DEVELOPMENT OF HYBRID OPTIC CHARACTER RECOGNITION SYSTEM

Onu Sunday[1], Ugah J.O.[2], Ituma Chinagorom[3], Ori Silas Ene[4] Ogbu Henry N.[5]
Ani Ukamaka Eucharia[6] Omeje Kingsley Nnabuike[7]
[1,4,6,7] David Umahi Federal University of Health Sciences, Uburu
[2,3,5] Department of Computer Science, Ebonyi State University, Abakaliki

Abstract
Handwritten documents remain an essential medium for communication in academic, administrative, and social contexts. However, converting them into digital formats with high accuracy continues to present challenges, especially in cases where legibility is poor or writing styles vary significantly. Optical Character Recognition (OCR) technologies have been widely adopted to automate text extraction from printed and handwritten sources, but they often produce error-prone outputs when handling cursive or irregular handwriting. At the same time, the emergence of advanced generative Artificial Intelligence (AI) models, such as OpenAI's GPT family, has enabled automated correction of textual errors while preserving meaning. This article presents OCRFlow, a novel hybrid system that combines Google Cloud Vision OCR and OpenAI's GPT with a user-friendly graphical interface built using Python's Tkinter. The study follows a combination of Object Oriented Analysis and Development (OOAD) and Design Science Research (DSR) Methodologies, involving iterative prototyping and user feedback to improve usability and accuracy. A series of evaluations were conducted to assess OCR accuracy, correction performance, usability, and efficiency. OCRFlow provides a practical proof of concept for hybrid AI-OCR systems that enhance digital transformation of handwritten content. The article contributes to both research and practice by demonstrating how existing AI services can be orchestrated into a cohesive tool that addresses real-world problems in document management. It also highlights the broader implications of integrating OCR with LLM-based correction for educational institutions, government offices, and industries where handwritten records remain prevalent.

Keywords: Artificial Intelligence, OCR, GPT, deep learning, machine learning, language model, transformer model

Introduction
The use of handwritten documents over the years cannot be over-emphasized. Even in the digital age, handwritten documents continue to play vital roles in academia, healthcare, legal practices, and government administration. Examination scripts, medical prescriptions, meeting notes, and archival manuscripts are still predominantly handwritten across much of the world. Recognizing such handwriting on paper has been a big problem for people over time even with the advent of technology. There are some common problems with handwriting. Perhaps the most obvious problem is poor quality or illegible handwriting. According to [1], poor handwriting can have a pervasive effect on school performance in mathematics because handwriting is a basic tool used in taking notes and doing classwork and assignments. To [2], another problem with handwriting is joined and cursive handwriting. The problem that cursive handwriting can cause is that letters

may not be as easily recognized and possibly cause false and incorrect information to be processed. As a result, the development of effective Optical Character Recognition (OCR) systems capable of handling handwriting has been a recurring theme in both academic and industrial research [3]. Optical Character Recognition (OCR) is a mechanism which enables the computer system and its applications to recognize different characters in different types of documents and interpret them into analyzable, editable and searchable data. But OCR is prone to errors. According to [3], variability in writing styles affects the accuracy of handwriting recognition. [4] stated that factors such as poor lighting, smudging, and camera distortions degrade recognition accuracy. According to [5], unlike printed characters, cursive scripts connect letters in fluid strokes, complicating segmentation.

Over time, the integration of computer vision and machine learning improved recognition capabilities. But even state-of-the-art OCR engines still produce errors, especially when confronted with cursive writing, low-quality images, or non-standard writing styles. Recent developments in Artificial Intelligence (AI), particularly Natural Language Processing (NLP) and Large Language Models (LLMs), present an opportunity to reduce these errors [6]. [7] stated that NLP is related to information retrieval, knowledge representation, computational linguistics, and more broadly with linguistics. The largest and most capable LLMs are generative pre-trained transformers (GPTs), based on a transformer architecture, which are largely used in generative chatbots such as ChatGPT, Gemini and Claude. The introduction of transformer-based language models, particularly OpenAI's GPT series and Google's BERT, revolutionized text correction and contextual understanding [8],[9].

As a result of the above, this research work aims to develop a handwritten document to digital document converter, known as OCRFlow, which integrates Google Cloud Vision OCR with OpenAI GPT to address the inherent problems associated with character recognition.

This research work aims to develop a hybrid system, known as OCRFlow, that integrates OCR and generative AI for accurate digitization of handwritten documents. The specific objectives:

i. To design a user-friendly Python application for real-time scanning, recognition, and correction of handwritten documents.
ii. To integrate Google Cloud Vision OCR for text recognition and OpenAI GPT for post-processing correction.
iii. To implement a transparent interface showing both raw OCR and corrected text for user validation.

Previous Related Work

According [10], the origins of OCR can be traced to the early 20th century, when mechanical devices were developed to assist visually impaired individuals in reading printed materials.

The first electronic OCR systems emerged in the 1950s and 1960s, designed to recognize machine-printed characters in restricted fonts, such as OCR-A and OCR-B. These systems relied on template matching and rule-based recognition, offering limited flexibility when confronted with handwriting [11]

A major milestone was the development of Tesseract, initially by Hewlett-Packard in the 1980s, and later open-sourced by Google in 2005 [4]. Tesseract became one of the most widely used OCR engines due to its open-source availability and adaptability. However, its performance was

optimal for printed text rather than handwriting. Research using Tesseract as a baseline highlights its limited accuracy in recognizing handwritten samples, often requiring preprocessing techniques such as binarization, noise removal, and segmentation [12]. Despite these limitations, Tesseract remains a benchmark in OCR research, with extensions and training datasets enabling limited handwritten recognition [13].

ABBYY FineReader was later developed as a commercial OCR engine known for high accuracy in printed text [14]. While it includes handwriting support, licensing costs and closed-source architecture limit its adaptability for research and educational purposes.

This was followed by end-to-end systems such as TrOCR [15] which outperforms traditional pipelines by learning holistic representations of handwritten text. However, these systems require large-scale training data and high-performance computing resources, limiting accessibility for non-specialist users.

The use of artificial neural networks in OCR began in the 1990s, with early models applying multi-layer perceptrons (MLPs) to character classification tasks [16]. However, these models lacked robustness to spatial dependencies in handwriting. The introduction of Recurrent Neural Networks (RNNs), specifically Long Short-Term Memory (LSTM) networks, revolutionized handwriting recognition by modeling sequential dependencies in strokes and characters [17]. LSTMs can capture long-term dependencies in sequential data making them ideal for tasks like language translation, speech recognition and time series forecasting [18].

Convolutional Neural Networks (CNNs) was extensively used in image-based handwriting recognition tasks, particularly for feature extraction [19]. It performs different tasks like image classification, text detection, pose estimation, object tracking, action detection, visual saliency detection, scene marking, speech and natural language processing.

The recent introduction of Transformer-based architectures reshaped the field of handwriting recognition. Microsoft's TrOCR model [15] applies pre-trained Vision Transformers (ViTs) and Transformer decoders to achieve state-of-the-art performance on handwritten text datasets. Transformers have the advantage of having no recurrent units, therefore requiring less training time than earlier recurrent neural architectures (RNNs) such as long short-term memory (LSTM). [20]

Cloud services have made OCR widely accessible, with providers offering pre-trained models as APIs. Google Cloud Vision API supports printed and handwritten text extraction in multiple languages. Studies show high accuracy for well-formed handwriting but persistent errors in messy scripts [21]). AWS Textract emphasizes structured document recognition, such as forms and tables [22]. Microsoft Azure Cognitive Services also provide handwriting recognition, though limited by subscription models [23]. Cloud-based OCR reduces the need for local training and model deployment but introduces challenges of billing, privacy, and internet dependency [24].

One of the most persistent challenges in OCR is that even when the system successfully identifies characters, the resulting text often contains errors. These errors arise from segmentation mistakes, noise, and ambiguous handwriting styles. To address these limitations,

post-processing correction techniques have been developed using Natural Language Processing (NLP), a sub-field of computer science, generally associated with artificial intelligence [6]

Early approaches to error corrections applied dictionary-based spell-checking to OCR outputs [25]. Later research introduced statistical language models, such as n-grams, to evaluate the likelihood of word sequences [26]. Advances in NLP brought machine learning methods to post-OCR correction. Conditional Random Fields (CRFs) and Hidden Markov Models (HMMs) were employed to identify probable corrections [27].

A language model trained with self-supervised machine learning on a vast amount of text, designed for natural language processing tasks, especially language generation, known as large language model (LLM) was introduced for error correction [6]. The largest and most capable LLMs are generative pre-trained transformers (GPTs), based on a transformer architecture, which are largely used in generative chatbots such as ChatGPT, Gemini and Claude. The introduction of transformer-based language models, particularly OpenAI's GPT series and Google's BERT, revolutionized text correction and contextual understanding [8], [9]. Research shows that GPT outperform traditional grammar checkers by recognizing long-range dependencies and understanding semantic context [28]. Despite their power, LLMs also introduced challenges. They are computationally intensive, rely on external APIs (raising concerns about latency and costs), and may sometimes over-correct, changing words incorrectly if the context is ambiguous [29].

Several recent studies have attempted to combine OCR engines with NLP or AI correction systems, forming hybrid frameworks similar to OCRFlow. These studies include the following:

i. [30] proposed an intelligent document processing framework that integrates OCR with AI-based error correction, demonstrating significant reductions in error rates compared to OCR alone.
ii. [31] explored sequence-to-sequence models that simultaneously recognize and correct handwritten text, bypassing the need for separate OCR and correction modules.
iii. [32] described a pipeline using Google Vision OCR combined with BERT-based correction for processing historical archives, showing that hybrid approaches can outperform standalone OCR or NLP methods.

The previous works reviewed highlighted three key gaps that OCRFlow addresses:

i. Usability Gap: Most OCR + AI correction frameworks are research prototypes without user-friendly interfaces. OCRFlow provides a GUI (Tkinter) that integrates live capture, side-by-side preview, and save options.
ii. Correction Gap: Existing systems often restructure text during correction, introducing unintended meaning. OCRFlow explicitly constrains GPT to only correct spelling and grammar, ensuring authenticity of the handwritten content.
iii. Integration Gap: While OCR and AI have been studied separately, practical hybrid applications that combine cloud OCR + LLM correction + export options (Word/PDF) are scarce in real-world deployments. OCRFlow provides such integration.
iv. Accuracy Gap: Existing OCR systems struggle with handwriting variability

Methodology

Software Development methodology refers to structured processes involved when working on a project. The methodologies adopted for this project are Design Science Research (DSR) approach and Object Oriented Analysis and Design Methodology (OOADM). Design Science Research Methodology is suitable for artifact construction (such as models, methods, frameworks, or software systems) in computing research to solve real-world problems [33]. OCRFlow is a software artifact that integrates cloud-based Optical Character Recognition (OCR) with artificial intelligence (AI)-powered text correction in a graphical user interface (GUI). DSR is the methodology adopted for the design and development stage.

While OOAD is a new system development approach, encouraging and facilitating re-use of software components. It employs International Standard Unified Modeling Language (UML) from the Object Management Group (OMG). Using this methodology, a system can be developed on a component basis, which enables the effective re-use of existing components. It facilitates the sharing of its other system components. Object Oriented Methodology asks the analyst to determine what the objects of the system are? What responsibilities and relationships an object has to do with the other objects? How they behave over time?

Both methodologies were combined due to the following reasons:

i. Both the Object Oriented Analysis (OOAD) and design Methodology and the Design Science Research Methodology (DSRM) are used together to complement each other. While the DSRM is used as the research roadmap, the OOAD as the engineering toolkit within the roadmap.
ii. While the DSRM is used to structure the research (to design the problem, set objectives, and design and build the artifact, demonstrate, evaluate, and communicate results), OOAD ensures that the artifact (software) is well-structured, maintainable, and practical. As the design/development technique within DSRM, OOAD principles are applied to model and build the artifact (software system, prototype, framework)
iii. While OOAD provides tools (UML, diagrams, use cases, class, models, etc.) to ensure that the artifact is systematically designed, DSRM ensures that the research is valid as a contribution to knowledge
iv. Together, they cover both academic rigor (DSRM) and technical rigor (OOAD)

SYSTEM ANALYSIS

Analysis of the Existing System

Over the last two decades, research has focused on developing intelligent document processing pipelines capable of handling both printed and handwritten inputs. Comparative studies highlight the advantages and limitations of existing OCR tools in different domains. These include:

1. Some of the existing systems perform sub-optimally on handwritten documents without extensive pre-processing, e.g Tesseract. Researchers have integrated binarisation, skew correction, and noise reduction pipelines to enhance their performance. However, the improvements remain marginal when applied to unconstrained cursive handwriting.

2. Some of the existing Systems are commercial OCR engines known for high accuracy in printed text e.g ABBYY FineReader. They include handwriting support, licensing costs and closed-source architecture, limiting their adaptability for research and educational purposes.

3. End-to-end systems such as outperform traditional pipelines by learning holistic representations of handwritten text. However, these systems require large-scale training data and high-performance computing resources, limiting accessibility for non-specialist users.

4. Some recent research prototypes integrate OCR with NLP-based post-processing. These approaches demonstrate the feasibility of hybrid systems but often remain confined to academic experimentation, with limited usability for end-users. OCRFlow distinguishes itself by implementing these ideas into a functional desktop GUI.

Figure 1 shows the architecture of an existing system

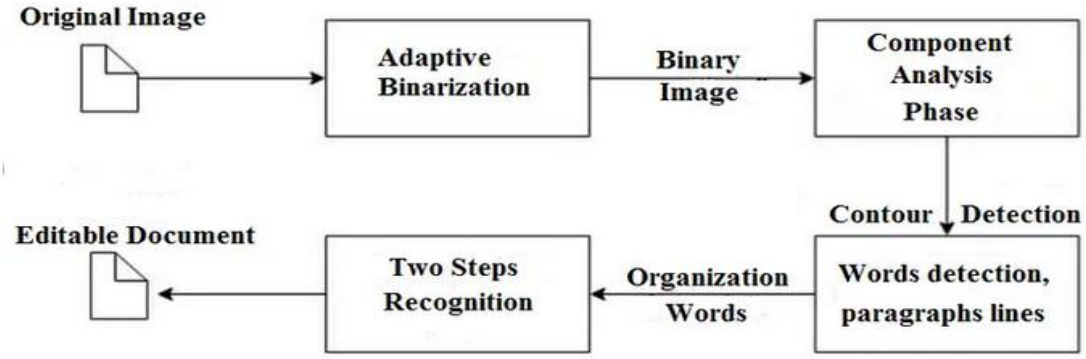

Figure 1: Tesseract OCR architecture

Analysis of the New System

The new system, OCRFlow, addresses these gaps by integrating Google Cloud Vision OCR, GPT correction, and a Tkinter GUI with Word/PDF export features. Building hybrid pipelines involves technical decisions regarding architecture, correction strategies, and user experience. While offline OCR engines like Tesseract are free, their accuracy on handwriting is limited. Google Cloud Vision OCR was selected for OCRFlow due to its superior handwriting recognition capabilities and support for multiple languages. The machine learning algorithm behind the Google Cloud vision OCR is the Deep Neural Network, specifically the Convolutional Neural Networks (CNNs) combined with Sequence Models. CNNs are the core algorithm for image feature extraction. They detect strokes, shapes, and patterns in scanned documents or handwritten text. They convert raw pixels into high-level features (edges, characters, word shapes, etc.). After the CNN extracted features, Recurrent Neural Networks (e.g., Long Short-Term Memory (LSTM)), are used to model text sequences. This is important because text has order or sequence (e.g., "on" and "no", "rat" and "tar", etc.). RNN helps to understand the flow of letters and words. The Connectionist Temporal Classification (CTC) loss function aligns predicted sequences with actual text, handling cases where spacing is irregular (common in handwriting or noisy scans), since OCR does not know where one character ends and another begins. Transformers and Attention Models help the model to focus on relevant parts of an image (like specific text regions) when recognizing characters.

Reasons for integrating Google Cloud Vision OCR and OpenAI in OCRFlow

Integrating Google Cloud Vision OCR with OpenAI GPT in OCRFlow has the following benefits:

1. High Accuracy Text Extraction: Google Cloud Vision is one of the most advanced OCR engines for extracting text from scanned documents, handwritten notes, images, and

PDFs. It supports multilingual text recognition and complex layouts like tables, forms, receipts, etc. This ensures that the first stage (raw text capture) is very accurate.

2. Intelligent Post-Processing and Error Correction: OCR often introduces errors (e.g., confusing "1" with "I" or misspellings in handwriting). OpenAI's language models can auto-correct spelling, grammar, and context-sensitive errors without losing meaning. For instance, OCR might read "decieve", OpenAI corrects to "deceive".
3. Contextual Understanding: Google OCR extracts text as it is, but OpenAI can interpret meaning. For instance:
   - OCR extracts "Total Purchase: #22.50, Subtotal: #21.20, Tax: #1.30
   - OpenAI can summarize it as: "The total purchase was #22.50 including tax"
   4. Natural Language Enhancement: OpenAI can summarize, translate, or reformat the OCR text.

In summary, while OCR extracts the raw text, OpenAI cleans, corrects, understands, and presents text leading to higher accuracy, context-awareness, and user-friendly OCR system.

Interface design

User interface design is critical in document processing systems. OCRFlow adopts a three-panel layout showing:

1. Live camera preview (capture input).
2. Raw OCR output.
3. Corrected AI output.

This ensures transparency, allowing users to validate corrections before saving.

Export options

Most research prototypes stop at text correction. OCRFlow integrates python-docx and ReportLab for direct export into Word and PDF formats. This feature enhances usability in professional and academic contexts

Strengths of OCRFlow

The evaluation highlights several strengths:

1. Hybrid Accuracy: Integration of Vision OCR and GPT correction significantly reduces errors.
2. Usability: GUI design with side-by-side comparison enhances user trust.
3. Export Functionality: Direct Word and PDF export aligns with user needs in academia and business.
4. Lightweight Design: Runs efficiently on mid-range hardware.
5. Reproducibility: Code structure with .env and organized folders enhances replicability.

Weaknesses and limitations

Despite these strengths, OCRFlow also presents weaknesses:

1. Internet Dependence: Both OCR (Google Vision) and correction (GPT) require constant connectivity.
2. Billing Requirements: Google Cloud Vision API requires billing activation, limiting accessibility in resource-constrained settings.
3. Language Limitations: Evaluation focused solely on English; multilingual capabilities remain untested.
4. No Batch Processing: Current design processes only one page at a time.
5. Potential GPT Over-Correction: In rare cases, GPT introduced subtle lexical substitutions.

6. Scalability: The current system processes single pages sequentially, lacking batch-processing features.
7. Domain Specificity: GPT was not fine-tuned for specialized domains (e.g., medical or legal handwriting).

Application Contexts of the New System

Hybrid OCR-AI System can be applied in the following fields:

1. Education: OCR-AI pipelines have potential to digitize handwritten examination scripts, classroom notes, and archival records. A study by [34] emphasizes how digitization improves accessibility for students with disabilities. OCRFlow could be deployed in universities to convert scanned lecture notes into shareable, corrected documents.
2. Healthcare: Medical transcription remains a high-risk task where errors can affect patient safety. Integrating OCR with AI-driven correction can reduce medication errors and improve record-keeping [35]. OCRFlow demonstrates the feasibility of low-cost hybrid solutions in this space, though domain-specific fine-tuning would be necessary.
3. Government Administration: Governments, particularly in developing countries, often rely on handwritten forms for census, taxation, and welfare distribution. Intelligent OCR systems can accelerate data entry and reduce corruption by ensuring accuracy [36]. OCRFlow's lightweight design suggests adaptability for such contexts.

5. SYSTEM DESIGN

The system is composed of five core modules (Figure 8):

1. Capture Module (OpenCV): Captures images of handwritten documents using a laptop camera.
2. OCR Module (Google Cloud Vision): Performs text extraction from captured images.
3. Correction Module (OpenAI GPT): Corrects spelling and grammar errors without restructuring sentences.
4. GUI Module (Tkinter): Displays live video preview, raw OCR output, and corrected output side by side.
5. Export Module (python-docx & ReportLab): Saves corrected text as Word or PDF documents.

See figure 2 below for the OCRFlow system architecture

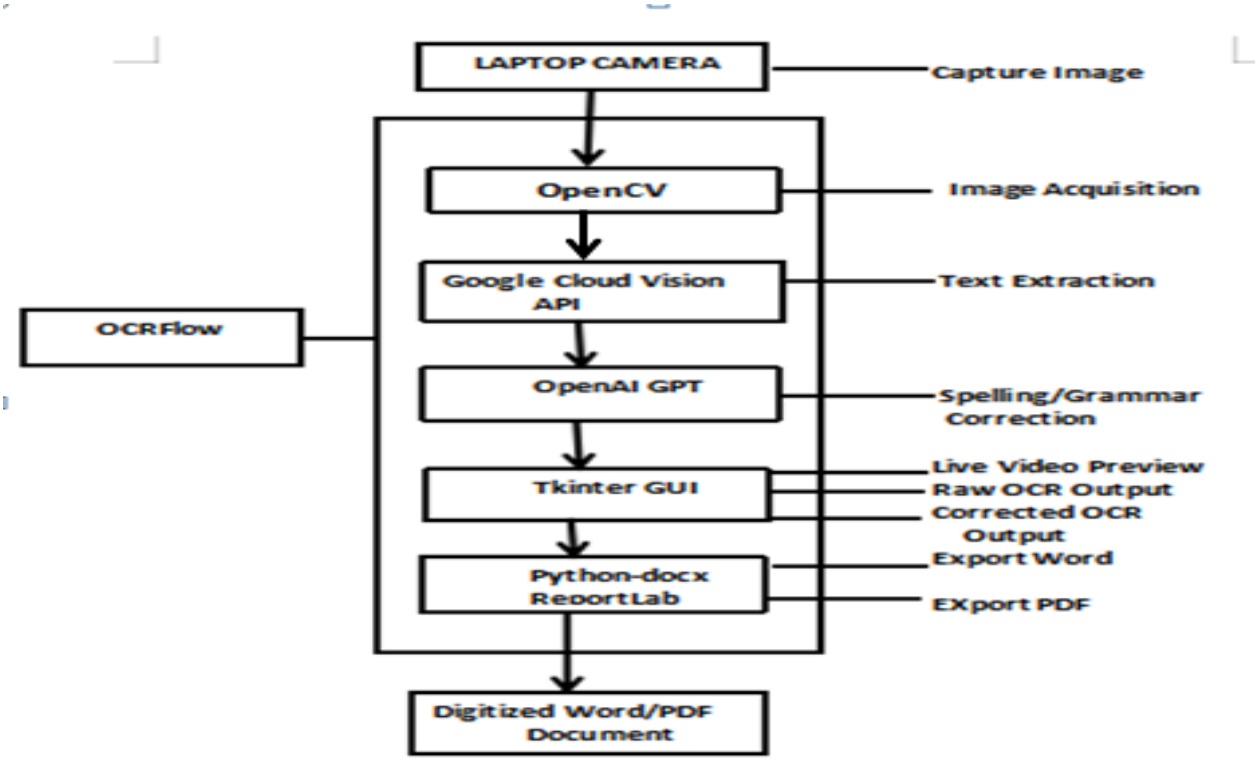

Figure 2: OCRFlow System Architecture

Use Case of the Proposed System

Use Case diagrams describe what a system does from the standpoint of an external observer. The emphasis of use case diagrams is on what a system does rather than how. They are used to show the interactions between users of the system and the system. A use case represents the several users called actors and the different ways in which they interact with the system. See figure 3 below

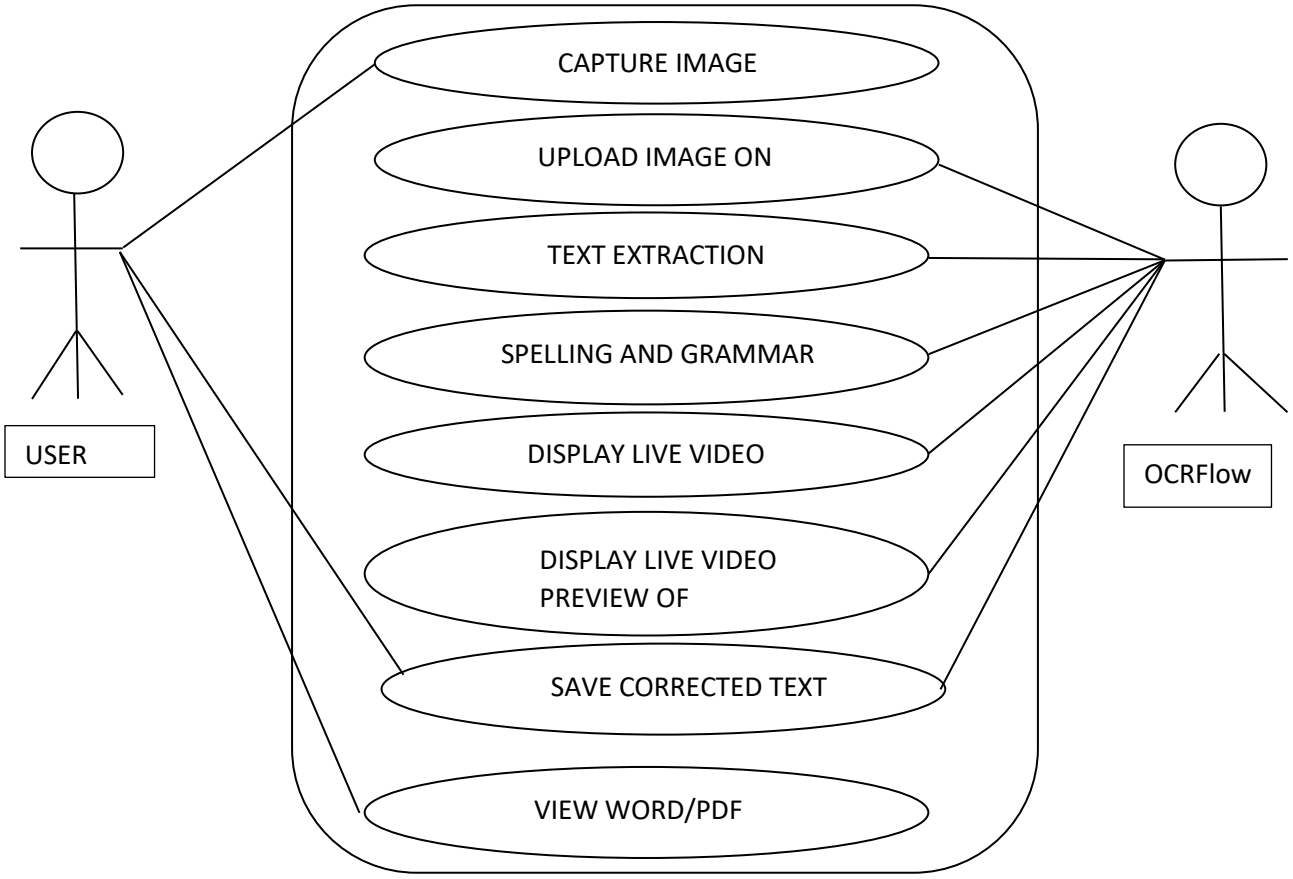

Figure 3: Use case of the Proposed System

User interface implementation

The interfaces for the system were implemented as shown in the figures below

1. Capture module (OpenCV)

The capture module uses the OpenCV library (cv2.VideoCapture) to access the laptop's webcam. The user interface provides a preview panel, allowing users to align the handwritten document. On pressing the "📸 Capture" button, a snapshot is taken and saved to the CaptureImage folder as captured.jpg.

The module ensures that if a previous image exists, it is overwritten, maintaining storage efficiency. See figure 4 below

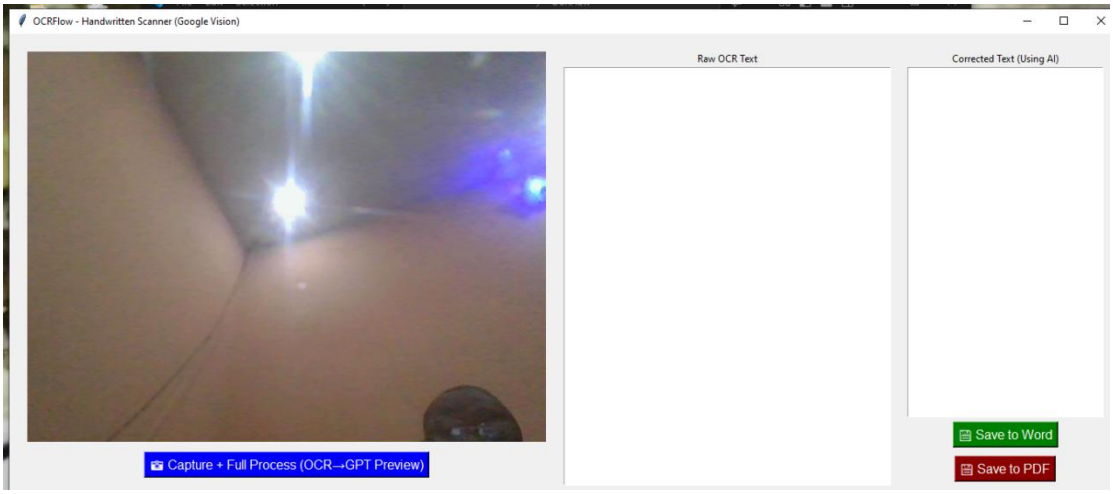

Figure 4: Interface for Image capture

2. OCR module (Google Cloud Vision API)

The Google Cloud Vision API was chosen due to its superior performance on handwritten text compared to open-source OCR engines such as Tesseract. This is because:

i. The captured image is sent to the Vision API via the Python client library.
ii. The method document_text_detection() returns structured annotations, including detected words and layout.
iii. Extracted text is returned as raw OCR output and displayed in the Raw OCR Text Box in the GUI.

This module requires authentication via a service account JSON key, securely stored in the system. See figure 5 below

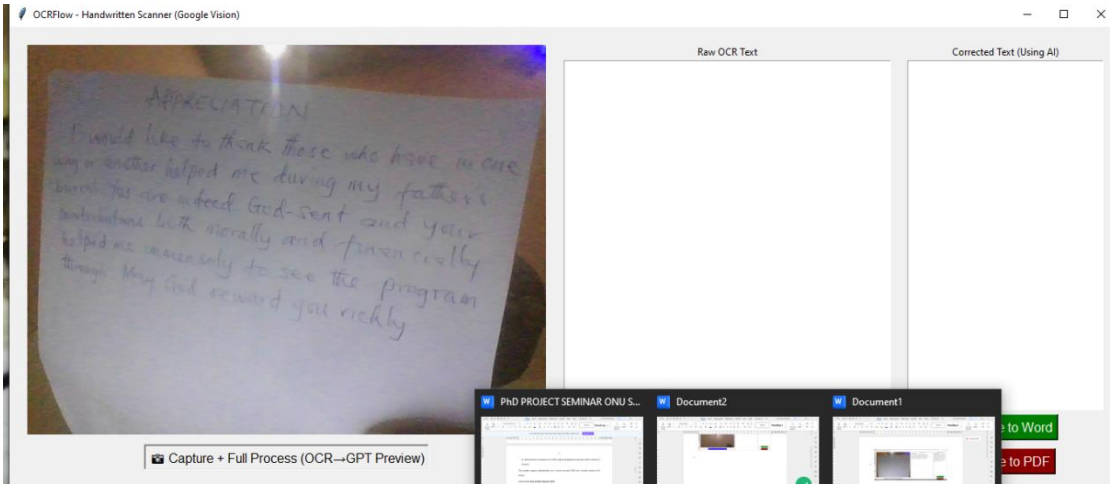

Figure 5: Interface for captured image

3. Correction module (OpenAI GPT)

The raw OCR output is often noisy, with spelling mistakes and grammatical inconsistencies. To address this, the OpenAI GPT API is employed thus:

i. The GPT model (gpt-4o-mini) is prompted with the raw OCR text.
ii. The system prompt is carefully engineered:

"You are an OCR text corrector. Correct miss-spelt words and fix grammar mistakes. Do NOT enhance or rewrite the text. Keep the original sentence structure and meaning intact."
This ensures corrections remain faithful to the original handwritten meaning. The corrected text is displayed in the Corrected Text Box in the GUI. See figure 6 below

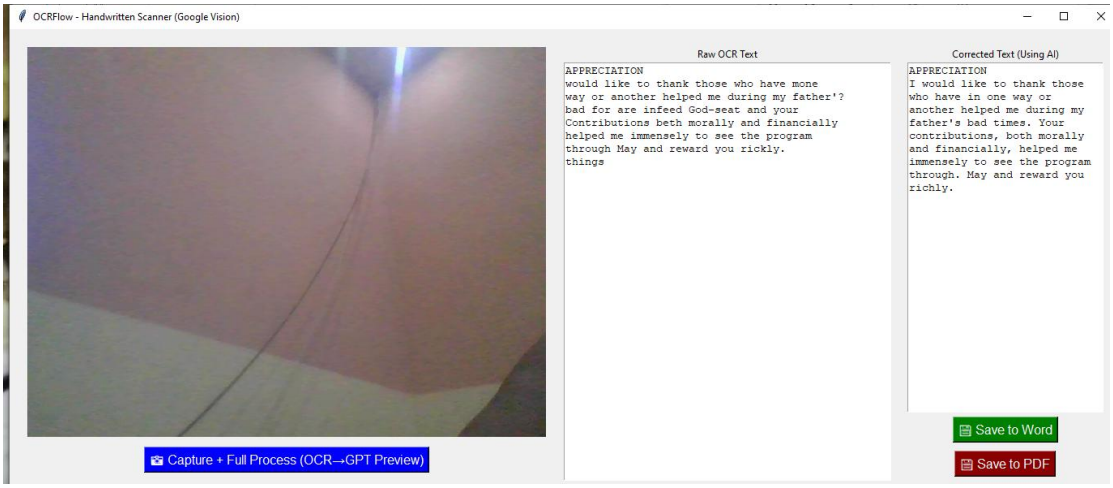

Figure 6: Interface for Raw and corrected Text

4. GUI module (Tkinter)

The interface was designed using Tkinter to balance functionality and simplicity, as stated below:

i. Left Panel: Live camera preview with capture button.
ii. Center Panel: Raw OCR text, uneditable, for transparency.
iii. Right Panel: Corrected text with Save to Word and Save to PDF buttons.

This three-panel design ensures usability, enabling users to see both raw and corrected outputs before deciding to save. See figure 7 below

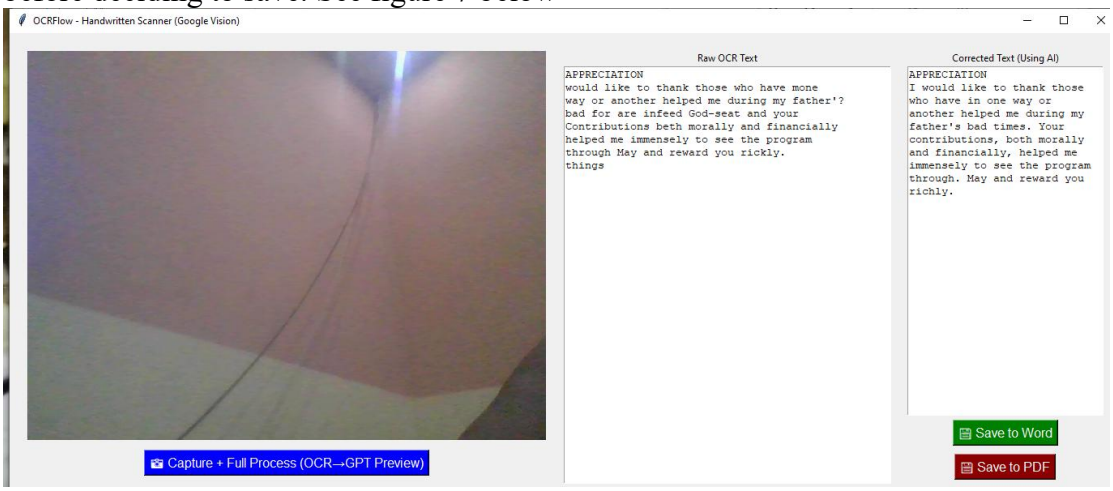

Figure 7: Interface for GUI Module

5. Export Module (Word/PDF)

Two export functionalities were implemented:

i. Word Export: Uses python-docx to create a .docx file in the Word Document folder, overwriting existing versions.

ii. PDF Export: Uses ReportLab to generate a .pdf file in the PDF folder, overwriting existing versions. See figure 8 below

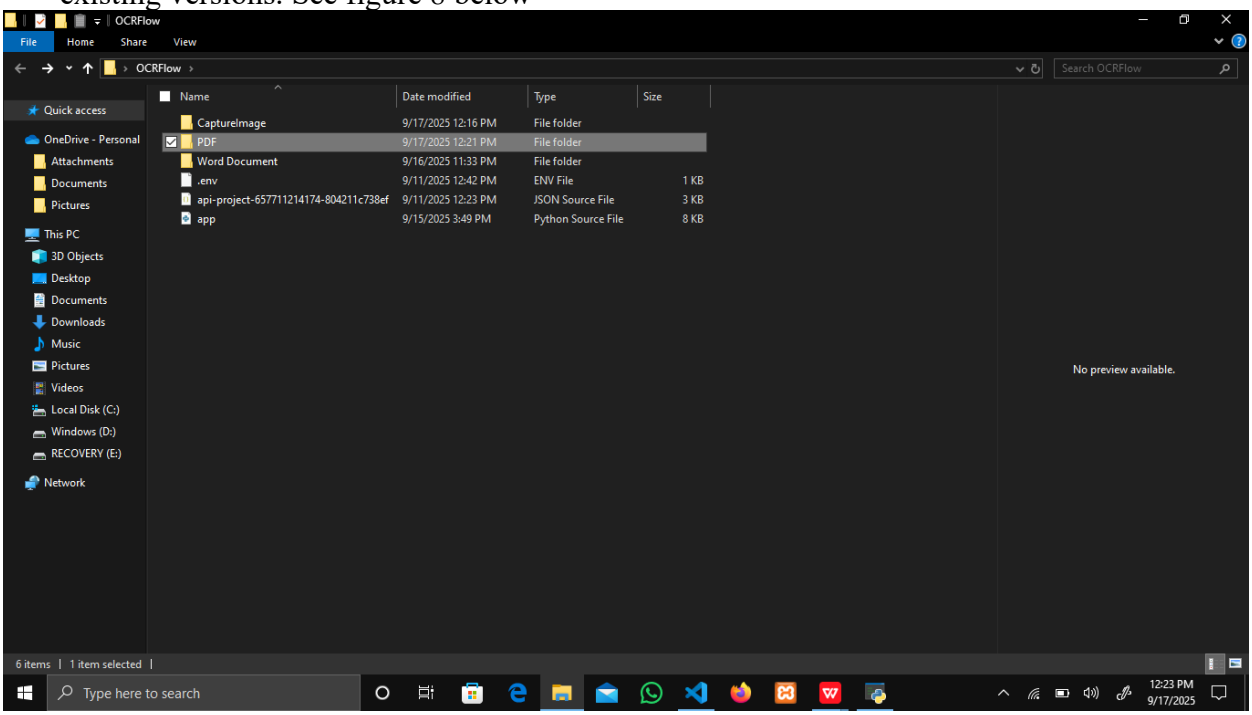

Figure 8: Interface for Word/PDF Digital Copy

Both exports include a header ("Digital Copy of Handwritten Document") followed by the corrected text. See figure 9 below

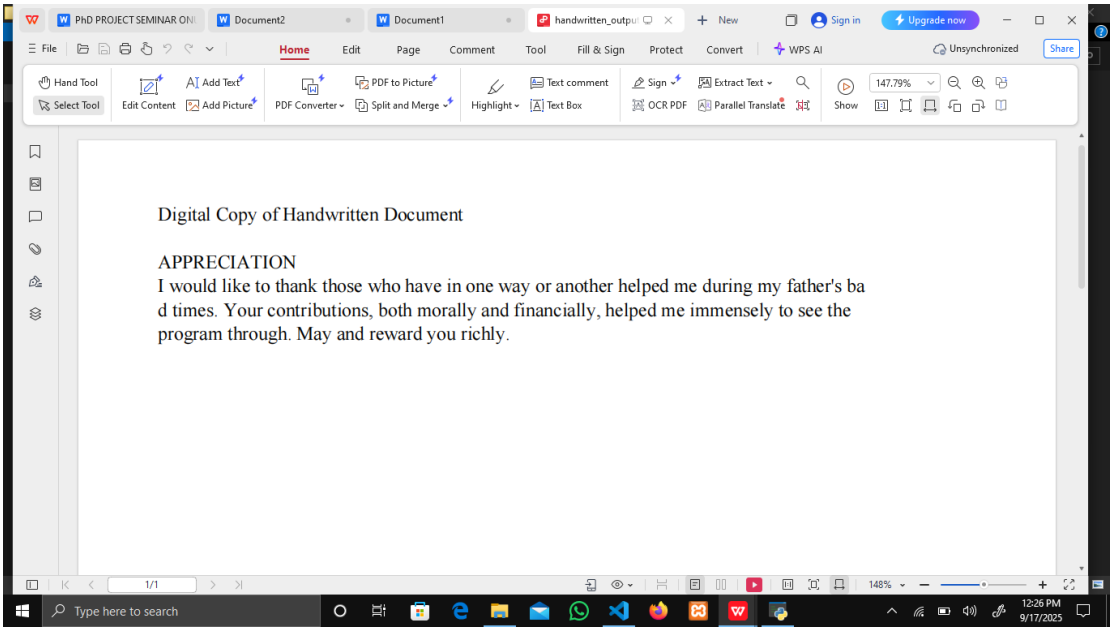

Figure 9: Digital Copy of Handwritten Document

Conclusion

In conclusion, the research identified the limitations of existing OCR systems in handling handwritten documents. While tools such as Tesseract offer robust recognition for printed text, their accuracy on cursive handwriting remains low. Cloud-based OCR engines like Google Vision provide improvements but still produce outputs with spelling and grammatical errors.

OCRFlow, as a solution, was designed with the following modules:

1. Capture Module (OpenCV): Enables real-time capture of handwritten documents via webcam/laptop camera.
2. OCR Module (Google Vision): Extracts raw text from captured images.
3. Correction Module (GPT): Corrects spelling and grammar errors without restructuring meaning.
4. GUI Module (Tkinter): Displays raw and corrected text side by side.
5. Export Module (Word/PDF): Saves corrected text for reuse in academic and professional contexts.

After testing the modules, results confirm that OCRFlow successfully addresses the challenges of handwritten OCR by combining Google Vision OCR with GPT correction in a transparent and usable interface. Compared to existing systems, OCRFlow offers a balanced approach that prioritises accessibility and accuracy over cutting-edge but resource-intensive models.

While limitations exist, particularly concerning internet dependence and billing, OCRFlow represents a practical, reproducible, and deployable solution to the problem of handwritten document digitization.

REFERENCES

1. Oche, E. S. (2017). The Influence of Poor Handwriting on Students Score Reliability in Mathematics. Mathematics Education Trends and Research, 1-15.
2. Chescoe David (2017, May 3rd). What Problem Can Handwriting Cause in Digital Age https://www.aacsystems.co.uk/handwriting-recognition-challanges/
3. Plamondon R & Srihari S N (2000). On-Line and Off-Line Haandwriting Recognition: A Comprehensive Survey. *IEEE Transaction on Pattern Analysis and Machine Intelligence*. Vol. 22(1), pp 63-84. Doi: 10.1109/34.824821
4. Smith, R. (2007). "An Overview of the Tesseract OCR Engine*". 9th International Conference on Document Analysis and Recognition (ICDAR) 2007*. Vol. 2, pp 629-633. DOI: 10.1109/ICDAR.2007.4376991
5. Graves, A., Liwicki, M., Fernandez, S., Bertolami, R., Bunke, H. and Schmidhuber, J., (2009). A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5), pp.855–868.
6. Wikipedia, (2025). Natural language processing [online] Available at: https://wikipedia.org/wikis/natural_language_processing/ [Accessed 15 September 2025].
7. Eisenstein, Jacob (October 1, 2019*).* Introduction to Natural Language Processing. *The MIT Press. p. 1.* ISBN 9780262042840.
8. Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K., (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of NAACL-HLT 2019*, pp.4171–4186.
9. Brown, Tom B, et al (ed) (2020*).* "Language Models are Few-Shot Learners" (PDF). *Advances in Neural Information Processing Systems. 33. Curran Associates, Inc.:* 1877*–1901.* Archived (PDF) *from the original on 2023-11-17*. Retrieved 2025-09-15.

DOI: 10.48550/arXiv.2005.14165
10. Handel, S., (2011). A History of the Development of OCR Technology. *International Journal of Document Analysis and Recognition*, 14(2), pp.67–84.
11. Mori, S., Suen, C. and Yamamoto, K., (1992). "Historical Review of OCR Research and Development". *Proceedings of the IEEE*, 80(7), pp.1029–1058.
12. Patel, C. Patel, Atul; & Patel Dharmendra (2012). Optical Character Recognition by Open Source OCR Too, Tesseract: A Case Study. *International Journal of Computer Application*. Vol. 55, Number 10, pp 50-56.
13. Kay, S. and Kim, H., (2018). Extending Tesseract for Cursive Handwriting Recognition. *Journal of Imaging Science and Technology*, 62(4), pp.405–414.
14. ABBYY, 2020. *ABBYY FineReader OCR*. [online] Available at: https://www.abbyy.com/finereader/ [Accessed 11 September 2025].
15. Li, M., Lv, T., Cui, L., Lu, Y., Florencio, D., Zhang, C., et al., (2021). TrOCR: "Transformer- based OCR with Pre-trained Vision and Language Models". *Proceedings of ICCV 2021*, pp.7464–7475.
16. LeCun, Y. *et al* (1998). Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE,* 86(11), 2278-2324 https://doi.org/10.1109/5.726791
17. Graves, A., Liwicki, M., Fernandez, S., Bertolami, R., Bunke, H. and Schmidhuber, J., (2009). A novel connectionist system for unconstrained handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(5), pp.855–868.
18. GeeksforGeeks (13 August, 2025) What is LSTM - Long Short Term Memory?[Online] available at https://www.geeksforgeeks.org/deep-learning/deep-learning-introduction-to-long-short-term-memory/ *[Accessed 17th September, 2025]*
19. Ciresan, D.C., Meier, U., Gambardella, L.M. and Schmidhuber, J., (2012). Deep, big, simple neural nets for handwritten digit recognition. *Neural Computation*, 22(12), pp.3207–3220.
20. Hochreiter, Sepp*;* Schmidhuber, Jürgen *(1997). Long Short-Term Memory. Neural Computation. 9(8):* 1735*–1780.* doi*:*10.1162/neco.1997.9.8.1735. ISSN 0899-7667
21. Alawad, M., Li, Y., Eskenazi, M. and Si, L., (2020). Evaluation of Google Cloud Vision API for OCR tasks on handwritten documents. *Proceedings of LREC 2020*, pp.4801–4807.
22. Amazon, (2022). *AWS Textract*. [online] Available at: https://aws.amazon.com/textract/ [Accessed 11 September 2025].
23. Microsoft, (2023). Azure Cognitive Services – Computer Vision. [online] Available at: https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/ [Accessed 11 September 2025].
24. Shafait, F. *et al* (2008). Document Cleanup Using Page Frame Detection. *International Journal on Document Analysis and recognition (IJDAR).* Vol. 11, Number 2, pp 81-96. DOI: 10.1007/s10032-008-0071-7
25. Kukich, K., (1992). Techniques for Automatically Correcting Words in Text. *ACM Computing Surveys*, 24(4), pp.377–439.
26. Manning, C.D. and Schütze, H., (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press.
27. Kolak, O. and Resnik, P., (2002). OCR Error Correction Using a Noisy Channel Model. *Proceedings of HLT 2002*, pp.257–262.

28. Wu, J. et al (2022). Training Language Models to follow Instructions with Human Feedback. DOI: 10. 48550/arXiv.2203.0215
29. Floridi, L. and Chiriatti, M., (2020). GPT-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30(4), pp.681–694.
30. Gupta, A., Sharma, P. and Singh, R., (2022). Intelligent Document Processing Using AI: A Survey. *IEEE Access*, 10, pp.45789–45802.
31. Wigington, C. et al (2017). Data Augmentation for Recognition of Handwritten Words and Lines Using a CNN-LSTM Network. *International Conference on Document Analysis and recognition.* Pp 639-645. DOI: 10.1109/ICDAR.2017.110
32. Kae, A., Chen, S. and Zhao, H., (2021). OCR and AI Correction for Historical Documents: A Hybrid Approach. *Proceedings of ICDAR 2021*, pp.350–357.
33. Hevner, A.R., March, S.T., Park, J. and Ram, S., (2004). Design science in information systems research. *MIS Quarterly*, 28(1), pp.75–105.
34. Al-Azawei, A., Serenelli, F. and Lundqvist, K., (2019). Universal design for learning (UDL): A content analysis of peer-reviewed journal papers from 2012 to 2015. *Journal of the Scholarship of Teaching and Learning*, 16(3), pp.39–56.
35. Kushniruk, A., Borycki, E., Kuwata, S. and Kannry, J., (2018). Predicting and Preventing Technology-induced Errors in Healthcare: The role of human factors. *Healthcare Quarterly*, 21(3), pp.25–30.
36. Heeks, R., (2002). Information systems and developing countries: Failure, success, and local Improvisations. *The Information Society*, 18(2), pp.101–112.