COMPARATIVE ANALYSIS OF OCR SERVICES IN AUTOMATED TEXT EXTRACTION FOR EFFICIENT SPORTS CARD INVENTORY MANAGEMENT

Terrence Brent Hernandez

A Capstone Project Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Department of Computer Science
Department of Information Systems and Operations Management

University of North Carolina Wilmington

2024

Approved By

Advisory Committee

Dr. Brittany Morago	Dr. Curry Guinn
Dr. Minoo Modaresnezhad	Dr. Lucas Layman, Chair

Accepted By

Dean, Graduate School

Table of Contents

List of Figures **4**

List of Tables **4**

1 Introduction **5**
1.1 Background 5
1.2 Motivation 5
1.3 Objective 9

2 Literature Review **10**
2.1 Image Processing 10
2.2 Optical Character Recognition (OCR) 10
2.2.1 AI-Based Text Recognition 10
2.3 Text Extraction Metrics 11
2.3.1 Levenshtein Distance 11
2.3.2 Character Accuracy 12
2.3.3 Character Error 13
2.3.4 Word Error 13
2.4 Analysis of OCR as a Service 14
2.4.1 Feature extraction 15
2.4.2 Google Vision: v.3.7.2 16
2.4.3 ChatGPT 4.0: v.gpt-4-0125-preview 18
2.4.4 Usage in Python 18

3 Methodology **24**
3.1 OCR Data Extraction Overview 24
3.1.1 Ground Truth Textual Data 24
3.1.2 Datasets for Text Extraction 24
3.1.3 Evaluation Metrics 27
3.2 Experiment 27
3.2.1 OCR API, Data Extraction, Metrics 27

4 Implementation **29**
4.1 Dataset Construction 29
4.2 Ground Truth Collection 30
4.3 Python Implementation 31
4.3.1 Implementation of Google Vision v.3.7.2 31
4.3.2 API Response Processing 32
4.3.3 Attribute Extraction 33
4.3.4 Implementation of ChatGPT 4.0 v.gpt-4-0125-preview 36
4.4 Metrics Generation 41

5 Evaluation **44**
5.1 Accuracy Evaluation 44
5.1.1 Time Evaluation 47
5.1.2 Cost Analysis 48

6 Conclusion **50**
6.1 Future Works 50

Abstract

This capstone project proposes a performance-based comparative analysis of Image Processing and Optical Character Recognition (OCR) services in the application of sports card attribute identification for efficient inventory. The analysis focuses on two web-based image processing services, Google Vision AI and ChatGPT 4.0. The performance analysis serves to determine the most accurate and overall higher-performing OCR service in the domain of attribute extraction from sports cards. Each service will be given image samples from a controlled source and an experimental source. This will provide variation concerning card features and image quality. These images will be used to generate OCR service responses that will be used to derive performance metrics. These metrics will represent the capabilities and accuracy of text detection extraction for each service to highlight the service with superior performance.

List of Figures

1 Manually Labeled Card (Sample 1) . . . 7
2 Manually Labeled Card (Sample 2) . . . 8
3 OCR Processing Overview [27] . . . 15
4 Google Vision Application Implementation [5] . . . 16
5 Google Vision API in a Python Environment . . . 19
6 Vision API Response Pre-Processing . . . 19
7 Google Vision Block Bound Image . . . 20
8 Returned Block Bound Data . . . 20
9 Google Vision Paragraph Bound Image . . . 21
10 Returned Paragraph Bound Data . . . 21
11 Google Vision Paragraph Bound Image . . . 22
12 Returned Word Bound Data . . . 22
13 ChatGPT API implemented in a Python Environment . . . 23
14 ChatGPT API Primary Console Response . . . 23
15 ChatGPT API Secondary Console Response . . . 24
16 Image samples in variability matrix layout . . . 26
17 Database ERD Structure . . . 31
18 ChatGPT module GUI . . . 31
19 Algorithm for processing images with Google Vision . . . 32
20 Vision API Response Pre-Processing . . . 33
21 Google API Response Post-Processing . . . 33
22 Regionalized Card Image . . . 35
23 Google Vision Output Post Processing . . . 36
24 Vision Module GUI . . . 37
25 ChatGPT Input Prompt . . . 37
26 ChatGPT Output Pre-processing . . . 38
27 ChatGPT String Text Evaluation Pseudocode . . . 38
28 ChatGPT Output Post-processing . . . 39
29 Google Vision Output Post Processing . . . 39
30 Algorithm for Processing Images with ChatGPT API. . . . 40
31 Example Metrics Computation . . . 44
32 Cost Comparison of Google Vision and ChatGPT . . . 48

List of Tables

1 Manually recorded attributes of Card Sample (1) . . . 7
2 Manually recorded attributes of Card Sample (2) . . . 8
3 Levenshtein function overview [26] . . . 12
4 Levenshtein Function Matrix Output . . . 12
5 Derived formula for C_A [22] . . . 13
6 Derived formula for C_{ER} [22] . . . 13
7 Derived formula for W_{ER} [22] . . . 14
8 Google Vision API Feature Options [6] . . . 17
9 Ground Truth and Simulated ORC Data for Card Sample (2) . . . 27
10 Simulated Metrics Generation for Card Sample (2) . . . 28
11 Project Phases . . . 29
12 Dataset Image Dimension Averages . . . 30
13 Height Percentage Difference . . . 30
14 Width Percentage Difference . . . 30
15 Google Vision Input Descriptions . . . 32
16 Google Dataset A - Card 1_dj01-a-01 Metrics . . . 42
17 ChatGPT Dataset A - Card 29_dj01-a-01 Metrics . . . 43
18 OCR Metrics Summary . . . 46
19 Platform Run Time Averages . . . 47

20 Service Cost Projections 49

1 Introduction

1.1 Background

The sports trading card hobby has undergone a transformative evolution in recent years, marked by a growing interest in sports cards featuring emerging athletes on the cusp of sports stardom. Simultaneously, vintage and iconic cards have made a resurgence, sparking enthusiasm among collectors and investors. Contributing to this growing interest is the presence of online marketplaces such as eBay, Fanatics, and Beckett. According to Market Statesville Group (MSG), the sports card market was valued at 14.2 billion dollars as of 2021 and is projected to surge to a staggering 117.3 billion dollars by 2030 [15]. This renewed interest and market growth presents a challenge for collectors and a lucrative opportunity for developers to find an efficient method for cataloging extensive card collections.

In application development, significant progress has been made in the domains of algorithmic image processing and text data analysis. These advancements in computer vision and deep learning have led to exceptional precision when identifying and categorizing objects and extracting details within images. The impact of these technological breakthroughs spans many fields, from medical diagnostics to autonomous vehicles and now potentially the lucrative sports trading card arena.

1.2 Motivation

The value of sports cards hinges on three primary factors: market trends among consumers, the performance and popularity of the player, and the inherent attributes of the card itself. Collectors have limited influence and control over the first two factors, like how stock market traders observe changes. Tracking market trends and player performance is relatively easy. Collectors track these factors to financially profit from their collections by selling the cards of the players whose market trend and or performance surge, coupled with highly sought-after card attributes such as player, rarity, and print type. The challenge for card hobbyists is knowing what specific cards with specific attributes exist in their collections at any given time.

Hobbyist collectors often use spreadsheets, documents, or pen-and-paper applications to create their personal collection inventory manually. Creating these inventories is time-consuming, tedious, inefficient, and subject to human error with no automation. This makes the card inventory process almost impossible when collections extend to thousands of cards. Several scanning apps provide solutions for identifying, matching, and valuing cards. These apps are typically custom-built solutions that rely on image matching to source the card attribute information from an already established dataset.

While some apps today employ Computer Vision, its application is typically seen in card grading, identification, matching, and valuations. However, the use of Computer Vision in the domain of sports cards is still new and significantly limited in documentation. This project's approach differs from other applications, focusing on accurately extracting textual card data for the future application of automated inventory generation without the need to source the card data to fulfill matching and valuation.

Some popular apps include Beckett Collect, CardBase, Center Stage and CollX [18]. While utilizing image recognition and data extraction technologies, these applications offer limited insights into the specific image processing technologies they employ. Their primary emphasis is on assessing and determining the value of cards rather than using textual extraction for inventory generation. The reason for this is that numerous apps often derive and extrapolate card data from sources beyond the actual card. This approach results in drawbacks such as misidentification of cards that resemble each other, confusion between cards with substantial design differences, inaccurate card valuations, and potential bias against notably older cards.

A solution lies in a shift of focus from data extrapolation for data outsourcing and valuations to focus on attribute extraction to provide accurate automation with a reliable Optical Character Recognition (OCR) service. Optical Character Recognition (OCR) is a process that encompasses a

wide range of image processing components and techniques. Together, these components carry out the detection and conversion of printed or handwritten text from images into machine-usable text. OCR technology has been the subject of research and development since the late 1800's [2]. The origins of OCR are rooted back to the 1860s with the development of the Optophone. This early OCR tool was designed to identify and convert text to audio, providing a reading aid for individuals with visual impairments [1]. Today, it utilizes advanced algorithms and machine learning techniques for automated character recognition from digital images and has become a readily available commodity. Today, OCR is offered as a service by numerous web service providers (Google, AWS, Azure, OpenAI) and is commonly used for digitizing printed documents, extracting text from images, and automating data entry processes.

OCR as a service is described by Amazon Web Services (AWS) as technology that provides a solution for "extracting text in images that cannot be processed by word processing software the same way as text documents" [30]. Using OCR as a service can extend image recognition technology to the sports card market. It offers an automated solution to the labor-intensive task of manually inspecting individual sports cards to extract textual data and database entry. The relevance of OCR for this project is to detect and extract the attributes that make the sports card itself valuable.

The card attributes of interest are the player's name, year, type, and brand, directly determining the card's value. The card details section is also an attribute of interest. This section can contain information that will further assist in identifying the card and other sought-after attributes. The details section is described as the large body of text on the back of the card. This section may contain a player's performance biography or player statistics if the card depicts a veteran player. The main challenge is the lack of knowledge of OCR technology in the sports card domain. This means that there is sparse literature available to assist in determining which OCR service performs the best in this area. This information is significant not only in assisting sports card collectors but also in providing an opportunity to reveal new insights into the strengths and weaknesses of OCR technology relative to its service provider.

Google Vision and ChatGPT 4.0 Vision primarily focus their OCR capabilities on broader computer vision tasks and do not specialize in image matching. This presents a critical issue in obtaining the sports card type with OCR services. Card type is an attribute that significantly influences card value. Card types, such as "parallels," exhibit distinct characteristics like color, orientation, and detail variations, making them exclusive and more valuable than their base counterparts. Unlike text-based attributes, OCR is not capable of extracting card types, necessitating manual inspection. Utilizing image recognition, distinct visual features associated with each card type can be identified. These unique characteristics can be identified, captured, and employed in image-matching processes to determine the card types accurately.

Using image recognition and feature extraction, a systematic approach can be employed to match the input images with reference images from eBay listings obtained using the textual information from OCR. Using these distinctive visual elements, a comparison process can cross-reference these features against the reference images to facilitate accurate identification of the card types. Furthermore, information can be gathered from eBay listings of matched images, including details on card types. This serves as valuable input that contributes to the refinement and enhancement of the identification process without reliance on a specific training model. This method ensures continual improvement in accurately recognizing and matching card types based on visual attributes observed in eBay listings.

Figure 1: Manually Labeled Card (Sample 1)

Card Sample (1) Summary		
Attributes	**Values**	**Label**
Manufacture	Panini America	1
Brand	Chronicles Donruss Optic	2
Type	Base, Rookie	3
Number	18	4
Year	2023	5
Player	Jaylin Hayatt	6
Team	Tennessee Volunteers	7
Details	Until 2022, Tennessee did not have a Biletnikoff Award winner in its history. Then, Hyatt checked that box. The junior put up nearly 19 yards a catch with a school-record 15 touchdown grabs. His five scores against Alabama also set a school record to highlight.	8

Table 1: Manually recorded attributes of Card Sample (1)

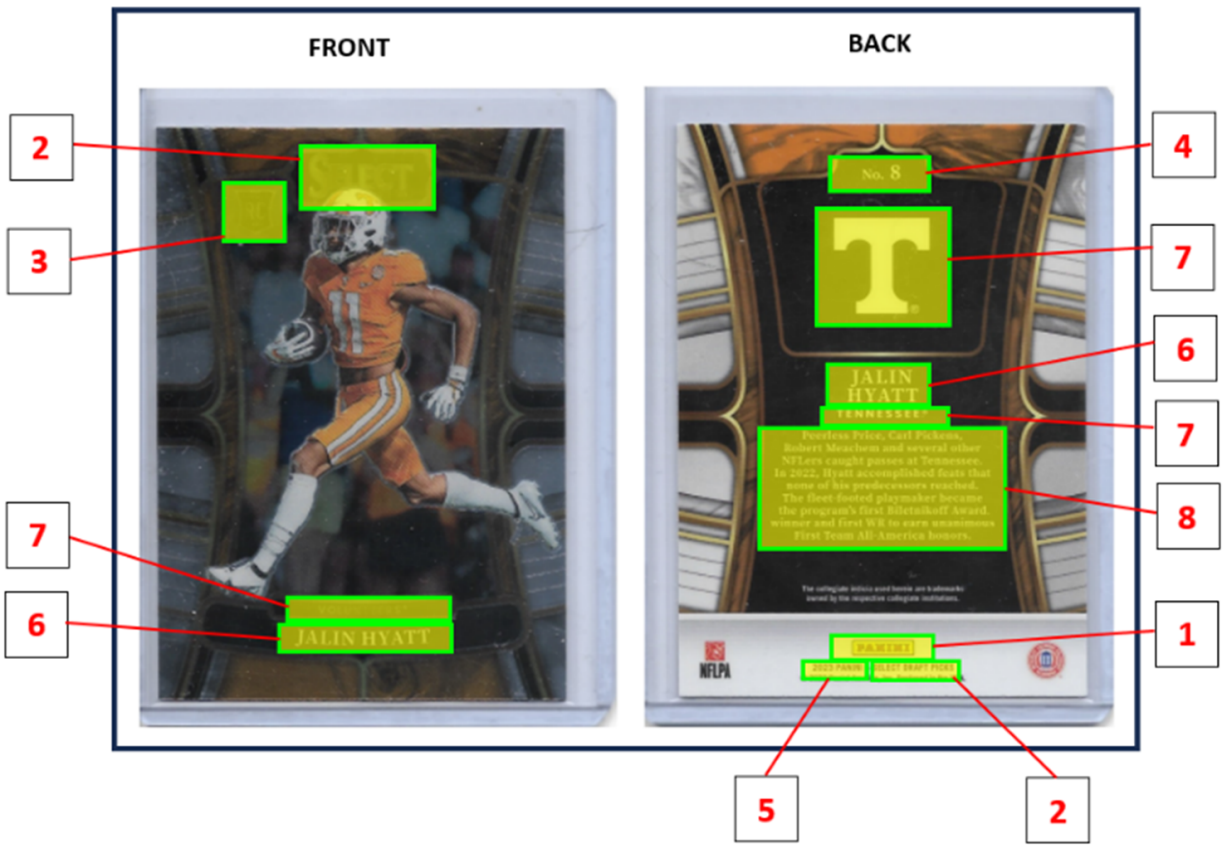

Figure 2: Manually Labeled Card (Sample 2)

Card Sample (2) Summary		
Attributes	**Values**	**Label**
Manufacture	Panini America	1
Brand	Select Draft Picks	2
Type	Base Rookie	3
Number	8	4
Year	2023	5
Player	Jaylin Hayatt	6
Team	Tennessee Volunteers	7
Details	Peerless Price, Carl Pickens, Robert Meachem and several other NFLers caught passes at Tennessee. In 2022, Hyatt accomplished feats that none of his predecessors reached. The fleet-footed playmaker became the programs first Biletnikoff Award, winners and first WR to earn unanimous First Team All-American honors.	8

Table 2: Manually recorded attributes of Card Sample (2)

1.3 Objective

The remainder of this paper is organized as follows: the introductory section elaborates on this capstone project's background, motivation, and objective. Chapter 2 presents the findings from exploring literature and research on algorithmic image processing, textual data analysis, sports card collecting, and market dynamics. Chapter 3 outlines the methodology adopted for the implementations to be developed and designed for evaluating the selected OCR services. Chapter 4 comprehensively records the implementations used to access and measure the selected OCR services. Chapter 5 is the analysis and assessment of the metrics obtained. Chapter 6 covers unsolved questions and/or obstacles, future work recommendations, and the project's conclusion.

2 Literature Review

This section establishes the knowledge base for the methodologies and technologies relevant to this capstone project. It conducts a thorough review encompassing academic and practical research on image processing, textual data analysis, and computer vision in the context of OCR. The primary goal of this chapter is to discern fundamental insights, obstacles, and innovations relevant to this project in its respective domains.

2.1 Image Processing

Image Processing encompasses a broad spectrum of techniques to modify input images and yield processed images as output. Image processing in computer vision is described as "the process of transforming an image into a digital form and performing certain operations to get useful information from it" [29]. The techniques are each designed to address specific challenges in working with images as inputs. This section will focus on phases and techniques directly correlated with deep learning and computer vision. Preparing input images for utilization within a computer vision system involves a sequence of fundamental phases. These phases are Image Acquisition, Image Enhancement, Segmentation, Representation and Description, and Recognition [29].

Image acquisition is the capturing of images from various sources such as digital cameras, scanners, or other imaging devices [29]. Low-quality image factors such as motion blur, noise, distortions, lighting conditions, device resolution, sensor sensitivity, and lens quality can cause difficulties in recognition, information extraction, data misinterpretation, and decreased accuracy within the system [23]. Image enhancement is the process of applying alteration techniques to an image to improve the visual quality and emphasize features of interest. These techniques include adjustments to noise, brightness, contrast, and sharpness to enhance the image's overall quality and the visibility of specific details [14]. Image segmentation involves breaking down a complex image into more manageable segments. This is achieved by utilizing algorithms to assess and group pixels according to criteria such as color, intensity, and texture [11]. Recognition is the process of labeling or classifying objects within the image based on the information derived from the representation and description [29].

2.2 Optical Character Recognition (OCR)

OCR is widely used in diverse domains such as business, government, finance, and healthcare. It is instrumental in tasks involving text extraction, translation, document digitization, data entry automation, and various image processing applications [24]. The use of OCR technology is well-documented across these domains.
The process involves the use of image processing algorithms to recognize patterns and shapes that correspond to characters in the input images. OCR systems analyze the shapes of individual characters, their spatial relationships, and contextual information to interpret and convert the images into machine-encoded text. The OCR process in a computer vision system is executed through a series of sequential steps. These key steps include Image Acquisition, Pre-Processing, Segmentation, Feature Extraction, Classification/Recognition, Post-Processing, and Output [2].

2.2.1 AI-Based Text Recognition

Traditional OCR techniques primarily depend on rule-based algorithms and heuristic-driven approaches. These traditional methods include Template-Based OCR, Feature Extraction-Based OCR, and Statistical Model-Based OCR [17]. While these traditional methods have demonstrated high effectiveness in text recognition, they are subject to various limitations, including variance in lighting, sharpness, rotation scale, and character deformation [17].

Integrating AI into traditional OCR has led to enhancements in domains where conventional OCR methods possess limitations. Integrating Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and other deep learning techniques offer feature detection, extraction, and prediction capabilities that aid in achieving higher OCR accuracy [17]. CNNs are deep learning models used

for image processing tasks such as image segmentation and feature extraction, while RNNs' are models designed to process sequential data in Natural Language Processing (NLP), sentiment analysis, and speech analysis [7]. The use of these deep learning models enhances OCR capabilities by capturing and learning patterns in sequential text data and image features used to isolate individual text characters.

Large Language Models (LLMs) are deep learning models trained using large amounts of textual data to facilitate the understanding and generation of human-like language from digitized data [28]. LLMs can enhance CNNs and RNNs by processing sequential textual data and generating human-like language using statistical relationships between sequential words [16]. LLMs contribute to RNN and CNN capabilities by their ability to detect language patterns. This pattern recognition allows RNNs and CNNs to maintain language context and semantic structure, further aiding in OCR accuracy improvement.

In AI-based Text recognition, traditional OCR methods prioritize image processing and feature detection, CNNs specialize in recognizing patterns of image features to detect characters, and RNNs specialize in capturing sequential textual data. LLMs contribute to this system by providing additional language semantic structure and context. This contribution improves text recognition by refining captured textual data and aids in handling character-related challenges such as variations in lighting, sharpness, rotation, and font types [16].

2.3 Text Extraction Metrics

Text extraction metrics in OCR are used to quantify the accuracy and performance of text extraction by the OCR service. For this evaluation, these metrics will include Character Accuracy (C_A) (Table 5), Character Error Rate (C_{ER}) (Table 6), and Word Error Rate (W_{ER}) (Table 7). It is worth noting that C_A and C_{ER} are found to be complements of one another due to their use of the Levenshtein Distance [26] value in their computations, the sum of their percentages is 100%. This means that given the set of characters that is C_A, its complement would be all the characters of the ground truth string that are incorrect, yielding the set that is C_{ER}. Computationally, this means that finding C_{ER}, the C_A percentage can be derived by subtracting the percentage C_{ER} from 100. These metrics will be applied to each of the following card attributes: player name, year, brand, and description.

2.3.1 Levenshtein Distance

The Levenshtein distance algorithm ($\mathbf{lev}_{a,b}(i, j)$) detailed in Table 3 is used to determine the edit distance between two given strings [26]. The piecewise function quantifies the difference between two strings by measuring the minimum number of single-character edits (insertions, deletions, or substitutions) required to transform one string into another. Named after the Soviet mathematician Vladimir Levenshtein, this distance metric provides a numerical value representing the degree of difference between two string sequences [26]. The lower the edit score, the more similar the strings are, indicating fewer edits required. It finds applications in various fields, such as spell-checking, DNA sequencing, and natural language processing, where understanding the similarity or dissimilarity between textual sequences is essential.

Insertions (I) pertain to the insertion of missing characters, **Substitutions (S)** pertain to the replacement of incorrect characters, and **Deletions (D)** pertain to the removal of characters that do not belong [26]. The provided piece-wise function calculates the number of operations needed to transform the string into the ground truth string. This metric of dissimilarity can also be used to calculate the "number of errors" that exist in a given string when compared to a ground truth string. The metrics that are used in this evaluation are based on the Levenshtein Distance function and use its output as a means to obtain other metrics such as **Character Accuracy (C_A)**, **Character Error Rate (C_{ER})**, and **Word Error Rate (W_{ER})**.

Levenshtein Distance - $\mathbf{lev_{a,b}(i, j)}$

Function

$$lev_{\mathrm{a,b}}(i,j) = \begin{cases} max(i,j) & \text{if} \quad min(i,j) = 0, \\ min \begin{cases} lev_{\mathrm{a,b}}(i-1,j) + 1 \\ lev_{\mathrm{a,b}}(i,j-1) + 1 \\ lev_{\mathrm{a,b}}(i-1,j-1) + 1_{a_i \neq b_j)} \end{cases} & \text{otherwise} \end{cases}$$

Variables

- a = Ground Truth string
- b = OCR output string
- i = terminal character position of string (a)
- j = terminal character position of string (b)

Table 3: Levenshtein function overview [26]

A matrix is used to describe the output of the Levenshtein function and yield the difference score given two strings. An example show Table 4, string (a) is the ground truth string (KAT) and string (b) is the OCR output string (COP). When comparing the characters of the strings, the piece-wise functions are used to compute the value of each matrix cell. The last lower right cell of the matrix is the found Levenshtein Distance of strings (a) and (b). This score is used in the subsequent computations to derive the metrics **Character Accuracy ($\mathbf{C_A}$)**, **Character Error Rate ($\mathbf{C_{ER}}$)**, and **Word Error Rate ($\mathbf{W_{ER}}$)**.

Levenshtein Matrix			
$_{\mathbf{string}}\mathbf{Character}_{\mathbf{index}}$	$_{\mathbf{b}}\mathbf{C}_{\mathbf{j=1}}$	$_{\mathbf{b}}\mathbf{O}_{\mathbf{j=2}}$	$_{\mathbf{b}}\mathbf{P}_{\mathbf{j=3}}$
$_{\mathbf{a}}\mathbf{K}_{\mathbf{i=1}}$	1	2	3
$_{\mathbf{a}}\mathbf{A}_{\mathbf{i=2}}$	2	1	2
$_{\mathbf{a}}\mathbf{T}_{\mathbf{i=3}}$	3	2	3

Table 4: Levenshtein Function Matrix Output

2.3.2 Character Accuracy

Character Accuracy (C_A) represents the accuracy with which the OCR service correctly identifies and recognizes individual characters [22]. The calculation for C_A begins with determining the number of characters in the image that make up the ground truth string. (C_M) is the number of sequence character matches of the ground truth string found in the OCR string. The number of found character matches is divided by the number of ground truth characters. The resulting quotient is multiplied by 100 to derive a percentage. This percentage signifies the proportion of characters accurately identified by the OCR service. A high C_A percentage indicates high accuracy in individual character recognition by the OCR service. Below is the formula implemented to derive the C_A metric.

Character Accuracy (C_A)	
Formula	**Variables**
$C_A = \frac{C_M}{C_T} \times 100$	• C_A = Character Accuracy • C_M = No. of character matches • C_T = No. of true characters

Table 5: Derived formula for C_A [22]

2.3.3 Character Error

Character Error (C_{ER}) is a metric representing the percentage of character-level errors in text recognition service [22]. It considers three types of operations: insertions, deletions, and substitutions. Each represents an error in character recognition. The formula for C_{ER} begins by calculating the total number of edit operations required to transform the recognized characters into ground truth characters. This sum is divided by the total number of ground truth characters. A lower C_{ER} percentage indicates a higher accuracy in recognizing individual characters by the OCR system. Below is the formula implemented to derive the C_{ER} metric.

Character Error Rate (C_{ER})	
Formula	**Variables**
$C_{ER} = \frac{I + S + D}{C_T} \times 100$	• C_{ER} = Character Error Rate • I = No. of Insertions • S = No. of Substitutions • D = No. of Deletions • C_T = No. of true characters

Table 6: Derived formula for C_{ER} [22]

2.3.4 Word Error

Word Error is a metric used to evaluate the accuracy of text recognition systems by measuring the percentage of word-level errors in the recognized text [22]. W_{ER} quantifies the difference between the recognized text and the ground truth text in terms of full words [25]. The calculation involves counting the total number of insertions, deletions, and substitutions required to transform the recognized text into the ground truth text. This sum is divided by the total number of words in the ground truth text. The resulting value is represented as a percentage of the total errors in the recognized text relative to the total number of words in the ground truth text. A lower WER percentage indicates a higher accuracy in recognizing full words by the OCR system. Below is the formula implemented to derive the W_{ER} metric.

Word Error Rate (W_{ER})	
Formula	**Variables**
$W_{ER} = \frac{I + S + D}{W_T} \times 100$	• W_{ER} = Word Error Rate • I = No. of Insertions • S = No. of Substitutions • D = No. of Deletions • W_T = No. of true words

Table 7: Derived formula for W_{ER} [22]

2.4 Analysis of OCR as a Service

Online AI and OCR services are gaining popularity today due to their ability to provide accessible and cost-effective solutions for automating various industry-related tasks. AI cloud computing services at affordable costs allow organizations the opportunity to optimize and automate industry services at lower costs compared to standard operating costs. Integrating these services provides businesses and organizations with advanced services into their processes, workflows, and operations. Performance comparison knowledge and literature between these three services is available but limited. While many comparisons have been conducted, virtually no literature exists pertaining to the performance comparison of online OCR services in the application of sports cards.

"A Comparison of Public Cloud Computer Vision Services" is a publication that details a comparative performance analysis of several OCR services including Google Vision, AWS, and Azure. This analysis conducted a trade-off assessment between the services regarding accuracy, performance, and speed [4]. The methods of evaluation for the mentioned publication were carried out by accessing the Application Programming Interface (API) of each service using a Python environment. The testing focused on distinctive OCR services using a pre-labeled dataset [4].

This comparative analysis publication provides an opportunity to expand on assessing differing OCR-providing platforms by conducting a similar comparison but in a different domain. The findings of this project contribute to these works by its application in the domain of sports cards without using a pre-existing sports card dataset. Using a fixed set of images to test each service, the publication concluded that Google Vision was the most accurate, Amazon AWS was the most cost-effective per image, and MS Azure was the best in terms of speed [4]. These findings provide a general foundation of OCR service comparison, which this project can expand upon by incorporating ChatGPT 4.0. The selection of Google Vision and ChatGPT 4.0 for this evaluation derives from the prominence and prevalence of these organizations in the AI computing industry. These platforms are known for their extensive infrastructure, accessible services, and capabilities, making them standard choices for businesses and developers looking to implement cloud-based AI services [3].

Due to the scarcity of literature about the implementations used to build computer vision systems such as Google Vision, an overview of how OCR systematically processes images is required. Each part of the multistage process is designed to convert images containing textual data into machine-usable data. The first relevant stage depicted in Figure 3 is **Preprocessing**. In this stage, image enhancement takes place in the form of noise reduction, along with brightness and contrast alterations. During the **Segmentation** phase, the now enhanced image is taken from a complex image and broken down into defined regions, such as object regions, text regions, and even character regions.

2.4.1 Feature extraction

Feature extraction is the phase in which features from the original image during the segmentation phase are used in the next phase. Classification and Recognition is where the extracted features and patterns are classified or labeled using machine learning or deep learning algorithms for OCR. This is where, based on the features provided, predictions about the contents of the segmented regions are made. **Post-processing** is the phase where further OCR application is done to detect and correct potential errors and format the output data into a more usable format. At this stage, the input image is considered completely processed.

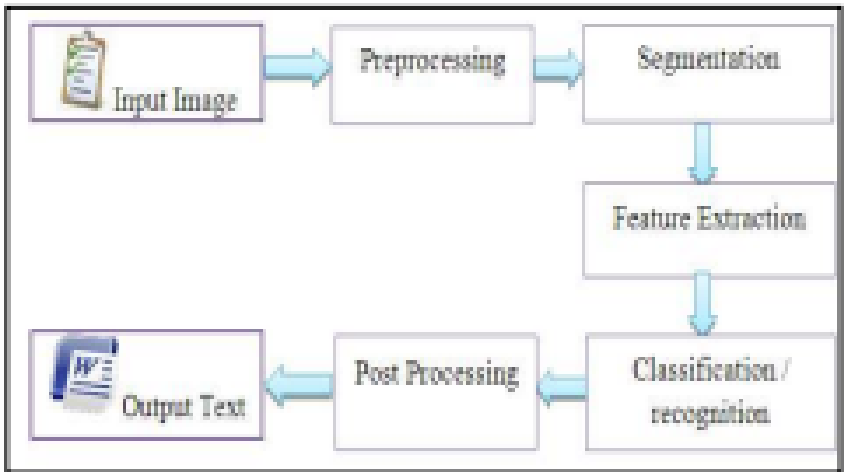

Figure 3: OCR Processing Overview [27]

2.4.2 Google Vision: v.3.7.2

Google Vision is a component of Google Cloud's machine learning and AI tooling parent suite. It provides various options for image processing, generation, and object detection. This particular API allows the integration of advanced image processing capabilities into local developer applications. This provides a given application with an extended set of tools to process and/or interpret image attributes, detect and decipher text, and recognize various objects.

Google Vision's image processing capabilities include optical character recognition and feature detection such as image labeling, face detection, landmark detection, and content tagging [5]. This API provides various processing options focused on extracting and understanding features and the context of interest within images. This allows for the identification, categorization, and detection of various types of image content and material. The wide range of functionalities Google Vision provides makes it a versatile tool and a viable option for image analysis at an affordable expense. The ease with which Google Vision can be integrated into various applications is visualized in Figure (4), as provided by the Google Vision documentation site.

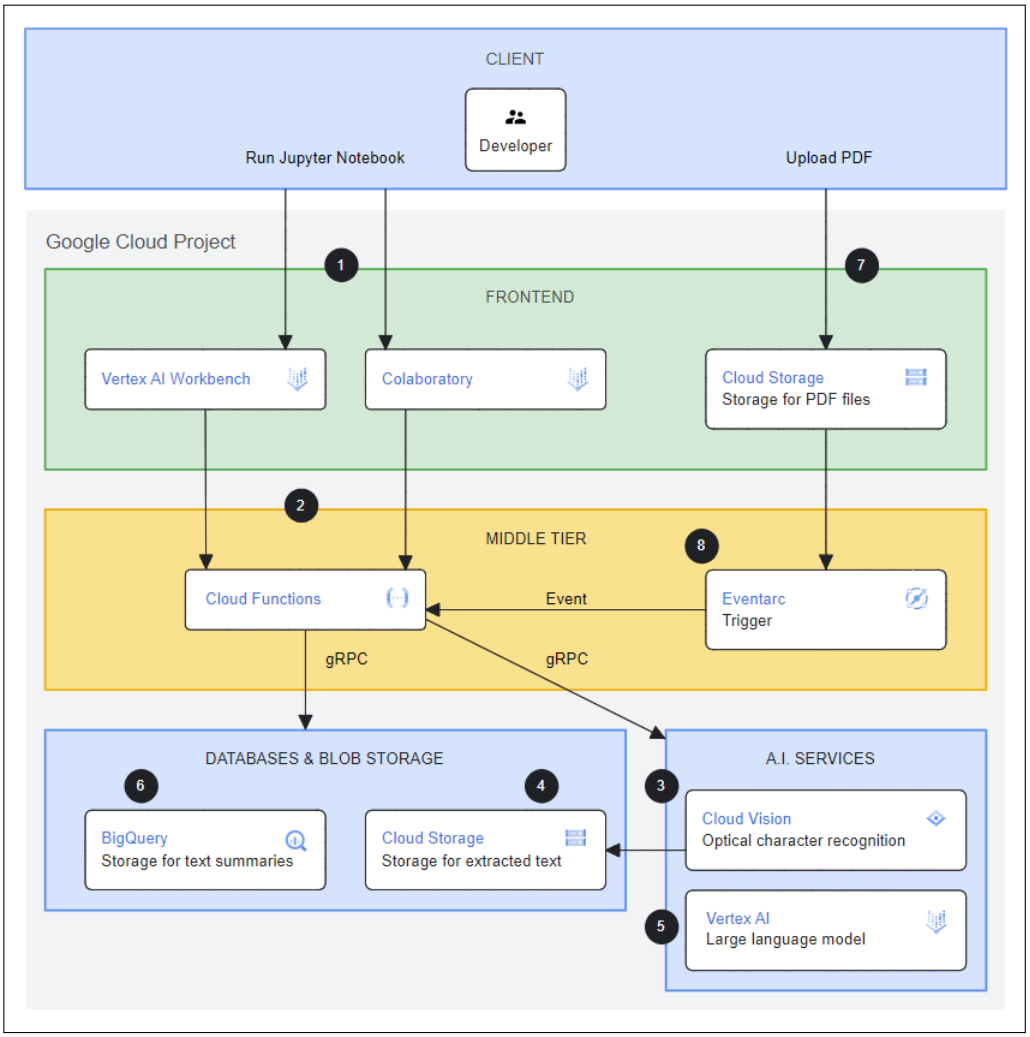

Figure 4: Google Vision Application Implementation [5]

Google Vision's API includes several feature options that provide a broad amount of functionality for application integration. These feature options demonstrate Google Vision's versatility in the domain of image processing. It offers a range of capabilities to detect faces, landmarks, logos, and multiple objects within a singular image [6]. In the domain of text detection and extraction, its features possess optical character recognition (OCR) functionalities that can differentiate between different types of text (sparse, dense) in images and documents. Google Vision can determine image properties to identify dominant colors, label images based on their content, and ensure safe content with its SafeSearch detection feature [6]. It also can correlate identified objects in images to broader topics like news events or celebrities through its Web Detection feature. These capabilities are detailed as features in Table (8), as provided by Google Vision Pricing documentation.

Vision API Feature Options	
Feature	**Description**
CROP_HINTS	Determine suggested vertices for a crop region on an image.
DOCUMENT_TEXT_DETECTION	Perform OCR on dense text images, such as documents (PDF/TIFF), and images with handwriting. TEXT_DETECTION can be used for sparse text images. Takes precedence when both DOCUMENT_TEXT_DETECTION and TEXT_DETECTION are present.
FACE_DETECTION	Detect faces within the image.
IMAGE_PROPERTIES	Compute a set of image properties, such as the image's dominant colors.
LABEL_DETECTION	Add labels based on image content.
LANDMARK_DETECTION	Detect geographic landmarks within the image.
LOGO_DETECTION	Detect company logos within the image.
OBJECT_LOCALIZATION	Detect and extract multiple objects in an image.
SAFE_SEARCH_DETECTION	Run SafeSearch to detect potentially unsafe or undesirable content.
TEXT_DETECTION	Perform Optical Character Recognition (OCR) on text within the image. Text detection is optimized for areas of sparse text within a larger image. If the image is a document (PDF/TIFF), has dense text, or contains handwriting, use DOCUMENT$_{T}EXT_{D}ETECTIONinstead.$
WEB_DETECTION	Detect topical entities such as news, events, or celebrities within the image, and find similar images on the web using the power of Google Image Search.

Table 8: Google Vision API Feature Options [6]

This platform's viability for this project is demonstrated by its capabilities analysis, coupled with the availability of its documentation. Implementing Google Vision's text extraction functionality within a Python module will allow experimentation and metrics derivation to gauge its performance against the secondary platform, ChatGPT 4.0. A basic implementation of Google Vision in a Python environment is demonstrated in Figure 5.

2.4.3 ChatGPT 4.0: v.gpt-4-0125-preview

ChatGPT 4.0: v.gpt-4-0125-preview is an API and artificial intelligence model developed and offered by the parent organization OpenAI [12]. Version gpt-4-0125-preview is one of the latest iterations of the Generative Pre-trained Transformer (GPT) models applicable for application integration. This API allows users to integrate natural language processing capabilities into their applications. This provides the applications with abilities such as text generation, conversation, summarization, translation, and image processing to some extent.

In the domain of image processing, this API and model allow for the uploading of images for analysis. Unlike Google Vision, this API lacks comparable user control over functionality. Users can upload multiple images through an API request and an attached prompt. A prompt is a text-based set of instructions or requests for service over the image by the API. The API's responses, however, are exclusively text-based. This means that while the API can deliver meaningful information and context about the images—similar to the insights provided by Google Vision—all outputs from the API are in textual form. An example of its implementation and production in a Python environment is demonstrated in Figure 5.

2.4.4 Usage in Python

The implementations of the Google Vision (Figure 5) and ChatGPT APIs (Figure 13) in a Python environment demonstrate that both platforms are easily integrated with minimal effort into a Python module. The advantages and limitations become evident upon their preliminary implementation via response outputs. Both API responses provide adequate information but require further processing to be usable.

Figure 6 illustrates that Google Vision offers an exhaustive data set, including extracted text and the corresponding vertices of defined bounding boxes. This data encompasses all the data of each API feature available to users under the text detection vision feature. Specifically, the JSON response from Google Vision includes the vertices that form bounding boxes on the blocks of text detected in the image. Textual data is identified at the Block, Paragraph, and Word levels within the supplied image, with the textual data existing within the bounding box defined by its corresponding vertices. The **Block** level refers to the largest grouping of text within an image, **Paragraph** is the structural unit of text representing an ordered sequence of words within a block, the **Word** is the smallest unit of text that isolates each word within a paragraph and is represented as an array of Symbols [8].

```python
from google.cloud import vision
import io
import os

def google_vision_request(image_path):
    client = vision.ImageAnnotatorClient()
    with io.open(image_path, 'rb') as image_file:
        content = image_file.read()

    image = vision.Image(content=content)
    response = client.document_text_detection(image=image)
    annotated_response = response.full_text_annotation.pages

    return annotated_response

def main():
    os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = '
       client_file_vision_ai_demo.json'

    image_file = '1_dj01-a-15-front.jpg'
    google_response = google_vision_request(image_file)
    print(google_response)

main()
```

Figure 5: Google Vision API in a Python Environment


```
{"front": {"textAnnotations": [{"locale": "en", "description": "nu\nE\
   nMOSAIC\nM\nCO\n8\nDANIEL JONES\nNEW YORK GIANTS", "boundingPoly":
   {"vertices": [{"x": 163, "y": 104}, {"x": 2020, "y": 104}, {"x":
   2020, "y": 2895}, {"x": 163, "y": 2895}]}}, {"description": "nu", "
   boundingPoly": {"vertices": [{"x": 240, "y": 2474}, {"x": 450, "y":
    2474}, {"x": 450, "y": 2583}, {"x": 240, "y": 2583}]}}, {"
   description": "E", "boundingPoly": {"vertices": [{"x": 252, "y":
   2635}, {"x": 242, "y": 2452}, {"x": 453, "y": 2441}, {"x": 463, "y"
   : 2624}]}}, {"description": "MOSAIC", "boundingPoly": {"vertices":
   [{"x": 206, "y": 111}, {"x": 390, "y": 107}, {"x": 391, "y": 133},
   {"x": 207, "y": 137}]}}, {"description": "M", "boundingPoly": {"
   vertices": [{"x": 165, "y": 177}, {"x": 428, "y": 164}, {"x": 439,
   "y": 394}, {"x": 176, "y": 407}]}}, {"description": "CO", "
   boundingPoly": {"vertices": [{"x": 797, "y": 585}, {"x": 916, "y":
   582}, {"x": 917, "y": 620}, {"x": 798, "y": 623}]}}
```


Figure 6: Vision API Response Pre-Processing

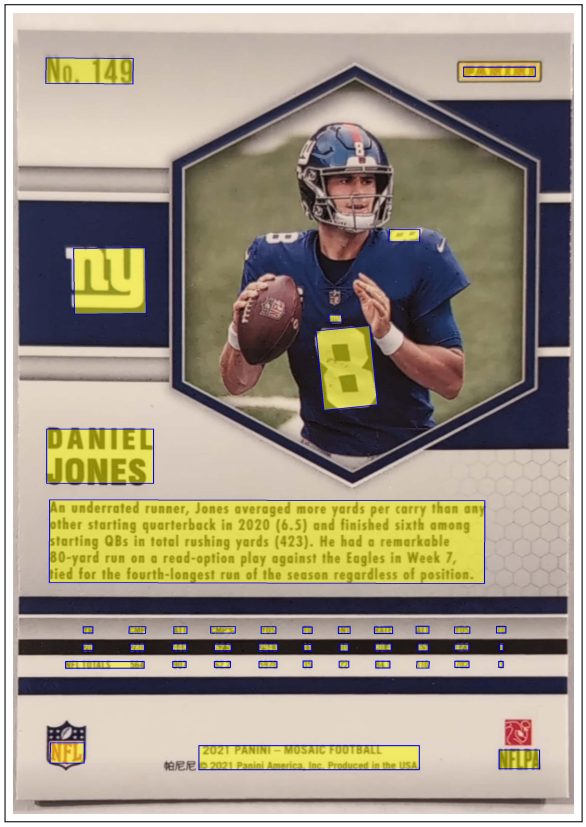

Figure 7: Google Vision Block Bound Image

```
30_dj01-a-01-back.jpg
['No. 149', 34320, [(127, 105), (457, 105), (457, 209), (127, 209)]]
['PAKIKI', 12150, [(1682, 124), (1952, 123), (1952, 168), (1682, 169)
    ]]
['nu', 219776, [(203, 855), (475, 856), (474, 1012), (202, 1011)]]
['E', 220968, [(222, 1058), (222, 836), (486, 837), (486, 1059)]]
['8', 71364, [(1132, 1161), (1360, 1149), (1375, 1433), (1147, 1445)]]
['B', 5546, [(1402, 765), (1520, 763), (1521, 810), (1403, 812)]]
['DANIEL JONES', 666246, [(116, 1543), (522, 1544), (521, 1758), (115,
     1757)]]
['An underrated runner , Jones averaged more yards per carry than any
    other starting quarterback in 2020 ( 6.5 ) and finished sixth among
     starting QBs in total rushing yards ( 423 ) . He had a remarkable
    80 - yard run on a read - option play against the Eagles in Week 7
    , tied for the fourth - longest run of the season regardless of
    position .', 3330608, [(129, 1820), (1777, 1822), (1777, 2152),
    (129, 2150)]]
```

Figure 8: Returned Block Bound Data

Figure 9: Google Vision Paragraph Bound Image

```
30_dj01-a-01-back.jpg
['No. 149', 34320, [(127, 105), (457, 105), (457, 209), (127, 209)]]
['PAKIKI', 12150, [(1682, 124), (1952, 123), (1952, 168), (1682, 169)
    ]]
['nu', 219776, [(203, 855), (475, 856), (474, 1012), (202, 1011)]]
['E', 220968, [(222, 1058), (222, 836), (486, 837), (486, 1059)]]
['8', 71364, [(1132, 1161), (1360, 1149), (1375, 1433), (1147, 1445)]]
['B', 5546, [(1402, 765), (1520, 763), (1521, 810), (1403, 812)]]
['DANIEL JONES', 666246, [(116, 1543), (522, 1544), (521, 1758), (115,
     1757)]]
['An underrated runner , Jones averaged more yards per carry than any
    other starting quarterback in 2020 ( 6.5 ) and finished sixth among
     starting QBs in total rushing yards ( 423 ) . He had a remarkable
    80 - yard run on a read - option play against the Eagles in Week 7
    , tied for the fourth - longest run of the season regardless of
    position .', 3330608, [(129, 1820), (1777, 1822), (1777, 2152),
    (129, 2150)]]
```

Figure 10: Returned Paragraph Bound Data

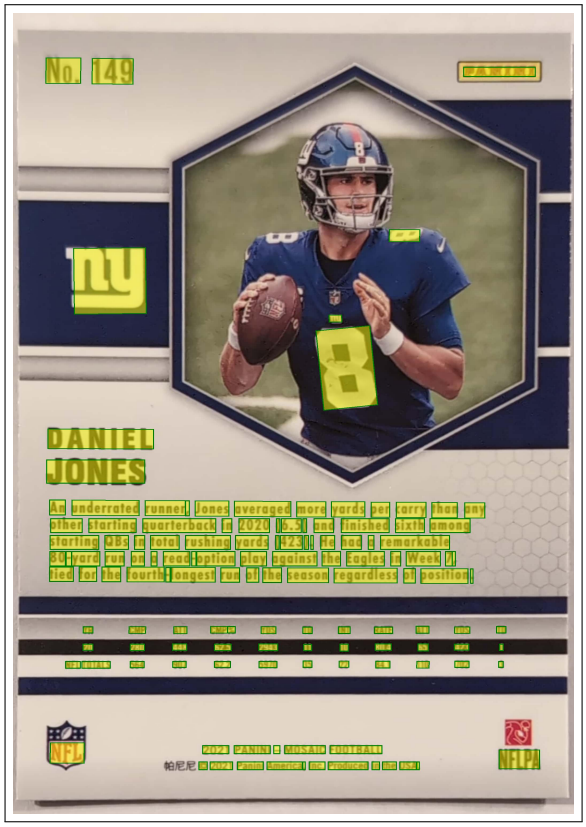

Figure 11: Google Vision Paragraph Bound Image

```
30_dj01-a-01-back.jpg
['No.', 14456, [(127, 105), (266, 105), (266, 209), (127, 209)]]
['149', 16744, [(296, 105), (457, 105), (457, 209), (296, 209)]]
['PAKIKI', 12150, [(1682, 124), (1952, 123), (1952, 168), (1682, 169)
    ]]
['nu', 219776, [(203, 855), (475, 856), (474, 1012), (202, 1011)]]
['E', 220968, [(222, 1058), (222, 836), (486, 837), (486, 1059)]]
['8', 71364, [(1132, 1161), (1360, 1149), (1375, 1433), (1147, 1445)]]
['B', 5546, [(1402, 765), (1520, 763), (1521, 810), (1403, 812)]]
['DANIEL', 596602, [(124, 1543), (522, 1546), (521, 1626), (123, 1623)
    ]]
['JONES', 612093, [(116, 1663), (489, 1663), (489, 1757), (116, 1757)
    ]]
['An', 114205, [(130, 1821), (195, 1821), (195, 1887), (130, 1887)]]
['underrated', 430732, [(211, 1821), (468, 1823), (468, 1889), (211,
    1887)]]
['runner', 218868, [(486, 1822), (642, 1823), (642, 1890), (486, 1889)
    ]]
[',', 21233, [(641, 1824), (658, 1824), (658, 1890), (641, 1890)]]
['Jones', 161595, [(675, 1823), (808, 1824), (808, 1891), (675, 1890)
    ]]
```

Figure 12: Returned Word Bound Data

Figure 13 shows the API response of ChatGPT. While the response is minimal compared to that of Google Vision, the precision with which ChatGPT satisfies the user's request prompt is substantial. Although users are not provided the level of functionality provided by Google Vision, if the initial API response is insufficient, users can submit alternative API requests with new prompts to improve the output accuracy or request more information, as shown in Figure 15. Figure 14 shows the original prompt and its output and the new prompt with the latest output. Coupling this API with Python scripts designed for text extraction incorporates the capabilities of OpenAI's machine-learning models. This provides analysis capabilities of textual data and images, including sports card images. This feature presents itself as a viable platform for image analysis in this project's context, where the effectiveness and precision of text extraction are metrics of interest.

```python
from openai import OpenAI
import base64

def encode_image(image_path):
    with open(image_path, "rb") as image_file:
        return base64.b64encode(image_file.read()).decode('utf-8')

def chatgpt_request(front_base64_image):
    client = OpenAI(api_key="API_KEY_HERE")
    response = client.chat.completions.create(
        model="gpt-4-turbo",
        messages=[{"role": "user", "content":
            [{"type": "text",
            "text": "Tell me the Name of this player.", },
             {"type": "image_url", "image_url":
                 {
                     "url": f"{
                        front_base64_image}",
                 }, }, ], }], max_tokens=300, )

    content = response.choices[0].message.content
    return content

def main():
    image_file = '1_dj01-a-15-front.jpg'
    base64_image = encode_image(image_file)
    chatgpt_response = chatgpt_request(base64_image)
    print(chatgpt_response)

main()
```

Figure 13: ChatGPT API implemented in a Python Environment

```
    "The name of the player in the image is Daniel Jones. He is a
    member of the New York Giants, as indicated on his sports card."

    Process finished with exit code 0
```

Figure 14: ChatGPT API Primary Console Response

```
    "Prompt 1: Tell me the Name of this player."
    "Response: The name of the player in the image is Daniel Jones. He
        plays for the New York Giants."


    "Prompt 2: Tell me the Name of this player, the team and jersey
       number"
    "Response: The player in the image is Daniel Jones. He plays for
       the New York Giants, and his jersey number is 8."

    Process finished with exit code 0
```

Figure 15: ChatGPT API Secondary Console Response

3 Methodology

3.1 OCR Data Extraction Overview

The goal of this evaluation was to identify the most effective OCR service in terms of text recognition in sports cards. The collective dataset includes a series of sub-sets (A, B, C, D, E), each addressing variations within the cards under controlled conditions. Dataset (E) introduces uncontrolled degrees of variability in lighting, angle, and distance by utilizing source images from eBay. The collective dataset is a total of (150) separate images or (75) pairs (card front, card back) of cards.

3.1.1 Ground Truth Textual Data

The attributes and textual data of interest on every card within the dataset are manually retrieved and recorded in a local relational database. This manually recorded data represents the expected output for each OCR service and is the "Ground Truth" used to compare the OCR output against. The output of each service was normalized and compared against the ground truth to derive the performance metrics C_A, C_{ER}, and W_{ER}. While the OCR service may or may not return ALL the textual data on the sports card, for this comparison, the data of concern is that of the desired attributes: player name, team name, brand name, manufacturer, and card description.

The primary challenge with the output from the Google Vision API existed in extracting and associating the OCR data returned with relevant attributes. The API's output consists of a JSON-like structure filled with deeply nested dictionaries, as shown in Figure 6. Although the necessary attribute information is present within the API's response, identifying and correctly mapping this data to the relevant attributes required careful analysis and validation.

The primary challenge associated with the output from the ChatGPT API lies in crafting the prompt sent to the API. It was essential to construct a clear and precise prompt to ensure that the API's response contained the specific attributes of interest. A secondary challenge arose with the text-based responses provided by the API. Since these responses are returned as strings, further processing was necessary not only to extract relevant information but also to convert it into a data type that can be utilized appropriately within the Python environment.

3.1.2 Datasets for Text Extraction

The datasets were designed to assess how each OCR service performed when faced with differences between the cards themselves rather than variations in environmental conditions. These datasets were created under controlled and consistent conditions to minimize environmental differences (lighting, angle, and distance). These images were captured using the same cellular device, in the same location, under uniform conditions in lighting, angle, and distance. A control baseline metric was established by **Dataset A**, with each subsequent dataset introducing increasing levels of card variability while

keeping environmental factors constant.

Dataset A - Control: This dataset consisted of 30 individual images (15 front sides, 15 back sides) of the same card brand, type, and player. The purpose of this dataset was to define metrics for each OCR service for images obtained in a controlled environment with almost no variability introduced concerning card brand, type, and player.

Dataset B - Player Variability: This dataset consisted of 30 individual images (15 front sides, 15 back sides) of the same card brand, type, and year with each card being a different player. The purpose of this dataset was to define metrics for each OCR service for images obtained in a controlled environment with minor variability introduced using cards of the same card brand and type but differing players.

Dataset C – Year-Brand Line Variability: This dataset consisted of 30 individual images (15 front sides, 15 back sides) of the same player, where the year and brand line vary. The purpose of this dataset was to define metrics for each OCR service for images obtained in a controlled environment with moderate variability, which was introduced using cards of the same player with different years and brands.

Dataset D – Type Variability: This dataset consisted of 30 individual images (15 front sides, 15 back sides) of different card types, but the brand, year, and player are the same. The purpose of this dataset was to define metrics for each OCR service for images obtained in a controlled environment with significant variability introduced using cards of different types but the same brand, year, and player.

Dataset E – Photograph Variability: This dataset evaluated how OCR each service performed when dealing with differences between the cards themselves and variations in the image. Real-world images were obtained from eBay, and each eBay image corresponded to the same card found in each dataset. These eBay images were kept original to preserve their natural diversity. This dataset mimicked the control dataset but introduced differing degrees of variability in lighting, distance, angle, and other factors by incorporating uncontrolled images sourced from eBay.

Figure 16: Image samples in variability matrix layout

3.1.3 Evaluation Metrics

C_A, C_{ER}, and W_{ER} were applied to the outputs of each OCR service. These metrics served as a representation of the accuracy of the extracted text. C_A assessed the correctness of individual characters, C_{ER} evaluated the percentage of incorrectly recognized characters, and W_{ER} measured the percentage of words that are incorrectly transcribed.

To apply these metrics to each dataset, the OCR-generated text was compared with the ground truth data, which held the correct card information. This comparison was performed for every dataset, from Dataset (A) with minimal variability to Dataset (E) with diverse variations in lighting, angle, and other factors. For each dataset, these metrics were calculated, allowing a nuanced assessment of OCR service performance across different levels of dataset variability. The results provided a comprehensive understanding of how well the OCR services handled various challenges in image recognition and transcription.

Simulated OCR Results		
Attributes	**Truth Values**	**Mock OCR Values**
Manufacture	Panini America	Ponini Ameriqa
Brand	Select Draft Picks	Select Draft Picks
Type	Base Rookie	Bose Rookie
Number	8	8
Year	2023	2028
Player	Jaylin Hayatt	JaylinHayatt
Team	Tennessee Volunteers	Tonnessoo Voluntaars
Description	Peerless Price, Carl Pickens, Robert Meachem and several other NFLers caught passes at Tennessee. In 2022, Hyatt accomplished feats that none of his predecessors reached. The fleet-footed playmaker became the programs first Biletnikoff Award, winners and first WR to earn unanimous First Team All-American honors.	Peerless Pric, Carl Pickons, Robert Meachem and several other NFLers caught passes at Tennosseo. In 2022, Hyatt accamqlishep feats that naqe of hiz predecessors raached. Te fleet-footed playmaker becume the programs firzt Biletnikott Award, winners and first WR to oarn qnanlmous First Team All-American honors.

Table 9: Ground Truth and Simulated ORC Data for Card Sample (2)

3.2 Experiment

This section provides an overview of the experiment conducted to evaluate each OCR service. The experiment was executed in distinct phases: Image Dataset Creation, Truth Data Extrapolation, Python API Implementation, API Response Processing, Metrics Generation, and Metrics Analysis. Each phase was composed of subsequent steps designed to handle tasks related to input automation, normalization, output collection, output normalization, automated performance computation, and metric documentation.

3.2.1 OCR API, Data Extraction, Metrics

To access each OCR service, a Python script was developed to interface with the respective API services provided by Google Vision and ChatGPT 4.0. The images from each dataset were fed into the Python script, leveraging the APIs to submit image data for optical character recognition. The results generated by each service were captured, normalized, and stored for further analysis. Subsequently,

an automated metrics script was employed to compute the desired performance metrics C_A, C_{ER}, and W_{ER}. This approach ensured a systematic and standardized evaluation of OCR services, allowing for a comprehensive assessment of their performance across different datasets and conditions. Table 10 illustrates the anticipated results for each processed image, highlighting the challenge of correctly aligning OCR data with their respective attributes. It is important to understand that this table shows the OCR accuracy metrics for each evaluated card. However, these findings do not specifically pertain to any particular OCR service being reviewed in the project.

Simulated OCR Results					
Attributes	**Truth Values**	**Mock OCR Values**	**C_A**	**C_{ER}**	**W_{ER}**
Manufacture	Panini America	Ponini Ameriqa	85.71%	14.29%	100.00 %
Brand	Select Draft Picks	Select Draft Picks	100.00%	0.00%	0.00%
Type	Base Rookie	Bose Rookie	90.91%	9.09%	50.00%
Number	8	8	100.00%	0.00%	0.00%
Year	2023	2028	75.00%	25.00%	100.00%
Player	Jaylin Hayatt	JaylinHayatt	92.31%	7.69%	100.00%
Team	Tennessee Volunteers	Tonnessoo Voluntaars	75.00%	25.00%	100.00%
Description	Peerless Price, Carl Pickens, Robert Meachem and several other NFLers caught passes at Tennessee. In 2022, Hyatt accomplished feats that none of his predecessors reached. The fleet-footed playmaker became the programs first Biletnikoff Award, winners and first WR to earn unanimous First Team All-American honors.	Peerless Pric, Carl Pickons, Robert Meachem and several other NFLers caught passes at Tennosseo. In 2022, Hyatt accamqlishep feats that naqe of hiz predecessors raached. Te fleet-footed playmaker becume the programs firzt Biletnikott Award, winners and first WR to oarn qnanlmous First Team All-American honors.	93.93%	6.07%	28.89%
Result Average			89.11%	10.89%	59.86%

Table 10: Simulated Metrics Generation for Card Sample (2)

4 Implementation

The project was structured into defined phases to implement the processes required for the systematic evaluation of Google Vision and ChatGPT 4.0. The implementation phases are outlined in Table 11.

Project Implementation Phases		
Phase	**Name**	**Description**
1	**Dataset Construction**	Capture and store card images into datasets.
2	**Ground Truth Collection**	Collect and store card attribute ground truth data.
3	**Python Implementation**	Access and use OCR API in a Python module.
4	**Response Processing**	Capture, process, format, and store API response data.
5	**Metrics Generation**	Generate and store metrics against truth dataset.

Table 11: Project Phases

4.1 Dataset Construction

The images for each dataset were captured using a Google Pixel 8 Pro smartphone to ensure high-quality image captures of each card. A smartphone was chosen as the image-capturing device to emulate the real-world scenario of a typical card collector using a cellular device to capture their collection. Each card selected to create the datasets underwent manual inspection to verify its compliance with the specific requirements of its assigned dataset.

For each card, front and back images were taken under controlled conditions. These conditions included maintaining a consistent distance between the phone and the card while ensuring that the location and lighting conditions remained unchanged throughout this phase. Images were taken standing without the use of a device stand. This was intended to simulate a given user taking a photograph under normal conditions. The distance between the device and the sports card was visually approximated and maintained. These standardization measures were intended to maintain the consistency and reliability of the visual data while minimizing variables that could impact the clarity and quality of the images.

A script using the Open-CV-Python library was implemented to automate the resolution adjustment of each image. While images were taken under controlled conditions, the height and width were slightly variable from image to image. Adjusting the resolution of each image to the same aspect ratio ensured this did not introduce a new variability factor. OpenCV-Python is a library of Python bindings designed to solve computer vision problems [13]. The height, width, and aspect ratio of each image were used to derive an average height, width, and aspect ratio for each dataset. These findings helped determine dimensions that would serve as the normalization standard for each image. The script sets the image aspect ratio to 0.70 using a width of (2100 px) and a height of (3000 px). This adjustment standardized the images to ensure uniform aspect ratios throughout the experiment.

The normalization values for image height and width were derived from the metrics provided in Table 12, Table 13, and Table 14. For the controlled datasets (A, B, C, D), a maximum percentage difference of 5% was allowed. The aspect ratio of 0.71 was used as a normalization standard due to three of the four control sets already possessing this value. The normalization values for height at 2100 and width at 3000 were decided due to their percentage difference being less than 3%, well below the maximum of 5%. Dataset E was not held to these constraints because it was the project's experimental group. The large percentage differences exhibited by the dataset allowed them to introduce another level of variance in the set.

Dataset Image Dimension Metrics			
Data Set	**Avg. Height**	**Avg. Width**	**Avg. Aspect Ratio**
A	3002.67	2146.17	0.71
B	3001.20	2094.22	0.70
C	3003.17	2142.23	0.71
D	3001.53	2131.13	0.71
E	551.7	423.53	0.78

Table 12: Dataset Image Dimension Averages

Average Height Percent Difference			
Data Set	**Avg. Height**	**Normalized Height**	**Percent Difference**
A	3002.67	3000	0.08%
B	3001.20	3000	0.04%
C	3003.17	3000	0.11%
D	3001.53	3000	0.50%
E	551.7	3000	137.87%

Table 13: Height Percentage Difference

Average Width Percent Difference			
Data Set	**Avg. Width**	**Normalized Height**	**Percent Difference**
A	2146.17	2100	2.17%
B	2094.22	2100	0.28%
C	2142.23	2100	1.99%
D	2131.13	2100	1.47%
E	423.53	2100	132.87%

Table 14: Width Percentage Difference

Each dataset comprised the front and back images of 15 selected cards, totaling 30 individual images for each dataset. Each card was represented by two images: the front and the back. The cards selected to comprise datasets A, B, C, and D satisfied the criteria outlined in section 3.1.1. Images for dataset E were sourced from the commerce site eBay. Four criteria dictated the selection of the images for this set. The criteria to be met were that the image must be of a card that exists in one of the control datasets, an image of the front and back of the card must exist, and the image must exhibit signs of variance. Variance was defined as variations in image distance, lighting, angle, and sharpness. The last criterion for selecting dataset E images was that the text in the image must be human-readable despite the image quality.

4.2 Ground Truth Collection

Each card for datasets A, B, C, and D was manually inspected to collect text data related to specific card attributes targeted for OCR testing. A Microsoft (MS) Access Database was established locally to store this extracted truth data. The choice to use an MS Access database was strategically made for its compatibility with the pyodbc Python module. This module utilizes the Open Database Connectivity (ODBC) API to facilitate connections with various database management systems, including

MS Access [19]. This allowed local Python scripts 'read and write' access to this database.

The database was intentionally kept small and simplistic. It comprises two tables: 'player' and 'card_truth.' The 'player' table is a comprehensive list of all the players featured across all datasets. It was established to maintain an independent record of the players and to act as an auxiliary reference for data rows in table 'card_truth', should the need arise in future works. Table 'card_truth' contained all extrapolated truth data from each card in each dataset. This table supplied the implementation scripts with the necessary information for carrying out the comparisons required for the experiment. All attributes collected in this table and those in table 'player' are illustrated in Figure 17.

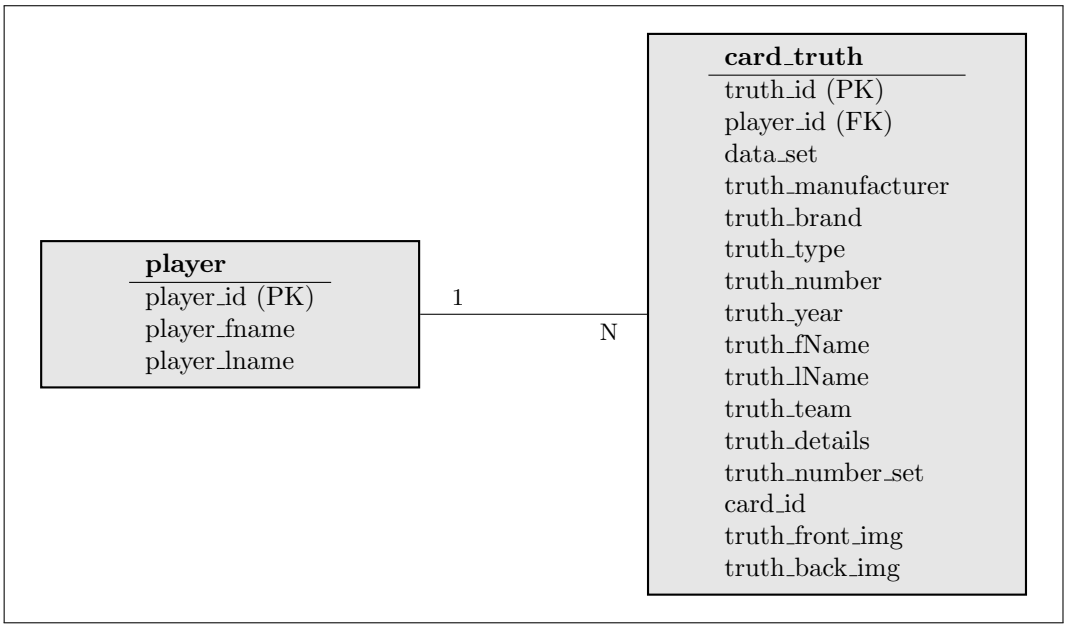

Figure 17: Database ERD Structure

4.3 Python Implementation

4.3.1 Implementation of Google Vision v.3.7.2

The project's Python module incorporating Google Vision services required the creation of a custom Graphical User Interface (GUI) to manage the execution sequence of Vision API requests, response collection, and processing. This ensured the required processes were carried out in proper order. This GUI accepts three inputs, which are passed to a driver file. The driver file passes the inputs to the functions responsible for carrying out API requests with their required parameters. The inputs taken in by the module interface include function type, feature type, and image set. The Google Vision Python module GUI is illustrated in Figure 18. Descriptions of the inputs are detailed in Table 15.

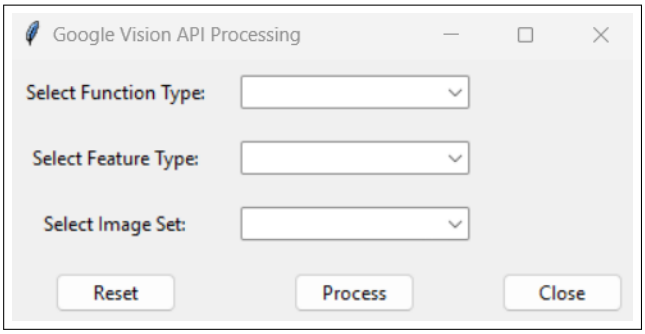

Figure 18: ChatGPT module GUI

Google Vision Input Descriptions		
Variable Name	**Type**	**Description**
function	String	Dictates the operation to be executed. Inputs include Run Vision, Extract Data, Run Metrics, and Run All.
feature	String	Dictates text detection level returned. Options include BLOCK, PARAGRAPH, WORD
image_set	String	Dictates the dataset to be processed. Options include A, B, C, D, E

Table 15: Google Vision Input Descriptions

Algorithm 1 Google Vision Image Processing

```
function PROCESS_REQUEST(feature, imageSet, imageSubset, jsonSet, textSet)
    imageSetPath ← REPO_PATH(imageSet)
    tupleList ← GET_IMAGES(imageSetPath)
    allJsonFiles ← {}
    allTextFiles ← []
    for fileTuple in tupleList do
        for image in fileTuple do
            imagePath ← PATH_TO_IMAGE(imageSet, image)
            googleData ← GOOGLE_VISION(imagePath, feature)
            allJsonData[key] ← googleData.pop()
            verticesList ← []
            for text, vertices in googleData do
                verticesList.append(vertices)
                textDataSet ← [text, vertices]
                area ← CALCULATE_AREA(vertices)
                textDataSet.Insert(1, area)
            end for
            WRITE(allTextFiles)
            WRITE(allJsonFiles)
```

Figure 19: Algorithm for processing images with Google Vision

4.3.2 API Response Processing

The documentation provided by Google Cloud resource sites lacked detail in the application of Google Vision in the domain of processing sports card images for attribute extraction. This obstacle required developing several custom-built supporting algorithmic processes to manage, manipulate, and extract the targeted attributes. The solution focused on processing and formatting Google Vision API response data to a usable format in the Python environment.

The supporting module scripts send card images from the selected dataset to the function responsible for carrying out the Vision API request. The API response was captured and further processed for formatting. The JSON response is hierarchical: text detected in blocks, paragraphs within blocks, and words within paragraphs, each with their corresponding bounding box vertices, as shown in Figure 20. The supporting algorithms extracted and organized the data by level (BLOCK, PARAGRAPH, and

WORD). This provided the ability to carry out data extraction and analysis at each level of data using the API response. The output of this function was a list of lists. Each element contained a string, the computed area of the associated vertices, and all vertices for the bounding box in which the textual data resides. Figure 20 and Figure 21 are visualizations of the API Response data pre-processing and post-processing.


```
{"front": {"textAnnotations": [{"locale": "en", "description": "nu\nE\
   nMOSAIC\nM\nCO\n8\nDANIEL JONES\nNEW YORK GIANTS", "boundingPoly":
   {"vertices": [{"x": 163, "y": 104}, {"x": 2020, "y": 104}, {"x":
   2020, "y": 2895}, {"x": 163, "y": 2895}]}}, {"description": "nu", "
   boundingPoly": {"vertices": [{"x": 240, "y": 2474}, {"x": 450, "y":
    2474}, {"x": 450, "y": 2583}, {"x": 240, "y": 2583}]}}, {"
   description": "E", "boundingPoly": {"vertices": [{"x": 252, "y":
   2635}, {"x": 242, "y": 2452}, {"x": 453, "y": 2441}, {"x": 463, "y"
   : 2624}]}}, {"description": "MOSAIC", "boundingPoly": {"vertices":
   [{"x": 206, "y": 111}, {"x": 390, "y": 107}, {"x": 391, "y": 133},
   {"x": 207, "y": 137}]}}, {"description": "M", "boundingPoly": {"
   vertices": [{"x": 165, "y": 177}, {"x": 428, "y": 164}, {"x": 439,
   "y": 394}, {"x": 176, "y": 407}]}}, {"description": "CO", "
   boundingPoly": {"vertices": [{"x": 797, "y": 585}, {"x": 916, "y":
   582}, {"x": 917, "y": 620}, {"x": 798, "y": 623}]}}
```


Figure 20: Vision API Response Pre-Processing


```
29_dj01-a-01-front.jpg
['nu', 492030, [(240, 2474), (450, 2474), (450, 2583), (240, 2583)]]
['E', 500492, [(252, 2635), (242, 2452), (453, 2441), (463, 2624)]]
['MOSAIC', 4784, [(206, 111), (390, 107), (391, 133), (207, 137)]]
['M', 63646, [(165, 177), (428, 164), (439, 394), (176, 407)]]
['CO', 4522, [(797, 585), (916, 582), (917, 620), (798, 623)]]
['8', 77822, [(1000, 1008), (1233, 969), (1288, 1295), (1055, 1334)]]
['DANIEL JONES', 1043290, [(1290, 2652), (2012, 2656), (2012, 2739),
   (1290, 2735)]]
['NEW YORK GIANTS', 1489520, [(1160, 2845), (2020, 2848), (2020, 2895)
   , (1160, 2892)]]
30_dj01-a-01-back.jpg
['No. 149', 34320, [(127, 105), (457, 105), (457, 209), (127, 209)]]
['PAKIKI', 12150, [(1682, 124), (1952, 123), (1952, 168), (1682, 169)
   ]]
```


Figure 21: Google API Response Post-Processing

4.3.3 Attribute Extraction

After the Google Vision API response data was successfully structured into lists, with the text data positioned at index (0), the bounding box area at index (1), and the list of vertices for that data at index (2), it became possible to extract specific attributes using a series of custom-built string searching functions.

Reference files, called 'Data Bank' files, were created to assist with attribute extraction. These files contain verified 'truth' values relevant to several attributes. The Data Bank consists of a player file with 27,520 NFL player names, a card manufacturers file of nine manufacturer names, an NFL team name file of 147 different team names, a card brands file of 79 different card brands, and a college team name file with over 900 different college team names.

The process of extracting target attributes followed the same general process, with minor deviations per attribute. Each attribute was extracted separately during this process. Once all attributes were extracted, the found attributes were then combined to create a singular dictionary that represented the individual card. The general process of extracting targeted card attributes consisted of iterating over the formatted API response string data and using either string matching or a similarity ratio to deem the substring an attribute of interest. If a given substring was not a direct match to the reference string, then a similarity ratio was derived to determine if the string possesses a high likeness to the reference string. This criterion was implemented to compensate for instances of the OCR system misspelling the target attribute by a few characters.

'FuzzyWuzzy' is a Python library that uses the Levenshtein distance formula to calculate the differences between sequences and patterns [9]. The **ratio** method of this library uses the Levenshtein distance formula output to return a ratio that represents the percentage of similarity between two strings. If a direct match was not found when comparing the reference string against the API response string, then the similarity percentage was determined. If a substring in the API response string achieved a similarity percentage greater than or equal to 90%, the substring that yielded this similarity percentage was extracted as the target attribute. Otherwise, the substring was extracted as the target attribute.

To find the player, team, brand, and manufacturer name, the search algorithm carried out separate searches for each attribute against its corresponding reference file. To identify the player name, the post-processed card front and back API response data was sent to a search function. The player names in the reference data was first compared against all words in all strings contained in the card's front data. If a direct match to the reference player data was not found, then a similarity metric was computed using the FuzzyWuzzy ratio method to determine if a substring yielding 90% similarity is found. If a viable substring was not returned from the search of the card front data, it is then when the same process was carried out over the card back data. If a value was not returned from a search of the card back data, an empty string was returned so as not to modify the behavior of the performance metric computations. The process of finding the card brand and manufacturer was the same. The only difference was the reference data used to find these attributes.

A problem in determining whether the team name was that there exist card parallels of an existing NFL player, but the card itself could be a 'Collegiate' card. This means that the specialty printed card was intended to depict the players' collegiate team instead of the professional team. Having a reference file for NFL teams and Collegiate teams rectified this issue, but it increased the number of comparisons that were carried out under certain cases. The process of finding the team name was the same as the process of finding the player name. The process was first conducted using the NFL teams reference file to search the API data for a team. If an NFL team was not returned from a search of the front and back, the process was then repeated using the college team reference file. If no team was found after iteration through both reference files, then an empty string was returned.

A characteristic worth noting was that the area of the bounding box that encompasses the card description yields the largest area out of all bounding boxes returned from the Vision API request. To find the card description featured on the back of most sports cards, the area of the bounding box created on the card by the vision API was computed after the immediate capture of the API response. This area value was inserted into the list of formatted API response data during the post-processing phase. To find the card description, the lists that contained the formatted API response data were sorted from largest to smallest using the area value that exists at index 1 of each list. The function dedicated to finding the card description was responsible for sorting the lists of data and then returning the string data at index 0 of the list that contains the largest area value. This return string is deemed

to be the card description.

Finding the card year and type has been reserved for future works. The tasks for determining the card year are of high complexity. This was due to the presence of statistical player data on the back of the card, along with the card's print year. Extraction of the card year can be achieved through regionalization of the card using the vertices provided by the API response. While this work was started, time constraints prevented its completion, and it was thereby categorized as future work. The determination of the card type was also reserved for future work due to concerns about complexity. An effective means of determining the card type are image matching and image recognition. This approach was deemed most appropriate due to the lack of text that defines or indicates the card type. A sample of a regionalized card image is shown in Figure 22.

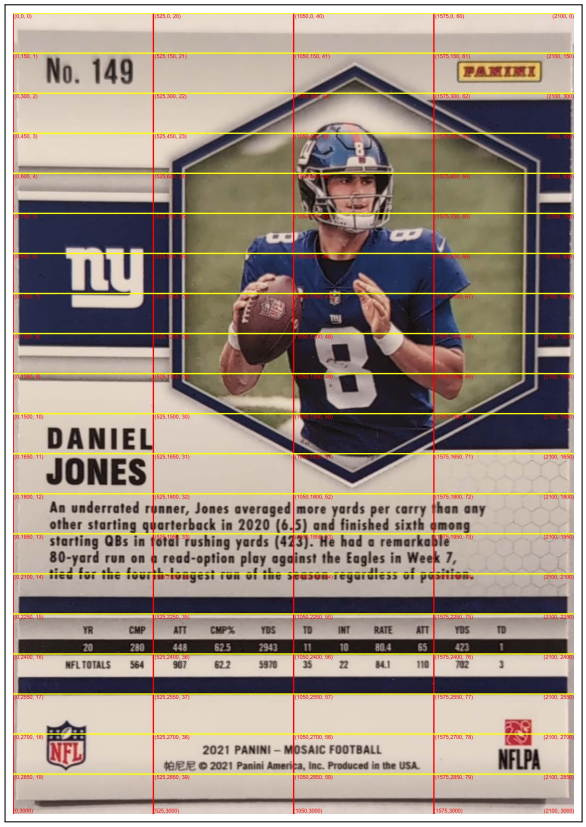

Figure 22: Regionalized Card Image

Once all attributes were assigned values that were either extracted substrings from the post-processed data or empty strings representing attributes not found, the searching of card attributes was stopped and a dictionary for that specific card was created. The dictionary used the card file name as the key to uniquely identify the dictionary belonging to that specific card. The dictionary value was a single list containing all attributes of interest once all post-processed data files had been processed and all card dictionaries had been created. This list containing all card dictionaries was written to its respective dataset folder for storage. A sample of the file containing all dictionaries, in which each dictionary is an 'attribute identified' card is shown in Figure 23.


```
{
        "29_dj01-a-01": [
            "DANIEL JONES",
            "New York Giants",
            "No. 149",
            "MOSAIC",
            "PANINI",
            "An underrated runner , Jones averaged more yards per
               carry than any other starting quarterback in 2020 ( 6.5
                ) and finished sixth among starting QBs in total
               rushing yards ( 423 ) . He had a remarkable 80 - yard
               run on a read - option play against the Eagles in Week
               7 , tied for the fourth - longest run of the season
               regardless of position ."
        ]
    },
    {
        "27_dj01-a-02": [
            "DANIEL JONES",
            "New York Giants",
            "No. 149",
            "MOSAIC",
            "PANINI",
            "An underrated runner , Jones averaged more yards per
               carry than any other starting quarterback in 2020 ( 6.5
                ) and finished sixth among starting QBs in total
               rushing yards ( 423 ) . He had a remarkable 80 - yard
               run on a read - option play against the Eagles in Week
               7 , tied for the fourth - longest run of the season
               regardless of position ."
        ]
    },
```


Figure 23: Google Vision Output Post Processing

4.3.4 Implementation of ChatGPT 4.0 v.gpt-4-0125-preview

Implementing ChatGPT 4.0 into a Python environment was less cumbersome than Google Vision. ChatGPT showed promising advantages over Google Vision during its implementation. One advantage worth noting is that ChatGPT can take in and process two images during a single API call. In contrast, Google Vision could only take one image per API call. The code base for interacting with Google Vision was reused with ChatGPT4.0 with minor modifications. As with the case of the Python implementation for Google Vision, the implementation of ChatGPT 4.0 required a GUI to exact control over the order of process execution in the module. An illustration of the Python ChatGPT module is provided in Figure 24.

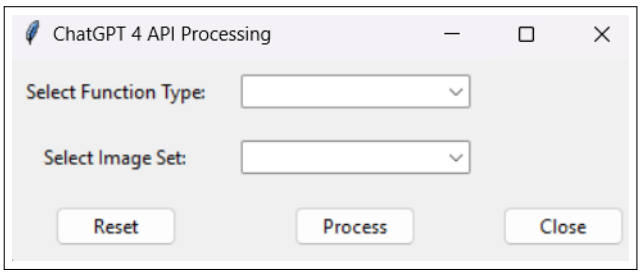

Figure 24: Vision Module GUI

The API calls to ChaptGPT with both images depended heavily on the user prompt. As mentioned earlier, the prompt is a string request, or a set of instructions passed to the API that was to be carried out over the image. The prompt input for this experiment was fixed and was not altered once a particular prompt yielded a usable output. The fixed prompt used during this implementation is illustrated in Figure 25.

One main difference in the GUI implementations between Google Vision and ChatGPT was the missing Type input box from the ChatGPT interface. The Google API response returned from the API was a JSON response that contained the detected text at the Block level, paragraph level, and word level. This large volume of returned data gave the users options regarding which level of textual data they wanted to access and extract. The type was not an option for the ChatGPT module because the API response from this API was a text or string-based response that only returned the data that was in the confines of the prompt instructions. This meant the volume of data returned was significantly less than that of the response returned from Google Vision, limiting its application in other areas such as regionalization and bounding box visualization.

```
1 f"Do a text extraction on these images, find the player, team, card
    number, brand, manufacturer and description. Return the response as
     python dictionary using these attributes as the keys. Find brands
    that only match this data {brands}"
```

Figure 25: ChatGPT Input Prompt

The 'brands' parameter was the list of brands from the Data Bank directory, which was used not for string matching as was done in the Google Vision implementation but for refining the ChatGPT response output. Once the ChatGPT API call was completed, the JSON response was formatted for metrics analysis. Due to ChaGPTs' inherent accuracy in adhering to the prompt constraints, this eliminated a vast amount of supplementary response data processing and formatting. ChatGPT's response contained attribute data formatted as a Python dictionary, as shown in Figure 26. As shown, the target attribute data was surrounded by other non-essential string string data. As mentioned earlier, the ChatGPT response was solely text-based. Meaning it was a long continuous string in which the target attributes are contained.

```
    ```
The card is from the 'MOSAIC' series and produced by Panini, which is
   a recognized manufacturer of sports cards.
 CHATGPT RESPONSE: From the images provided, here is the extracted
    information about the sports card formatted as a Python dictionary
    :
```python
card_info = {
    "player": "Daniel Jones",
    "team": "New York Giants",
    "card_number": "No. 149",
    "brand": "MOSAIC",
    "manufacturer": "Panini",
    "description": "An underrated runner, Jones averaged more yards
       per carry than any other starting quarterback in 2020 (6.5) and
        finished sixth among starting QBs in total rushing yards (423)
       . He had a remarkable 80-yard run on a read-option play against
        the Eagles in Week 7, tied for the fourth-longest run of the
       season regardless of position."
}
```
The card is from the 'MOSAIC' series and produced by Panini, which is
   a recognized manufacturer of sports cards.
```

Figure 26: ChatGPT Output Pre-processing

To extract the string dictionary that was contained within the text response, Regular Expression Pattern matching was implemented to extract all the string data that was contained within the curly brackets of the response string. Upon extracting the string dictionary, the Python 'eval()' was used to convert the string dictionary into an actual Python dictionary. The eval() function evaluates the specified expression; if the expression is a legal Python statement, it will be executed as such [20]. The function shown in Figure 27 transformed the string response data of Figure 26 into the same format as the post-processed data of the Google Vision module.

**Algorithm 2** ChatGPT Image Processing

**function** CONVERT_RESPONSE($cadID$, $resonseText$)

    $pattern \leftarrow r'*$

    $dictString \leftarrow$ SEARCH($pattern, responseText$)

    **if** $dictString$ **then**

        $realDict \leftarrow$ EVAL($dictString$) **return** $realDict$

    **else**: **return** $\{cardID : [responseText]\}$

Figure 27: ChatGPT String Text Evaluation Pseudocode
```

```
{
    "1_dj01-a-15": [
        "Daniel Jones",
        "New York Giants",
        "No. 149",
        "MOSAIC",
        "Panini",
        "An underrated runner, Jones averaged more yards per carry
           than any other starting quarterback in 2020 (6.5) and
           finished sixth among starting QBs in total rushing yards
           (423). He had a remarkable 80-yard run on a read-option
           play against the Eagles in Week 7, tied for the fourth-
           longest run of the season regardless of position.",
        13.93
    ]
},
```

Figure 28: ChatGPT Output Post-processing

```
{
        "29_dj01-a-01": [
            "DANIEL JONES",
            "New York Giants",
            "No. 149",
            "MOSAIC",
            "PANINI",
            "An underrated runner , Jones averaged more yards per
               carry than any other starting quarterback in 2020 ( 6.5
                ) and finished sixth among starting QBs in total
               rushing yards ( 423 ) . He had a remarkable 80 - yard
               run on a read - option play against the Eagles in Week
               7 , tied for the fourth - longest run of the season
               regardless of position ."
        ]
    },
    {
        "27_dj01-a-02": [
            "DANIEL JONES",
            "New York Giants",
            "No. 149",
            "MOSAIC",
            "PANINI",
            "An underrated runner , Jones averaged more yards per
               carry than any other starting quarterback in 2020 ( 6.5
                ) and finished sixth among starting QBs in total
               rushing yards ( 423 ) . He had a remarkable 80 - yard
               run on a read - option play against the Eagles in Week
               7 , tied for the fourth - longest run of the season
               regardless of position ."
        ]
    },
```

Figure 29: Google Vision Output Post Processing

Algorithm 3 ChatGPT Image Processing

```
1: function PROCESS_REQUEST(imageSet)
2:
3:     imageSetPath ← REPO_PATH(imageSet)
4:
5:     tupleList ← GET_IMAGES(imageSetPath)
6:
7:     brandsData ← GET_BANK_DATA()
8:
9:     cardsDictList ← []
10:
11:    for fileTuple in tupleList do
12:
13:        frontImage ← fileTuple[0]
14:
15:        backImage ← fileTuple[1]
16:
17:        stringResponse ← CHATGPT_REQUEST(frontImage, backImage, brandsData)
18:
19:        cardDict ← CONVERT_RESPONSE(stringResponse)
20:
21:        cardDictList.append(cardDict)
22:
23:    end for
24:
25:    WRITE(cardDictList)
26:
27: end function=0
```

Figure 30: Algorithm for Processing Images with ChatGPT API.

Due to ChatGPT's advanced modeling capability, the dictionary shown in Figure 23 returned and converted contains all attributes of interest because of the API prompt's requirements. This further highlighted the API's accuracy and precision being heavily reliant on the prompt's requirements. Once all images within all image sets had been successfully processed by the ChatGPT API and converted into a viable data format, the algorithm immediately wrote the list of identified cards to their respective image repository folder, as shown in Figure 30, line 25.

4.4 Metrics Generation

Automated metric generation was critical due to the volume of data produced by the processing scripts. Each module implementation source folder contains a Python script to compute the Character Accuracy (C_A), Character Error Rate (C_{ER}), and Word Error Rate (W_{ER}) as discussed in section 2.1.2. These metrics are implemented in a Python environment as individual functions using the Python "Levenshtein" module. The Levenshtein Python C extension module contains functions for fast computation of edit distance, string similarity, string averaging, and sequence similarity [21].

The various text detection levels (Block vs. Paragraph vs. Word) provided by Google Vision significantly influence the outcomes. This variation stems from the differing accuracies in string detection across these levels. A manual review of the data output revealed that the Paragraph level yielded the most comprehensive and accurate data. In contrast, the Block and Word level detections often missed or excluded portions of strings, leading to discrepancies in the metrics data.

The current module implementation of Google Vision possesses the capability to process and extract target data attributes with high precision and accuracy at the block, paragraph, and word levels. For the purpose of this service examination, the paragraph text level was used due to its yield of the highest precision and accuracy in attribute detection. Further refinement of the data processing algorithms will enhance the viability of the Block and Word level API response data in future iterations of this project. This will allow for more precise data extraction and improved accuracy. This improvement will enable the module to more effectively handle a wider range of text formats and complexities found in various datasets and cards.

A script that exists in each implementation module is responsible for the automated reading of data files, computing comparison metrics, and writing the metric results to a comma-separated file (CSV) located in the image set's respective folder. For computing the metrics for each card dictionary, the unique key assigned to each card's dictionary is utilized to locate and extract its matching set of truth data from the Access database. A predefined SQL query using the dictionary key runs against the database to return the truth data values used for the computations against the dictionary card attributes. The values from the database response, with the values of the card dictionary, are sent to the functions responsible for carrying out the computations. For each attribute in the card dictionary, the Character Accuracy (C_A), Character Error Rate (C_{ER}), and Word Error Rate (W_{ER}) are computed. Once all the metrics have been completed for a given image set, the results are written to the CSV file and stored. Table 16 and Table 17 are samples from the API metric data pool for each OCR platform.

Table 16: Google Dataset A - Card 1_dj01-a-01 Metrics

Google API - Data Set A Metrics					
Card ID: 1_dj01-a-01					
Attributes	**Truth Values**	**OCR Values**	**C_A**	**C_{ER}**	**W_{ER}**
Player	DANIEL JONES	DANIEL JONES	100%	0%	0%
Number	149	149	100%	0%	0%
Brand	MOSAIC	MOSAIC	100%	0%	0%
Manufac.	PANINI	PANINI	100%	0%	0%
Description	An underrated runner , Jones averaged more yards per carry than any other starting quarterback in 2020 (6.5) and finished sixth among starting QBs in total rushing yards (423) . He had a remarkable 80 - yard run on a read - option play against the Eagles in Week 7 , tied for the fourth - longest run of the season regardless of position .	An underrated runner , Jones averaged more yards per carry than any other starting quarterback in 2020 (6.5) and finished sixth among starting QBs in total rushing yards (423) . He had a remarkable 80 - yard run on a read - option play against the Eagles in Week 7 , tied for the fourth - longest run of the season regardless of position .	99.96%	4.26%	40%
Result Average			99.99 %	0.85%	8%
Google API Response Time:					2.46s

ChatGPT API - Data Set A Metrics					
Card ID: 29_dj01-a-01					
Attributes	**Truth Values**	**OCR Values**	$\mathbf{C_A}$	$\mathbf{C_{ER}}$	$\mathbf{W_{ER}}$
Player	DANIEL JONES	DANIEL JONES	100%	0%	0%
Number	149	149	100%	0%	0%
Brand	MOSAIC	MOSAIC	100%	0%	0%
Manufac.	PANINI	PANINI	100%	0%	0%
Description	An underrated runner, Jones averaged more yards per carry than any other starting quarterback in 2020 (6.5) and finished sixth among starting QBs in total rushing yards (423). He had a remarkable 80-yard run on a read-option play against the Eagles in Week 7, tied for the fourth-longest run of the season regardless of position.	An underrated runner , Jones averaged more yards per carry than any other starting quarterback in 2020 (6.5) and finished sixth among starting QBs in total rushing yards (423) . He had a remarkable 80 - yard run on a read - option play against the Eagles in Week 7 , tied for the fourth - longest run of the season regardless of position .	100%	0%	0%
Result Average			100 %	0%	0%
ChatGPT API Response Time:					12.21s

Table 17: ChatGPT Dataset A - Card 29_dj01-a-01 Metrics

5 Evaluation

5.1 Accuracy Evaluation

For each dataset and each targeted attribute, the character accuracy, character error rate, and word error rate are analyzed to determine their minimum, median, maximum and mean values for both Google Vision and ChatGPT. The character accuracy and character error rate can take potential percentage values within the range of [0,100]. A character accuracy of 0% indicates that the OCR system identified a string that lacked any sequential character matches with the ground truth, or no string value was detected for that attribute. A character accuracy of 100% indicates a perfect match of sequential characters in the OCR string against the ground truth string. This indicates a successful detection of the given target attribute.

Character accuracy values of 100% can yield character error rates and word error rate values ranging from $[0, \infty)$. When a character accuracy of 100% yields a character error or word error value, it indicates the presence of additional characters in the OCR string or the necessity for further edits (insertions, substitutions, deletions) to align it completely with the ground truth string. This scenario is exemplified in Dataset A of the ChatGPT metrics in the detection of the 'brand' attribute. In this case, the ground truth string is 'PANINI', while the OCR string is 'PANINI AMERICA, INC.'. Figure 31 provides a detailed elaboration of how the metric values for this attribute are calculated.

Let ($\mathrm{C_T}$) be the length of the ground truth string.
Let ($\mathrm{C_H}$) be the length of the hypothesis (OCR) string.
Let ($\mathrm{C_M}$) be the character matches of ($\mathrm{C_T}$) in ($\mathrm{C_H}$).
Let ($\mathrm{W_{ER}}$) be the number of words in the ground truth string.

$$C_{\mathrm{T}} = 6 \leftarrow \mathrm{len}('PANINI')$$

$$C_{\mathrm{H}} = 20 \leftarrow \mathrm{len}('PANINI \;\; AMERICA, \;\; INC.')$$

$$C_{\mathrm{M}} = 6 \leftarrow \mathrm{MATCH}('PANINI', \;\; 'PANINI \;\; AMERICA, \;\; INC.')$$

$$C_{\mathrm{A}} = \left(\frac{C_{\mathrm{M}}}{C_{\mathrm{T}}} \times 100 \right) = \frac{6}{6} \times 100 = 100\%$$

$$C_{\mathrm{ER}} = \left(\frac{I + S + D}{C_{\mathrm{T}}} \times 100 \right) = \frac{14}{6} \times 100 = 233.33\%$$

$$W_{\mathrm{ER}} = \left(\frac{I + S + D}{W_{\mathrm{T}}} \times 100 \right) = \frac{2}{1} \times 100 = 200\%$$

Figure 31: Example Metrics Computation

The character error metric represents the percentage of incorrect characters in the OCR string. In Figure 31, a character error percentage of 233.33% means approximately 2.3 times more characters in the OCR string than in the ground truth string. The OCR string in Figure 31 has 20 characters, with the first 6 matching the ground truth characters. Multiplying the number of matching characters by 2.3 yields an approximate value 13.8. This resulting value is the approximate number of incorrect characters present in the OCR string.

The word error metric shows the percentage of word-level errors in the OCR string compared to the ground truth string. If the word error percentage exceeds 100%, it means there are more corrections needed to match the OCR string with the ground truth string than there are words in the ground truth string. This occurs when the OCR string contains many incorrect or extra words. Word error rate values can range from $[0, \infty)$. In Figure 31, the ground truth string has one word, but the OCR string has three words. This results in an edit distance of 2 between the two strings, requiring 2 deletions to align the OCR string with the ground truth string. Similarly to the character error percentage, this indicates that there are 2 times as many words in the OCR string as in the ground truth string. This is how the character error and word error percentages in Table 18 are to be interpreted.

Immediate analysis of the computed metrics showed that for each provider, the maximum character accuracy values for all targeted attributes fell within the range of [99 , 100] in Dataset A. Both Google Vision and ChatGPT exhibited near-perfect performance, consistently achieving a minimum and maximum accuracy value of 100% in detecting all targeted attributes. With exception for the card description attribute for Google Vision, which scored 99.96%. This near-perfect performance of both OCR services in Dataset A highlights their ability to maintain accuracy consistency in text detection across all attributes with no card variance among all images in Dataset A.

Minimum values for character accuracy ranged [0, 100]. Instances of 0% minimum values indicate a failure to detect the specified attribute. Both Google and ChatGPT demonstrated this in Dataset C for the Team, Number, and Brand attributes. In Dataset C, card images feature the same player but vary in the year and brand line, showing a moderately high level of card variance. Both services also exhibited extreme percentage variance in the maximum word error rate percentages across all attributes except for the player name attribute. Showing difficulty in the services ability to maintain a consist amount of word detection for each attribute regarding image text length and complex text structures. This highlights challenges faced by both services in maintaining consistent text detection accuracy in the presence of card variation, particularly regarding year and brand.

In Dataset E, Google Vision failed to detect the Team, Number, and Brand attributes, whereas ChatGPT only missed the Number attribute. This dataset mimics the control dataset but introduces variations in lighting, distance, angle, and other factors affecting image quality. Similar to Dataset C, both services showed significant variance in word error percentage across all attributes except for the Player Name attribute. This highlights challenges faced by both services in maintaining consistent word detection for other attributes regarding image text length and complex text structures. However, these findings highlight Google Vision's struggles in consistently and accurately detecting text under conditions of extremely high image quality and card variance. The most frequent occurrence percentage variance in both OCR providers is observed in the Description attribute across all datasets. This indicates a consistent difficulty with the OCR providers in accurately detecting the card descriptions. This variability is evident in the vast range of Word Error Rates that appear to fluctuate from extremely low (0) to extremely high (500).

Based on these metrics, both Google Vision and ChatGPT demonstrate comparable levels of accuracy across the all datasets. These findings indicate that neither service consistently outperforms the other in terms of accuracy alone. Further examination of other service attributes was required to discover alternate service strengths and weaknesses.

Table 18: OCR Metrics Summary

Provider (Set)	Attribute	(C_A)%				(C_{ER})%				(W_{ER})%			
		min	med	max	M	min	med	max	M	min	med	max	M
Google (A)	Player	100	100	100	100	0	0	0	0	0	0	0	0
	Team	100	100	100	100	0	0	0	0	0	0	0	0
	Number	100	100	100	100	0	0	0	0	0	0	0	0
	Brand	100	100	100	100	0	0	0	0	0	0	0	0
	Manufact.	100	100	100	100	0	0	0	0	0	0	0	0
	Desc.	99.9	99.9	99.9	99.9	4.2	4.2	4.2	4.2	40	40	40	40
ChatGPT (A)	Player	100	100	100	100	0	0	0	0	0	0	0	0
	Team	100	100	100	100	0	0	0	0	0	0	0	0
	Number	100	100	100	100	0	0	0	0	0	0	0	0
	Brand	100	100	100	100	0	0	0	0	0	0	0	0
	Manufact.	100	100	100	100	0	0	233.3	15.5	0	0	200	13.3
	Desc.	100	100	100	100	0	0	0	0	0	0	0	0
Google (B)	Player	100	100	100	100	0	0	0	0	0	0	0	0
	Team	99.8	100	100	99.9	0	0	11.11	0.7	0	0	33.3	2.3
	Number	100	100	100	100	0	0	0	0	0	0	0	0
	Brand	100	100	100	100	0	0	0	0	0	0	0	0
	Manufact.	100	100	100	100	0	0	0	0	0	0	0	0
	Desc.	99.9	99.9	99.9	99.9	2.4	4.0	6.0	4.0	22.8	32.1	59.6	35.0
ChatGPT (B)	Player	100	100	100	100	0	0	0	0	0	0	0	0
	Team	99.8	100	100	99.9	0	0	11.1	0.7	0	0	33.3	2.3
	Number	100	100	100	100	0	0	0	0	0	0	0	0
	Brand	100	100	100	100	0	0	0	0	0	0	0	0
	Manufact.	100	100	100	100	0	0	0	0	0	0	0	0
	Desc.	99.9	99.9	100	99.9	0	1	15.5	2.3	0	5.5	18.5	5.5
Google (C)	Player	100	100	100	100	0	0	0	0	0	0	0	0
	Team	0	100	100	86.6	0	0	100	15.1	0	0	100	15.5
	Number	0	100	100	79.8	0	0	133	34.4	0	0	500	120
	Brand	0	100	100	73.3	0	7.1	100	35.1	0	33.3	100	43.3
	Manufact.	99	100	100	99.8	0	0	100	13.3	0	0	100	13.3
	Desc.	99.0	99.9	99.9	99.6	1.9	4.5	96.5	32.5	19.61	40	112	57.5
ChatGPT (C)	Player	100	100	100	100	0	0	0	0	0	0	0	0
	Team	0	100	100	60	0	0	99.3	26.2	0	0	100	28.8
	Number	0	100	100	93.3	0	0	140	9.3	0	0	100	6.6
	Brand	0	100	100	86.6	0	0	150	32.0	0	0	200	40
	Manufact.	99	100	100	99.9	0	0	100	6.6	0	0	100	6.6
	Desc.	99.0	100	100	99.8	0	0.3	91.3	11.1	0	1.8	98.0	9.9
Google (D)	Player	100	100	100	100	0	0	0	0	0	0	0	0
	Team	100	100	100	100	0	0	0	0	0	0	0	0
	Number	99.6	100	100	99.9	0	0	40	2.6	0	0	300	20
	Brand	0	100	100	66.6	0	0	100	33.3	0	0	100	33.3
	Manufact.	100	100	100	100	0	0	0	0	0	0	0	0
	Desc.	99.9	99.9	99.9	99.9	3.8	6.1	7.0	6.1	23.0	41.1	54	43.7
ChatGPT (D)	Player	100	100	100	100	0	0	0	0	0	0	0	0
	Team	100	100	100	100	0	0	0	0	0	0	0	0
	Number	0	100	100	86.6	0	0	633.3	48.8	0	0	300	26.6
	Brand	100	100	100	100	0	0	0	0	0	0	0	0
	Manufact.	100	100	100	100	0	0	0	0	0	0	0	0
	Desc.	0	99.9	99.9	86.6	1.8	2.1	98.1	16.4	9.8	12	100	25.3
	Player	100	100	100	100	0	0	0	0	0	0	0	0
	Team	0	100	100	93.3	0	0	100	7.4	0	0	100	8.8
	Number	0	100	100	79.9	0	0	150	28.6	0	0	300	60

Continued on next page

Table 18 continued from previous page

Provider (Set)	Attribute	(C_A)%				(C_{ER})%				(W_{ER})%			
Google (E)	Brand	0	100	100	80	0	0	100	24.4	0	0	100	26.6
	Manufact.	0	100	100	93.3	0	0	100	6.6	0	0	100	6.6
	Desc.	99.1	99.9	99.9	99.8	2.8	4.9	87.1	17.08	23.0	39.2	116	50.0
ChatGPT (E)	Player	100	100	100	100	0	0	0	0	0	0	0	0
	Team	99.8	100	100	99.9	0	0	11.1	0.74	0	0	33.3	2.2
	Number	0	100	100	80	0	0	150	23.3	0	0	100	20
	Brand	99.1	100	100	99.9	0	0	85.7	5.7	0	0	100	6.6
	Manufact.	100	100	100	100	0	0	233.3	31.1	0	0	200	26.6
	Desc.	99.2	99.9	100	99.8	0	1.8	76.2	14.6	0	5.7	98	21.9

5.1.1 Time Evaluation

Table 19 shows the average duration's for completing API requests and processing response data for each data. Highlighting performance differences between Google Vision and ChatGPT's across the datasets in terms of time. Google Vision shows quick API response times, averaging about 2.5 seconds, with additional data processing that extends the total runtime to approximately 4 to 5 seconds. Despite the increases in the overall duration due to the required API response processing, it remains significantly shorter than ChatGPT's total runtime.

ChatGPT shows significantly longer API response times, which average between 13 and 14 seconds, without local data processing time. This lack of local processing means no additional time was spent manipulating the API response, facilitating a streamlined integration process where the data was immediately usable upon capture. These larger response times may become problematic in cases where rapid response is required when handling large user volumes.

Table 19: Platform Run Time Averages

Platform (Set)	API Response (s)	Data Processing (s)	Combined Time (s)
Google (A)	2.34	2.60	4.94
ChatGpt (A)	11.20	0	11.20
Google (B)	2.35	1.60	3.95
ChatGpt (B)	13.80	0	13.80
Google (C)	2.90	1.72	4.10
ChatGpt (C)	13.59	0	13.59
Google (D)	2.79	2.18	4.97
ChatGpt (D)	14.00	0	14.00
Google (E)	2.33	1.98	4.31
ChatGpt (E)	14.13	0	14.13

5.1.2 Cost Analysis

The **Google Vision** API provides the initial allotment of 1,000 API calls for free. Charges apply after exceeding this allotment at a fixed rate of $1.50 for every 1,000 additional calls over [6]. Figure 32 illustrates a proportionate increase in cost as the volume of API calls increases. This shows a linear rise in cost with each additional 1,000 calls made. While the graph shows a consistent rise in cost, the overall cost remains relatively small in relation to the volume of API calls allowed. The projected data shows that 100,000 API calls would result in a cost of $150.00, reducing t0 $0.0015 per call. This cost metric is notable for applications when handling a large user base, offering an attractive cost-to-service ratio that can be beneficial for scaling purposes.

The cost of **ChatGPT 4.0** is based on prompt tokens (input text size) and completion tokens (output text size). The current rate for input to ChatGPT is $0.01 per 1,000 tokens and $0.03 per 1,000 tokens of output [10]. The total cost per API call can be calculated as $(InputTokens * (0.01/1000)) + (OutputTokens * (0.03/1000))$. The input size for each call made in each dataset was approximately 2600 tokens, with an average output size of 300 tokens. These token sizes make the cost of each ChatGPT API call approximately $.0035 per call. Using these values as constants, projections can be made to estimate the service cost of ChatGPT as the volume of API calls increases. Figure 32 illustrates the cost projection of ChatGPT per 1,000 API calls using the fixed constants. Using these projections, the estimated cost for 1,000 API calls is approximately $35.04. Continuing along the linear trend, the estimated cost of 100,000 API calls amounts to the approximate cost of $3,500.04. This characteristic is critical due to its impact on the ratio of cost to usage, providing insights into the cost of application over a large user base.

Figure 32 compares the linear cost trajectories of each service as the volume of API calls, or 'usage' increases in terms of API calls. The graph shows a significantly faster rate of cost increase for ChatGPT when compared to Google Vision. Despite the similarities in their pricing, ChatGPT's service generates greater expenses at similar call volumes. This is a notable factor to consider for applications involving significantly larger API call volumes and/or user base.

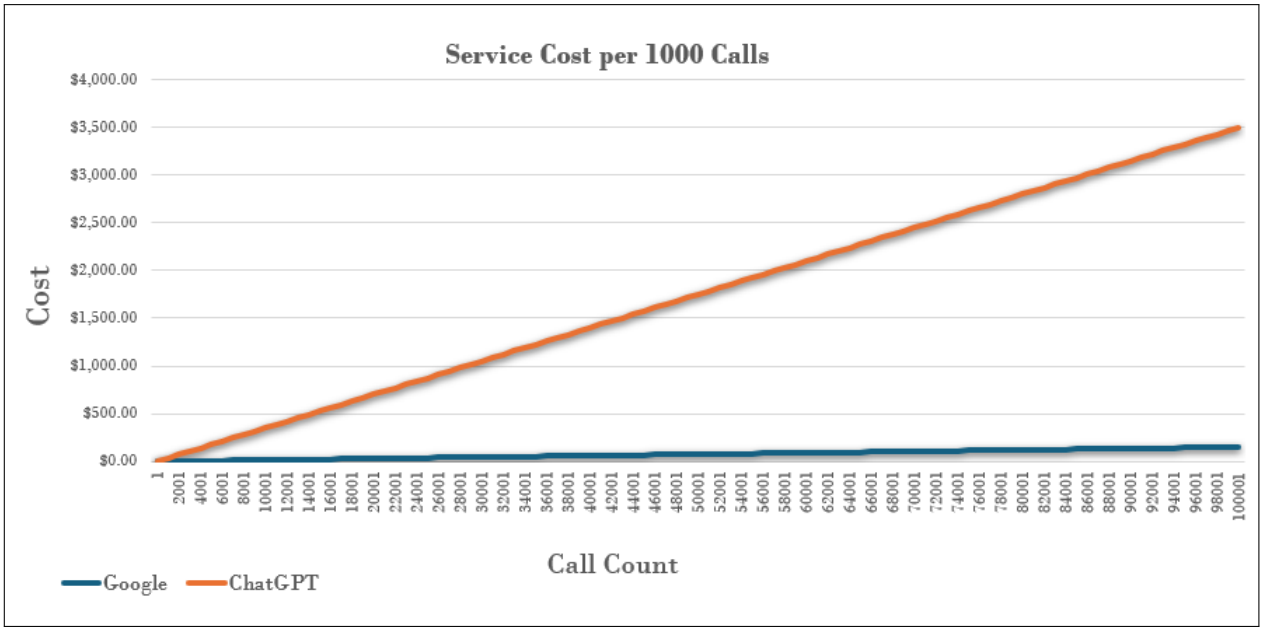

Figure 32: Cost Comparison of Google Vision and ChatGPT

Service Cost Projections		
Call Size	**Google Vision Cost**	**ChatGPT 4.0 Cost**
1	0	0.035
1,001	0	35.035
10,001	15.00	350.03
20,001	30.00	700.03
30,001	45.00	1,015.03
40,001	60.00	1,400.03
50,001	75.00	1,750.03
60,000	90.00	2,100.03
70,001	105.00	2,450.03
80,001	120.00	2,800.03
90,001	135.00	3,150.05
100,001	150.00	3,500.03

Table 20: Service Cost Projections

6 Conclusion

This comparative performance analysis between Google Vision and ChatGPT reveals slight variances in consistent accuracy, with ChatGPT exhibiting marginally better results in this domain. However, Google Vision demonstrates substantial benefits in terms of API response time, data extraction speed, and cost efficiency. The empirical data from this study shows both Google Vision and ChatGPT maintain near-perfect character accuracy with very little variability across the differing datasets. This quantifies their abilities to process text from sports card images accurately. However, Google Vision outperforms ChatGPT in operational efficiency, demonstrated by its quicker API responses and data processing times. These are considered critical factors for applications requiring fast and reliable functionality. Cost analysis also suggests that Google Vision presents a more economically viable option for sustained use over a large user base. Considering these metrics, Google Vision is identified as a more suitable choice for the integration of OCR services in the sports card domain, with a focus of efficiency and cost-effectiveness.

6.1 Future Works

Future works for expanding and enhancing the project's implementation and OCR evaluation capabilities. Accurately determining the card year from the data returned from the OCR service proved challenging. The abundance of numbers present on both sides of a given sports card make it difficult to identify the card year in the presence of other text numeric values. Algorithms that use contextual analysis and image regionalization may offer one solution. The goal is to use OCR data and other supporting processes to develop a system that accurately identifies the card type. Card type refers to a card that presents an alternative, variation, or parallel version of an existing base card. This will require further research into image matching and creating custom algorithms using image recognition techniques to identify card types and variations. This project's OCR extraction and evaluation implementation will be modified into a backend support system for a football card inventory application. By integrating advanced OCR technology, the system will efficiently extract and process text data from football cards, facilitating convenient inventory management. Further enhancements will focus on optimizing data accuracy, scalability, and integration capabilities to meet the specific needs of the inventory application.

References

[1] *A Century Ago, The Optophone Allowed Blind People to Hear the Printed Word*. 2021. URL: https://spectrum.ieee.org/a-century-ago-the-optophone-allowed-blind-people-to-hear-the-printed-word (visited on 05/02/2024).

[2] Muna Ahmed Awel and Ali Imam Abidi. "Review on Optical Character Recognition". In: *International Research Journal of Engineering and Technology* (2019).

[3] Collins Ayuya. *AWS vs Azure vs Google Cloud — Comparing Solutions 2023*. 2023. URL: https://www.channelinsider.com/cloud-computing/aws-vs-azure-vs-google-cloud/ (visited on 11/28/2023).

[4] Sanjeev Chaudhary et al. "A Comparison of Public Cloud Computer Vision Services". In: (2012). (Visited on 10/01/2023).

[5] *Cloud Vision Documentation*. 2024. URL: https://cloud.google.com/vision/docs#:~:text=Google's%20Cloud%20AutoML%20Vision%20enables,machine%20learning%20expertise%20train%20custom (visited on 04/19/2024).

[6] *Cloud Vision Pricing*. 2024. URL: https://cloud.google.com/vision/pricing#prices (visited on 04/19/2024).

[7] *CNN vs. RNN: How are they Different*. 2023. URL: https://www.techtarget.com/searchenterpriseai/feature/CNN-vs-RNN-How-they-differ-and-where-they-overlap (visited on 05/06/2024).

[8] *Dense Document Text Detection Tutorial*. 2024. URL: https://cloud.google.com/vision/docs/fulltext-annotations (visited on 04/19/2024).

[9] *Fuzzy String Matching in Python: Introduction to FuzzyWuzzy*. 2023. URL: https://builtin.com/data-science/fuzzy-matching-python (visited on 04/19/2024).

[10] *How Much Does GPT-4 Cost*. 2024. URL: https://help.openai.com/en/articles/7127956-how-much-does-gpt-4-cost (visited on 04/19/2024).

[11] *Image Segmentation: The Basics and 5 Key Techniques*. 2023. URL: https://datagen.tech/guides/image-annotation/image-segmentation/ (visited on 11/07/2023).

[12] *Introduction*. 2024. URL: https://platform.openai.com/docs/introduction (visited on 04/19/2024).

[13] *Introduction to OpenCV-Python Tutorials*. 2024. URL: https://docs.opencv.org/4.x/d0/de3/tutorial_py_intro.html#:~:text=OpenCV%2DPython%20is%20a%20library,to%20solve%20computer%20vision%20problems. (visited on 04/21/2024).

[14] Amanjot Kaur and Gagandeep Kaur. "A Review on Image Enhancement with Deep Learning Approach". In: *ACCENTS Transactions on Image Processing and Computer Vision* (2018). URL: https://www.proquest.com/docview/2492711744?pq-origsite=gscholar&fromopenview=true (visited on 10/04/2023).

[15] Eric Martin. *Sports Trading Cars Market: Global Scenario, Leading Players and Growth by 2023*. 2023. URL: https://www.linkedin.com/pulse/sports-trading-cards-market-global-scenario-leading-players-martin (visited on 11/28/2023).

[16] Fanfei Meng and Braden Ghena. "Applications of integration of AI-based Optical Character Recognition (OCR) and Generative AIin Document Understanding and Processing". In: *Applied Research in Artificial Intelligence and Cloud Computing* (2023). URL: https://researchberg.com/index.php/araic/article/view/171/159 (visited on 05/04/2024).

[17] Fanfei Meng and Braden Ghena. "Research on Text Recognition Methods Based on Artificial Intelligence and Machine Learning". In: *Advances in Computer and Communications* (2023). URL: https://www.hillpublisher.com/UpFile/202311/20231130191638.pdf (visited on 05/04/2024).

[18] Dhruv Parman. *9 Best Sports Card Scanners to Assess Your Collections Value*. 2023. URL: https://geekflare.com/best-sports-card-scanners/ (visited on 11/27/2023).

[19] *pyodbc 5.1.0*. 2024. URL: https://pypi.org/project/pyodbc/ (visited on 04/19/2024).

[20] *Python eval() Function*. 2024. URL: https://www.w3schools.com/python/ref_func_eval.asp (visited on 04/19/2024).

[21] *python-Levenshtein 0.25.1*. 2024. URL: https://pypi.org/project/python-Levenshtein/ (visited on 04/19/2024).

[22] Ashutosh Saitwal. *How to Calculate and Improve OCR Accuracy*. 2023. URL: https://www.klearstack.com/how-to-calculate-and-improve-ocr-accuracy/#:~:text=In%20the%20case%20of%20that,accuracy%20indicates%20better%20OCR%20performance. (visited on 11/28/2023).

[23] Sumit Kumar Sharma. *Image Representation in Computer Vision*. 2023. URL: https://medium.com/@sumit-kr-sharma/image-representation-in-computer-vision-364e47c4e69a (visited on 11/10/2023).

[24] *The Power of OCR Automation in the Age of Digital Transformation*. 2024. URL: https://www.artsyltech.com/blog/Power-Of-OCR-Automation (visited on 05/02/2024).

[25] Amit Timalsina. *Analysis and Benchmarking of OCR Accuracy for Data Extraction Models*. 2023. URL: https://www.docsumo.com/blog/ocr-accuracy (visited on 11/16/2023).

[26] *Understanding the Levenshtein Distance Equation for Beginners*. 2019. URL: https://medium.com/@ethannam/understanding-the-levenshtein-distance-equation-for-beginners-c4285a5604f0 (visited on 11/28/2023).

[27] Dr. S. Vijayarani and S. Sakila. "Performance Comparison of OCR Tools". In: *International Journal of UbiComp (IJU)* (2015). URL: https://www.researchgate.net/publication/281583162_Performance_Comparison_of_OCR_Tools (visited on 11/28/2023).

[28] *What are Language Models (LLMs)?* 2024. URL: https://www.ibm.com/topics/large-language-models (visited on 05/06/2024).

[29] *What Is Image Processing: Overview, Applications, Benefits, and More*. 2023. URL: https://www.simplilearn.com/image-processing-article (visited on 11/04/2023).

[30] *What is OCR (Optical Character Recognition)*. 2023. URL: https://aws.amazon.com/what-is/ocr/ (visited on 11/04/2023).