Automated Administrative Document Processing

Arkadii Shekhovtsov
Department of Business
Univ. of Europe for Applied Sciences
Potsdam 14469, Germany
sh.ark1843@gmail.com

Raja Hashim Ali
Department of Business
Univ. of Europe for Applied Sciences
Potsdam 14469, Germany
hashim.ali@ue-germany.de

Talha Ali Khan
Department of Business
Univ. of Europe for Applied Sciences
Potsdam 14469, Germany
talhaali.khan@ue-germany.de

***Abstract*—Receipt processing is a frequent yet time-consuming task for both personal use and business reporting. Automating this process, especially when people have the opportunity to mark items by hand, helps save time and avoid errors in expense tracking.**

Most receipt scanning systems do not support user-added marks such as hand-drawn checkmarks and do not offer customization options. This project addresses this by building a system that detects such marks and uses them to generate the final list of expenses.

The system combines OCR tools (Tesseract and EasyOCR) with an object detection model (YOLOv8) to extract both text and checkmarks from receipt photos. A custom dataset with real and synthetic checkmarks was created to train the detection model for accurate mark recognition.

The final pipeline correctly extracted structured data from 57 percent of real-world receipts. While this may not seem impressive, the problems in the remaining 43 percent are clear and can be fixed without changing the original project requirements or limits.

Overall, this work presents a modular, open-source pipeline that combines OCR and object detection to accurately extract structured data from marked paper receipts. The full pipeline implementation is available as a Kaggle notebook at: https://www.kaggle.com/code/arka2711/receipt-recognition-pipeline.

***Index Terms*—receipt processing, OCR, checkmark detection, YOLOv8, Tesseract, EasyOCR, document automation, expense extraction**

I. Introduction

Automated receipt processing is useful for tasks such as tracking expenses or preparing invoices. Typically, processing such information requires manual data entry, which is inconvenient and sometimes time consuming. People often just want to take a photo of a receipt and mark certain items to include, exclude, or classify them.

In this work, I used a combination of OCR and deep learning-based object detection to extract structured data from marked receipts. Tesseract and EasyOCR were applied for text extraction. A YOLOv8 model was trained to detect hand-drawn checkmarks, allowing users to exclude items directly on the paper. This approach enabled flexible and accurate receipt processing without relying on fixed templates.

A. Gap Analysis

Despite ease of use, most existing commercial platforms are 'black-box' solutions. They rarely support automated recognition of user-added checkmarks. Customization is typically limited to the template configuration. Moreover, some platforms 'go a step further' by human verification of all extracted data that does not match the word 'automation' at all.

Fig. 1. Receipt with correctly recognized item prices (orange boxes). A user-drawn checkmark was successfully detected (green boxes), while a printed letter 'w' was incorrectly classified as a checkmark.

B. Research Questions

1) RQ1: Is it possible to build a complete pipeline that extracts structured data from receipts with user-drawn checkmarks?
2) RQ2: Which object detection model performs best for recognizing hand-drawn checkmarks?
3) RQ3: Which OCR setup performs best for extracting text from real-world receipts?

The main contribution of this work is a modular pipeline that combines OCR and object detection to support flexible user-guided receipt parsing. Unlike most existing systems, this approach allows users to manually mark relevant items on receipts and processes them automatically without requiring fixed templates.

C. Problem Statement

This study solves the problem of lacking a tool that can automatically process receipts with user-drawn checkmarks and can be used both for business and personal tasks. While many systems extract receipt information, none allow users to mark specific items on the paper. The aim was to build

a system that fills this gap by combining text recognition and mark detection to give users control over how the data is processed.

D. Novelty of this study

This study presents a unique approach to receipt processing by enabling user-driven item selection through hand-drawn checkmarks, a feature not supported by existing systems. To address the identified gaps, the following novel contributions were made:

- Developed a complete end-to-end pipeline that combines OCR and object detection to extract structured data from receipt images with user annotations.
- Created a synthetic and real-world dataset of checkmarks for training and evaluating object detection models.
- Achieved reliable results without relying on fixed templates, making the system more flexible for varied receipt layouts and use cases.

E. Significance of Our Work

This work introduced a complete pipeline for processing receipt images with user-drawn checkmarks by combining OCR (Tesseract and EasyOCR) with object detection (YOLOv8). The system was trained and tested on both synthetic and real-world data, and it successfully extracted structured data in 57 percent of test cases. Although this result is not very high, the errors are well understood and can be fixed without changing the original project scope. Overall, this is a working prototype that shows that the method is effective and can be further improved.

II. Literature Review

Various techniques have been explored in the field of document and receipt processing, particularly for text extraction and visual mark detection. [1] [2] Traditional OCR engines like Tesseract use rule-based layout analysis and character pattern matching, while modern alternatives such as EasyOCR and TrOCR apply deep learning to handle more diverse fonts and noisy inputs. These OCR systems often follow a stage-wise pipeline involving preprocessing, text line segmentation, and character recognition [1]. End-to-end frameworks like Donut and PaddleOCR have further improved document understanding by integrating layout and content parsing in a single model. For visual element detection, object detection models like YOLOv8, Faster R-CNN, and SSD have been used, although prior work has rarely combined them with OCR to support user-drawn marks like checkmarks on receipts.

A. Optical Character Recognition (OCR)

Optical Character Recognition (OCR) is a technique used to convert printed or handwritten text in images into machine-readable form. It plays a central role in document processing tasks such as receipt parsing, form extraction, and invoice analysis [1] [3]. Several OCR systems have been developed, each following different architectural principles and offering varying levels of robustness and flexibility. Tesseract is a classical OCR engine that uses layout analysis and character pattern recognition to identify text in structured documents. [4] EasyOCR applies a deep learning-based approach, combining the CRAFT detector with a CRNN-based recognizer to handle more complex and varied layouts. [5] TrOCR is a transformer-based model that treats OCR as a sequence-to-sequence problem and has been proposed for recognizing both printed and handwritten text. [6] These systems represent different architectural approaches to OCR and are commonly used in document analysis pipelines. [7] [8]

B. Object Detection for Checkmark Recognition

Object detection is a computer vision technique used to locate and classify objects within an image. It is commonly based on convolutional neural networks and has been used in tasks such as document layout analysis, form parsing, and symbol recognition. YOLO (You Only Look Once) models are single-shot detectors known for their speed and efficiency in real-time applications [9]. Two-stage detectors like Faster R-CNN provide higher accuracy by generating region proposals before classification [10]. SSD (Single Shot Multibox Detector) offers a compromise between speed and accuracy and has been applied in various small-object detection tasks [11]. These methods provide the basis for detecting hand-drawn marks or other visual elements in scanned documents.

See Table I for a comparison of object detection methods.

C. End-to-End Document Understanding

End-to-end document understanding refers to systems that extract structured information from documents without separating OCR and layout analysis into distinct stages. These models often combine visual features with language modeling to interpret both content and structure. Donut is a recent example of such a system, using a transformer-based encoder-decoder architecture to process documents directly as images and generate structured outputs without relying on traditional OCR. [12] PaddleOCR is another modular framework that integrates detection, recognition, and post-processing components, supporting multilingual text and key-value extraction. [5] These systems aim to simplify document pipelines by minimizing intermediate steps and leveraging large pretrained models. They are commonly used in scenarios where documents follow semi-structured formats, such as forms, invoices, or receipts.

III. Methodology

The proposed system combines OCR and object detection to extract structured data from receipt images containing user-drawn checkmarks. The workflow consists of image preprocessing, text extraction using two OCR engines, and checkmark detection using a YOLOv8 model. The results from both components are merged and filtered to build a structured list of selected items. An overview of the pipeline is shown in Fig. **??**.

TABLE I
COMPARISON OF KEY DEEP LEARNING OBJECT DETECTION METHODS

Method	Architecture Summary	Speed / Accuracy	Suitable?
R-CNN	2-stage: region proposals + CNN + SVM classification	∼0.03 fps, mAP = 66%	No: too slow
Fast R-CNN	2-stage: shared conv layers, RoI pooling, Softmax	∼0.6 fps, mAP = 70%	No: still slow
Faster R-CNN	2-stage: RPN + end-to-end training	2–9 fps, mAP = 70–78%	Yes
YOLO (v1–v3)	1-stage: grid-based regression, real-time	40–60 fps, mAP = 57–80%	No: outdated
YOLO (v8)	1-stage: anchor-free, decoupled head	50–120 fps, mAP = 85–90%	Yes
SSD	1-stage: multi-scale feature maps	19–46 fps, mAP = 74–82%	Yes
Mask R-CNN	2-stage: Faster R-CNN + masks	5–6 fps, mAP = 78–82%	No: for segmentation

A. Datasets

Checkmark Dataset: To train the checkmark detection model, a dataset was created using both real and synthetic images. Handmade dataset was created by manually collecting and annotating images of checkmarks. Each image was labeled in YOLO format, specifying the coordinates of each checkmark using bounding boxes. Manual annotation was performed using makesense.ai, a web-based tool that supports direct export to YOLO format 2. To further increase the size and variability of the dataset, a synthetic checkmark generator was implemented in Python. The generator creates images by drawing checkmarks with randomized parameters: starting position, length and angle of each line and thickness. 3

The entire dataset was also converted to COCO format for compatibility with different detection frameworks. After combining all sources and augmentations, the final dataset consisted of about 400 training checkmarks and 100 validation checkmarks. Each image is accompanied by a YOLO-format annotation file and COCO annotations.

Receipt Dataset: To evaluate the full pipeline, a test set of 14 real-world receipt images was collected. The images were manually marked and grouped into three levels of difficulty.

1) Easy: 1 or 2 items. No prices inside item names and no discounts (no negative prices).
2) Medium: Fewer than 10 items. May include prices inside item names.
3) Hard: 10 or more items, or includes both prices in names and discounts.

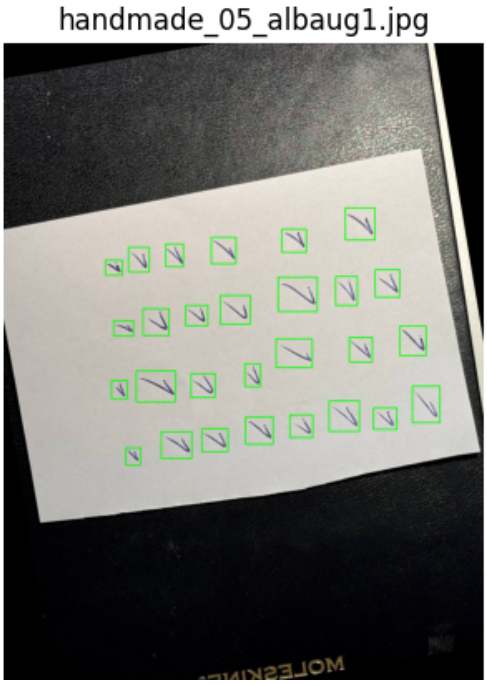

Fig. 2. Examples of handmade checkmarks and bounding boxes.

Fig. 3. Examples of synthetically generated checkmarks and bounding boxes.

B. Checkmark Recognition Models Comparison

Three object detection models were evaluated for the task of checkmark recognition: YOLOv8n, Faster R-CNN, and SSD300-VGG16. YOLOv8n, a lightweight anchor-free one-stage detector, was trained for 20 epochs on 640x640 images using the Ultralytics API. It showed excellent performance and fast training thanks to built-in augmentations and adaptive learning. Faster R-CNN with ResNet50-FPN backbone was trained for 5 epochs with the Adam optimizer and small batch size due to its high memory demand. SSD300-VGG16, a balanced model between speed and accuracy, was trained for 10 epochs under similar settings. All models were trained on the same dataset and evaluated using mAP@0.5, precision, and recall.(See Table II for a summary of the evaluation results.) YOLOv8n significantly outperformed the others, especially on real hand-drawn checkmarks, and was chosen as the primary model for integration in the final pipeline.

TABLE II
CHECKMARK RECOGNITION MODELS COMPARISON

Model	mAP@0.5	Precision	Recall
YOLOv8n	0.995	0.999	1.000
SSD300-VGG16	0.693	0.693	0.650
Faster R-CNN	0.296	0.296	1.000

C. Text Recognition Models Evaluation

Several OCR and document understanding models were evaluated for extracting structured data from receipts. Donut, an end-to-end transformer model that bypasses traditional OCR, showed initial promise but failed in practice, misclassifying simple elements and omitting key items without fine-tuning, which limits its applicability to diverse receipt formats. TrOCR, known for its robustness and high text recognition accuracy, also underperformed, often returning empty or unusable outputs due to its lack of receipt-specific training. PaddleOCR, despite its flexibility and modern architecture, similarly produced no usable results without further configuration or fine-tuning. In contrast, Tesseract demonstrated solid baseline performance, accurately detecting prices and bounding boxes, though it misinterpreted checkmarks and favored raw over preprocessed images. EasyOCR performed consistently well with real-world receipts, enabling effective item extraction through bounding box post-processing. A combination of Tesseract and EasyOCR proved sufficient for reliable receipt text extraction in the initial phase, with room for further enhancement by incorporating or fine-tuning additional models.

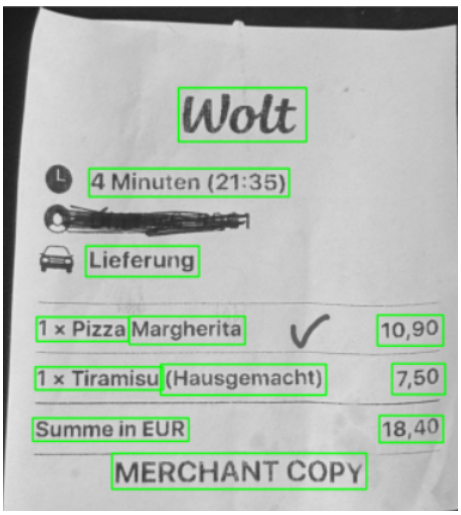

Fig. 4. Example of receipt text extraction using EasyOCR. Detected text elements, including item names, prices, and labels, are highlighted with bounding boxes.

D. Full processing pipeline

The processing pipeline for automated administrative document recognition consists of several main stages, each responsible for a specific part of image analysis and data extraction. The system requires a photo of a receipt with the specified check amount and transforms it into a structured digital table. The implementation is modular, allowing each stage to be improved or adapted independently.

The first stage is text extraction. The system initially attempts to extract data using Tesseract OCR. If the result is incomplete or invalid, EasyOCR is used as a fallback to improve reliability. This process is first applied to the raw receipt image. If the sum of the calculated elements does not match the real total, the system repeats the same sequence on a preprocessed version of the image. If both attempts fail, the pipeline stops to avoid unreliable output.

The next stage is checkmark detection. The image that successfully passed validation is analyzed using YOLO to locate user-drawn checkmarks. Detected checkmarks are then matched to corresponding items in the recognized table.

In the final stage, marked items are filtered out to produce a list of considered expenses. The total sum of these items is compared to a final value to determine whether the recognition was successful. An overview of this process is shown in Fig 6. An example of recognition results is shown in Fig 5.

	Item	Count	Price	bbox price	is_marked
0	B		4.79	[643, 665, 699, 691]	False
1	ENTDECKER MIX 4 B		3.99	[644, 693, 699, 719]	True
2	CASHEW-CRANBERRY 4 B		4.29	[643, 721, 699, 747]	True
3	BUTTER B		3.59	[644, 750, 699, 780]	False
4	BROT CLASSIC GF B		3.29	[645, 778, 699, 804]	False
5	HORCESTERSAUCE 7°		2.19	[644, 807, 728, 837]	True
6	EXCELLENCE 100% B		3.99	[644, 835, 699, 865]	False
7	SPEZI ZERO v A		0.85	[644, 863, 699, 888]	True
8	PFAND 0, 25 EURD A x		0.25	[644, 891, 699, 916]	False

```
------------------------------
The result is Success
Real/Predicted final price: 15.91/15.91
```

Fig. 5. Output of the receipt recognition pipeline showing detected items, prices, bounding box coordinates, and whether each item was marked (e.g., with a checkmark). The total predicted price matches the real total, indicating a successful extraction.

E. Evaluation Metrics

To evaluate the effectiveness of the pipeline, specific metrics were defined for each of its major components: text extraction, checkmark detection, and the full end-to-end process. These metrics reflect real-world performance and were used to compare methods and select the best configuration during development.

For OCR, the main metric was Price Recognition Accuracy. A strict total sum check was used. The sum of all detected prices was compared to the total printed on the receipt. If the values matched exactly, the receipt was considered successfully processed at this stage.

Checkmark detection was evaluated using standard object detection metrics: precision, recall, and mean average precision (mAP) at an IoU threshold of 0.5. These metrics were calculated using manually annotated checkmark positions from both synthetic and real receipts.

At the pipeline level, the key metric was the percentage of receipts where the final calculated sum (after excluding marked items) matched the expected ground truth value. Additionally, an error analysis was conducted to identify common issues,

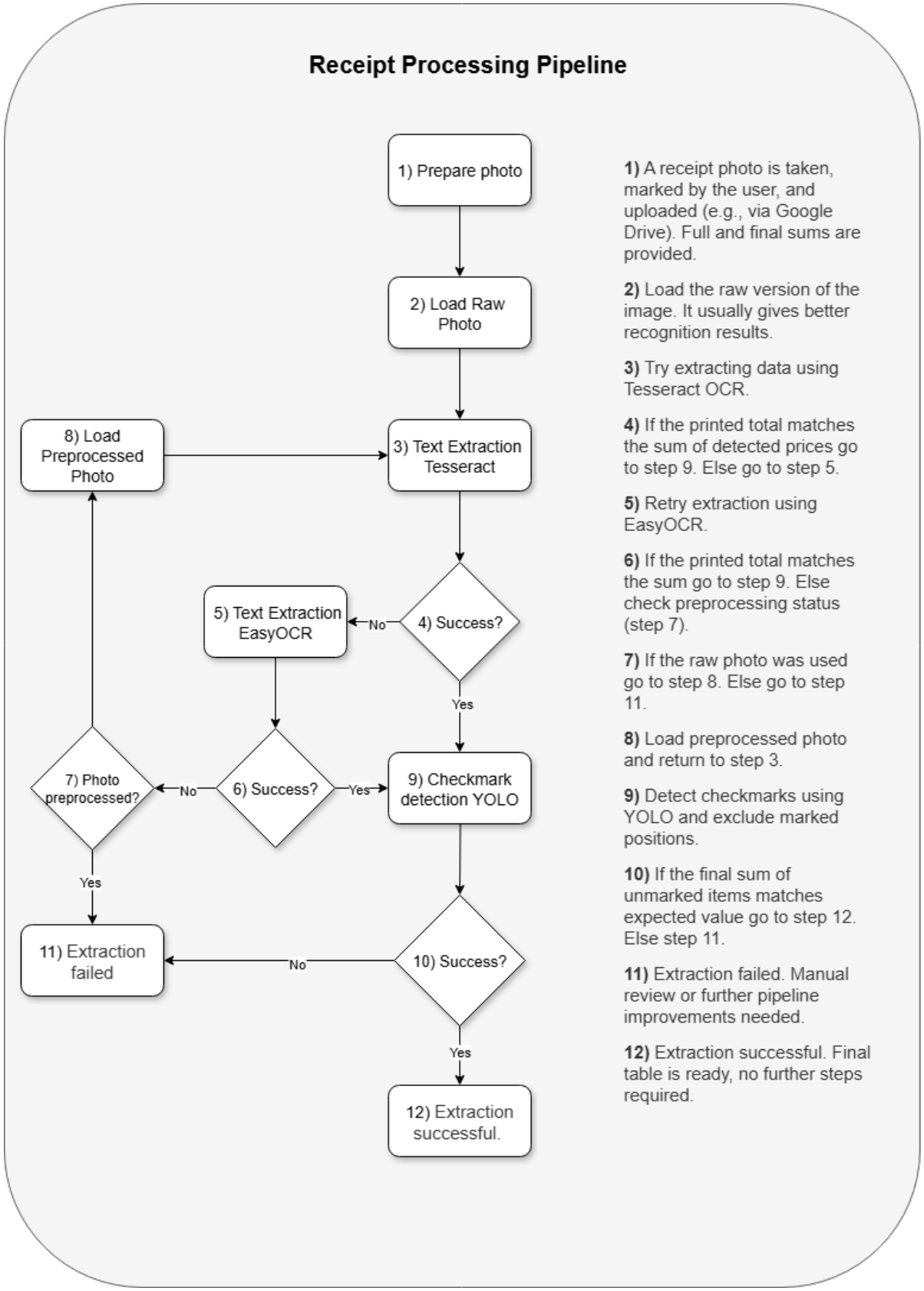

Fig. 6. Full receipt processing pipeline.

such as incorrect text recognition or missed checkmarks, and to understand how they affected the final output.

F. Experimental settings

All experiments were conducted in a Kaggle environment with access to NVIDIA Tesla T4 GPUs. Tesseract and EasyOCR were used for text extraction, with fallback logic applied when needed. The object detection component was implemented using the Ultralytics YOLOv8 framework and trained on a custom dataset of real and synthetic checkmark images. Checkmark dataset consisted of approximately 400 training and 100 validation checkmarks, with each image annotated in both YOLO and COCO formats. The pipeline was evaluated on a test set of 14 real-world receipts, each manually annotated with user-drawn checkmarks.

IV. Results

A. Pipeline Evaluation Summary

A total of 14 real-world receipts were tested to evaluate the effectiveness of the complete receipt processing pipeline. Successful extraction was achieved for 8 out of 14 receipts, resulting in an end-to-end success rate of 57.1 percent. EasyOCR contributed to correct extraction in two cases where Tesseract failed to recognize item data properly. Preprocessing was attempted on receipts where initial extraction failed, but it did not lead to success in any case. All successfully processed receipts were handled using the raw image without additional preprocessing. See Table III for full results of pipeline testing on real receipts.

B. Performance by Receipt Complexity

Receipts were grouped into three levels of complexity: easy, medium, and hard. All three receipts in the “easy” category were processed successfully, resulting in a 100 percent success rate. Among the five “medium” receipts, three were processed correctly, yielding a 60 percent success rate. For the six “hard” receipts, only two were successfully processed, corresponding to a 33 percent success rate. These results demonstrate a clear drop in performance as receipt complexity increases.

C. Observed Failure Reasons

For the six failed receipts, the causes were manually categorized. Three failures were attributed to OCR issues, such as incorrect price recognition or failure to detect the total amount. Two cases failed due to checkmark detection error. A checkmark was missed or incorrectly localized. One failure was caused by a formatting mismatch during postprocessing. In summary, the most frequent sources of error were text extraction mistakes and detection inaccuracies. An example of recognition failure is shown in Fig 7.

Fig. 7. Example of a failure case in the receipt recognition pipeline. The keyword ’SUMME’ was misrecognized as ’SUMHE’ by Tesseract, leading to incorrect item boundary detection. As a result, the payment line was mistakenly included as regular item

V. Discussion

The main goal of this study was to develop a modular pipeline for receipt processing that supports user-drawn checkmarks, allowing selective item recognition. The results show that the system is able to successfully process real-world receipts in more than half of the test cases, which confirms the general feasibility of the proposed approach. Although the end-to-end success rate is currently limited to 57 percent, the sources of errors are well understood and can be addressed with further improvements. This section discusses the results in detail with respect to the research questions and highlights the novelty and implications of the findings.

RQ1: Is it possible to build a complete pipeline that extracts structured data from receipts with user-drawn checkmarks? The results confirm that such a pipeline is feasible. The proposed system correctly processed 8 out of 14 real-world receipts. The modular structure, consisting of OCR, checkmark detection, and postprocessing, proved functional under realistic input conditions. Moreover, the system required no external APIs, relied only on open-source tools, and handled unstandardized receipt layouts, which makes it suitable for both personal and business use cases.

RQ2: Which object detection model performs best for recognizing hand-drawn checkmarks? YOLOv8 demonstrated strong performance on both synthetic and real samples and was selected for integration into the pipeline. While other models like SSD and Faster R-CNN were tested, they performed worse in detecting small marks with varied shapes and colors. The high detection accuracy of YOLOv8 contributed directly to the success of end-to-end receipt recognition in complex cases. Errors related to checkmark detection were mostly due to cases where marks were too light, too small, or outside the expected item zone.

TABLE III
RECEIPTS RECOGNITION RESULTS

ID	Level	EasyOCR Helped?	Preprocess Helped?	Succeed?	Possible Fail Reason
1	Easy	N/A	N/A	Yes	
2	Easy	N/A	N/A	Yes	
3	Easy	Yes	N/A	Yes	
4	Medium	N/A	N/A	Yes	
5	Medium	Yes	N/A	Yes	
6	Medium	No	No	No	Sum keyword detection
7	Medium	N/A	N/A	Yes	
8	Medium	No	No	No	Right price recognition (OCR)
9	Hard	No	No	No	"Summe" recognition (OCR)
10	Hard	N/A	N/A	No	Missed checkmark (YOLO)
11	Hard	N/A	N/A	Yes	
12	Hard	N/A	N/A	Yes	
13	Hard	No	No	No	Price boxes detection
14	Hard	N/A	N/A	No	Checkmark position (YOLO)

RQ3: Which OCR setup performs best for extracting text from real-world receipts? The combination of Tesseract and EasyOCR proved more effective than either engine used alone. In several failed cases, switching to the alternative OCR engine enabled recovery of usable data. The fallback mechanism increased the robustness of the text recognition stage. However, the most frequent failures in the pipeline were still due to OCR issues suggesting that further tuning and adding more OCR models could be beneficial.

Novelty and contribution The key novelty of this study lies in supporting user-drawn objects on printed receipts. Such feature is not present in any of the commercial or open-source solutions reviewed and can be improved and expanded. Existing systems such as Expensify, Veryfi, ABBYY FlexiCapture and others focus only on extracting predefined fields and do not allow user-driven item selection. [13] [14] [15] [16] [17] [18] [19] This work addresses that gap by introducing a hybrid approach that combines vision-based mark detection with text parsing and filtering. Additionally, a dedicated dataset for checkmark detection was created, consisting of both real and synthetic annotated images.

Comparison with existing solutions Compared to existing tools, the proposed pipeline is open-source, flexible, and supports visual marks like user-drawn checkmarks. It does not rely on cloud services or vendor-specific formats. Although its overall recognition accuracy is still low, it can be further improved to a competitive level. The ability to let users select items by hand makes it useful for real-life tasks such as bill splitting, personal expense tracking, or customized reporting.

A. Limitations

Certain initial requirements were set for receipt images: they had to be clear and readable.. As a result, the test dataset did not include low-quality receipts or photos, and the system's reliability under such conditions was not evaluated. The test set included only 14 real receipts, which limits the generalizability of the results. Additionally, the system is not designed for real-time processing, priority was given to extraction quality rather than speed. This makes it less suitable for scenarios that require high throughput or real-time image streams.

B. Future Directions

There are several directions for improving the current pipeline. First, increasing image resolution from 960×1280 to 1536×2048 could improve both text and checkmark recognition, especially for larger receipts. This change is unlikely to affect performance significantly and can resolve issues in several failed test cases. Second, the logic used to locate the receipt total could be improved. Currently, the pipeline stops after detecting a "sum" keyword, which sometimes appears too early or is misrecognized. A smarter fallback mechanism could increase reliability. Third, adding additional OCR engines such as TrOCR would increase recognition robustness in cases where both Tesseract and EasyOCR fail. Improvements in checkmark detection are also possible by expanding the dataset with more diverse examples. Beyond these fixes, other directions include refining preprocessing steps (which currently do not help), improving how item names are parsed, and optionally exploring large language models (LLMs) for fallback recognition, though their use must be carefully managed.

VI. CONCLUSION

This work presents a complete and modular pipeline for receipt processing with support for user-drawn checkmarks, combining OCR and object detection methods. The system was tested on real-world receipts and demonstrated successful extraction in over half of the cases. Although the overall accuracy is not yet ideal, the errors were well understood and can be addressed through targeted improvements. The results confirm that integrating visual mark detection into traditional text extraction pipelines is both feasible and useful for flexible data capture.

REFERENCES

[1] N. Islam, Z. Islam, and N. Noor, "A survey on optical character recognition system," *Journal of Information*, vol. 10, no. 2, 2016.

[2] Z.-Q. Zhao, P. Zheng, S. Xu, and X. Wu, "Object detection with deep learning: A review," *arXiv preprint arXiv:1807.05511*, 2019.

[3] S. G. Dedgaonkar, A. A. Chandavale, and A. M. Sapkal, "Survey of methods for character recognition," *International Journal*, vol. 1, no. 5, 2012.

[4] R. Smith, "An overview of the tesseract ocr engine," in *9th International Conference on Document Analysis and Recognition (ICDAR)*, 2007, pp. 629–633.

[5] M. Flores, D. Valiente, M. Alfaro, M. Fabregat-Jaén, and L. Payá, "Evaluation of open-source ocr libraries for scene text recognition in the presence of fisheye distortion," in *Proceedings of the 21st International Conference on Informatics in Control, Automation and Robotics*, 2024, pp. 133–140.

[6] M. Li *et al.*, "Trocr: Transformer-based optical character recognition with pre-trained models," *arXiv preprint arXiv:2109.10282*, 2022. [Online]. Available: https://arxiv.org/abs/2109.10282

[7] Wikipedia contributors, "Optical character recognition," https://en.wikipedia.org/w/index.php?title=Optical_character_recognition&oldid=1293503121, 2025, accessed: 2025-06-19.

[8] ——, " ," https://ru.wikipedia.org/w/index.php?title=$\mu\mu$_μ_&oldid=145245515, 2025, accessed: 2025-06-19.

[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *arXiv preprint arXiv:1506.02640*, 2016.

[10] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *arXiv preprint arXiv:1506.01497*, 2016.

[11] W. Liu *et al.*, "Ssd: Single shot multibox detector," in *European Conference on Computer Vision (ECCV)*, vol. 9905, 2016, pp. 21–37.

[12] G. Kim *et al.*, "Ocr-free document understanding transformer," *arXiv preprint arXiv:2111.15664*, 2022. [Online]. Available: https://arxiv.org/abs/2111.15664

[13] "Spend management software for receipts expenses," https://www.expensify.com/, accessed: 2025-07-03.

[14] "Ai document automation software — abbyy flexicapture," https://www.abbyy.com/flexicapture/, accessed: 2025-07-03.

[15] "Veryfi," https://www.veryfi.com/, accessed: 2025-07-03.

[16] "Online bookkeeping software for small businesses," https://dext.com/en, accessed: 2025-07-03.

[17] "1 receipt scanner app free mileage tracker — shoeboxed," https://www.shoeboxed.com/, accessed: 2025-07-03.

[18] "Wellybox — organize receipts and invoices with the power of ai," https://www.wellybox.com/, accessed: 2025-07-03.

[19] "Gorilla expense," https://www.gorillaexpense.com/, accessed: 2025-07-03.