University of Science and Technology of Southern Philippines

College of Information Technology and Computing
Department of Computer Science

Augmented Reality-Aided Card Counting in BlackJack Using YOLOv5 and WebSocket Communication

A Final Performance Innovative Task presented to the
Department of Computer Science
College of Information Technology and Computing
for Networks and Communication CC313

By:

Cainday, Joeniño

Galceran, Rozel

To:

Gefferson A. Balase

December 16, 2024

1. Project Overview & Objectives

1.1 Background

Augmented Reality (AR):

Augmented Reality is a cutting-edge technology that overlays digital content onto the physical environment, enhancing the user's perception of reality. AR has become a cornerstone in various fields, including gaming, education, and interactive applications, offering users immersive and engaging experiences. Devices such as AR glasses, like the Brilliant Labs Frame Spectacles, integrate AR functionalities seamlessly into wearable hardware, making them ideal for hands-free applications.

Computer Vision:

Computer vision, a subfield of artificial intelligence, focuses on enabling machines to understand and interpret visual data. Through algorithms and deep learning models like YOLOv5, computer vision can detect, identify, and track objects in real time, providing practical solutions in fields like gaming, surveillance, and augmented reality.

YOLOv5:

YOLOv5 (You Only Look Once, Version 5) is a state-of-the-art object detection model renowned for its speed and accuracy. It can process visual data in real time, making it ideal for AR applications like card recognition in gaming. Its lightweight and efficient design make it compatible with resource-constrained devices like AR glasses.

WebSockets*:*

WebSockets enable real-time, bidirectional communication between client and server systems. A full-duplex protocol that maintains a continuous connection between a client and a server. WebSocket is ideal for real-time updates or applications that require continuous data exchange.

WebSocket is compatible with HTTP, but it's not the same thing:

HTTP
A request-response protocol that's used to serve static resources and make API requests. HTTP requires a new connection for each data exchange.

WebSocket is designed to work over HTTP ports 443 and 80, and it supports HTTP proxies and intermediaries. To achieve compatibility, WebSocket uses the HTTP Upgrade header to change from the HTTP protocol to the WebSocket protocol.

Here are some other differences between WebSocket and HTTP:

- Connection type
  WebSocket is stateful, meaning it maintains a persistent connection between the client and server. HTTP is stateless, meaning each request is handled in isolation.

- Latency
  WebSocket has lower latency than HTTP because data is sent as soon as it's available. With HTTP, the client needs to open a new connection if they need more data.

- Security
  WebSocket offers a secure (encrypted) version. The address of the unencrypted version starts with ws:// and the encrypted version with wss://.

Brilliant Labs Frame Spectacles:
The Brilliant Labs Frame Spectacles are smart AR glasses equipped with built-in cameras and sensors, offering developers a platform for creating augmented reality applications. These glasses are designed to process visual input and display digital overlays in real time, making them an excellent choice for innovative projects like integrating computer vision into AR gaming experiences.

1.2 Problem Statement

The project, commissioned by Algo Vision and represented by Chief of Operations Justeen Bondoc, involved creating a BlackJack card-counting system integrated into the Brilliant Labs Frame Spectacles. The system was intended to use augmented reality and computer vision powered by YOLOv5. Despite its innovative concept, the project faced several challenges:

- Remote Programming Limitations: Developing and debugging the system remotely for the AR glasses proved to be excessively time-consuming and inefficient.

- Frequent Scope Changes: Constant adjustments to the project requirements disrupted the workflow and extended development time.

- Compensation Discrepancy: While the initial offer for the project was $50,000, only $5,000 was allocated for creating the computer vision client-server prototype, limiting resources and progress.

- Practical Concerns: Warnings about the legal and ethical implications of using AI-driven systems in casino environments were overlooked, adding complexity to the project's goals.

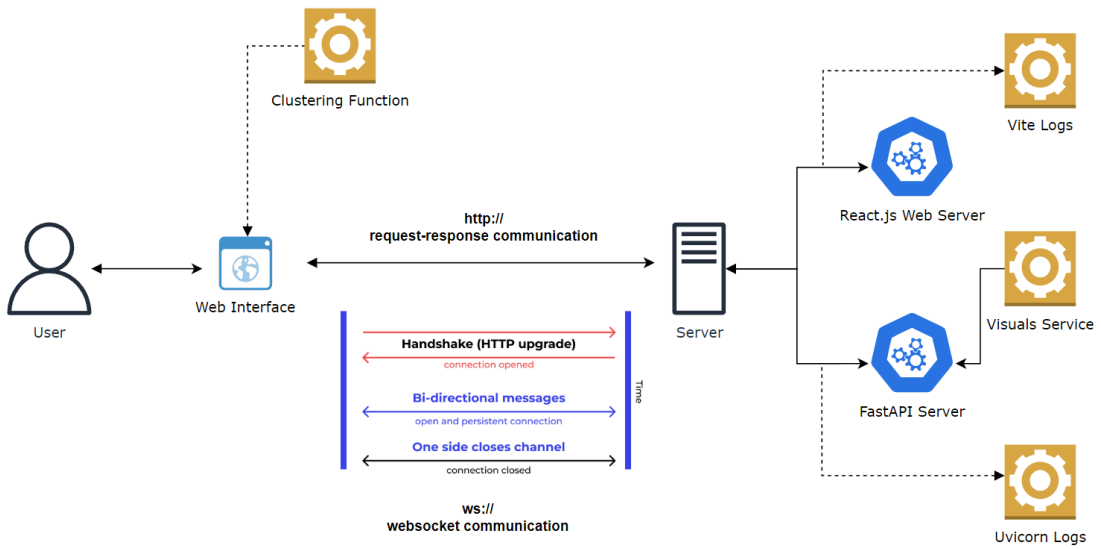
1.3 Objective

The primary objective of this project is to design and implement a prototype system that leverages augmented reality, YOLOv5, and WebSockets for real-time BlackJack card counting on the Brilliant Labs Frame Spectacles. Specifically, the system aims to:

- Utilize YOLOv5 for accurate and fast card detection and recognition.
- Leverage WebSockets to ensure low-latency communication between the AR glasses and the server.
- Integrate these technologies into the Brilliant Labs Frame Spectacles for a seamless AR experience.
- Deliver a proof-of-concept system that meets the stakeholder's expectations within the constraints of time, resources, and remote programming challenges.

2. System Architecture

The system is built on a robust architecture integrating a Client, Server, Protocol Layer, and Network Layer, ensuring efficient real-time communication and processing of augmented reality blackjack card counting using computer vision.



2.1. Components

Client

The Client serves as the user-facing interface. It provides real-time interaction with the system, utilizing the Web Interface built with Vanilla JavaScript and optimized using Vite.

- Technologies:
  - Vite (Version: ^5.2.0) for fast development and production builds.
  - Vanilla JavaScript for building the frontend without additional frameworks.
  - Node.js (Version: 20.18.0) for server-side processing, running in conjunction with Vite.
- Key Functions:
  - Displays the web interface for interacting with the system.
  - Establishes communication with the backend server via WebSockets and HTTP requests.
  - Visualizes processed data, including detected cards, via augmented reality.

Server

The Server component handles all backend processes, including computer vision tasks (card detection), clustering, and communication with the client.

- Technologies:
  - Python (Version: 3.12.5) for backend logic.
  - FastAPI for creating fast and efficient RESTful APIs.
  - Ultralytics YOLOv5 for object detection, particularly for blackjack card recognition.
  - Uvicorn for running the FastAPI application with ASGI support.
- Key Functions:
  - Processes incoming client requests, performing object detection on images or video streams.
  - Handles clustering of detected objects to count blackjack cards.

    - Provides real-time updates back to the client via WebSocket communication.

Protocol Layer

The Protocol Layer ensures smooth communication between the Client and Server.

- WebSocket: Enables real-time, bi-directional communication between the Client and Server, allowing the server to send continuous updates (e.g., detected card information) to the client in real time.
- HTTP: Used for initial setup and request-response interactions, establishing the connection between the client and the backend server before upgrading to WebSocket for continuous communication.

Network Layer

The Network Layer is responsible for the underlying communication infrastructure that connects the Client and Server.

- Protocol: WebSocket (ws://) is used for persistent, low-latency communication, ensuring that the data exchange between the Client and Server happens in real time.
- Technologies:
    - Vite for bundling frontend assets and facilitating smooth client-server interaction.
    - Uvicorn for running the FastAPI server, enabling efficient HTTP and WebSocket connections.
    - FastAPI for handling API requests and WebSocket connections, while ensuring scalability.

2.2. Technologies Used

Language:

- Frontend: Vanilla JavaScript for client-side programming, ensuring lightweight and fast performance.
- Backend: Python 3.12.5, chosen for its compatibility with computer vision libraries and fast API development.

Libraries:

- Frontend:
    - Vite (Version: ^5.2.0): A fast build tool and development server.
    - Density Clustering (Version: ^1.3.0): A clustering algorithm used for organizing the detected objects.
- Backend:
    - FastAPI: A modern framework for building APIs quickly with automatic documentation.
    - Ultralytics YOLOv5: A state-of-the-art object detection model for detecting blackjack cards.
    - Uvicorn: An ASGI server for serving FastAPI applications with asynchronous support.

Protocol:

- HTTP: Used for initial communication (request-response) to set up the connection.
- WebSocket (ws://): Upgraded from HTTP to WebSocket for continuous, bi-directional communication between the Client and Server.

2.3. System Workflow

1. User Interaction:
    - The user accesses the Web Interface built with Vanilla JavaScript and powered by Vite.
2. Frontend Communication:
    - The frontend sends HTTP requests to initialize communication with the FastAPI Server.
    - The HTTP connection is upgraded to WebSocket for persistent, real-time interaction.
3. Backend Processing:
    - The FastAPI Server processes image or video frames to detect blackjack cards using the YOLOv5 model.

- The detected cards are clustered and counted for blackjack.

4. Data Visualization:
    - Processed results (e.g., detected cards) are streamed back to the frontend in real-time, displayed through augmented reality visuals.

3. Code Structure

The project consists of both frontend and backend components, each organized in dedicated directories. The frontend handles user interactions through a client-side application, while the backend manages server-side logic, including the machine learning model and computer vision tasks.

```
Folder PATH listing for volume Windows-SSD
Volume serial number is DECC-298C
C:
📂 Project Root
├── .dockerignore          # Specifies files and directories to ignore when building a Docker image
├── .gitignore             # Specifies files and directories to exclude from Git version control
├── structure.txt          # Describes the file structure of the project
├── README.md           # Documentation and setup instructions for the backend

📂 client                   # Frontend client application
│   ├── index.html         # Main HTML file for the client-side user interface
│   ├── package.json       # Defines dependencies and scripts for the client application
│   ├── pnpm-lock.yaml     # Lockfile for managing exact dependency versions with pnpm
│   ├── script.js          # Client-side JavaScript logic
│   └── 📂 node_modules    # Dependencies installed by pnpm (not included in version control)

📂 server                   # Backend server application
│   ├── Dockerfile         # Docker configuration for containerizing the server
│   ├── main.py            # Entry point of the backend server application
│   ├── requirements.txt    # Python dependencies for the server

│   📂 model               # Machine learning model directory
│   │   ├── train.py       # Script for training the machine learning model
│   │   └── 📂 detect      # Directory for model detection results and configurations
│   │       └── blackjack-model-final
│   │           ├── args.yaml         # Configuration and parameters for the detection model
│   │           ├── results.csv       # Results from the model's detection tests
│   │           └── 📂 weights        # Directory containing the model weights
│   │               ├── best.pt       # Best-performing model weights
│   │               └── last.pt       # Weights from the most recent training

│   📂 service             # Server services and logic
│   │   └── vision.py      # Vision service for handling computer vision tasks

│   📂 test                # Folder containing the test scripts
│   │   └── vision.test.py  # Testing the core functions of the api
│   │   └── main.test.py    # Testing the http and websockets communications of fastapi

│   📂 venv                # Python virtual environment for managing dependencies (not included in version control)
```

3.1 Root Directory

- `.dockerignore`: Specifies which files and directories should be ignored when building the Docker image. It is similar to `.gitignore` but for Docker builds.
- `.gitignore`: Specifies files and directories to exclude from version control in Git. Typically includes dependencies, configuration files, and local environment settings.
- `structure.txt`: A document that likely describes the file and folder structure of the project for better understanding and maintenance.
- `README.md`: Provides documentation and setup instructions for the backend application. This could include steps for running the server, installing dependencies, and using the API.

3.2 Frontend (Client)

The client directory contains all the necessary files for the frontend application.

- `index.html`: The main HTML file for the client-side user interface. It is the entry point for the frontend, and other JavaScript and CSS files are linked from here.
- `package.json`: Defines the dependencies and scripts required for the client-side application. It is used by the package manager (e.g., pnpm) to install and manage dependencies.
- `pnpm-lock.yaml`: A lockfile used by pnpm to ensure that the exact same versions of dependencies are installed across different environments.
- `script.js`: Contains the main JavaScript logic for the frontend, handling interactions with the user interface and possibly communicating with the backend.
- `node_modules/`: This folder contains all the dependencies installed by pnpm. It is not included in version control because it can be regenerated from the `package.json` file.

3.3 Backend (Server)

The server directory contains the backend application logic, including server configuration, machine learning models, and services.

- `Dockerfile`: A configuration file used to build the Docker image for the backend server. It contains instructions on how to set up the server environment, install dependencies, and run the application.
- `main.py`: The entry point of the backend server application. This file likely contains the configuration for the FastAPI (or other web framework) server, defining endpoints and handling incoming requests.
- `requirements.txt`: A list of Python dependencies required by the backend server. These are installed using pip to set up the Python environment.

3.4 Machine Learning Model

The model directory contains everything related to the machine learning model used for computer vision tasks, such as object detection (in this case, likely to detect blackjack cards).

- `train.py`: A script used to train the machine learning model. This may include preprocessing data, training the model, and saving the final model weights.
- `detect/`: A subdirectory containing files and results related to model detection.
    - `blackjack-model-final/`: The final directory for the trained blackjack card detection model.
        - `args.yaml`: A configuration file containing parameters for the detection model, such as input dimensions, hyperparameters, etc.
        - `results.csv`: The results of the model's detection tests, possibly including accuracy, precision, recall, etc.
        - `weights/`: Contains the trained weights of the model.
            - `best.pt`: The best-performing weights from training, likely the model that achieved the highest performance during evaluation.
            - `last.pt`: The most recent model weights, potentially from the latest training iteration.

3.5 Server Services and Virtual Environment

- `service/vision.py`: A Python file containing the logic for handling computer vision tasks. This likely includes functions to process images, run the detection model, and interpret results.
- `venv/`: A virtual environment that isolates the Python dependencies for the backend server. This ensures that the project dependencies don't conflict with other projects or global Python installations. This directory is not included in version control.

4. Client-Side Implementation

The client-side application is a web-based augmented reality interface designed to assist in card counting during a Blackjack game. Developed using modern web technologies, the application leverages WebSocket communication, real-time video processing, and strategic decision support. It provides a robust, interactive interface for augmented reality-assisted card counting, combining real-time video processing, strategic analysis, and intuitive visualization.

4.1. Camera Initialization and Device Management

The client application manages video input through several critical functions:

Device Enumeration

- `getConnectedDevices()`: Identifies available video input devices
- Populates a dropdown menu allowing users to select different cameras
- Provides flexibility in camera selection

Stream Initialization

- `startStream()`: Establishes video stream with selected camera
- Configures video constraints matching display dimensions
- Manages stream lifecycle by stopping previous streams before initiating new ones

4.2. Frame Capture and WebSocket Communication

The client facilitates real-time image processing through WebSocket communication:

Frame Extraction

- `getFrame()`: Converts live video stream into base64-encoded PNG images
- Creates temporary canvas element to capture current video frame
- Enables seamless transmission of video data to server-side processing

WebSocket Connectivity

- Establishes persistent WebSocket connection to server at `ws://localhost:8000/ws`
- Sends video frames continuously
- Processes detection results returned by server

4.3. Detection Visualization and Interpretation

Upon receiving detection data, the client performs complex visual and analytical tasks:

Bounding Box Rendering

- Dynamically generates HTML div elements representing detected card locations
- Applies styled overlays with detection metadata (card name, confidence score)
- Provides visual feedback on detected cards' positions

Card Tracking and Scoring

- Maintains running count of cards using predefined Omega Count strategy
- Updates grid display to track remaining deck composition
- Calculates cumulative running count based on detected cards

4.4. Strategic Decision Support

The `getStrategy()` function implements comprehensive Blackjack decision-making logic:

Strategy Determination

- Analyzes player's hand composition
- Considers dealer's face-up card
- Recommends actions based on established Blackjack basic strategy
- Handles various hand scenarios:
    - Pair splitting
    - Soft hands
    - Hard hands
    - Special situations like doubling down

4.5. Clustering and Game State Inference

Utilizes density-based spatial clustering to interpret game state:

Cluster Analysis

- Uses DBSCAN clustering algorithm to group detected cards
- Distinguishes between dealer and player card clusters
- Identifies dealer's face-up card
- Calculates hand totals
- Determines game outcomes (win/lose/tie/blackjack)

Technical Components

- Frontend Framework: Vanilla JavaScript
- WebSocket Library: Native WebSocket API
- Clustering Library: Density-clustering
- Video Capture: Web Media Devices API

Communication Protocol

1. Client captures video frame
2. Sends frame to server via WebSocket
3. Receives detection results

4. Processes and visualizes results
5. Repeats process continuously

Error Handling and Device Compatibility

- Graceful handling of camera access permissions
- Fallback mechanisms for device selection
- Compatible with modern web browsers supporting WebSocket and Media Devices APIs

5. Server-Side Implementation for Image Detection and Websocket Communication

The server-side of this application is built using Python, specifically leveraging the FastAPI framework for handling HTTP and WebSocket requests, and the Ultralytics YOLO (You Only Look Once) model for image detection. The server listens for incoming client connections, processes image data, and returns the detection results.

This server-side setup enables real-time communication and processing of images for the purpose of detecting and tracking playing cards in a dynamic environment. The server is scalable and efficient, handling multiple clients concurrently, thanks to FastAPI's asynchronous capabilities.

Key Server Functions:

5.1 Dependencies:

- FastAPI: A modern, fast (high-performance) web framework for building APIs with Python based on standard Python type hints.
- Uvicorn: An ASGI server for Python, required to run FastAPI applications.
- Ultralytics YOLO: A pre-trained YOLOv8 model used to detect objects, in this case, playing cards.
- OpenCV: Used for handling image conversion between base64 format and OpenCV images.

5.2. Installation: To set up the server-side environment, install the required dependencies using the following commands:

```
pip install uvicorn[standard] ultralytics fastapi opencv-python
```

The `ultralytics` package contains the necessary functions to interact with the YOLO model for object detection.

5.3 Server Initialization:

The FastAPI server is initialized with the following command:

```
uvicorn main:app --reload
```

1. This starts the server with auto-reloading enabled, making it convenient during development. The `main:app` indicates that the `FastAPI` instance `app` is located in the `main.py` file.

WebSocket Communication for Image Processing:

A WebSocket endpoint is used for real-time, bidirectional communication between the client and the server. This allows for sending and receiving image data continuously.

WebSocket Endpoint: In `main.py`, a WebSocket route is defined

```
@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket):
    await websocket.accept()
    while True:
        img = await websocket.receive_text()
        await websocket.send_text(detect(img))
```

1.
    - `/ws`: This is the WebSocket route that clients will connect to for sending images and receiving processed data.
    - `receive_text()`: This function waits for the client to send base64-encoded image data (in text format).

    - `send_text()`: After processing the image, the server sends back the detection results as a JSON-encoded list of card detections.
    - `detect(img)`: This function performs the image detection, using the `YOLO` model from the custom service.vision module.
2. Image Detection: The server receives a base64-encoded image string, decodes it into an OpenCV image, and processes it using the YOLO model to detect playing cards. The `detect` function handles this as follows:
    - from_b64(uri): Decodes the base64 string into an OpenCV image.
    - YOLO model inference: The YOLO model processes the image and returns bounding boxes, class names, and confidence scores.
    - cards list: The function creates a list of detected cards with their positions, class names, and confidence scores.
3. Returning Results: The detection results are returned in JSON format, which includes the following information:
    - Bounding box coordinates (left, top, right, bottom).
    - Class name of the detected card (e.g., `ace`, `jack`).
    - Confidence score: A value representing the accuracy of the detection.
    - Object ID: Each detected card is assigned a unique ID, used to track the card across frames.

Image Conversion Utility Functions:

- from_b64(uri): Converts the base64-encoded string into a NumPy array that OpenCV can use for further image processing.
- to_b64(img): Converts an OpenCV image back into a base64-encoded string, ready for transmission.

Server Flow:

1. Client Connection: The client establishes a WebSocket connection to the server at the `/ws` route.
2. Image Transmission: The client sends base64-encoded image data over the WebSocket connection.

3. Image Processing: The server decodes the image, passes it to the YOLO model, and performs object detection.
4. Sending Results: After processing, the server sends the detection results (coordinates, class names, confidence) back to the client in a JSON format.
5. Real-time Interaction: The server continues to receive and process new images, sending back updated detection results as long as the WebSocket connection is open.

Backend Technologies and Tools:

1. Python 3.12.5: The server is implemented using Python 3.12.5, which provides support for the latest features in the language and libraries.
2. Uvicorn: A high-performance ASGI server used to run FastAPI applications.
3. FastAPI: A Python web framework for creating APIs with asynchronous capabilities, enabling efficient handling of WebSocket connections.
4. Ultralytics YOLO: A state-of-the-art object detection model used to detect playing cards from the image data.
5. OpenCV: A powerful library used to handle image manipulation, including decoding base64 images and performing object detection.

annotated-types==0.7.0
anyio==4.7.0
certifi==2024.12.14
charset-normalizer==3.4.0
click==8.1.7
colorama==0.4.6
contourpy==1.3.1
cycler==0.12.1
fastapi==0.115.6
filelock==3.16.1
fonttools==4.55.3
fsspec==2024.10.0
h11==0.14.0
httptools==0.6.4
idna==3.10
Jinja2==3.1.4
kiwisolver==1.4.7
lap==0.5.12
MarkupSafe==3.0.2
matplotlib==3.10.0
mpmath==1.3.0
networkx==3.4.2
numpy==2.2.0
opencv-python==4.10.0.84
packaging==24.2
pandas==2.2.3
pillow==11.0.0
psutil==6.1.0
py-cpuinfo==9.0.0
pydantic==2.10.3
pydantic_core==2.27.1
pyparsing==3.2.0
python-dateutil==2.9.0.post0
python-dotenv==1.0.1
pytz==2024.2
PyYAML==6.0.2
requests==2.32.3
scipy==1.14.1
seaborn==0.13.2
setuptools==75.6.0
six==1.17.0
sniffio==1.3.1
starlette==0.41.3
sympy==1.13.1
torch==2.5.1
torchvision==0.20.1
tqdm==4.67.1
typing_extensions==4.12.2
tzdata==2024.2
ultralytics==8.3.49

ultralytics-thop==2.0.13	uvicorn==0.34.0	websockets==14.1
urllib3==2.2.3	watchfiles==1.0.3	

6. Protocol Design

This section documents the custom protocol used for communication between the client - a frontend application and the server - FastAPI backend with WebSocket support. The communication involves sending base64-encoded images to the server for processing using YOLO-based card detection and receiving the results back in JSON format.

This protocol ensures clear and structured communication between the client and server, using WebSockets for continuous interaction. It includes robust error handling and ensures that both images and results are efficiently encoded and decoded using base64.

Protocol Flow

1. Handshake Process
    - When the client accesses the website, it establishes a WebSocket connection with the server.
    - The client sends an initial connection request to the WebSocket endpoint (`/ws`), and the server responds with an acceptance message, allowing the client to send data.
2. Client Sends Image for Detection
    - The client captures an image (e.g., via a camera, file input, or canvas).

The image is converted to a base64-encoded string and sent to the server through the WebSocket connection. The client sends this image as a string payload:

```
{
  "type": "image",
  "data": "..."
}
```

    - The server processes this image, applies card detection via YOLO, and returns the result in JSON format.

3. Server Processes the Image

The server decodes the base64-encoded image, runs card detection using the YOLO model, and compiles the results. Each detected card is sent with the coordinates of the bounding box, the class of the detected object (card name), the confidence score, and a unique ID for tracking:

```
{
  "type": "detection_result",
  "data": [
    {
      "left": 100,
      "top": 200,
      "right": 150,
      "bottom": 250,
      "card": "Ace of Spades",
      "confidence": 0.98,
      "id": 1
    },
    {
      "left": 300,
      "top": 400,
      "right": 350,
      "bottom": 450,
      "card": "King of Hearts",
      "confidence": 0.95,
      "id": 2
    }
  ]
}
```

- The `data` field contains an array of detected card details with the following attributes:
  - `left`, `top`, `right`, `bottom`: Coordinates of the bounding box around the detected card.
  - `card`: The name of the detected card.
  - `confidence`: The confidence score of the detection (between 0 and 1).

- `id`: A unique identifier for the tracked card.

4. Error Handling

If an error occurs during the image processing (e.g., if the image is not valid or the YOLO model encounters an issue), the server sends an error message back to the client:

```
{
  "type": "error",
  "message": "Invalid image format or detection error."
}
```

   - The client should handle these errors by displaying appropriate messages or triggering retries. The error messages include the type of error to help diagnose the issue (e.g., invalid base64 data, detection failure).

5. Closing the WebSocket Connection

   - After receiving the detection results or an error message, the WebSocket connection remains open for further communication. The client can send additional images for detection as needed.
   - If the client chooses to close the WebSocket connection, it can send a close signal, and the server will gracefully disconnect.

Example Protocol Flow

1. Client Establishes WebSocket Connection:

   - The client sends a WebSocket connection request to `/ws`.
   - The server responds with acceptance, and the connection is established.

2. Client Sends Image for Detection:

The client sends an image in base64 format to the server via WebSocket:

```
{
  "type": "image",
```

```
  "data": "..."
}
```

3. Server Processes the Image and Responds:

The server processes the image and sends back the detection results:

```
{
  "type": "detection_result",
  "data": [
    {
      "left": 100,
      "top": 200,
      "right": 150,
      "bottom": 250,
      "card": "Ace of Spades",
      "confidence": 0.98,
      "id": 1
    }
  ]
}
```

4. Error Handling:

If the image is invalid or there is an issue with the detection, the server sends:

```
{
  "type": "error",
  "message": "Invalid image format or detection error."
}
```

7. Error Handling and Logging

The implemented logging and error handling strategy provides comprehensive monitoring, rapid issue detection, and robust system resilience. By leveraging Vite and Uvicorn's native logging capabilities, the Augmented Reality Card Counting system maintains high reliability and provides actionable insights into its operational performance.

7.1 Vite Development Environment Logging

Configuration

Vite provides built-in logging mechanisms that offer comprehensive insights into the application's development and build processes. The configuration is defined in the `vite.config.js` file:

```
export default {
    build: {
      rollupOptions: {
        onLog(level, log, handler) {
          // Custom logging strategy for build process
          if (level === 'error') {
            console.error('Build Error:', log.message);
          }
        }
      }
    },
    server: {
      // Enhanced error reporting for development server
      strictPort: true,
      hmr: {
        overlay: true  // Display runtime errors directly in browser
      }
    }
  }
```

Key Logging Features

- Real-time error overlay during development
- Detailed build process error reporting
- Hot Module Replacement (HMR) error tracking

7.2 Uvicorn Server Logging

Logging Configuration

Uvicorn offers robust logging capabilities for server-side operations:

```
import uvicorn
import logging

logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s: %(message)s',
    handlers=[
        logging.FileHandler('server.log'),
        logging.StreamHandler()
    ]
)

def start_server():
    uvicorn.run(
        "main:app",
        host="0.0.0.0",
        port=8000,
        reload=True,
        log_level="info",
        log_config={
```

```
            "version": 1,
            "formatters": {
                "default": {
                    "format": "%(asctime)s - %(levelname)s: %(message)s"
                }
            }
        }
```

Logging Strategies

- Dual logging to file and console
- Configurable log levels
- Timestamped log entries
- Detailed server startup and runtime information

7.3 WebSocket Error Handling

```
@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket):
    try:
        await websocket.accept()
        while True:
            try:
                img = await websocket.receive_text()
                detection_results = detect(img)
                await websocket.send_text(detection_results)
            except Exception as process_error:
                logging.error(f"WebSocket processing error: {process_error}")
                await websocket.send_json({
                    "status": "error",
                    "message": str(process_error)
                })
    except Exception as connection_error:
        logging.critical(f"WebSocket connection error: {connection_error}")
```

7.4 Error Classification

Network Errors

- Connection timeouts
- Disconnection events
- Transmission failures

Processing Errors

- Image decoding issues
- YOLO detection failures
- Data validation errors

7.5 Logging Best Practices

1. Capture Contextual Information

    - Timestamp of events
    - Detailed error messages
    - System state at error occurrence
2. Implement Granular Logging Levels
    - DEBUG: Detailed diagnostic information
    - INFO: General system operations
    - WARNING: Potential issues
    - ERROR: Significant failures
    - CRITICAL: System-threatening events

8. Documentation

This documentation provides an overview of the system and detailed instructions for setting up, running, and testing both the client and server components of the augmented reality Blackjack card counting system. Additionally, it offers guidelines on configuring the application, troubleshooting common issues, and running unit tests to ensure the system works as expected.

Overview

The system enables real-time Blackjack card counting using augmented reality (AR) glasses integrated with a computer vision model (YOLOv5). It consists of two main components: the Client (Web Interface) and the Server (Backend), both communicating via WebSocket for low-latency, bidirectional communication. The server processes video frames to detect cards, and the client visualizes detected cards through AR.

Requirements

Client-Side:

- Vite (Version: ^5.2.0): Build tool and development server
- Vanilla JavaScript: For client-side code
- Node.js (Version: 20.18.0): Required for running the frontend server

Server-Side:

- Python (Version: 3.12.5)
- FastAPI: Web framework for building the backend API
- Uvicorn: ASGI server to run the FastAPI application
- Ultralytics YOLOv5: Object detection model for detecting Blackjack cards
- OpenCV: For image processing (converting between base64 and OpenCV formats)
- Density Clustering (Version: ^1.3.0): Clustering algorithm used to organize detected objects

Running the Server

To set up and run the server:

Install dependencies:

```
pip install uvicorn[standard] ultralytics fastapi opencv-python
```

1. Start the server: Run the following command to launch the FastAPI server: `uvicorn main:app --reload`
2. This command will start the server and enable auto-reloading for development purposes. The server will be accessible at `http://localhost:8000/` and will use WebSockets for communication at `ws://localhost:8000/ws`.

Running the Client

Clone the repository:

```
git clone https://github.com/mihkuno/BLACKJACK.git
cd BLACKJACK
```

1. Install Node.js dependencies: Install the necessary Node.js packages using npm:
   `npm install`
2. Start the client: Run the following command to start the client:
   `npm run dev`
3. The client will be accessible at `http://localhost:3000/` and will initiate a WebSocket connection to the server at `ws://localhost:8000/ws`.

Troubleshooting

Common Issues:

1. WebSocket connection fails:
   - Ensure the server is running and accessible at `ws://localhost:8000/ws`.
   - Check firewall and network settings to ensure WebSocket communication is not blocked.
2. Video frame not captured:
   - Ensure the camera is properly connected and accessible.
   - Check browser permissions for camera access.
3. Card detection issues (e.g., no cards detected):
   - Ensure the camera feed is clear and cards are well-lit.
   - Check the YOLOv5 model's accuracy—retraining the model may improve results for specific card types.
4. Server crash or high latency:
   - Monitor server logs for errors. Ensure there are sufficient system resources to handle image processing.

Testing

1. Unit Testing for `service.visual.py` Functions

Functions to Test:

- `from_b64`: Converts a base64 image string to an OpenCV image.

- `to_b64`: Converts an OpenCV image to a base64 string.
- `detect`: Detects cards using YOLO in a base64 image.

Tests to Run:

- Test `from_b64`: Verifies it correctly converts base64 to an image.
- Test `to_b64`: Ensures the function converts an image to a valid base64 string.
- Test `detect`: Mocks YOLO detection and verifies the card details.
- Test invalid image: Checks that invalid base64 data results in an empty detection response.

How to Run the Tests:

1. Open the terminal and install the necessary testing dependencies:
   `pip install pytest unittest`
2. Run the unit tests for `service.visual.py` using
   `python -m unittest visual.test.py`

2. Testing the FastAPI App (`main.py`)

Key Components to Test:

- HTTP endpoint: `/test`
- WebSocket endpoint: `/ws` for image detection

Tests to Run:

- Test `/test` endpoint: Verifies it returns the correct test message.
- Test WebSocket `/ws` endpoint: Simulates sending a base64 image and verifies the response.
- Test WebSocket with invalid data: Checks how the WebSocket handles invalid base64 input.

How to Run the Tests:

Install dependencies for FastAPI testing:
`pip install pytest httpx pytest-asyncio`

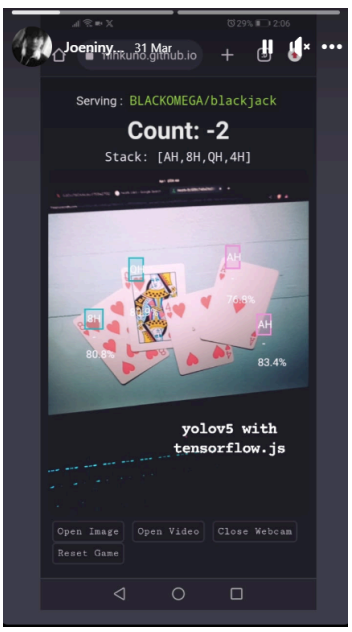
1. Run the tests for the FastAPI app using:
   `pytest`
2. This simplified guide walks you through the essential steps to test your `service.visual.py` functions and the FastAPI app's endpoints using `unittest` and `pytest`. Just run the relevant commands and ensure that the expected results are returned.

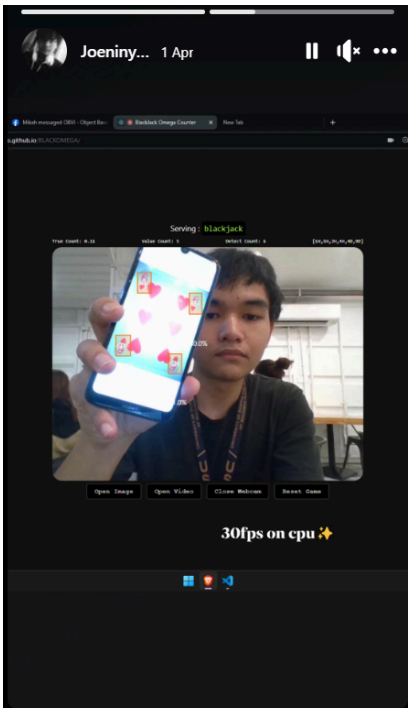
Repository:
https://github.com/mihkuno/BLACKJACK

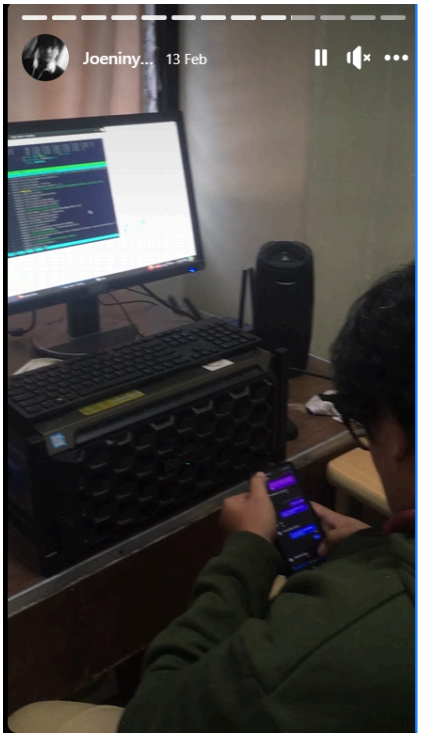
Videos:
https://www.facebook.com/share/v/1Bwt2gxn2p/

https://www.facebook.com/share/v/1PYwSfpzag/

Images:







9. Testing

To create a unit test for the **service.visual.py** code, we will utilize Python's unittest framework. We need to focus on testing each function, particularly the

from_b64, to_b64, and detect functions. Using unit tests like these, we can ensure that the core functions of your card detection system are working as expected and handle errors gracefully.

These unit tests ensure that the FastAPI app handles both the HTTP and WebSocket endpoints correctly and that the detection logic is functioning as expected. The tests also handle the case where invalid data is passed, ensuring that your WebSocket logic can handle errors gracefully.

Breakdown of the Tests:

1. test_from_b64:
    - Verifies that the from_b64 function correctly converts a base64 string to an OpenCV image (a NumPy array).
    - Checks the number of channels in the resulting image to ensure it's a color image (RGB).
2. test_to_b64:
    - Tests that the to_b64 function correctly converts an OpenCV image into a base64 string, with the expected prefix ().
    - Ensures that the base64 string is non-empty.
3. test_detect:
    - Mocks the YOLO.track method to return a simulated detection result.
    - Passes a base64-encoded image string to the detect function and checks that the response contains the expected card information, including the correct card name and confidence.
4. test_invalid_image:
    - Tests how the detect function handles invalid base64 image data (e.g., incorrect or corrupted data).
    - Ensures that the function returns an empty list if no valid detection occurs.

How to Run the Tests:

Step 1: Save the code in a Python file, visual.test.py.

```
import unittest
import base64
import json
```

```python
import cv2
from unittest.mock import patch

# Assuming your code is saved in a file named `card_detection.py`
from server.service.vision import from_b64, to_b64, detect

class TestCardDetection(unittest.TestCase):

    def setUp(self):
        # This method will be executed before each test
        self.img_b64 = self.encode_image_to_base64("test_image.jpg")
        self.sample_image = cv2.imread("test_image.jpg")  # Load a sample image for testing
        self.mock_card_model = patch('card_detection.YOLO').start()  # Mock YOLO model
        self.mock_card_model.names = {0: "card1", 1: "card2"}  # Mock card names

    def tearDown(self):
        # This method will be executed after each test
        patch.stopall()

    def encode_image_to_base64(self, image_path):
        """
        Helper function to encode an image file to base64.
        """
        with open(image_path, "rb") as img_file:
            return base64.b64encode(img_file.read()).decode('utf-8')

    def test_from_b64(self):
        """Test the from_b64 function"""
        img = from_b64(f"{self.img_b64}")
        self.assertIsInstance(img, np.ndarray)  # Check if the result is an OpenCV image (numpy array)
        self.assertEqual(img.shape[2], 3)  # Ensure it's a color image (3 channels)

    def test_to_b64(self):
        """Test the to_b64 function"""
        b64_img = to_b64(self.sample_image)
        self.assertTrue(b64_img.startswith(''))  # Check if the result starts with correct prefix
        self.assertTrue(len(b64_img.split(',')[1]) > 0)  # Check if the base64 string is non-empty

    @patch('card_detection.YOLO.track')  # Mocking the track method of the YOLO model
    def test_detect(self, mock_track):
        """Test the detect function"""
        # Setup a mock response for the detection
        mock_track.return_value = [
            # Simulating the result from a YOLO model with detection boxes
            type('Result', (), {
                'boxes': [type('Box', (), {
                    'id': 1,
                    'xyxy': [[50, 50, 150, 150]],
                    'cls': 0,
                    'conf': 0.8
                })]
            })
        ]

        # Simulate a base64 image passed to detect()
        response = detect(f"{self.img_b64}")

        # Assert the returned JSON format is as expected
        cards = json.loads(response)
        self.assertEqual(len(cards), 1)
        self.assertEqual(cards[0][4], 'card1')  # The detected card class
        self.assertEqual(cards[0][5], 0.8)  # The confidence score

    def test_invalid_image(self):
        """Test the detect function with invalid image input"""
        response = detect("_data")
        self.assertEqual(response, '[]')  # If no valid detection occurs, an empty list is returned

if __name__ == '__main__':
    unittest.main()
```

Step 2: In the terminal, navigate to the directory where the test file is located.

Run the tests using the following command:

`python -m unittest visual.test.py`

Step 3: This will execute all the test cases, and you'll see the results in the terminal.

To create unit tests for the **main.py** FastAPI app that you have shared, we will utilize `pytest` and `httpx` (for HTTP requests) and `starlette.testclient` (for WebSocket testing). The goal is to test the HTTP endpoint (`/test`) and the WebSocket functionality (`/ws`), as well as to ensure that the `detect` and `from_b64` functions are working correctly when used in the FastAPI app.

Here's how to write the unit tests for this FastAPI app:

Step 1: Install Dependencies

Require the following dependencies for the unit tests:

```
pip install pytest httpx pytest-asyncio
```

Step 2: Create the Unit Test File

```python
import pytest
from fastapi.testclient import TestClient
from fastapi import FastAPI
from unittest.mock import patch
from server.service.vision import detect

# Import the app from your FastAPI code (assuming the app is in a file named `main.py`)
from main import app

@pytest.fixture
def client():
    """Create a test client for the FastAPI app."""
    return TestClient(app)

@pytest.fixture
def mock_detect():
    """Mock the detect function to avoid actual detection during testing."""
    with patch("service.vision.detect") as mock:
        yield mock

def test_test_endpoint(client):
    """Test the /test endpoint."""
    response = client.get("/test")
    assert response.status_code == 200
    assert response.json() == {"message": "Test is working!"}

def test_websocket_endpoint(client, mock_detect):
    """Test the /ws WebSocket endpoint."""
    mock_detect.return_value = '{"message": "Detection successful"}'

    with client.websocket_connect("/ws") as websocket:
        # Send a base64 image string (simulated for testing)
        websocket.send_text("_image_data")

        # Receive the detection response from the WebSocket
        response = websocket.receive_text()
```

```
        assert response == '{"message": "Detection successful"}'

        # Ensure the detect function was called with the correct arguments
        mock_detect.assert_called_once_with("_image_data")

def test_invalid_image_websocket(client, mock_detect):
    """Test the /ws WebSocket endpoint with invalid image input."""
    mock_detect.return_value = '[]'

    with client.websocket_connect("/ws") as websocket:
        # Send an invalid base64 image string
        websocket.send_text("_data")

        # Receive the detection response from the WebSocket
        response = websocket.receive_text()
        assert response == '[]'  # Should return an empty list for invalid input
```

Explanation of Tests:

1. `test_test_endpoint`:
    - This test checks the `/test` endpoint to ensure that it returns the correct response. It expects a JSON object with a message: `{"message": "Test is working!"}`.
2. `test_websocket_endpoint`:
    - This test mocks the `detect` function to simulate the behavior of the WebSocket endpoint (`/ws`).
    - It simulates sending a base64-encoded image string to the WebSocket and checks the response to ensure that the mocked detection result is returned.
    - The `mock_detect.assert_called_once_with()` statement checks that the `detect` function was called once with the correct arguments.
3. `test_invalid_image_websocket`:
    - This test checks how the WebSocket endpoint handles an invalid base64 image string.
    - It mocks the `detect` function to return an empty list (`[]`) when an invalid image is received.
    - The test checks that the WebSocket returns an empty list when invalid data is provided.

Step 3: Run the Tests

To run the tests, use the following command:

```
pytest
```

This will run all the tests and show the results in the terminal.

Mocking the `detect` Function:

In the tests, we use `unittest.mock.patch` to mock the `detect` function. This ensures that we don't actually perform any card detection during the unit tests. Instead, we simulate the detection function's behavior by returning predefined values.

10. Appendices

Front-End

Index.html

```
<html>
<head>
    <title>Streamer</title>

    <style>
        body {
            margin: 0;
        }

        #left-info {
            position: absolute;
            top: 200;
            left: 50;
            color: white;
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            background-color: #1d1d1d1d;
            padding: 50;
        }
        .grid-container {
            width: 400px;
            display: flex;
            flex-wrap: wrap;
            position: absolute;
            right: 20;
            top: 50;
        }
        .grid-item-remaining {
            width: 25px;
            margin: 2px;
            box-sizing: border-box;
            border: 1px solid #ccc;
            font-family: Arial, Helvetica, sans-serif;
            background-color: rgba(255, 255, 255, 0.5);
            padding: 1px;
            text-align: center;
            font-weight: bold;
            font-size: 12px;
        }
        .grid-item-tracked {
            width: 25px;
            margin: 2px;
            box-sizing: border-box;
            border: 1px solid rgba(70, 70, 70, 1);
            font-family: Arial, Helvetica, sans-serif;
            background-color: rgba(70, 70, 70, 0.5);
            padding: 1px;
            text-align: center;
            font-weight: bold;
            font-size: 12px;
        }
        #grid-remain-text {
            position: absolute;
            font-size: 20;
            right: 35;
            top: 20;
            color: white;
            font-weight: bold;
            font-family: Arial, Helvetica, sans-serif;
        }
        #running-count-text {
            position: absolute;
            font-size: 20;
            right: 295;
            top: 20;
            color: white;
            font-weight: bold;
            font-family: Arial, Helvetica, sans-serif;
        }
    </style>
```

```
</head>
<body>

    <div>
        <select id="videoSource" style="position: absolute; padding: 1px;"></select>

        <span id="grid-remain-text">Cards Remaining</span>
        <div class="grid-container" id="grid-container"></div>

        <span id="running-count-text">Running Count : 0</span>

        <div id="detection-container">
            <!-- <video id="stream" autoplay muted src="video2.mp4"></video> -->
            <video id="stream" autoplay width="100%"></video>
            <!-- <img id="stream" src="final.png"> -->
        </div>

    </div>
    <script src="script.js" type="module"></script>
</body>
</html>
```

package.json

```
{
  "name": "client",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview",
    "deploy": "gh-pages -d dist"
  },
  "devDependencies": {
    "gh-pages": "^6.1.1",
    "vite": "^5.2.0"
  },
  "dependencies": {
    "density-clustering": "^1.3.0"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/mihkuno/BLACKJACK"
  },
  "author": "Joeninyo Cainday"
}
```

script.js

```
import clustering from 'density-clustering';

var deck_cards = [
    '10C', '10D', '10H', '10S', '2C', '2D', '2H', '2S', '3C', '3D',
    '3H', '3S', '4C', '4D', '4H', '4S', '5C', '5D', '5H', '5S', '6C',
    '6D', '6H', '6S', '7C', '7D', '7H', '7S', '8C', '8D', '8H', '8S',
    '9C', '9D', '9H', '9S', 'AC', 'AD', 'AH', 'AS', 'JC', 'JD', 'JH',
    'JS', 'KC', 'KD', 'KH', 'KS', 'QC', 'QD', 'QH', 'QS'
];
var omega_value = {
    '10C': -2, '10D': -2, '10H': -2, '10S': -2,
    '2C': 1, '2D': 1, '2H': 1, '2S': 1,
    '3C': 1, '3D': 1, '3H': 1, '3S': 1,
    '4C': 2, '4D': 2, '4H': 2, '4S': 2,
    '5C': 2, '5D': 2, '5H': 2, '5S': 2,
    '6C': 2, '6D': 2, '6H': 2, '6S': 2,
    '7C': 1, '7D': 1, '7H': 1, '7S': 1,
    '8C': 0, '8D': 0, '8H': 0, '8S': 0,
    '9C': -1, '9D': -1, '9H': -1, '9S': -1,
    'AC': -2, 'AD': -2, 'AH': -2, 'AS': -2,
    'JC': -2, 'JD': -2, 'JH': -2, 'JS': -2,
    'KC': -2, 'KD': -2, 'KH': -2, 'KS': -2,
    'QC': -2, 'QD': -2, 'QH': -2, 'QS': -2
}
```

```
var basic_strategy = [
// D  2   3   4   5   6   7   8   9  10   A            P

    // Hard Totals
    ['H','H','H','H','H','H','H','H','H','H'], //       0
    ['H','H','H','H','H','H','H','H','H','H'], //       1
    ['H','H','H','H','H','H','H','H','H','H'], //       2
    ['H','H','H','H','H','H','H','H','H','H'], //       3
    ['H','H','H','H','H','H','H','H','H','H'], //       4
    ['H','H','H','H','H','H','H','H','H','H'], //       5
    ['H','H','H','H','H','H','H','H','H','H'], //       6
    ['H','H','H','H','H','H','H','H','H','H'], //       7
    ['H','H','H','H','H','H','H','H','H','H'], //       8
    ['H','D','D','D','D','H','H','H','H','H'], //       9
    ['D','D','D','D','D','D','D','D','H','H'], //       10
    ['D','D','D','D','D','D','D','D','D','D'], //       11
    ['H','H','S','S','S','H','H','H','H','H'], //       12
    ['S','S','S','S','S','H','H','H','H','H'], //       13
    ['S','S','S','S','S','H','H','H','H','H'], //       14
    ['S','S','S','S','S','H','H','H','Z','H'], //       15
    ['S','S','S','S','S','H','H','Z','Z','Z'], //       16
    ['S','S','S','S','S','S','S','S','S','S'], //       17
    ['S','S','S','S','S','S','S','S','S','S'], //       18
    ['S','S','S','S','S','S','S','S','S','S'], //       19
    ['S','S','S','S','S','S','S','S','S','S'], //       20

    // Soft Totals
    ['H','H','H','D','D','H','H','H','H','H'], // A,2   21
    ['H','H','H','D','D','H','H','H','H','H'], // A,3   22
    ['H','H','D','D','D','H','H','H','H','H'], // A,4   23
    ['H','H','D','D','D','H','H','H','H','H'], // A,5   24
    ['H','D','D','D','D','H','H','H','H','H'], // A,6   25
    ['d','d','d','d','d','S','S','H','H','H'], // A,7   26
    ['S','S','S','S','d','S','S','S','S','S'], // A,8   27
    ['S','S','S','S','S','S','S','S','S','S'], // A,9   28

    // Pairs
    ['y','y','Y','Y','Y','Y','H','H','H','H'], // 2,2   29
    ['Y','Y','Y','Y','Y','Y','H','H','H','H'], // 3,3   30
    ['H','H','H','y','y','H','H','H','H','H'], // 4,4   31
    ['D','D','D','D','D','D','D','D','H','H'], // 5,5   32
    ['y','Y','Y','Y','Y','H','H','H','H','H'], // 6,6   33
    ['Y','Y','Y','Y','Y','Y','H','H','H','H'], // 7,7   34
    ['Y','Y','Y','Y','Y','Y','Y','Y','Y','Y'], // 8,8   35
    ['Y','Y','Y','Y','Y','S','Y','Y','S','S'], // 9,9   36
    ['S','S','S','S','S','S','S','S','S','S'], // 10,10 37
    ['Y','Y','Y','Y','Y','Y','Y','Y','Y','Y']  // A,A   38
];

var strategy_description = {
    'H': 'Hit',
    'S': 'Stand',
    'D': 'Double if possible, otherwise Stand',
    'd': 'Double if possible, otherwise Hit',
    'Y': 'Split',
    'y': 'Split if Double After Split is allowed',
    'Z': 'Surrender or Hit if Surrender is not allowed'
}

function getCardValue(card) {
    let value = card.slice(0, -1);

    if (value == 'A') {
        return 11;
    }
    else if (value == 'J' || value == 'Q' || value == 'K') {
        return 10;
    }
    else {
        return parseInt(value);
    }
}


function getStrategy(count, cards, dealerFaceCard=null) {
```

```
    // check if player has a blackjack or bust
    if (count >= 21) return;

    // check if dealer is null
    else if (dealerFaceCard == null) {
        return 'Waiting for next round..';
    }

    // row index of the basic strategy table
    const dealerIndex = getCardValue(dealerFaceCard) - 2;

    // used on pairs and soft hands
    const isTwoCard = cards.length == 2;
    const card1 = getCardValue(cards[0]);
    const card2 = getCardValue(cards[1]);

    // check if player has a pair
    if (isTwoCard && (card1 == card2) ) {
        return strategy_description[basic_strategy[29 + card1 - 2][dealerIndex]];
    }

    // check if player has a soft hand
    else if (isTwoCard && (card1 == 11 || card2 == 11)) {
        // Identify the value of the non-Ace card
        const nonAceCard = (card1 == 11) ? card2 : card1;
        return strategy_description[basic_strategy[21 + nonAceCard - 2][dealerIndex]];
    }

    // check if player has a hard hand
    else {
        return strategy_description[basic_strategy[count][dealerIndex]];
    }
}


var deck_count   = window.prompt("How many decks of cards do you want to use? \n(Enter a number between 1 to 10)");

// var deck_count = 1;

for (let i = 0; i < deck_count-1; i++) {
    deck_cards = deck_cards.concat(deck_cards);
}

const container = document.getElementById('grid-container');

deck_cards.forEach(str => {
    const item = document.createElement('div');
    item.className = 'grid-item-remaining';
    item.textContent = str;
    item.detectId = -1;
    container.appendChild(item);
});

// get video dom element
var stream = document.getElementById('stream');




// request access to webcam
//    navigator.mediaDevices.getUserMedia({video:   {width:   stream.offsetWidth,   height:   stream.offsetHeight   }}).then((frame)   =>
stream.srcObject = frame);


// get video dom element
// var stream = document.getElementById('stream');
var videoSelect = document.getElementById('videoSource');

function getConnectedDevices(type, callback) {
    navigator.mediaDevices.enumerateDevices()
        .then(devices => {
            const filtered = devices.filter(device => device.kind === type);
            callback(filtered);
        });
}


function startStream(deviceId) {
```

```
    if (window.stream && window.stream.getTracks) {
        window.stream.getTracks().forEach(track => track.stop());
    }
    const constraints = {
        video: {
            deviceId: deviceId ? { exact: deviceId } : undefined,
            width: stream.offsetWidth,
            height: stream.offsetHeight

        }
    };
    navigator.mediaDevices.getUserMedia(constraints)
        .then((frame) => {
            stream.srcObject = frame;
            window.stream = frame; // Assign the MediaStream object to window.stream
        })
        .catch((error) => {
            console.error('Error accessing media devices.', error);
        });
}


function gotDevices(deviceInfos) {
    videoSelect.innerHTML = '';
    deviceInfos.forEach(deviceInfo => {
        const option = document.createElement('option');
        option.value = deviceInfo.deviceId;
        option.text = deviceInfo.label || `Camera ${videoSelect.length + 1}`;
        videoSelect.appendChild(option);
    });
    videoSelect.onchange = () => startStream(videoSelect.value);
}

function initializeCamera() {
    navigator.mediaDevices.enumerateDevices()
        .then(devices => {
            const videoDevices = devices.filter(device => device.kind === 'videoinput');
            if (videoDevices.length > 0) {
                startStream(videoDevices[0].deviceId);
            }
        });
}

getConnectedDevices('videoinput', gotDevices);
initializeCamera(); // Initialize camera with the first available device

// Other parts of your code...



// request access to webcam
//    navigator.mediaDevices.getUserMedia({video:    {width:    stream.offsetWidth,    height:    stream.offsetHeight    }}).then((frame)    =>
stream.srcObject = frame);



// returns a frame encoded in base64
const getFrame = () => {
    const canvas = document.createElement('canvas');
    canvas.width = stream.offsetWidth;
    canvas.height = stream.offsetHeight;
    canvas.getContext('2d').drawImage(stream, 0, 0);
    const data = canvas.toDataURL('image/png');
    return data;
}

// Change this if you serve on different server or port
const WS_URL = 'ws://localhost:8000/ws'
// const WS_URL = 'wss://2217o5b2zh9g6g-8000.proxy.runpod.net/ws';

const ws = new WebSocket(WS_URL);

const dbscan = new clustering.DBSCAN();

ws.onopen = () => {
    console.log(`Connected to ${WS_URL}`);

    ws.send(getFrame());
```

```
}

ws.onmessage = message => {

    const detection_container = document.getElementById('detection-container');
    // Select all div elements within the parent div
    var divsToRemove = detection_container.querySelectorAll("div");
    // Iterate over the div elements and remove them
    divsToRemove.forEach(function(div) {
        div.remove();
    });

    const detections = JSON.parse(message.data);
    let coordinates = [];

    detections.forEach((detection) => {
        const [x1, y1, x2, y2, name, score, id] = detection;

        coordinates.push([x1, y1]);

        // handling the output bounding boxes
        // create a div element
        var div = document.createElement('div');

        // set div style
        div.style.position = 'absolute';
        div.style.left = `${x1}px`;
        div.style.top = `${y1}px`;
        div.style.width = `${x2 - x1}px`;
        div.style.height = `${y2 - y1}px`;
        div.style.border = '4px solid #a29bfe';
        div.style.color = 'black';
        div.style.display = 'flex';
        div.style.justifyContent = 'center';
        div.style.alignItems = 'center';
        div.style.textAlign = 'center';
        div.style.fontSize = '20px';
        div.style.fontWeight = 'bold';
        div.style.fontFamily = 'arial';
        div.style.backgroundColor = 'rgba(162,155,254, 0.4)';
        div.innerHTML = `${name} <br> ${score * 100}% <br> ${id}`;

        // append div to detection_container
        detection_container.appendChild(div);


        // handling the count of cards
        for (let item of container.childNodes) {
            if (item.textContent == name && item.detectId == id) {
                break;
            }
            else if (item.textContent == name && item.detectId == -1) {
                item.detectId = id;
                item.className = 'grid-item-tracked';

                const running_count = document.getElementById('running-count-text');
                const running_value = parseInt(running_count.textContent.split(' : ')[1]) + parseInt(omega_value[name]);
                running_count.textContent = 'Running Count : ' + running_value.toString();
                break;
            }
        }
    });

    // start checking for player and dealer clusters if there are at least 3 detections
    if (detections.length < 3) {
        setTimeout(() => { ws.send(getFrame()); }, 1);
        return;
    }

    var clusters = dbscan.run(coordinates, 500, 2);
    console.log(clusters, dbscan.noise);


    let dealerFaceCard = null;

    // check if there is one dealer face card up detected (noise)
    if (dbscan.noise.length >= 1) {
```

```
    // find the noise that has the minimum y value
    let miny = Infinity;
    let bucket = [];
    dbscan.noise.forEach(noise_point => {
        const [x1, y1, x2, y2, card, score, id] = detections[noise_point];

        if (y1 < miny) {
            miny = y1;
            dealerFaceCard = card;
            bucket = detections[noise_point];
        }
    });
    const [x1, y1, x2, y2, card, score, id] = bucket;


    // create a div element
    var div = document.createElement('div');

    // set div style
    div.style.position = 'absolute';
    div.style.left = `${x1-40}px`;
    div.style.top = `${y1-40}px`;
    div.style.width = `${x2 - x1 + 40}px`;
    div.style.height = `${y2 - y1 + 40}px`;
    div.style.border = '4px solid #a29bfe';
    div.style.color = 'black';
    div.style.fontSize = '20px';
    div.style.fontWeight = 'bold';
    div.style.fontFamily = 'arial';
    div.style.backgroundColor = 'rgba(162,155,254, 0.2)';
    div.style.zIndex = 1;
    div.style.padding = '10px';
    div.style.borderRadius = '20px';
    div.innerHTML = `Dealer's Card`;

    // append div to detection_container
    detection_container.appendChild(div);
}


// if no dealer face card, find the dealer's cluster that contains the cluster point minimum y value
let dealerClusterIndex = null;
let dealerClusterCount = 0;

if (dealerFaceCard == null) {
    let miny = Infinity;

    clusters.forEach((cluster, index) => {
        cluster.forEach(cluster_point => {
            const [x1, y1, x2, y2, card, score, id] = detections[cluster_point];
            if (y1 < miny) {
                miny = y1;
                dealerClusterIndex = index;
            }

            let numberOfAceDetected = 0;
            if (getCardValue(card) == 11) {
                numberOfAceDetected += 1;
            }

            dealerClusterCount += getCardValue(card);

            while (dealerClusterCount > 21 && numberOfAceDetected > 0) {
                dealerClusterCount -= 10;
                numberOfAceDetected -= 1;
            }

        });
    });
}


clusters.forEach((cluster, index) => {

    let minx = Infinity;
    let miny = Infinity;
    let maxx = -Infinity;
```

```
        let maxy = -Infinity;

        let count = 0;
        let cards = [];

        let numberOfAceDetected = 0;
        cluster.forEach(cluster_point => {
            const [x1, y1, x2, y2, card, score, id] = detections[cluster_point];
            if (x1 < minx) minx = x1;
            if (y1 < miny) miny = y1;
            if (x2 > maxx) maxx = x2;
            if (y2 > maxy) maxy = y2;

            if (getCardValue(card) == 11) {
                numberOfAceDetected += 1;
            }

            count += getCardValue(card);
            cards.push(card);
        });

        while (count > 21 && numberOfAceDetected > 0) {
            count -= 10;
            numberOfAceDetected -= 1;
        }


        // handling the output bounding boxes for clusters
        // create a div element
        var div = document.createElement('div');
        // set div style
        div.style.position = 'absolute';
        div.style.left = `${minx-40}px`;
        div.style.top = `${miny-40}px`;
        div.style.width = `${maxx - minx + 40}px`;
        div.style.height = `${maxy - miny + 40}px`;
        div.style.border = '4px solid #a29bfe';
        div.style.color = 'black';
        div.style.fontSize = '20px';
        div.style.fontWeight = 'bold';
        div.style.fontFamily = 'arial';
        div.style.zIndex = 1;
        div.style.padding = '10px';
        div.style.borderRadius = '20px';


        const strategy = getStrategy(count, cards, dealerFaceCard);

        if (dealerFaceCard == null && dealerClusterIndex == index) {
            div.style.border = '4px solid rgba(255,165,0, 1)';

            if (count == 21) {
                div.style.backgroundColor = 'rgba(0,208,98, 0.6)';
                div.innerHTML = `Dealer: ${index} <br> Count: ${count} <br> Cards: [${cards}] <br> BLACKJACK`;
            }
            else if (count > 21) {
                div.style.backgroundColor = 'rgba(255,99,71, 0.6)';
                div.innerHTML = `Dealer: ${index} <br> Count: ${count} <br> Cards: [${cards}] <br> BUST`;
            }
            else {
                div.style.backgroundColor = 'rgba(255,165,0, 0.6)';
                div.innerHTML = `Dealer: ${index} <br> Count: ${count} <br> Cards: [${cards}]`;
            }

        }
        else if (dealerFaceCard == null) {
            if (count == 21) {
                div.style.backgroundColor = 'rgba(0,208,98, 0.6)';
                div.innerHTML = `Player: ${index} <br> Count: ${count} <br> Cards: [${cards}] <br> BLACKJACK`;
            }
            else if (count > 21) {
                div.style.backgroundColor = 'rgba(255,99,71, 0.6)';
                div.innerHTML = `Player: ${index} <br> Count: ${count} <br> Cards: [${cards}] <br> BUST`;
            }
            else if (count > dealerClusterCount) {
                div.style.backgroundColor = 'rgba(0,208,98, 0.6)';
                div.innerHTML = `Player: ${index} <br> Count: ${count} <br> Cards: [${cards}] <br> WINNER`;
            }
```

```
            else if (count < dealerClusterCount) {
                div.style.backgroundColor = 'rgba(255,99,71, 0.6)';
                div.innerHTML = `Player: ${index} <br> Count: ${count} <br> Cards: [${cards}] <br> LOSER`;
            }
            else if (count == dealerClusterCount) {
                div.style.backgroundColor = 'rgba(255,165,0, 0.6)';
                div.innerHTML = `Player: ${index} <br> Count: ${count} <br> Cards: [${cards}] <br> TIE`;
            }
        }
        else {
            if (count == 21) {
                div.style.backgroundColor = 'rgba(0,208,98, 0.6)';
                div.innerHTML = `Player: ${index} <br> Count: ${count} <br> Cards: [${cards}] <br> BLACKJACK`;
            }
            else if (count > 21) {
                div.style.backgroundColor = 'rgba(255,99,71, 0.6)';
                div.innerHTML = `Player: ${index} <br> Count: ${count} <br> Cards: [${cards}] <br> BUST`;
            }
            else {
                div.style.backgroundColor = 'rgba(162,155,254, 0.2)';
                div.innerHTML = `Player: ${index} <br> Count: ${count} <br> Cards: [${cards}] <br> Strategy: ${strategy}`;
            }
        }


        // append div to detection_container
        detection_container.appendChild(div);
    });

    setTimeout(() => { ws.send(getFrame()); }, 1);
}
```

Back-end

main.py

```
from fastapi import FastAPI
from starlette.responses import FileResponse
from starlette.websockets import WebSocket

from service.vision import detect, from_b64

app = FastAPI()

@app.get("/test")
async def test_endpoint():
    return {"message": "Test is working!"}


@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket):
    await websocket.accept()
    while True:
        img = await websocket.receive_text()
        await websocket.send_text( detect(img) )
```

service.vision.py

```
import cv2
import base64
import numpy as np
import json
from ultralytics import YOLO

card_model = YOLO('./model/detect/blackjack-model-final/weights/best.pt')

def from_b64(uri):
    '''
        Convert from b64 uri to OpenCV image
        Sample input: '......'
    '''
    encoded_data = uri.split(',')[1]
    data = base64.b64decode(encoded_data)
```

```
    np_arr = np.fromstring(data, np.uint8)
    img = cv2.imdecode(np_arr, cv2.IMREAD_COLOR)
    return img


def to_b64(img):
    '''
        Convert from OpenCV image to b64 uri
        Sample output: '......'
    '''
    _, buffer = cv2.imencode('.jpg', img)
    img = base64.b64encode(buffer).decode('utf-8')

    return f"{img}"


def detect(img):
    global card_model, corner_model, orientation_model, orientation_names

    cards = []

    try:

        # get the converted image from the b64 uri
        img = from_b64(img)

        # get whole card detection results
        result_card = card_model.track(img, conf=0.65, persist=True, tracker="botsort.yaml")


        for r in result_card:
            for box in r.boxes:
                id = box.id.item()
                b = box.xyxy[0]                         # get box coordinates in (left, top, right, bottom) format
                c = box.cls                             # class index
                confidence = round(float(box.conf), 2)  # confidence score
                left, top, right, bottom = map(int, b.tolist())  # Convert numpy scalars to Python integers

                print(id)

                card = card_model.names[int(c)] # class name
                cards.append((left, top, right, bottom, card, confidence, id))

    except:
        print('No image found.')

    return json.dumps(cards)
```

Dockerfile

```
FROM python:3.12.2-slim

WORKDIR /app

COPY . /app

RUN apt-get update && apt-get install ffmpeg libsm6 libxext6 -y

RUN pip install -r requirements.txt

CMD gunicorn main:app -k uvicorn.workers.UvicornWorker --bind 0.0.0.0:8000
```

model.train.py

```
from ultralytics import YOLO

# Build a YOLOv9c model from pretrained weight
model = YOLO('yolov8s.pt')

# Display model information (optional)
model.info()

# Train the model for 100 epochs
results = model.train(data='/home/mihkuno/Desktop/datasets/data.yaml', epochs=20, imgsz=1280, verbose=True, batch=4)
```

model.results.csv

```
epoch, train/box_loss, train/cls_loss, train/dfl_loss, metrics/precision(B), metrics/recall(B), metrics/mAP50(B), metrics/mAP50-95(B), val/box_loss, val/cls_loss, val/dfl_loss, lr/pg0, lr/pg1, lr/pg2
1, 1.2327, 5.2279, 1.9175, 0.18559, 0.48981, 0.25682, 0.16937, 1.2015, 2.4489, 2.1164, 0.00023702, 0.00023702, 0.00023702
2, 0.94057, 3.4377, 1.5943, 0.47917, 0.74412, 0.68558, 0.54285, 0.86461, 1.4084, 1.7089, 0.00047032, 0.00047032, 0.00047032
3, 0.88361, 2.4034, 1.5067, 0.69636, 0.78524, 0.83923, 0.68266, 0.77561, 1.1449, 1.5621, 0.00069891, 0.00069891, 0.00069891
4, 0.81452, 1.6846, 1.4353, 0.85411, 0.8711, 0.93992, 0.75627, 0.7851, 0.82764, 1.5426, 0.00069279, 0.00069279, 0.00069279
5, 0.81827, 1.2903, 1.4154, 0.92154, 0.90647, 0.96902, 0.80239, 0.70894, 0.66051, 1.4341, 0.00068573, 0.00068573, 0.00068573
6, 0.78284, 1.0644, 1.3867, 0.94868, 0.8909, 0.96726, 0.79931, 0.74061, 0.6428, 1.4517, 0.00067866, 0.00067866, 0.00067866
7, 0.79057, 0.98428, 1.3845, 0.93889, 0.93221, 0.97563, 0.81867, 0.67659, 0.57712, 1.3788, 0.00067159, 0.00067159, 0.00067159
8, 0.74418, 0.85359, 1.3279, 0.95425, 0.93749, 0.98245, 0.82727, 0.67317, 0.52768, 1.3766, 0.00066452, 0.00066452, 0.00066452
9, 0.73596, 0.78214, 1.3187, 0.97408, 0.94204, 0.987, 0.84258, 0.64818, 0.46657, 1.3555, 0.00065745, 0.00065745, 0.00065745
10, 0.69613, 0.72651, 1.2999, 0.96337, 0.95371, 0.98299, 0.83199, 0.66046, 0.48126, 1.3858, 0.00065038, 0.00065038, 0.00065038
11, 0.71566, 0.71068, 1.3055, 0.96829, 0.95751, 0.98649, 0.84228, 0.65194, 0.46704, 1.3392, 0.00064331, 0.00064331, 0.00064331
12, 0.68948, 0.65529, 1.2765, 0.97398, 0.958, 0.9878, 0.83865, 0.6551, 0.44565, 1.343, 0.00063625, 0.00063625, 0.00063625
13, 0.68463, 0.63422, 1.275, 0.97654, 0.96008, 0.98928, 0.85422, 0.61496, 0.41304, 1.2999, 0.00062918, 0.00062918, 0.00062918
14, 0.69947, 0.6316, 1.285, 0.9681, 0.96078, 0.98645, 0.83936, 0.6478, 0.42763, 1.3296, 0.00062211, 0.00062211, 0.00062211
15, 0.69083, 0.61926, 1.272, 0.96375, 0.95879, 0.98684, 0.8497, 0.62743, 0.43908, 1.3333, 0.00061504, 0.00061504, 0.00061504
16, 0.68703, 0.59467, 1.2693, 0.97352, 0.96614, 0.98862, 0.84362, 0.64309, 0.41998, 1.3306, 0.00060797, 0.00060797, 0.00060797
17, 0.67636, 0.56943, 1.2589, 0.98296, 0.97209, 0.99179, 0.85481, 0.61408, 0.38799, 1.3059, 0.0006009, 0.0006009, 0.0006009
18, 0.6684, 0.55033, 1.2443, 0.97541, 0.97013, 0.98871, 0.854, 0.61444, 0.38399, 1.3058, 0.00059383, 0.00059383, 0.00059383
19, 0.66316, 0.53805, 1.2508, 0.97922, 0.97044, 0.99039, 0.85931, 0.59723, 0.37556, 1.2814, 0.00058677, 0.00058677, 0.00058677
20, 0.67231, 0.55494, 1.2523, 0.96986, 0.97042, 0.98933, 0.86216, 0.6039, 0.36863, 1.2841, 0.0005797, 0.0005797, 0.0005797
21, 0.65226, 0.50498, 1.2334, 0.9725, 0.96871, 0.9886, 0.86209, 0.59615, 0.36405, 1.2765, 0.00057263, 0.00057263, 0.00057263
22, 0.66627, 0.52132, 1.2426, 0.97247, 0.97251, 0.98938, 0.85973, 0.6027, 0.36909, 1.2633, 0.00056556, 0.00056556, 0.00056556
23, 0.63694, 0.4838, 1.2216, 0.98335, 0.97821, 0.99075, 0.86285, 0.60608, 0.34054, 1.2845, 0.00055849, 0.00055849, 0.00055849
24, 0.64204, 0.50358, 1.2138, 0.97698, 0.97554, 0.99097, 0.86634, 0.59419, 0.34923, 1.2685, 0.00055142, 0.00055142, 0.00055142
```

```
                25,              0.62968,              0.46527,               1.214,              0.9766,
0.97777,           0.99162,              0.86479,              0.59892,              0.35632,              1.2699,
0.00054435,           0.00054435,           0.00054435
                26,              0.62183,              0.48014,              1.2149,              0.97885,
0.98062,           0.99262,              0.86955,               0.5851,              0.33227,               1.259,
0.00053728,           0.00053728,           0.00053728
                27,              0.65023,              0.49597,              1.2302,              0.98409,
0.96892,           0.99114,              0.86727,              0.58123,               0.3377,              1.2487,
0.00053022,           0.00053022,           0.00053022
                28,              0.61547,              0.46279,              1.2084,              0.97818,
0.9726,            0.99156,              0.8675,               0.59472,              0.3415,               1.2668,
0.00052315,           0.00052315,           0.00052315
                29,              0.62457,              0.46005,              1.2096,               0.9817,
0.97061,           0.99207,               0.8726,              0.57532,              0.32572,              1.2373,
0.00051608,           0.00051608,           0.00051608
                30,              0.63698,              0.46621,              1.2148,              0.98229,
0.96956,           0.99041,              0.87296,               0.582,               0.33109,              1.2513,
0.00050901,           0.00050901,           0.00050901
                31,              0.62283,              0.45749,              1.2187,              0.98174,
0.97156,           0.99082,               0.8693,              0.58759,              0.33666,              1.2469,
0.00050194,           0.00050194,           0.00050194
                32,              0.60841,              0.44481,              1.2068,              0.98159,
0.96628,            0.9913,              0.87309,              0.56903,              0.31941,              1.2377,
0.00049487,           0.00049487,           0.00049487
                33,              0.61684,              0.43599,              1.2081,               0.9807,
0.97219,           0.99101,              0.87093,              0.58125,              0.32452,               1.255,
0.0004878,            0.0004878,            0.0004878
                34,              0.63289,              0.43596,              1.2111,              0.98543,
0.97307,           0.99218,              0.87457,              0.57935,              0.31828,              1.2422,
0.00048074,           0.00048074,           0.00048074
                35,              0.61915,              0.42838,              1.1935,              0.99018,
0.96915,           0.99222,              0.87435,              0.57905,              0.31564,              1.2437,
0.00047367,           0.00047367,           0.00047367
                36,              0.61541,              0.42458,               1.203,              0.98101,
0.97784,           0.99269,              0.87833,              0.56125,              0.30403,              1.2244,
0.0004666,            0.0004666,            0.0004666
                37,              0.60301,              0.41466,              1.1825,              0.98503,
0.97613,           0.99234,              0.87971,              0.56336,              0.30718,              1.2132,
0.00045953,           0.00045953,           0.00045953
                38,              0.58669,              0.41189,              1.1789,              0.98244,
0.97869,           0.99193,               0.8762,              0.57192,              0.29858,              1.2333,
0.00045246,           0.00045246,           0.00045246
                39,              0.61884,               0.4286,              1.1941,              0.98546,
0.98318,            0.9928,              0.87621,              0.56847,              0.29745,               1.232,
0.00044539,           0.00044539,           0.00044539
                40,              0.59153,              0.39518,               1.188,               0.9759,
0.986,             0.99317,              0.87783,              0.5695,               0.30069,              1.2368,
0.00043832,           0.00043832,           0.00043832
                41,              0.60212,              0.40307,              1.1764,              0.98541,
0.97691,           0.99289,              0.87777,              0.57273,              0.29809,              1.2435,
0.00043126,           0.00043126,           0.00043126
                42,              0.57519,              0.40061,              1.1693,               0.9801,
0.97511,           0.99197,               0.8761,               0.5716,              0.30159,              1.2427,
0.00042419,           0.00042419,           0.00042419
                43,               0.5927,              0.39889,              1.1742,              0.98424,
0.98316,           0.99342,              0.87891,              0.56634,              0.29131,              1.2269,
0.00041712,           0.00041712,           0.00041712
                44,              0.58858,              0.39772,              1.1873,              0.98582,
0.97523,           0.99194,              0.87894,              0.56038,              0.28963,              1.2313,
0.00041005,           0.00041005,           0.00041005
                45,              0.59067,              0.39333,              1.1733,              0.98301,
0.97346,           0.99206,              0.88195,              0.56221,              0.29349,              1.2177,
0.00040298,           0.00040298,           0.00040298
                46,              0.58337,              0.39261,              1.1817,               0.9837,
0.97998,            0.9931,              0.88318,              0.56012,               0.2886,              1.2238,
0.00039591,           0.00039591,           0.00039591
                47,              0.58331,              0.39047,               1.168,              0.9824,
0.98016,            0.9925,               0.878,               0.56659,              0.30059,              1.2292,
0.00038884,           0.00038884,           0.00038884
                48,              0.59681,              0.40547,              1.1665,               0.983,
0.9823,            0.99238,              0.88262,              0.56606,              0.28714,              1.2246,
0.00038178,           0.00038178,           0.00038178
                49,              0.58087,               0.386,               1.1615,              0.98671,
0.97914,           0.99347,               0.8835,              0.55484,              0.27931,              1.2159,
0.00037471,           0.00037471,           0.00037471
                50,               0.5894,              0.39917,              1.1633,               0.9837,
0.98348,           0.99288,               0.8823,              0.55732,               0.2892,              1.2154,
0.00036764,           0.00036764,           0.00036764
```

```
                51,              0.58145,              0.38769,               1.1537,              0.98732,
0.98006,              0.99305,              0.87878,              0.56653,              0.28857,               1.2304,
0.00036057,           0.00036057,           0.00036057
                52,              0.57694,              0.38727,               1.1594,              0.97999,
0.98293,              0.99303,              0.88187,              0.56238,              0.28156,               1.2292,
0.0003535,            0.0003535,            0.0003535
                53,              0.56847,              0.36638,               1.1701,              0.98641,
0.97348,              0.99238,              0.88383,              0.55655,              0.28305,               1.2154,
0.00034643,           0.00034643,           0.00034643
                54,              0.59205,              0.37999,               1.1576,              0.98852,
0.97951,              0.99259,              0.87924,               0.5618,              0.28497,                1.226,
0.00033936,           0.00033936,           0.00033936
                55,              0.55684,                0.356,               1.148,              0.98192,
0.98092,               0.9928,              0.88285,              0.55947,              0.28263,                1.215,
0.0003323,            0.0003323,            0.0003323
                56,              0.57437,              0.37302,               1.1615,               0.9875,
0.97702,              0.99218,               0.8864,              0.55225,              0.27587,               1.2139,
0.00032523,           0.00032523,           0.00032523
                57,              0.56306,              0.35674,               1.1315,              0.98511,
0.97535,              0.99271,              0.88496,              0.55718,              0.28272,               1.2225,
0.00031816,           0.00031816,           0.00031816
                58,              0.57006,              0.36135,                1.142,              0.98239,
0.97898,              0.99214,               0.8834,              0.55574,              0.28041,               1.2237,
0.00031109,           0.00031109,           0.00031109
                59,              0.55538,              0.36523,               1.1372,              0.98531,
0.97639,               0.9919,              0.88754,              0.55015,              0.27815,               1.2133,
0.00030402,           0.00030402,           0.00030402
                60,               0.5413,              0.35414,               1.1247,               0.9849,
0.98108,              0.99245,              0.88986,              0.54671,              0.27166,               1.1997,
0.00029695,           0.00029695,           0.00029695
                61,              0.53981,              0.34739,               1.1367,              0.98579,
0.98196,              0.99247,              0.88593,              0.55092,              0.27337,               1.2082,
0.00028988,           0.00028988,           0.00028988
                62,              0.57436,              0.36658,               1.1564,               0.9798,
0.98363,              0.99303,              0.88837,              0.55031,              0.27079,               1.2102,
0.00028282,           0.00028282,           0.00028282
                63,              0.56374,              0.36031,                1.152,              0.98493,
0.97671,              0.99255,              0.88579,              0.54843,              0.27299,                1.207,
0.00027575,           0.00027575,           0.00027575
                64,              0.56217,              0.35232,               1.1541,              0.98854,
0.98329,              0.99266,              0.88684,              0.55529,              0.27411,               1.2144,
0.00026868,           0.00026868,           0.00026868
                65,              0.55836,              0.34589,               1.1333,              0.98665,
0.98116,              0.99259,              0.88808,               0.5497,              0.26851,               1.2103,
0.00026161,           0.00026161,           0.00026161
                66,              0.53983,              0.34102,               1.1376,              0.98594,
0.98097,              0.99285,              0.88855,              0.55059,              0.26973,               1.2079,
0.00025454,           0.00025454,           0.00025454
                67,              0.54308,              0.35025,               1.1348,              0.98674,
0.97975,              0.99207,              0.88832,              0.54483,              0.27139,                1.194,
0.00024747,           0.00024747,           0.00024747
                68,              0.54151,              0.33646,               1.1131,               0.9861,
0.98245,              0.99299,              0.88756,              0.54902,              0.26587,               1.2089,
0.0002404,            0.0002404,            0.0002404
                69,              0.55677,              0.34577,               1.1377,              0.98485,
0.98362,              0.99223,              0.88625,              0.55123,              0.26635,               1.2125,
0.00023334,           0.00023334,           0.00023334
                70,              0.55906,              0.33824,                1.115,              0.98782,
0.9836,               0.99237,              0.88664,              0.54807,              0.26349,               1.2096,
0.00022627,           0.00022627,           0.00022627
                71,              0.54239,              0.34287,               1.1138,              0.98666,
0.98079,              0.99187,              0.88964,              0.54465,              0.26534,               1.1978,
0.0002192,            0.0002192,            0.0002192
                72,              0.54711,              0.34255,               1.1312,              0.98638,
0.98314,              0.99309,               0.8899,               0.5485,               0.2617,               1.2077,
0.00021213,           0.00021213,           0.00021213
                73,               0.5483,              0.34593,               1.1392,              0.98804,
0.98244,               0.9928,              0.88762,              0.54536,              0.26586,               1.2039,
0.00020506,           0.00020506,           0.00020506
                74,              0.53811,              0.33558,               1.1217,              0.98783,
0.98558,              0.99264,              0.88796,              0.54998,              0.26511,               1.2079,
0.00019799,           0.00019799,           0.00019799
                75,              0.54966,              0.33607,               1.1178,              0.98826,
0.98305,              0.99295,              0.89023,              0.54548,              0.26114,               1.2071,
0.00019092,           0.00019092,           0.00019092
                76,              0.53194,              0.32284,               1.1081,              0.98677,
0.98646,              0.99247,               0.8903,              0.54863,              0.26364,               1.2055,
0.00018385,           0.00018385,           0.00018385
```

```
                77,              0.54277,              0.33339,              1.1265,              0.98816,
0.98511,              0.99308,              0.88962,              0.55044,              0.26253,              1.2125,
0.00017679,              0.00017679,              0.00017679
                78,              0.52468,               0.3191,              1.1234,              0.98613,
0.98534,              0.99266,              0.88869,               0.5511,               0.264,               1.212,
0.00016972,              0.00016972,              0.00016972
                79,              0.53776,              0.33507,              1.1133,              0.98577,
0.98452,              0.99266,              0.88845,              0.54931,               0.2617,              1.2075,
0.00016265,              0.00016265,              0.00016265
                80,              0.55293,              0.34238,              1.1237,              0.98745,
0.98179,              0.99245,               0.8901,              0.54715,              0.26352,              1.2026,
0.00015558,              0.00015558,              0.00015558
                81,              0.54328,              0.32546,              1.1255,              0.98959,
0.98214,              0.99285,              0.88981,              0.54554,              0.26087,              1.2076,
0.00014851,              0.00014851,              0.00014851
                82,              0.52729,              0.31968,              1.1164,              0.98573,
0.98321,              0.99316,              0.89108,              0.54844,              0.25787,               1.209,
0.00014144,              0.00014144,              0.00014144
                83,              0.50871,              0.30265,              1.1024,              0.98631,
0.98374,              0.99286,              0.89051,              0.54974,              0.25951,              1.2111,
0.00013437,              0.00013437,              0.00013437
                84,              0.54027,              0.32311,               1.112,              0.98942,
0.98353,              0.99317,              0.89196,              0.54701,              0.25584,              1.2078,
0.00012731,              0.00012731,              0.00012731
                85,              0.51483,              0.31073,              1.1011,              0.99005,
0.98233,              0.99274,              0.89418,              0.54495,              0.25735,              1.1996,
0.00012024,              0.00012024,              0.00012024
                86,              0.52498,              0.31851,              1.1063,              0.98789,
0.98135,              0.99293,              0.89394,              0.54251,              0.25697,              1.2024,
0.00011317,              0.00011317,              0.00011317
                87,              0.52842,              0.31935,              1.1093,              0.98772,
0.98288,              0.99278,               0.8922,              0.54323,              0.25866,              1.2059,
0.0001061,               0.0001061,               0.0001061
                88,              0.51417,              0.30461,              1.0988,              0.98624,
0.98303,              0.99248,              0.89262,              0.54094,              0.25878,              1.2009,
9.9032e-05,              9.9032e-05,              9.9032e-05
                89,              0.52176,              0.31451,              1.0917,              0.98492,
0.98269,              0.99259,              0.89143,              0.54823,              0.25976,              1.2101,
9.1963e-05,              9.1963e-05,              9.1963e-05
                90,              0.51574,              0.31516,              1.0965,              0.98757,
0.98495,               0.9928,              0.89351,               0.5437,              0.25847,              1.2063,
8.4895e-05,              8.4895e-05,              8.4895e-05
                91,               0.5533,              0.25772,              0.98284,              0.98689,
0.98409,              0.99271,              0.89423,              0.54208,              0.25568,              1.2037,
7.7826e-05,              7.7826e-05,              7.7826e-05
                92,              0.53707,              0.24761,              0.97912,              0.98756,
0.98334,              0.99243,              0.89364,              0.54292,              0.25721,              1.2084,
7.0757e-05,              7.0757e-05,              7.0757e-05
                93,              0.54767,              0.24912,              0.98069,              0.98328,
0.98496,              0.99257,              0.89462,              0.54052,               0.2577,              1.2064,
6.3689e-05,              6.3689e-05,              6.3689e-05
                94,              0.55468,              0.25132,              0.97874,              0.98451,
0.98371,              0.99217,              0.89356,              0.54384,              0.26007,              1.2067,
5.662e-05,               5.662e-05,               5.662e-05
                95,              0.53347,              0.24076,              0.96963,              0.98429,
0.98543,              0.99219,              0.89567,              0.54189,              0.25875,              1.2078,
4.9552e-05,              4.9552e-05,              4.9552e-05
                96,               0.5545,              0.24884,              0.97365,              0.98263,
0.98264,              0.99187,              0.89312,              0.54323,              0.25761,              1.2109,
4.2483e-05,              4.2483e-05,              4.2483e-05
                97,              0.56518,              0.25041,              0.96477,              0.98652,
0.98037,               0.9918,              0.89188,              0.54241,              0.25751,              1.2126,
3.5414e-05,              3.5414e-05,              3.5414e-05
                98,              0.54692,              0.24727,              0.96057,              0.98583,
0.98099,              0.99191,              0.89435,              0.54191,              0.25672,              1.2078,
2.8346e-05,              2.8346e-05,              2.8346e-05
                99,              0.54955,              0.24653,              0.96336,              0.98466,
0.98198,              0.99197,              0.89431,              0.54122,              0.25671,              1.2099,
2.1277e-05,              2.1277e-05,              2.1277e-05
               100,              0.55397,               0.2455,              0.96144,              0.98415,
0.9829,               0.99245,              0.89462,               0.54227,              0.25685,              1.2113,
1.4209e-05,              1.4209e-05,              1.4209e-05
```