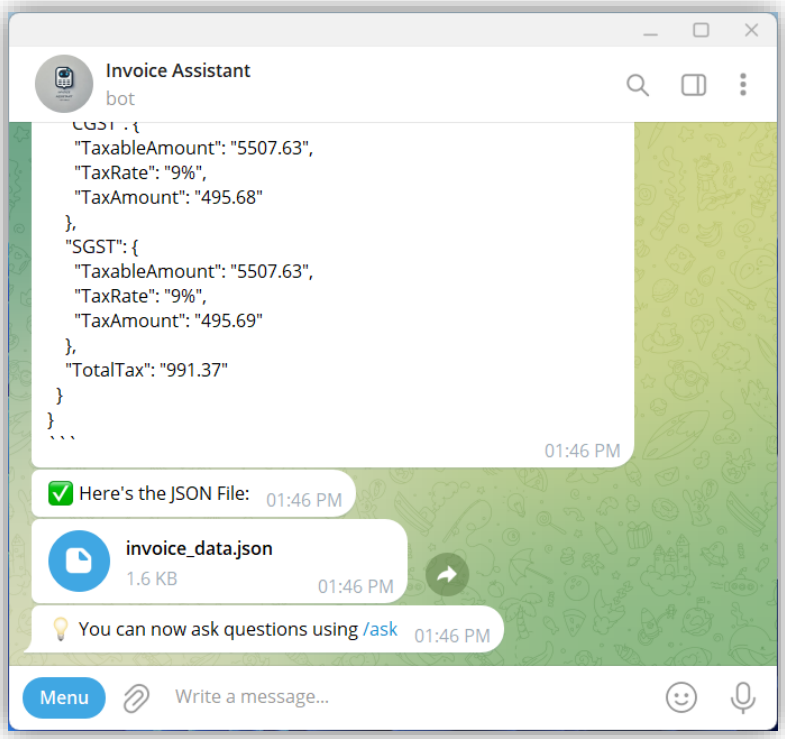


Figure 9-6 Telegram Bot Screenshot 4

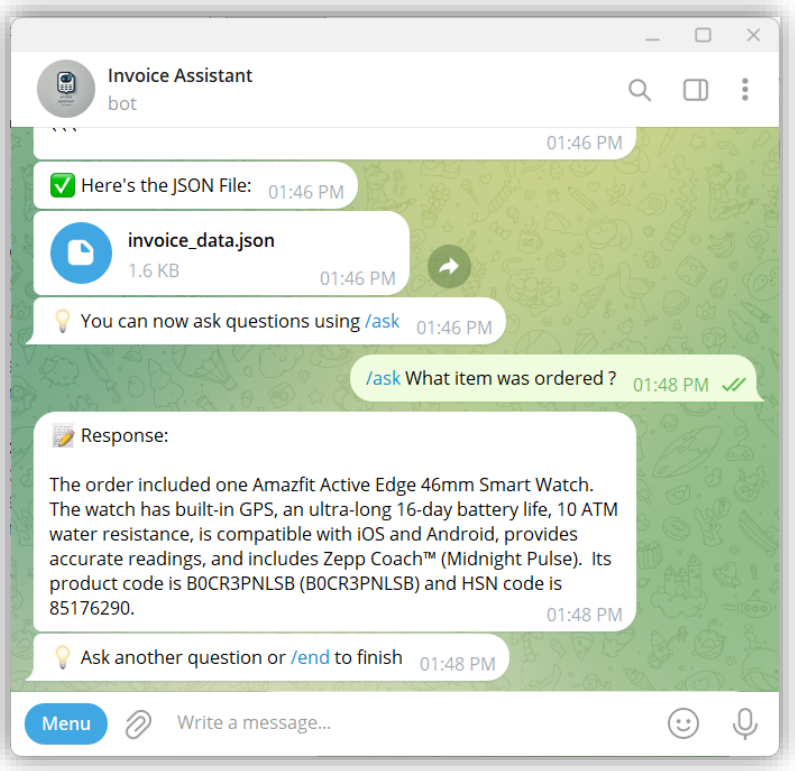


Figure 9-7 Telegram Bot Screenshot 5

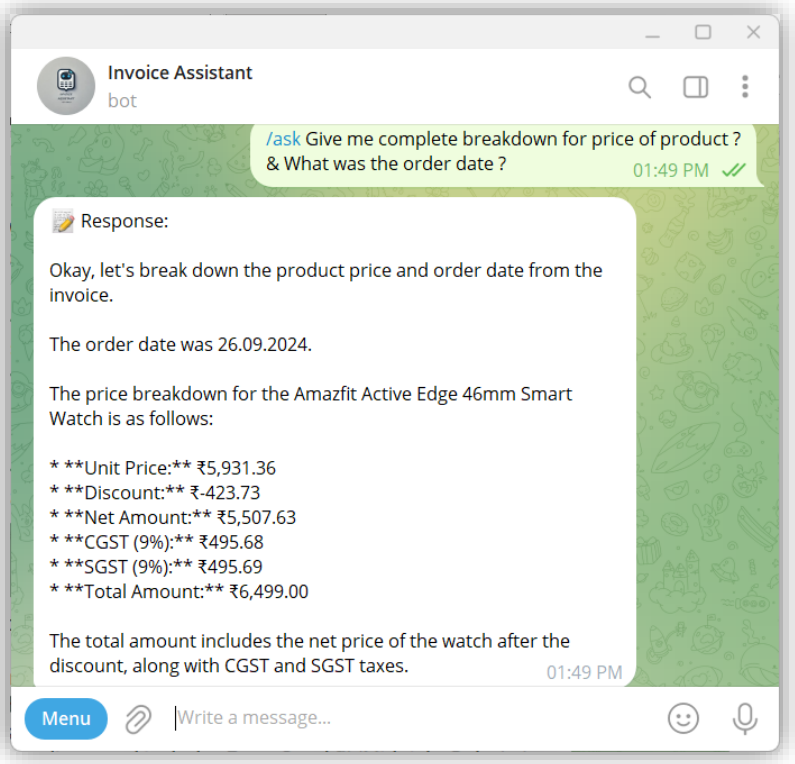


Figure 9-8 Telegram Bot Screenshot 6

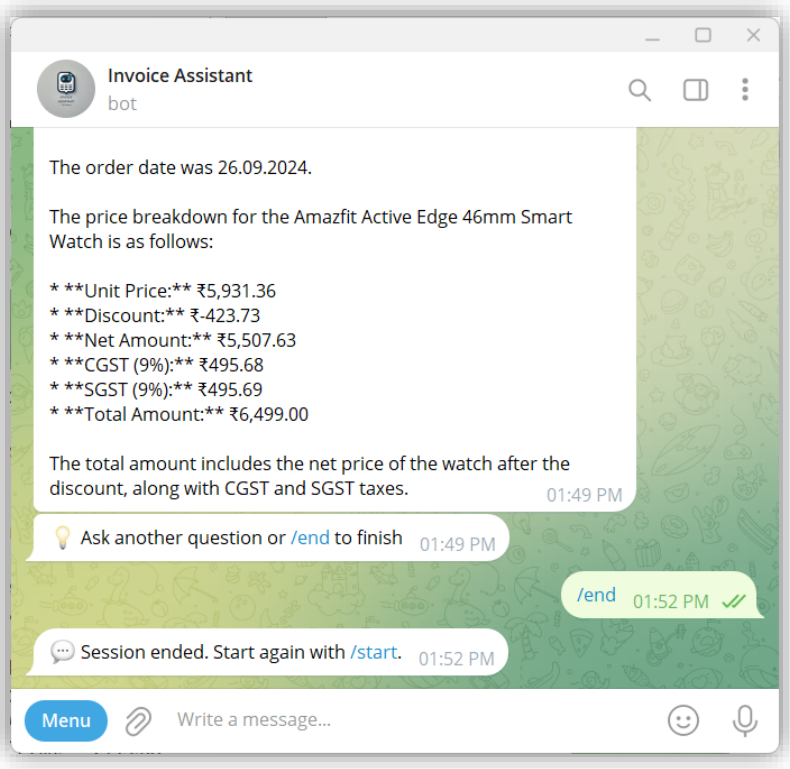


Figure 9-9 Telegram Bot Screenshot 7

Conclusion

The Invoice Assistant makes it easier to extract and analyse invoice data by automatically pulling structured details from PDFs and images. With Google Gemini, the system can process different invoice formats and accurately answer user queries. Features like retry mechanisms and logging help keep the bot running smoothly, even when network issues occur.

It's also important to note that the bot runs only while the code is actively running in VS Code; if the code is terminated, the bot stops responding. For a more robust deployment, platforms like AWS, Google Cloud Platform, Microsoft Azure, or similar services could be used.

That said, there are a few limitations. The accuracy of extraction depends on how clear the uploaded invoice is, blurry or low-quality images can lead to missing details. The free version of Google Gemini also has request limits, which might slow things down when processing multi-page invoices. Additionally, the bot has difficulty handling handwritten invoices or those with unusual layouts, which may require further improvements in OCR technology.

Despite these challenges, the Invoice Assistant is a useful tool for managing invoices efficiently, reducing manual work, and improving overall accuracy.