C.V. RAMAN GLOBAL UNIVERSITY

Bhubaneswar, Odisha

COMPUTER SCIENCE & ENGINEERING DEPARTMENT

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

TOPIC:

Medical Perception Processing using NLP and OCR

SUBMITTED BY:

SL. NO	NAME	REGD NO.	GROUP
1.	SANTTUN RAY	CL2025010601912359	4
2.	ADITYA MALLICK	CL2025010601915692	4
3.	SASMITA DASH	CL20250106018813119	4
4.	SUBHANGI PRIYADARSHINI	CL2025010601889981	4
5.	ANURAG JENA	CL2025010601958129	4

GUIDED BY:

ARIB NAWAL

ACKNOWLEDGMENT

We would like to express our deepest gratitude and sincere thanks to our Case Study project guide, Arib Nawal sir, whose cooperative guidance has been instrumental in the successful completion of our project on "**Medical Perception Processing using Nlp and Ocr**".We are truly grateful for them to guidance and mentorship throughout this journey.

DECLARATION:

We hereby declare that the Case Study project titled “Sign Language Detection System” submitted by us in partial fulfilment of the requirements for the 6th semester of Bachelor of Technology, was carried out under the supervision and guidance of Arib Nawal Sir.

CONTENT

INTRODUCTION:

This powerful document processing tool combines optical character recognition (OCR) with natural language processing (NLP) to transform images and PDFs into clean, analyzed text. It extracts content from documents, cleans OCR artifacts, corrects spelling and grammar issues, and provides detailed text analysis—all in an easy-to-use pipeline.

The tool features a comprehensive text processing workflow that begins with extraction using EasyOCR for images and PyMuPDF for PDFs. It then applies intelligent cleaning algorithms to fix common OCR errors, followed by spelling correction and grammar enhancement. The processed text undergoes readability analysis and named entity recognition to provide insights into text complexity and important content elements.

For users seeking additional improvements, the tool offers optional integration with Google's Gemini AI through a simple API key input. This feature allows for customized text enhancement based on user prompts. All processing results are neatly compiled into a downloadable PDF report that includes the processed text alongside detailed analysis, making it ideal for researchers, students, and professionals working with document digitization and text extraction tasks.

Setting up and using this tool is straightforward even for those with limited technical experience. After installing the required dependencies, users simply upload their document, optionally provide a Gemini API key if AI suggestions are desired, and the tool handles the rest—extracting text, performing multi-stage processing, and generating a comprehensive report. The interactive prompts guide users through each step, ensuring a smooth experience from document upload to final output.

PROBLEM STATEMENT

Healthcare professionals and institutions face significant challenges in efficiently processing and analyzing the vast amounts of unstructured medical documentation generated daily. Handwritten notes, scanned medical records, printed lab reports, and legacy documents contain critical patient information that remains inaccessible to digital analysis without proper extraction and processing. The manual conversion of these documents is time-consuming, error-prone, and diverts valuable healthcare resources away from patient care.

There is an urgent need for an automated system that can accurately perceive, extract, and process medical text from diverse document formats. By combining Optical Character Recognition (OCR) technology with advanced Natural Language Processing (NLP) techniques, we aim to develop a solution that can transform unstructured medical documents into standardized, analyzable text while preserving medical terminology integrity, identifying key clinical entities, and ensuring compliance with healthcare documentation standards.

This solution will address the critical gap between paper-based or image-based medical documentation and modern electronic health record systems, enabling more efficient clinical workflows, improved data accessibility for research and analysis, enhanced patient data integration, and ultimately better healthcare outcomes through more complete information availability at the point of care.

ABSTRACTION

This project presents an intelligent system that automates the extraction, correction, analysis, and reporting of textual data from medical documents such as scanned images and PDFs. Utilizing EasyOCR for Optical Character Recognition, the system efficiently retrieves text from unstructured visual formats. The extracted text undergoes extensive Natural Language Processing (NLP), including spelling correction, grammar refinement using spaCy, readability scoring via textstat, and named entity recognition to identify key medical terms, dates, and patient details.

The processed output is compiled into a well-formatted PDF report that includes both the cleaned text and its analytical insights. Additionally, integration with Google's Gemini API allows for context-aware AI suggestions to further improve clarity and usefulness of the extracted content. This system enhances the accessibility, interpretability, and digitization of healthcare records, providing significant value in clinical documentation, medical research, and health informatics workflows.

SOURCE CODE

Dependencies to Install:

```
!pip install easyocr
!pip install PyMuPDF
!pip install fpdf
!pip install openai
!pip install spellchecker
!pip install spacy
!python -m spacy download en_core_web_sm
!pip install pyspellchecker
!pip install textstat
!pip install nltk
!pip install --upgrade pip setuptools wheel
!pip install indexer
```

Main Code:

```
import easyocr
from fpdf import FPDF
import fitz  # PyMuPDF
from google.colab import files
import google.generativeai as genai
import re
import nltk
import spacy
from nltk.corpus import stopwords
from spellchecker import SpellChecker
from textstat import textstat

# Download necessary NLTK resources with explicit download path
print("📦 Downloading NLTK resources...")
nltk.download('punkt')  # Just punkt, not punkt_tab
nltk.download('stopwords')
nltk.download('wordnet')
print("✅ NLTK resources downloaded")

# Make sure we're using the right tokenizers
from nltk.tokenize import word_tokenize, sent_tokenize

# Load spaCy model
try:
    nlp = spacy.load('en_core_web_sm')
    print("✅ spaCy model loaded")
except Exception as e:
    print(f"⚠️ spaCy model error: {e}")
```

```
    print("Installing spaCy model...")
    import os
    os.system('python -m spacy download en_core_web_sm')
    try:
        nlp = spacy.load('en_core_web_sm')
        print("✅ spaCy model loaded after installation")
    except:
        print("❌ Could not load spaCy model")
        # Fallback option
        nlp = None

# Initialize spellchecker
spell = SpellChecker()

# --- TEXT EXTRACTION FUNCTIONS ---

def extract_text_from_image(image_path):
    """Extract text from an image using EasyOCR"""
    reader = easyocr.Reader(['en'], gpu=False)
    result = reader.readtext(image_path, detail=0, paragraph=True)
    return "\n".join(result)

def extract_text_from_pdf(pdf_path):
    """Extract text from a PDF using PyMuPDF"""
    doc = fitz.open(pdf_path)
    text = ""
    for page in doc:
        text += page.get_text("text") + "\n"
    return text

# --- NLP PROCESSING FUNCTIONS ---

import re

def clean_ocr_text(text):
    """Clean OCR-extracted text by fixing common character errors and formatting issues."""

    # --- STEP 1: Basic OCR format fixes ---
    text = re.sub(r'([A-Za-z])_([A-Za-z])', r'\1 \2', text)  # Underscores between letters → space
    text = re.sub(r'([a-z])([A-Z])', r'\1 \2', text)         # Lowercase followed by uppercase →
space

    # --- STEP 2: Common static replacements ---
    replacements = {
        'Fo': 'to',
        'Fhe': 'the',
        'Fhis': 'this',
        'Fhat': 'that',
        'Falking': 'talking',
        'Fearning': 'learning',
```

```
    'Fhings': 'things',
    'Talso': 'I also',
    'Fhink': 'think',
    'lfe': 'life',
    '&th': '8th',
    'Zgo': 'I go',
    '1 am': 'I am',

    # Additional likely OCR-based typos
    'Teh': 'The',
    'Ths': 'This',
    'Fere': 'There',
    'Fom': 'From',
    'Fime': 'Time',
    'Foday': 'Today',
    'Fey': 'They',
    'Tey': 'They',
    'Ferefore': 'Therefore',
    'Frue': 'True',
    'Fust': 'Just',
    'l': 'I',  # lowercase L often used for uppercase I
    'i m': 'I am',
    'ive': 'I have',
    'i ve': 'I have',
    'thw': 'the',
    'thid': 'this',
    'thar': 'that',
    'woukd': 'would',
    'coud': 'could',
    'shoukd': 'should',
    'Sth': '8th',
    'gth': '9th',
    'Oth': '10th'
}

for wrong, right in replacements.items():
    text = text.replace(wrong, right)

# --- STEP 3: Regex-based dynamic corrections ---
# Replace 'F' + vowel-starting word → 'Th'
text = re.sub(r'\bF(?=[aeiouAEIOU])', 'Th', text)

# Replace '1' at word beginning with 'I' (common OCR confusion)
text = re.sub(r'\b1(?=\w+)', 'I', text)

# Replace standalone lowercase l with uppercase I
text = re.sub(r'\bl\b', 'I', text)

# Fix weird ordinal numbers (e.g., &th)
text = re.sub(r'&th', '8th', text)
```

```
    # Fix "Z" at start of word, usually meant to be "I"
    text = re.sub(r'\bZ(?=\w+)', 'I', text)

    # --- STEP 4: Whitespace cleanup ---
    text = re.sub(r'\s+', ' ', text)
    text = re.sub(r'\n\s*\n', '\n\n', text)

    return text.strip()

def correct_spelling(text):
    """Correct spelling errors in the text"""
    # Simple tokenization using Python's split to avoid NLTK issues
    words = text.split()
    corrected_words = []

    for word in words:
        # Only check words with letters (not numbers or punctuation alone)
        if re.search('[a-zA-Z]', word):
            # Remove punctuation attached to the word for spell checking
            clean_word = re.sub(r'[^\w\s]', '', word)

            if clean_word and clean_word.lower() not in ['i', 'a', 'an', 'the', 'and', 'or', 'but', 'to', 'for',
'in', 'on', 'at', 'by']:
                misspelled = spell.unknown([clean_word])
                if misspelled:
                    # Get the most likely correction
                    correction = spell.correction(clean_word)
                    if correction:
                        # Replace just the word part, maintaining original punctuation
                        corrected_word = word.replace(clean_word, correction)
                        corrected_words.append(corrected_word)
                    else:
                        corrected_words.append(word)
                else:
                    corrected_words.append(word)
            else:
                corrected_words.append(word)
        else:
            corrected_words.append(word)

    return ' '.join(corrected_words)

def fix_grammar_with_spacy(text):
    """Use spaCy for basic grammar correction"""
    if nlp is None:
        # Fallback basic correction if spaCy is not available
        sentences = re.split(r'(?<=[.!?])\s+', text)
        corrected_sentences = []
```

```
        for sentence in sentences:
            sentence = sentence.strip()
            if sentence:
                # Capitalize first letter
                if sentence[0].isalpha() and not sentence[0].isupper():
                    sentence = sentence[0].upper() + sentence[1:]

                # Add period if missing ending punctuation
                if sentence[-1] not in ['.', '!', '?']:
                    sentence += '.'

                corrected_sentences.append(sentence)

        return ' '.join(corrected_sentences)

    try:
        doc = nlp(text)
        sentences = []

        for sent in doc.sents:
            # Convert to string and capitalize first letter
            sentence = sent.text.strip()
            if sentence:
                if sentence[0].isalpha() and not sentence[0].isupper():
                    sentence = sentence[0].upper() + sentence[1:]

                # Make sure sentence ends with punctuation
                if sentence[-1] not in ['.', '!', '?']:
                    sentence += '.'

                sentences.append(sentence)

        return ' '.join(sentences)
    except Exception as e:
        print(f"⚠️ Grammar correction error: {e}")
        return text  # Return original text if there's an error

def analyze_text_complexity(text):
    """Analyze readability of the text"""
    if len(text.strip()) == 0:
        return "No text to analyze"

    try:
        results = {
            "flesch_reading_ease": textstat.flesch_reading_ease(text),
            "flesch_kincaid_grade": textstat.flesch_kincaid_grade(text),
            "automated_readability_index": textstat.automated_readability_index(text)
        }

        analysis = f"Readability Analysis:\n"
```

```
        analysis += f"- Flesch Reading Ease: {results['flesch_reading_ease']:.1f}/100 "

        if results['flesch_reading_ease'] > 90:
            analysis += "(Very Easy to Read)\n"
        elif results['flesch_reading_ease'] > 80:
            analysis += "(Easy to Read)\n"
        elif results['flesch_reading_ease'] > 70:
            analysis += "(Fairly Easy to Read)\n"
        elif results['flesch_reading_ease'] > 60:
            analysis += "(Standard/Plain English)\n"
        elif results['flesch_reading_ease'] > 50:
            analysis += "(Fairly Difficult to Read)\n"
        elif results['flesch_reading_ease'] > 30:
            analysis += "(Difficult to Read)\n"
        else:
            analysis += "(Very Difficult to Read)\n"

        analysis += f"- Grade Level: {results['flesch_kincaid_grade']:.1f}\n"
        analysis += f"- Automated Readability Index:
{results['automated_readability_index']:.1f}"

        return analysis
    except Exception as e:
        print(f" ⚠️ Readability analysis error: {e}")
        return "Unable to analyze text complexity"

def extract_key_entities(text):
    """Extract named entities from the text using spaCy"""
    if nlp is None:
        return "Entity extraction not available (spaCy model not loaded)"

    try:
        doc = nlp(text)
        entities = {}

        for ent in doc.ents:
            entity_type = ent.label_
            if entity_type not in entities:
                entities[entity_type] = []
            if ent.text not in entities[entity_type]:
                entities[entity_type].append(ent.text)

        if not entities:
            return "No named entities found in the text"

        result = "Named Entities:\n"
        for entity_type, items in entities.items():
            result += f"- {entity_type}: {', '.join(items)}\n"

        return result
```

```
    except Exception as e:
        print(f"⚠️ Entity extraction error: {e}")
        return "Unable to extract entities"

def process_text_with_nlp(text):
    """Process text using various NLP techniques"""
    print("1. Cleaning OCR artifacts...")
    # Initial cleaning of OCR artifacts
    text = clean_ocr_text(text)

    print("2. Correcting spelling...")
    # Correct spelling
    text = correct_spelling(text)

    print("3. Fixing grammar...")
    # Fix basic grammar issues
    text = fix_grammar_with_spacy(text)

    print("4. Analyzing text complexity...")
    # Additional analysis
    complexity_analysis = analyze_text_complexity(text)

    print("5. Extracting entities...")
    entity_analysis = extract_key_entities(text)

    # Return processed text and analysis
    return {
        "processed_text": text,
        "complexity_analysis": complexity_analysis,
        "entity_analysis": entity_analysis
    }

# --- PDF GENERATION FUNCTION ---

def create_pdf_from_text(text, analysis, output_pdf):
    """Create a PDF from extracted text and analysis"""
    pdf = FPDF()
    pdf.add_page()
    pdf.set_auto_page_break(auto=True, margin=15)

    # Add original processed text
    pdf.set_font("Arial", 'B', size=14)
    pdf.cell(0, 10, "Processed Text", ln=True)
    pdf.set_font("Arial", size=12)
    pdf.multi_cell(0, 10, text)

    # Add analysis section
    pdf.add_page()
    pdf.set_font("Arial", 'B', size=14)
    pdf.cell(0, 10, "Text Analysis", ln=True)
```

```
    pdf.set_font("Arial", 'B', size=12)
    pdf.cell(0, 10, "Complexity Analysis", ln=True)
    pdf.set_font("Arial", size=11)
    pdf.multi_cell(0, 8, analysis["complexity_analysis"])

    pdf.set_font("Arial", 'B', size=12)
    pdf.cell(0, 10, "Entity Analysis", ln=True)
    pdf.set_font("Arial", size=11)
    pdf.multi_cell(0, 8, analysis["entity_analysis"])

    # Add AI suggestions if available
    if "ai_suggestions" in analysis:
        pdf.add_page()
        pdf.set_font("Arial", 'B', size=14)
        pdf.cell(0, 10, "AI Improvement Suggestions", ln=True)
        pdf.set_font("Arial", size=12)
        pdf.multi_cell(0, 10, analysis["ai_suggestions"])

    pdf.output(output_pdf)

# --- GEMINI IMPROVEMENT SUGGESTIONS ---
def suggest_improvements_with_gemini(extracted_text, api_key):
    """
    Prompt the user to enter a custom instruction, then use Gemini API
    to generate suggestions or improvements on the extracted text.
    """
    # Ask user to enter their prompt
    print("\n 📥 Please enter your instruction or prompt for Gemini (e.g., 'Fix grammar and
summarize'):")
    user_prompt = input(" 📝 Your Prompt: ").strip()

    genai.configure(api_key=api_key)

    try:
        model = genai.GenerativeModel('gemini-2.0-flash')

        # Limit extracted text to avoid overload
        trimmed_text = extracted_text[:1500]

        # Create the full prompt
        full_prompt = f"""{user_prompt}

OCR-extracted text:
{trimmed_text}
"""

        # Generate content
        response = model.generate_content(full_prompt)
        return response.text
```

```
    except Exception as e:
        print(f"⚠️ Gemini API Error: {e}")
        return f"Unable to generate suggestions using Gemini. Error: {str(e)}"

# --- MAIN PROCESSING FUNCTION ---

def process_document(file_path, file_type, output_pdf, api_key=None):
    """Extract text, process with NLP, and generate a PDF from it"""
    # Extract raw text from document
    if file_type == 'image':
        extracted_text = extract_text_from_image(file_path)
    elif file_type == 'pdf':
        extracted_text = extract_text_from_pdf(file_path)
    else:
        extracted_text = ""
        print("❌ Unsupported file type")
        return None

    print("\n=== 📝 Raw Extracted Text ===")
    print(extracted_text[:500] + "..." if len(extracted_text) > 500 else extracted_text)

    # Process text with NLP
    print("\n=== 🧠 Processing with NLP... ===")
    nlp_results = process_text_with_nlp(extracted_text)

    processed_text = nlp_results["processed_text"]
    print("\n=== 📝 NLP Processed Text ===")
    print(processed_text[:500] + "..." if len(processed_text) > 500 else processed_text)

    # Add complexity and entity analysis
    print("\n=== 📊 Text Analysis ===")
    print(nlp_results["complexity_analysis"])
    print("\n" + nlp_results["entity_analysis"])

    analysis_results = nlp_results

    # Get AI suggestions if API key is provided
    if api_key:
        print("\n=== 🤖 Requesting AI suggestions... ===")
        suggestions = suggest_improvements_with_gemini(processed_text, api_key)
        analysis_results["ai_suggestions"] = suggestions
        print("\n=== 💡 AI Suggestions ===")
        print(suggestions)

    # Create and save PDF
    create_pdf_from_text(processed_text, analysis_results, output_pdf)

    return {
```

```
        "raw_text": extracted_text,
        "processed_text": processed_text,
        "analysis": analysis_results
    }

# --- INTERACTIVE EXECUTION ---

# Install required packages if they're not already installed
try:
    import pkg_resources
    required_packages = ['spacy', 'textstat', 'pyspellchecker']
    installed = {pkg.key for pkg in pkg_resources.working_set}
    missing = [pkg for pkg in required_packages if pkg.lower() not in installed]

    if missing:
        print(f"📦 Installing missing packages: {', '.join(missing)}")
        import os
        os.system(f"pip install {' '.join(missing)}")
        print("✅ Packages installed successfully")
except Exception as e:
    print(f"⚠️ Package check error: {e}")

# Input Gemini API Key (optional)
api_key = input("🔑 Enter your Gemini API key (press Enter to skip AI suggestions):
").strip()

if api_key:
    print("✅ API key received.")
else:
    print("⏩ Skipping AI suggestions.")

print("\n📤 Upload a file (image or PDF) to process:")
uploaded = files.upload()

if uploaded:
    filename = list(uploaded.keys())[0]
    file_extension = filename.split('.')[-1].lower()

    if file_extension in ['jpg', 'jpeg', 'png', 'bmp']:
        file_type = 'image'
    elif file_extension in ['pdf']:
        file_type = 'pdf'
    else:
        print("❌ Unsupported file format.")
        file_type = None

    if file_type:
        print(f"\n📄 Processing {filename}...")
        output_pdf = "processed_document_with_nlp.pdf"
```

```
    # Process the document with all NLP features
    results = process_document(filename, file_type, output_pdf, api_key if api_key else
None)

    if results:
        print("\n⬇️ Downloading PDF...")
        files.download(output_pdf)
        print("\n✅ Processing complete!")
    else:
        print("\n❌ Processing failed.")
```

OUTPUT

📦 Downloading NLTK resources...

✅ NLTK resources downloaded

[nltk_data] Downloading package punkt to /root/nltk_data...

[nltk_data] Package punkt is already up-to-date!

[nltk_data] Downloading package stopwords to /root/nltk_data...

[nltk_data] Package stopwords is already up-to-date!

[nltk_data] Downloading package wordnet to /root/nltk_data...

[nltk_data] Package wordnet is already up-to-date!

✅ spaCy model loaded

🔑 Enter your Gemini API key (press Enter to skip AI suggestions): **********

✅ API key received.

📤 Upload a file (image or PDF) to process:

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

WARNING:easyocr.easyocr:Using CPU. Note: This module is much faster with a GPU.

Saving p3.jpg to p3.jpg

📄 Processing p3.jpg...

=== 📝 Raw Extracted Text ===

DEA# GB 05455616 LIC # 976269 MEDICAL CENTRE New York; NY 91743, USA 824 14u Street NAME Jola Smitl AGE 34 ADDRESS 162 Example St, NT DATE 09-11-12 Betaloc I0O~3 -144L Bid Dorzolamizvm I0 ~J +45 Bid Cinetizine 50 ~J 2 +4L, TID Oxprelol 50~a t45 QD e 1 Dc. Steve_JoLason signature OLABEL 1 REFILL 0(12 3 4 5 PRN WTXSNY PRES7OO 1

=== 🧠 Processing with NLP... ===

1. Cleaning OCR artifacts...

2. Correcting spelling...

3. Fixing grammar...

4. Analyzing text complexity...

5. Extracting entities...

=== 📝 NLP Processed Text ===

Dead# go 05455616 lie # 976269 MEDICAL center New York; my 91743, us 824 you Street NAME join smite AGE 34 ADDRESS. I example St, it DATE 09-I1-I2 betaine I0O~3. -i'll. Bid DorzoIamizvm i ~J +45 Bid cimetidine 50 ~J 2 +al, did OxpreIoI 50~a the ad e 1 Dc. Steve Jo jason signature label 1 REFILL 0(I2 3 4 5 pin WTXSNY preston 1.

=== 📊 Text Analysis ===

Readability Analysis:

- Flesch Reading Ease: 80.6/100 (Easy to Read)
- Grade Level: 6.0
- Automated Readability Index: 6.0

Named Entities:

- DATE: 05455616, 91743, AGE 34, 50
- MONEY: 976269
- ORG: MEDICAL, WTXSNY
- GPE: New York
- CARDINAL: 824, 50, 1, 3
- PERSON: Bid DorzoIamizvm, Steve Jo jason

=== 🤖 Requesting AI suggestions... ===

📩 Please enter your instruction or prompt for Gemini (e.g., 'Fix grammar and summarize'):

📝 Your Prompt: i want all that preception in structured way and after that give me the list of the medicines along with the suitable dosages

=== 💡 AI Suggestions ===

Okay, let's break down the information from the OCR text and then provide a structured summary and medication list with dosages.

Structured Information:

* **Patient Information:**
  * Name: Steve Jo Jason
  * Age: 34
  * Address: Example St, New York, US
  * Other Identifiers (likely): Dead# go 05455616 lie # 976269, my 91743, us 824 you Street NAME join smite, WTXSNY preston 1

* **Date:** 09-11-12 (Likely September 11, 2012; Date format depends on the regional system. I'm assuming MM-DD-YY format.)

* **Prescription Medications:**
  * Betaine: 100mg, frequency unclear (likely BID, or twice a day)
  * Dorzolamide: 1mg + 45 (unit unclear; likely drops if it's for eyes), BID (twice a day)
  * Cimetidine: 50mg + 2 (unit unclear), BID (twice a day)
  * Oxprenolol: 50mg, route of administration unclear, AD (frequency unclear; likely once a day)

* **Refills:** 1 refill

Medication List and Dosages (with Clarifications and Important Considerations):

* **Betaine 100mg** (Frequency Unclear but likely BID):
  * **Dosage:** 100mg
  * **Frequency:** BID (Twice a day - *presumed, verify with the prescribing doctor or pharmacist*)
  * **Route of Administration:** Oral (Implied, but confirm)
  * **Notes:** Betaine is often used to treat homocystinuria.

* **Dorzolamide 1mg + 45 (Unit Unclear) BID:**
  * **Dosage:** 1mg + 45(Unit unclear. Likely drops for eyes)
  * **Frequency:** BID (Twice a day)
  * **Route of Administration:** If for eyes then use as eye drops.
  * **Notes:** Dorzolamide is a carbonic anhydrase inhibitor, often used to treat glaucoma by reducing intraocular pressure. Verify dosage and route with the prescribing doctor or pharmacist.

* **Cimetidine 50mg + 2 (Unit Unclear) BID:**
  * **Dosage:** 50mg + 2(Unit unclear. Likely drops for eyes)
  * **Frequency:** BID (Twice a day)
  * **Route of Administration:** Oral.
  * **Notes:** Cimetidine is a histamine H2 receptor antagonist, used to reduce stomach acid production. Verify dosage and route with the prescribing doctor or pharmacist.

* **Oxprenolol 50mg AD:**
  * **Dosage:** 50mg
  * **Frequency:** AD (Unclear; likely once a day, *presumed, verify with the prescribing doctor or pharmacist*)
  * **Route of Administration:** Oral (Implied, but confirm)
  * **Notes:** Oxprenolol is a non-selective beta-blocker, used to treat hypertension, angina, and arrhythmias.

IMPORTANT DISCLAIMERS:

* **Accuracy:** This information is based solely on the provided OCR text. OCR can be inaccurate, and critical details may be missing or misinterpreted.
* **Completeness:** The prescription information may be incomplete. Important information like route of administration (oral, topical, etc.) and specific instructions (e.g., "take with food") are often not fully captured.
* **Verification is Essential:** **ABSOLUTELY DO NOT rely on this information for self-treatment.** You MUST verify the medications, dosages, frequencies, and routes of administration with the prescribing doctor, dispensing pharmacist, or a qualified healthcare professional.

*   **Dosage Unit Discrepancies:** Several dosages have unclear units or unclear abbreviations. This MUST be clarified by a medical professional.
*   **Date:** Date may be misinterpreted as the format is unclear.
*   **This is NOT medical advice.** This is simply a structured presentation of the information provided in the text. Consult with a healthcare professional for any medical concerns or before taking any medications.
*   **'AD' Frequency:** The abbreviation "AD" is ambiguous. It most likely means 'once a day'.

⬇️ Downloading PDF...

✅ Processing complete!

ADVANTAGES

Automation of Manual Work:

Reduces the need for manual transcription and interpretation of handwritten or printed medical documents.

Robust NLP Processing:

Applies multiple natural language processing techniques (cleaning, spell correction, grammar fixes, entity recognition) to produce clear and structured text.

Readability Analysis:

Provides readability scores and grade-level metrics, helping assess how easy the content is to understand—especially useful in patient education material.

AI Integration for Insights:

Enhances the output with Gemini-powered suggestions, such as summaries, grammar improvements, or rewriting instructions based on user input.

Custom PDF Report Generation:

Automatically generates organized PDF reports combining the processed text, analytical insights, and optional AI suggestions.

Flexible File Support:

Supports both image files (JPG, PNG, etc.) and PDFs, making it applicable to a wide range of healthcare documentation formats.

Offline Compatibility:

Most functionalities work without requiring constant internet access, except for AI-powered enhancements.

DISADVANTAGES

OCR Accuracy Depends on Image Quality:

Poor lighting, low resolution, or noisy backgrounds in scanned images can reduce OCR performance and result in incorrect text extraction.

Limited Handwriting Support:

While EasyOCR supports handwritten text, its accuracy is lower compared to printed text, especially with complex or cursive handwriting.

Spell Correction May Introduce Errors:

The spelling and grammar correction modules might misinterpret medical terms as incorrect and alter them, potentially changing the intended meaning.

Heavy Dependency on NLP Libraries:

The system relies on several NLP libraries and models that may increase complexity and require regular updates to maintain compatibility.

Performance on Large Files:

Processing large documents or PDFs with many pages can be slow and memory-intensive, especially on low-end systems.

AI Features Require Internet Access:

Integration with Gemini or other generative AI tools requires an internet connection, limiting functionality in offline environments.

Lack of Contextual Understanding:

The system might struggle to fully understand the medical context or relationships between extracted entities without deeper domain-specific models.

CONCLUSION

The developed system successfully bridges the gap between raw medical documents and actionable digital insights by combining OCR with advanced NLP techniques. By automating the extraction, cleaning, and analysis of unstructured text from images and PDFs, it significantly reduces manual effort and minimizes human error in interpreting clinical documents. The integration of readability metrics and AI-driven suggestions further enhances the clarity and usability of the output. Overall, this solution not only improves the efficiency of healthcare data processing but also lays a strong foundation for scalable applications in medical record digitization, patient data analysis, and intelligent health documentation systems

REFERENCES

https://github.com/madmaze/pytesseract

https://github.com/JaidedAI/EasyOCR

https://github.com/mindee/doctr

https://github.com/medspacy/medspacy

https://spacy.io/

https://portal.dbmi.hms.harvard.edu/projects/n2c2-nlp/