🎥 Browser-Based Video Summary System

This system converts your existing video summary pipeline into a browser-based application that can capture video from your camera and generate AI-powered summaries in real-time.

📋 System Architecture

```
Browser Frontend (HTML/JS)
    ↓ WebSocket Connection
WebSocket Backend (Python)
    ↓ Processes frames using
Your Existing Pipeline:
   - YOLO Object Detection
   - Depth Estimation (MIDAS)
   - Image Captioning (ViT-GPT2)
   - License Plate Recognition
   - LLM Summarization (Gemini)
```

🛠️ Setup Instructions

1. Install Dependencies

```bash
# Install Python dependencies
pip install -r requirements.txt

# If you don't have the requirements.txt, install manually:
pip install websockets torch torchvision transformers ultralytics opencv-python numpy Pillow langchain langchain-goo
```

2. Required Files

Make sure you have all these files in your project directory:

```
📁 Your Project Directory/
├── 📄 ocr_server.py              # Your OCR server
├── 📄 utils_video_summary.py     # Your utility functions
├── 📄 websocket_backend.py       # New WebSocket backend
├── 📄 video_summary_frontend.html  # New browser frontend
├── 📄 startup.py                 # Startup script (optional)
├── 📄 requirements.txt           # Dependencies
├── 🤖 best.pt                    # Your YOLO model
├── 🤖 license_plate_detector.pt  # Your license plate detector
└── 📄 README.md                  # This guide
```

3. Model Files

Ensure you have your trained model files:

- `best.pt` - Your custom YOLO model
- `license_plate_detector.pt` - License plate detection model

4. Google API Key

Update the Google API key in `websocket_backend.py`:

```
python

os.environ["GOOGLE_API_KEY"] = "your_actual_google_api_key_here"
```

🚀 How to Run

Option 1: Using the Startup Script (Recommended)

```
bash

python startup.py
```

This will automatically start both servers and provide you with status updates.

Option 2: Manual Startup

1. **Start the OCR server:**

```
bash

python ocr_server.py
```

Wait for: `[OCR Server] Listening on 0.0.0.0:6000...`

2. **Start the WebSocket backend:**

```
bash

python websocket_backend.py
```

Wait for: `✅ Server is ready! Open the frontend HTML file in your browser.`

3. **Open the frontend:**
   - Open `video_summary_frontend.html` in your web browser
   - Grant camera permissions when prompted

🎮 How to Use

1. **Open the frontend** in your browser (Chrome/Firefox recommended)
2. **Grant camera access** when prompted
3. **Click "Start Recording"** - the system will:
    - Capture video for 10 seconds
    - Show a countdown timer
    - Stream frames to the backend for processing
4. **Wait for processing** - after recording stops:
    - Frames are processed through your AI pipeline
    - Object detection, depth estimation, and captioning occur
    - A summary is generated using Gemini
5. **View the summary** - the final summary appears on the page

🔧 Configuration

Recording Duration

Change the recording duration in the frontend:

```javascript
this.recordingDuration = 10; // seconds
```

Frame Rate

Adjust frame capture rate in the frontend:

```javascript
}, 100); // Capture every 100ms (~10 FPS)
```

Detection Interval

Modify how often object detection runs in the backend:

```python
self.DETECTION_INTERVAL = 10  # Process every 10th frame
```

Confidence Threshold

Adjust detection confidence in the backend:

```python
self.CONFIDENCE_THRESHOLD = 0.4  # 40% confidence minimum
```

🌐 Network Configuration

Default Ports:

- **WebSocket Server**: `localhost:8765`
- **OCR Server**: `localhost:6000`
- **Frontend**: File-based (no server needed)

Changing Ports:

1. **WebSocket Backend** (`websocket_backend.py`):

```python
start_server = websockets.serve(
    server.handle_client,
    "localhost",
    8765,  # Change this port
    # ...
)
```

2. **Frontend** (`video_summary_frontend.html`):

```javascript
this.ws = new WebSocket('ws://localhost:8765'); // Update port here
```

3. **OCR Server** (`ocr_server.py`):

```python
def start_server(host="0.0.0.0", port=6000): # Change port here
```

🔍 Troubleshooting

Common Issues:

1. **"Camera access denied"**
   - Ensure you're using HTTPS or localhost
   - Check browser permissions for camera access
2. **"Not connected to server"**
   - Make sure both OCR server and WebSocket server are running
   - Check console for connection errors
3. **"Models not loading"**
   - Ensure all model files (`.pt`) are in the correct directory
   - Check CUDA availability if using GPU
4. **"OCR server error"**
   - Verify OCR server is running on port 6000
   - Check if docTR and PaddleOCR are properly installed
5. **"Google API error"**
   - Verify your Google API key is valid
   - Ensure Gemini API is enabled in your Google Cloud Console

Debug Mode:

To enable debug output, check the browser console and terminal outputs. The system provides extensive logging:

- 🚀 Server startup messages
- 📹 Frame reception logs
- 🎬 Processing progress
- ✅ Success confirmations
- × Error messages

📱 Browser Compatibility

Recommended Browsers:

- Chrome 88+ (best performance)
- Firefox 85+
- Safari 14+ (limited)
- Edge 88+

Note: Camera access requires HTTPS or localhost for security reasons.

⚡ Performance Tips

1. **GPU Usage**: Ensure CUDA is available for faster processing
2. **Frame Rate**: Lower frame rate = less bandwidth, faster processing
3. **Resolution**: Camera resolution affects processing speed
4. **Detection Interval**: Higher interval = faster processing, less accuracy

🔒 Security Considerations

- The system runs locally, no data leaves your machine
- Camera access is requested explicitly
- WebSocket connections are unencrypted (local use only)
- For production use, implement HTTPS/WSS

📈 System Requirements

Minimum:

- 8GB RAM
- Modern CPU (Intel i5+ or AMD equivalent)
- Python 3.8+
- Modern web browser

Recommended:

- 16GB+ RAM
- NVIDIA GPU with CUDA support
- Python 3.9+
- Chrome browser

🆘 Getting Help

If you encounter issues:

1. Check the terminal output for error messages
2. Open browser developer tools (F12) and check the console
3. Verify all model files are present and accessible
4. Ensure all dependencies are properly installed
5. Test with a simple video first

📝 License

This system integrates with your existing codebase. Please ensure compliance with all underlying model licenses (YOLO, MIDAS, ViT-GPT2, etc.).