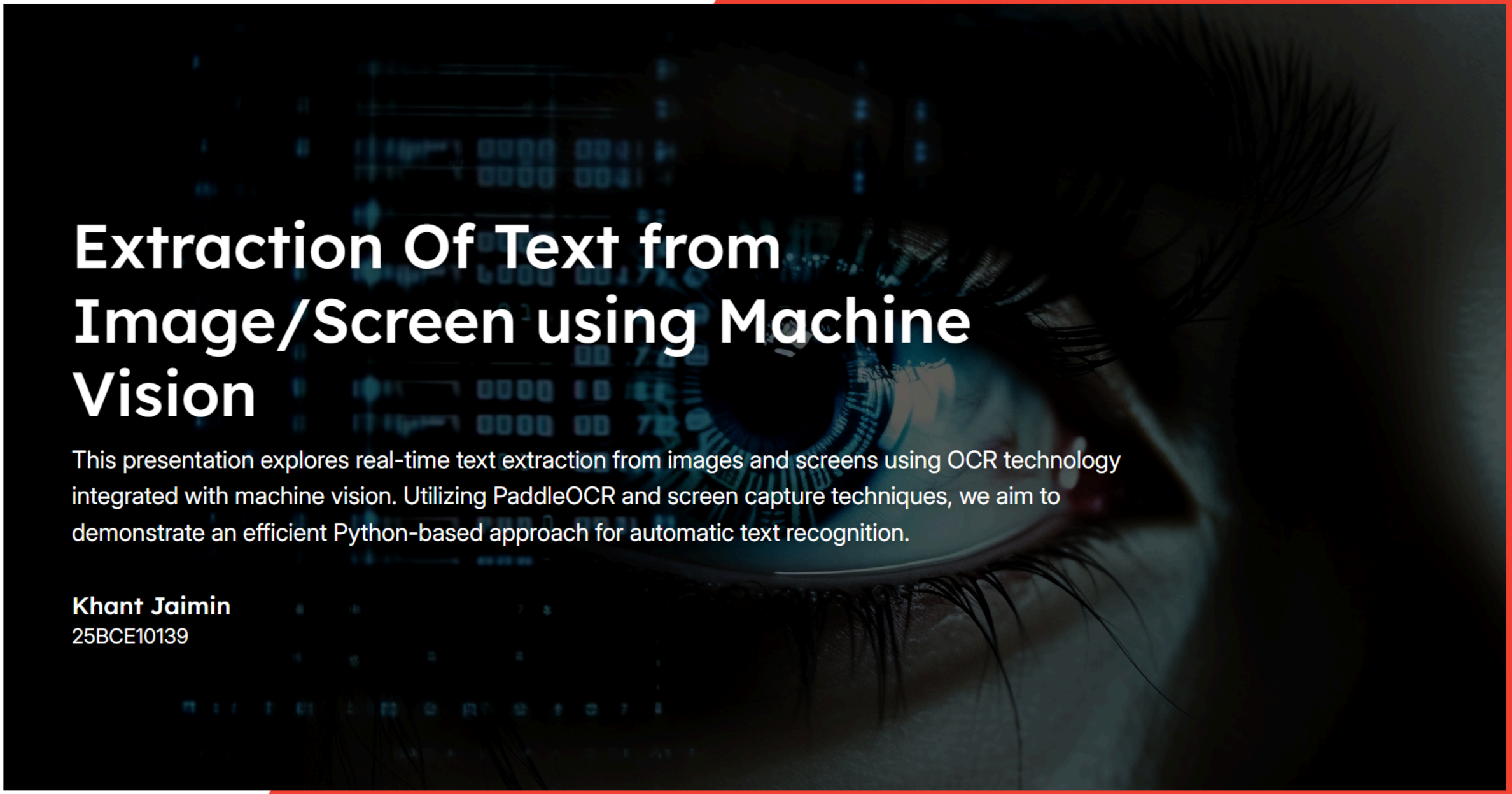


Extraction Of Text from Image/Screen using Machine Vision

This presentation explores real-time text extraction from images and screens using OCR technology integrated with machine vision. Utilizing PaddleOCR and screen capture techniques, we aim to demonstrate an efficient Python-based approach for automatic text recognition.

Khant Jaimin
25BCE10139

Problem: essential data trapped in images and scans; manual transcription is slow and error-prone

Solution: combine **OCR** and **machine vision** to detect and recognize text within images

Implementation: use pre-trained **PaddleOCR** models for rapid, out-of-the-box extraction

Benefits: supports **multilingual** input, full-screen captures, and improves accessibility of visual data

Impact: faster processing, fewer transcription errors, scalable text extraction workflow

Extract Text Faster with OCR + Machine Vision

Automate text detection and recognition from screenshots, images, and scans using pre-trained **PaddleOCR** for multilingual, full-screen extraction

Speed up text capture from images

Solve time-consuming, error-prone retyping by using screen capture + OCR with visual context

Problem: extracting text from non-editable sources (screenshots, scans) is slow and error-prone

01

Scale issue: manual retyping fails with large volumes or multiple images

02

Accuracy risk: human entry introduces transcription errors and inconsistencies

03

Goal: Python tool to capture screen, run OCR, and display extracted text with visual context

04

Outcome: faster, more accurate extraction and simplified user experience

05

Functional Requirements: OCR Capture → Recognition → Visualization

Clear workflow for screenshot capture, PP-OCRv5 detection, overlayed text, and simple user exit

Capture full screenshots with Pillow ImageGrab

- Grab full-screen image using Pillow's ImageGrab
- Ensure image passed intact to detection pipeline
- Support real-time or on-demand capture modes

Detection with PaddleOCR using PP-OCRv5 mobile models

- Use PaddleOCR PP-OCRv5 mobile models for speed
- Detect text regions and compute bounding boxes
- Recognize text strings from detected regions

Visualization overlay via OpenCV

- Draw bounding boxes to highlight detected text
- Overlay recognized text on the image
- Display processing time and recognized text log

User interaction simple window control

- Show processed image in a window
- Allow easy close via keypress
- Design for practical visualization and straightforward interaction

Non-functional Requirements: Performance, Accuracy, Usability

Targets, platform, reliability and future extensibility

Performance: ~0.8s processing per full-screen image with GPU acceleration for near real-time operation

Accuracy: PaddleOCR multilingual models supporting over 109 languages

Usability: single-command run with minimal configuration for simple operation

Platform: Windows, Python 3.10+, requires CUDA/cuDNN; portable and extensible

Reliability: graceful handling of empty or low-text screens

Extensibility: designed for future features like partial capture and continuous processing

System Architecture Overview

Modular OCR pipeline for real-time screen capture, detection, recognition, and visualization

Screen Capture

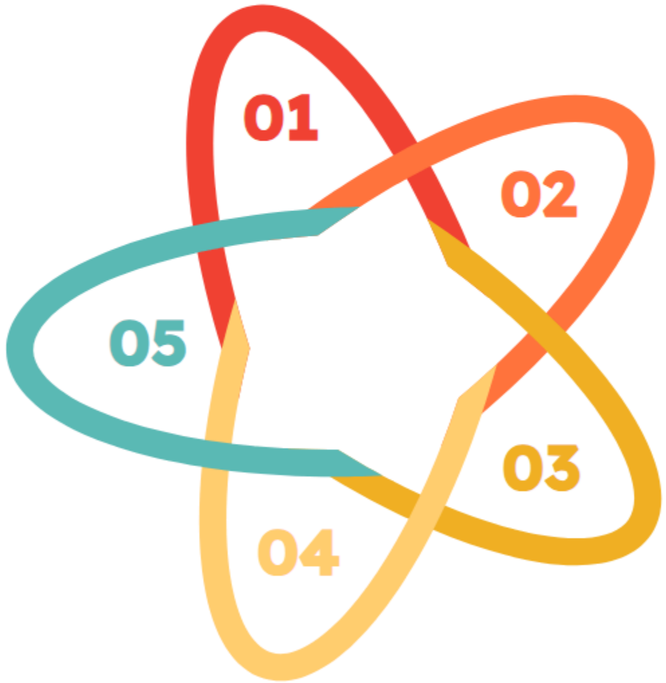
Capture screen frames using Pillow ImageGrab for input.

Timing & Output

Measure latency with timing utilities; show annotated window and optional text logs.

Visualization

Draw bounding boxes and overlays using **OpenCV** for annotated output.

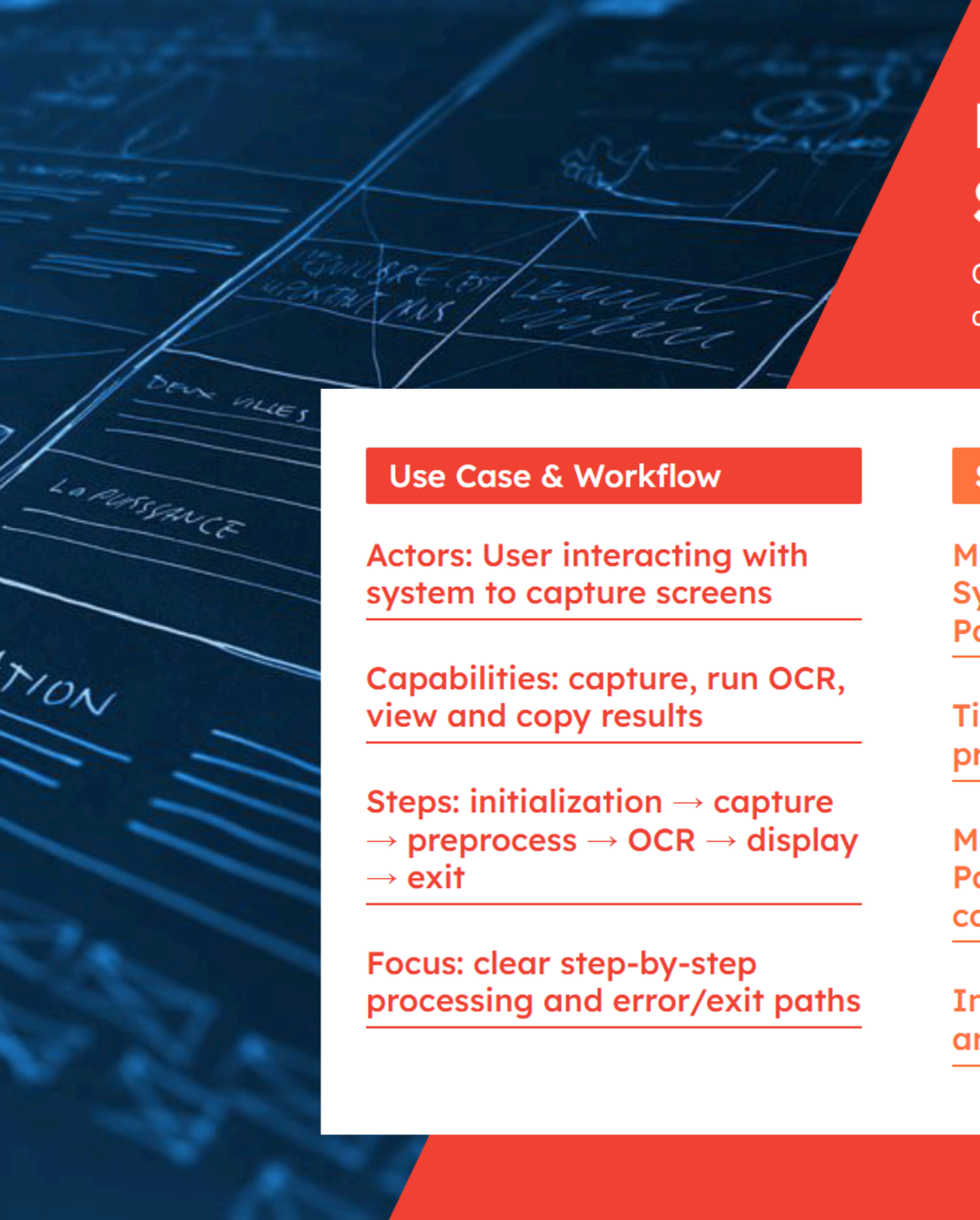


OCR Detection

Locate text regions using **PaddleOCR** detector (PP-OCRv5 mobile det).

OCR Recognition

Convert regions to text with **PaddleOCR** recognizer (PP-OCRv5 mobile rec).



Design Diagrams Overview — System Structure & Flows

Concise visual map of Use Case, Workflow, Sequence, Component, and ER diagrams for capture + OCR pipeline

Use Case & Workflow

Actors: User interacting with system to capture screens

Capabilities: capture, run OCR, view and copy results

Steps: initialization → capture → preprocess → OCR → display → exit

Focus: clear step-by-step processing and error/exit paths

Sequence & Component

Message flows between User, System, ImageGrab, PaddleOCR, OpenCV

Timing: request → capture → process → response

Modules: ImageGrab, PaddleOCR, OpenCV, UI, controller

Interactions: module interfaces and data flows

ER (future)

No DB currently; ER suggests schema for captures and OCR results

Entities: Capture, OCRResult, Metadata, User (if needed)

Design Decisions & Rationale

Why each component was chosen to balance accuracy, speed, compatibility, and usability

01 **PaddleOCR** — pre-trained, production-ready models with broad multilingual support; avoids building custom OCR networks

02 **PP-OCRv5 mobile** — mobile models tuned for a balance of accuracy and speed; suitable for GPU-powered environments

03 **ImageGrab** — simple, generic full-screen capture across application contexts

04 **OpenCV** — clear text visualization for debugging and user feedback

05 **Python 3.10+** — ensures compatibility with modern libraries and GPU acceleration frameworks

06 Summary — choices prioritize **production readiness**, **multilingual support**, **GPU performance**, and **developer simplicity**

Implementation Details & Key Components

Python 3.10+ OCR pipeline using PaddleOCR, OpenCV and optimized screen capture

TECHNOLOGIES: PYTHON 3.10+, **PADDLEOCR**, **PADDLE-GPU**, PILLOW, NUMPY, OPENCV

MAIN.PY FLOW: INITIALIZE PADDLEOCR WITH DETECTION & RECOGNITION MODELS; CAPTURE SCREEN → CONVERT IMAGE FORMATS → RUN OCR

OCR OUTPUTS: BOUNDING BOXES + TEXT STRINGS; OPENCV DRAWS OVERLAYS AND RENDERS RESULTS

DISPLAY & CONTROL: SHOW PROCESSED IMAGES; WAIT FOR USER INPUT TO CLOSE

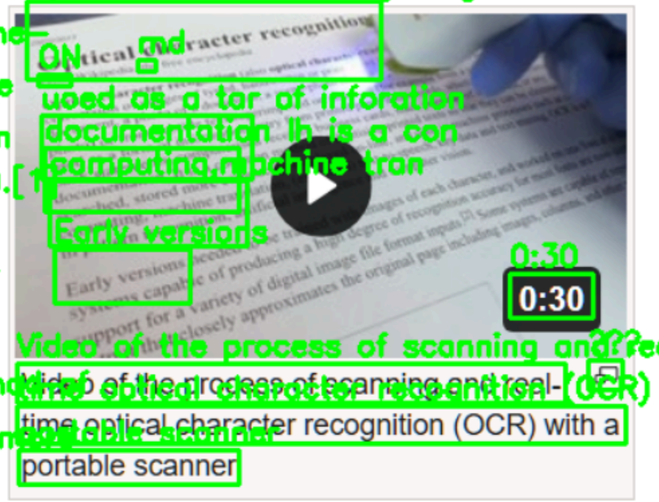
DESIGN FOCUS: MODULARITY AND PERFORMANCE VIA PADDLE-GPU AND FORMAT CONVERSION

OCR Results: Accurate Detection & Fast Processing

Screenshots validate detection of most on-screen text and ~0.8s full-screen processing

From Wikipedia, the free encyclopedia

Optical character recognition or **optical character reader** (**OCR**) is the electronic or mechanical conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene photo (for example the text on signs and billboards in a landscape photo) or from subtitle text superimposed on an image (for example: from a television broadcast).[1]



Video of the process of scanning and real-time optical character recognition (OCR) with a portable scanner

Widely used as a form of data entry from printed paper data records – whether passport documents, invoices, bank statements, computerized receipts, business cards, mail, printed data, or any suitable documentation – it is a common method of digitizing printed texts so that they can be electronically edited, searched, stored more compactly, displayed online, and used in machine processes such as cognitive computing, machine translation, (extracted) text-to-speech, key data and text mining. OCR is a field of research in pattern recognition, artificial intelligence and computer vision.

Early versions needed to be trained with images of each character, and worked on one font at a time. Advanced systems capable of producing a high degree of accuracy for most fonts are now common, and with support for a variety of image file format inputs.[2] Some systems are capable of reproducing formatted output that closely approximates the original page including images, columns, and other non-textual components.

History [edit]

History [edit]

See also: Timeline of optical character recognition

Early optical character recognition may be traced to technologies involving telegraphy and creating reading devices for the blind.[3] In 1914, Emanuel Goldberg developed a machine that read characters and converted them into standard telegraph code.[4] Concurrently, Edmund Fournier d'Albe developed the Optophone, a handheld scanner that when moved across a printed page, produced tones that corresponded to specific letters or characters.[5]

Testing Approach for Functional & Performance Validation

Validate OCR accuracy and speed across real-world screen scenarios

Functional: Verify recognized text accuracy across web pages, IDEs, and multilingual content

Performance: Measure OCR processing time across multiple resolutions

Edge Cases: Test low-contrast text, very small fonts, cluttered backgrounds, and screens without text

Robustness & Failure Modes: Ensure graceful failure and clear feedback when text can't be recognized

Outcome: Comprehensive testing to ensure **reliability** and **user satisfaction** in real-world scenarios

Technical Challenges During OCR Development

Root causes and technical implications for accuracy, performance, and integration

Configuring **PaddleOCR with GPU**: requires correct CUDA and cuDNN versions and driver compatibility to enable GPU acceleration

Windows build dependencies: **Visual C++ Build Tools** and system libs needed for compiling or installing native wheels

Performance for full-screen high-res: high memory and compute demand; need batching, resizing, or model optimization to maintain speed

Low-contrast / poor-quality text: degrades OCR accuracy; requires preprocessing (contrast enhancement, denoising) and robust models

Library integration: **Pillow, PaddleOCR, OpenCV** require careful data-format handoffs and ordered processing to avoid errors

Data flow and format mismatches: color channels, array types, and coordinate systems must be normalized between libraries

Mitigations summary: validate CUDA/cuDNN versions, install Visual C++ tools, optimize models/resizing, apply preprocessing, standardize data formats

Project Learnings: OCR & Machine Vision in Practice

Hands-on integration, GPU acceleration, Python tooling, and image-quality impacts on OCR accuracy

Integrated pipeline using pre-trained deep learning OCR and machine vision models

GPU acceleration is critical for real-time processing performance

Leverage **Python ecosystem**—libraries like Pillow and OpenCV enable rapid development

Image quality (clarity, contrast, resolution) directly affects OCR accuracy

Emphasize **input preparation** and system robustness to improve results

Future Enhancements for Expanded Capabilities

Planned features to improve capture, OCR, language support, storage, and accessibility

Region-based capture — allow user-selected area captures for precise extraction

Continuous real-time OCR — run OCR on video streams instead of single snapshots

Direct export — copy recognized text to clipboard or save to file

Multi-language support — automatic language detection and combined OCR passes

Database archival & search — store OCR history for retrieval and indexing

Simple GUI app — desktop application to broaden accessibility for non-technical users

Summary — scoped additions: region capture, live OCR, export, multilingual OCR, archival, and GUI

References & Technical Resources

Authoritative docs and repos for implementing and extending OCR and image processing

- **PaddleOCR GitHub** — source code, models, examples: https://github.com/PaddlePaddle/PaddleOCR
- **PaddlePaddle Docs** — framework guides and API reference: https://www.paddlepaddle.org.cn
- **Pillow (PIL) Docs** — image I/O and processing: https://pillow.readthedocs.io
- **OpenCV Python Docs** — computer vision APIs and tutorials: https://docs.opencv.org
- **Python Language Docs** — core language reference: https://docs.python.org
- Consult these resources for **building, configuring, and extending** OCR/image-processing applications