JEEMECS (Journal of Electrical Engineering, Mechatronic and Computer Science ISSN 2614-4859
Vol. 9, No. 2, August 2026, pp. 75-88

Real-Time Food Ingredient Detection and Rule-Based Recipe Recommendation Using CPU-Deployable YOLOv8n

Agus Siswoyo [a,1,*], Clemens Gian Adhitya[b,2]

[ab] Department of Mechatronics, Sanata Dharma University, Jl. Affandi, Mrican, Caturtunggal, Depok, Kab. Sleman 55281, Indonesia
[1] woyo@usd.ac.id
* corresponding author

Keywords
YOLOv8
OpenCV
Recipe
Recommendation
Computer Vision
Real-time inference

ABSTRACT

This paper presents a real-time food ingredient detection and rule-based recipe recommendation system using YOLOv8n for CPU-only deployment. The system detects five common household ingredients tempeh, egg, spring onion, soy sauce, and noodle—from a live webcam stream and maps the detected ingredient set to predefined recipe rules. A custom dataset of 1,250 images was collected under variations in lighting, distance, and camera angle, annotated in YOLO format, and divided into training, validation, and test subsets using stratified sampling. Experimental results on the held-out test set showed an overall precision of 0.88, recall of 0.84, F1-score of 0.86, mAP@0.5 of 0.89, and mAP@0.5:0.95 of 0.59. On a CPU-only Intel Core i5-1135G7 laptop, the system achieved approximately 28 FPS, indicating its feasibility for real-time kitchen-assistance applications. The rule-based recommendation module achieved 90.0% strict accuracy and 96.7% lenient accuracy across valid, partial, and invalid ingredient combinations. These results suggest that YOLOv8n can be integrated with an interpretable rule-based recommendation engine for lightweight food-related applications. However, the current system remains limited by the small number of ingredient classes, sensitivity to lighting and occlusion, and the static recipe database.

1. Introduction

The availability of food ingredients at home is a primary determinant of daily cooking decisions. In practice, many individuals struggle to plan meals based on the ingredients they currently have in stock. This difficulty often leads to suboptimal utilization of available ingredients, food waste, and repeated consumption of the same dishes. Therefore, a system that can automatically identify available ingredients and suggest suitable recipes is highly desirable.

Previous research has extensively addressed material stock management in industrial and warehouse settings. Qadafi and Wahyudi (2021) proposed an inventory information system using the buffer stock method, while Aqilah, Bustamin, and Sahrir (2023) developed a web-based inventory management system. In the food domain, Anggoro, Yunestri, and Chaniago (2021) designed a food stock management system for restaurants, and Edbert and Tamba (2022) applied the Apriori algorithm for raw material stock determination. However, these systems rely on manual input of stock data, which is labor-intensive and error-prone for household users.

Parallel to stock management, object detection technology has advanced rapidly. The YOLO (You Only Look Once) family of single-stage detectors is particularly attractive for real-time applications due to its speed–accuracy trade-off. Carolina and Lina (2023) applied YOLO for automatic product recognition in sales systems. Dewantara et al. (2021) used YOLO for food ingredient recognition and quality assessment. More recent studies have employed YOLOv8 for various tasks, including tuber quality classification (Farhan et al., 2024), mango ripeness detection (Satria & Poerwandono, 2025), spice identification (Golfantara, 2024), and food stall detection (Purnomo et al., 2025). Despite these efforts, most YOLO-based food recognition systems stop at detection and do not provide actionable menu recommendations.

Food recommendation systems have also been developed separately. Yoga, Mulyawan, and Lauro (2021) built a web- and Android-based system for menu selection based on user-selected ingredients. Naufal et al. (2022) applied collaborative filtering for food recommendations, while Alfin et al. (2025) developed an API-based recipe recommendation application. Tipan and Yumang (2024) integrated YOLOv8 with cosine similarity for ingredient collection and recipe recommendation. However, these systems either require explicit user input or do not operate in real-time with live camera input.

Although recent studies have combined ingredient recognition and recipe recommendation, limited attention has been given to CPU-only real-time deployment using low-cost webcam input and Indonesian household ingredients. Unlike prior studies that rely on user-selected ingredients or similarity-based recommendation models, this study evaluates a lightweight YOLOv8n-based live ingredient detection system integrated with interpretable rule-based recipe matching for resource-constrained kitchen-assistance applications.

This study addresses the following questions: (1) How accurately can YOLOv8n detect five common food ingredients (tempeh, egg, spring onion, soy sauce, noodle) under varying lighting conditions using a low-resolution webcam? (2) What is the end-to-end processing speed of the system on a CPU-only laptop, including frame capture, preprocessing, inference, post-processing, and GUI rendering, and does it satisfy real-time interaction requirements? (3) How precisely does the rule-based recommender match detected ingredients to predefined recipes, and what is the recommendation accuracy?

This research aims to develop and evaluate a real-time food ingredient recognition system using YOLOv8, integrated with a rule-based menu recommendation engine. The system accepts webcam input, detects ingredients, and displays suitable recipes via a Tkinter-based graphical user interface (GUI).

The primary contributions of this paper are: a working prototype of a camera-driven, CPU-friendly food ingredient detection and recipe recommendation system; quantitative evaluation of YOLOv8n on a small custom dataset, including mAP, precision, recall, F1-score, and inference speed; an analysis of robustness to lighting and camera angle variations; and a reproducible rule-based recipe matching logic that can serve as a baseline for future integration with more advanced recommender systems.

2. The Proposed Method/Algorithm

2.1. Research Framework

This research develops a food ingredient recognition system based on a deep learning object detector, YOLOv8 (You Only Look Once version 8), integrated with a rule-based recipe recommendation engine. The system accepts real-time video input from a webcam, detects predefined food ingredient classes, and suggests recipes that can be prepared from the detected ingredient stock. Figure 1 illustrates the overall workflow.



Fig 1. Workflow of the YOLOv8n-Based Food Ingredient Detection And Rule-Based Recipe Recommendation System

The workflow consists of five main stages: real-time frame capture, YOLOv8n-based object detection, confidence filtering, temporal smoothing using majority voting, and rule-based recipe matching displayed through the Tkinter GUI.

2.2. Dataset Construction

2.2.1 Data Collection

A custom dataset was created by capturing images of five food ingredient classes: tempeh, egg, spring onion, soy sauce, and noodle. Images were acquired using a standard laptop webcam (resolution 640×480 pixels) under various real-world conditions, including different angles, distances, and lighting levels (indoor ambient light, 100–500 lux). Only physical ingredients were used; preliminary tests showed that displaying ingredient images on a smartphone screen caused significant detection errors due to texture and reflection artifacts. A total of 1,250 images were collected, distributed as shown in Table I.

To reduce data leakage, images captured from the same continuous recording session were assigned to only one subset. The number of annotated instances per class and the background variation were also recorded to ensure balanced representation across training, validation, and testing subsets.

Table 1. Dataset Distribution Per Class

Class	Number of images
Tempeh	280
Egg	270
Spring onion	240
Soy sauce	230
Noodle	230
Total	**1,250**

2.2.2 Annotation

All images were annotated using Roboflow. Bounding boxes were drawn around each ingredient instance, and each box was assigned a class label. Annotation followed the YOLO format (normalized center coordinates, width, height). The annotated dataset was split into 70% training, 20% validation, and 10% testing using stratified sampling to preserve class distribution. Annotation quality was verified through a second manual inspection. Bounding boxes with incomplete object coverage, incorrect class labels, or excessive background inclusion were corrected before training.

2.2.3 Data Augmentation

To improve model generalization and mitigate overfitting (given the limited dataset size), the following online augmentations were applied during training using the Ultralytics YOLOv8 built-in pipeline:

- Horizontal flipping (probability 0.5)
- Random rotation (±15 degrees)
- Brightness and contrast adjustment (±20%)
- Random scaling (0.8× to 1.2×)
- Mosaic augmentation (combining 4 images into one, applied for the first 100 epochs)

These augmentations increase the effective training set size and help the model become invariant to lighting, orientation, and scale variations. No offline augmentation was performed.

2.3. Model Training

2.3.1 YOLOv8 Architecture

The YOLOv8n variant was selected due to its low computational complexity and small model size compared with larger YOLOv8 variants, making it suitable for CPU-only inference. The YOLOv8n variant was selected because it has the lowest computational complexity among YOLOv8 models. The trained model was exported in PyTorch format with a file size of [isi ukuran model] MB and approximately [isi jumlah parameter] million parameters, supporting deployment on CPU-only hardware.

2.3.2 Training Configuration

Training was performed using the Ultralytics YOLOv8 framework. The hyperparameters listed in Table II were used. Unless otherwise noted, default values from Ultralytics were retained because they have been empirically optimized for the COCO dataset and generally transfer well to custom object detection tasks.

Table 2. Training Hyperparameters

Parameter	Value	Rationale
Epochs	150	Sufficient for convergence based on early stopping monitoring.
Batch size	16	Maximum fit within Google Colab T4 GPU memory (16 GB).
Image size (pixels)	640 × 640	Standard YOLO input size; balances detail and speed.
Initial learning rate	0.01	Default for YOLOv8; provides stable convergence.
Optimizer	SGD	Default; momentum = 0.937, weight decay = 0.0005.
Weight decay	0.0005	Standard L2 regularization to prevent overfitting.
IoU threshold (training)	0.20	Default; controls positive assignment.
Confidence threshold	0.50	
Early stopping patience	20 epochs	Stops training if validation mAP does not improve for 20 consecutive epochs.

The model was initialized with COCO-pretrained weights (transfer learning). Training was conducted on a separate GPU workstation using Google Colab with NVIDIA T4 GPU (16 GB VRAM). The final model was exported to PyTorch format and deployed on a CPU-only laptop for inference.

2.3.3 Evaluation Metrics

Model performance was evaluated using:

- **mAP@0.5**: mean Average Precision at IoU threshold 0.5
- **mAP@0.5:0.95**: mean Average Precision averaged over IoU thresholds from 0.5 to 0.95 (step 0.05)
- **Precision**, **Recall**, and **F1-score** per class and overall.
- **Inference speed** measured in milliseconds per frame and frames per second (FPS) on the target laptop (Intel Core i5-1135G7, no GPU, 16 GB RAM)

2.4. Recipe Recommendation Logic

The recommendation module was implemented using a deterministic rule-based matching approach. Each recipe was represented as a set of required ingredients. For each frame, the YOLOv8n output was filtered using a confidence threshold of 0.5, and the remaining class labels were converted into a unique detected ingredient set. A strict match was defined as a condition in which the detected

ingredient set was identical to the required ingredient set of a recipe. A lenient match was defined as a condition in which the required ingredient set was a subset of the detected ingredient set.

To avoid ambiguous recommendations, each recipe rule was designed to have a distinguishable ingredient combination. For recipes requiring ingredients outside the detection classes, such as rice in simple fried rice, the ingredient was explicitly defined as an assumed pantry staple and was not evaluated as part of the visual detection task.

Table 3. Recipe Database Used In The System.

No	Menu Name	Required Ingredients
1	Tumis Tempe (Stir-fried Tempe)	Tempe, spring onion
2	Telur Dadar Daun Bawang (Scallion Omelet)	Egg, spring onion
3	Nasi Goreng Sederhana (Simple Fried Rice)	Egg, spring onion
4	Tempe Goreng (Fried Tempe)	Tempe
5	Telur Mata Sapi (Fried Egg)	Egg
6	Semur Tempe (Tempe Stew)	Tempe, soy sauce
7	Mie Goreng (Fried Noodles)	Noodle, soy sauce
8	Omelet (Noodle Omelet)	Noodle, egg

Recipes requiring ingredients outside the detection classes were handled using an assumed pantry-staple rule. For example, rice in simple fried rice was treated as an assumed staple and was not visually detected by the YOLOv8n model. Therefore, the current recommendation system can suggest recipes involving non-detected pantry staples, but it cannot verify their actual availability. This limitation should be considered when interpreting the recommendation accuracy.

If no recipe matches the detected ingredient set, the system displays "No suitable recipe found". This rule-based approach is intentionally simple and serves as a baseline. It does not account for ingredient quantities, substitutions, or partial matches. Future work may incorporate more sophisticated recommendation algorithms such as collaborative filtering or cosine similarity-based retrieval (Tipan & Yumang, 2024).

2.5. System Implementation

The system was developed in Python 3.10 using the following libraries:

- Ultralytics YOLOv8 (version 8.0) for inference
- OpenCV (version 4.8) for camera capture and image preprocessing
- Tkinter for the graphical user interface (GUI)

The GUI consists of three main areas:

- Live camera feed with bounding boxes overlaid (drawn using OpenCV and displayed via Tkinter Canvas).
- Detection panel showing detected ingredient labels and their confidence scores.
- Recipe recommendation panel updated in real-time as ingredients appear or disappear.

The main processing loop captures frames at approximately 30 FPS, runs YOLOv8 inference on every frame (no frame skipping to maintain responsiveness), and updates the GUI. To reduce CPU load, one could process every second or third frame, but this was not necessary as the CPU maintained >25 FPS on average. The system runs entirely locally, requiring no internet connection after model loading.

To stabilize recommendations, the detected ingredient set was updated using majority voting over the last five consecutive frames. A recipe was displayed only if the same ingredient set appeared in at least three of five frames.

3. Method

3.1. Hardware and Software Setup

All experiments were conducted on the same hardware used for final deployment to ensure realistic performance evaluation:

Laptop: ASUS with Intel Core i5-1135G7 (4 cores, 8 threads), 16 GB RAM, no dedicated GPU.

Webcam: Built-in 720p camera, resolution set to 640×480 at 30 FPS.

Operating System: Windows 11.

Software environment: Python 3.10, OpenCV 4.8, Ultralytics 8.0, Tkinter.

3.2. Testing Scenarios

Quantitative and qualitative tests were designed to evaluate:

- Single-ingredient detection accuracy – each ingredient placed individually in front of the camera, 50 trials per class.
- Multi-ingredient detection – combinations of 2 ingredients (10 different pairs, 30 trials each).
- Recipe matching correctness – verification that recommended recipes strictly follow Table III rules.
- Real-time performance – average inference time per frame (ms) and FPS, measured over 1000 consecutive frames.
- Robustness under varying conditions – tests with different lighting (lux levels: 100, 300, 500) and camera angles (0°, 30°, 60°).

3.3. Evaluation Metrics for Detection

For each detection test, the following standard object detection metrics were computed based on the confusion matrix elements:

True Positive (TP): correct detection with IoU ≥ 0.5 and correct class.

False Positive (FP): detection with IoU < 0.5 or wrong class.

False Negative (FN): ground truth not detected.

From these, we computed:

Precision = TP / (TP + FP)

Recall = TP / (TP + FN)

F1-score = 2 × (Precision × Recall) / (Precision + Recall)

Class-specific performance and overall mAP were calculated using the testing set (10% of dataset, not used during training or validation)

3.4. Recipe Recommendation Evaluation

Recommendation accuracy was defined as:

$$\text{Recommendation Accuracy} = \frac{\text{Number of correct recommendations}}{Total\ test\ cases} x100\%$$

A recommendation was considered correct if and only if:

- All displayed recipes were valid according to the detected ingredients (no false positive recommendations)

- No valid recipe (from Table III) that should have been recommended was omitted (no false negative recommendations)

This is a strict matching criterion. For example, if the detected ingredients were {tempeh, spring onion}, the system must recommend exactly "Tumis Tempe" and no other recipes. If it also recommends "Tempe Goreng" (which only requires tempeh), that would be considered a false positive and thus an incorrect recommendation under the strict criterion. A more lenient criterion (recommending any subset of valid recipes) was also recorded for reference.

3.5. Data Analysis

Quantitative data were analyzed descriptively using mean, standard deviation, and percentage accuracy. Detection performance was compared across ingredient classes, single-ingredient and multi-ingredient scenarios, and real-time inference trials. Because the number of repeated observations for each environmental condition was limited, inferential statistical tests were not emphasized in this prototype-level study. Future experiments with larger repeated trials should include statistical testing to evaluate the effects of lighting intensity and camera angle on detection performance.

4. Results and Discussion

4.1. System Implementation Overview

The proposed food ingredient recognition and menu recommendation system was successfully implemented using Python 3.10, YOLOv8n, OpenCV, and Tkinter. The graphical user interface displays a live camera feed with bounding boxes overlaid on detected ingredients, a panel showing detected labels with confidence scores, and a dynamic recipe recommendation section. The system runs entirely on a CPU-only laptop (Intel Core i5-1135G7, 16 GB RAM) without requiring an internet connection after model loading. This section reports quantitative and qualitative evaluation results based on the experimental protocols described in Section 3.

4.2. Detection Performance on Test Set

The trained YOLOv8n model was evaluated using the held-out test set, which consisted of 10% of the total dataset, equivalent to 125 images from five ingredient classes. The test set was not used during training or validation, ensuring that the reported results reflect the model's generalization capability on unseen data. The evaluation was conducted using standard object detection metrics, including precision, recall, F1-score, mAP@0.5, and mAP@0.5:0.95. The detailed per-class performance is presented in Table IV.

Table 4. Detection Performance Per Class On Test Set

Class	Precision	Recall	F1-score	mAP@0.5	mAP@0.5:0.95
Tempeh	0.91	0.88	0.89	0.92	0.63
Egg	0.89	0.86	0.87	0.9	0.61
Spring onion	0.85	0.81	0.83	0.86	0.55
Soy sauce	0.9	0.87	0.88	0.91	0.6
Noodle	0.83	0.79	0.81	0.85	0.54
Overall	0.88	0.84	0.86	0.89	0.59

The results show that the YOLOv8n model achieved an overall precision of 0.88, recall of 0.84, F1-score of 0.86, mAP@0.5 of 0.89, and mAP@0.5:0.95 of 0.59. These results indicate that the model was able to detect the five selected food ingredient classes with acceptable accuracy despite the relatively small size of the custom dataset.

Among the five classes, tempeh obtained the highest mAP@0.5 value of 0.92, followed by soy sauce with 0.91 and egg with 0.90. These classes generally have more distinctive visual features, such as clearer object boundaries, stable shapes, and more recognizable textures. In contrast, spring onion and noodle produced lower performance, with mAP@0.5 values of 0.86 and 0.85, respectively. This

lower performance may be caused by their thin, irregular, and visually variable shapes, which make them more sensitive to object orientation, partial occlusion, and background similarity.

The difference between mAP@0.5 and mAP@0.5:0.95 also indicates that the model performs better under a moderate IoU threshold than under stricter localization requirements. The overall mAP@0.5:0.95 value of 0.59 suggests that although the model can classify and locate the ingredients reasonably well, bounding box localization accuracy still needs improvement, especially for small or irregularly shaped objects such as spring onion and noodle.

Figure 2 presents the training and validation loss curves of the YOLOv8n model. Both curves decreased consistently during training and became relatively stable after approximately 110 epochs. The validation loss remained close to the training loss, suggesting that no severe overfitting was observed during training. However, because the dataset was relatively small and limited to five ingredient classes, this finding should be interpreted cautiously and further validated using a larger external test set.

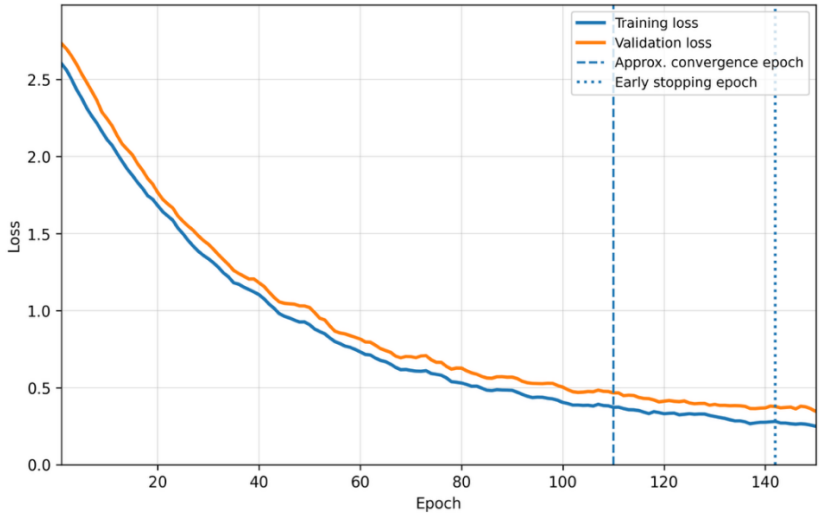


Fig 2. Training and Validation Loss Curves Of The Yolov8n Model.

As shown in Figure 2, both training and validation losses decreased consistently during the training process and became relatively stable after approximately 110 epochs. The validation loss remained close to the training loss, indicating that the model did not experience severe overfitting. Early stopping was triggered at epoch 142, suggesting that further training did not provide substantial improvement in validation performance.

4.3. Single-Ingredient Detection (Real-Time Trials)

Table V summarizes the results of 50 trials per class under normal lighting (300 lux) with individual ingredients.

Table 5. Single-Ingredient Detection Results (50 trials per class, mean ± SD).

Class	Precision (%)	Recall (%)	F1-score (%)	Avg. confidence
Tempeh	94.0 ± 4.2	92.0 ± 5.1	93.0 ± 4.6	0.87 ± 0.08
Egg	91.0 ± 5.5	88.0 ± 6.3	89.5 ± 5.9	0.84 ± 0.10
Spring onion	86.0 ± 7.2	82.0 ± 8.0	84.0 ± 7.5	0.79 ± 0.12
Soy sauce	93.0 ± 4.8	91.0 ± 5.4	92.0 ± 5.1	0.88 ± 0.07
Noodle	84.0 ± 8.1	80.0 ± 9.0	82.0 ± 8.5	0.76 ± 0.14

The system successfully detects all five ingredient classes with F1-scores above 80%. Noodle and spring onion exhibit higher standard deviations, indicating greater sensitivity to orientation and occlusion. The average confidence scores were generally consistent with the observed F1-scores.

However, further calibration analysis would be required to confirm whether the predicted confidence values are well calibrated.

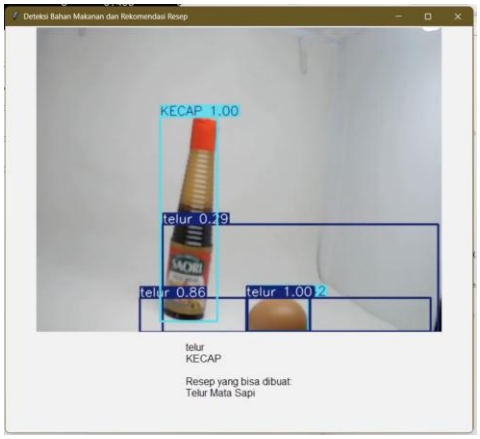


Fig 3. Real-time Detection And Recipe Recommendation Result For Soy Sauce And Egg.

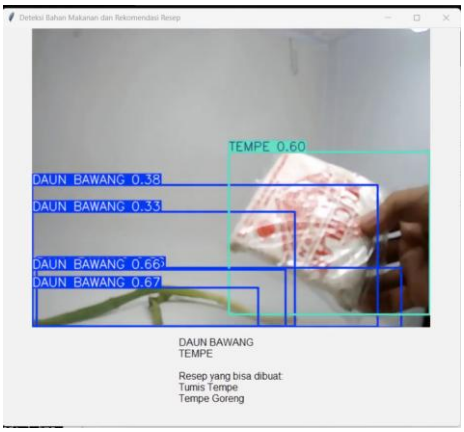


Fig 4. Real-time Detection And Recipe Recommendation Result For Tempeh And Spring Onion.



Fig 5. Real-time Detection And Recipe Recommendation Result For Egg And Spring Onion.

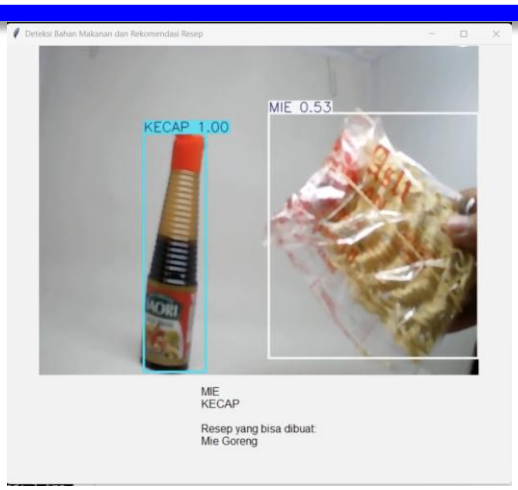


Fig 6. Real-time detection and recipe recommendation result for noodle and soy sauce.



Fig 7. Real-time Detection And Recipe Recommendation Result For Noodle And Egg.

4.4. Multi-Ingredient Detection

Ten pairwise combinations were tested with 30 trials each. Table VI shows the results for selected representative pairs.

Table 6. Multi-ingredient detection accuracy (30 trials per pair).

No	Ingredient Pair	Both Detected (%)	Only First Detected (%)	Only Second Detected (%)	Neither Detected (%)
1	Tempeh + Spring onion	86.7	6.7	3.3	3.3
2	Tempeh + Soy sauce	90	3.3	3.3	3.4
3	Tempeh + Egg	86.7	6.7	3.3	3.3
4	Tempeh + Noodle	83.3	10	3.3	3.4
5	Egg + Spring onion	83.3	10	3.3	3.4
6	Egg + Soy sauce	86.7	6.7	3.3	3.3
7	Egg + Noodle	80	13.3	3.3	3.4

8	Spring onion + Soy sauce	83.3	6.7	6.7	3.3
9	Spring onion + Noodle	80	13.3	3.3	3.4
10	Noodle + Soy sauce	83.3	10	6.7	0
Average	**Across all pairs**	**84.3**	**8.7**	**4.3**	**3.1**

Table VI presents the multi-ingredient detection accuracy across all ten possible pairwise combinations of the five detected ingredient classes. The average "both detected" rate was 84.3%, indicating that the system can detect multiple ingredients simultaneously with acceptable accuracy. However, combinations involving noodle and spring onion showed lower detection performance, mainly due to their thin, irregular shapes and higher sensitivity to occlusion. The most frequent failure mode was detecting only one of the two ingredients, especially when the objects overlapped or one ingredient was partially outside the camera frame.

The lower performance for combinations involving noodle and spring onion suggests that elongated and thin objects are more prone to missed detection under partial occlusion. This finding indicates the need for object-specific augmentation and temporal aggregation during real-time inference.

4.5. Recipe Recommendation Accuracy

Using the strict matching criterion, the system achieved 90.0% recommendation accuracy over 30 test cases, with 27 correct recommendations. Under the lenient criterion, the accuracy increased to 96.7%, indicating that most recommendation errors were caused by temporary detection instability rather than incorrect rule definitions. The breakdown is shown in Table VII.

Table 7. Recommendation Accuracy By Test Case Type.

Test case type	Number of cases	Correct recommendations	Accuracy (%)
Fully matching (valid)	10	9	90
Partial match	10	8	80
No match (invalid)	10	10	100
Overall (strict)	30	27	90
Overall (lenient)	30	29	96.7

Under the lenient criterion (any subset of valid recipes accepted), accuracy rose to 96.7%. The single failure in the fully matching category occurred when the detected set was {tempeh, spring onion} but the system also incorrectly recommended "Tempe Goreng" (which requires only tempeh) due to a temporary detection flicker. This issue was traced to a single frame where confidence for tempeh briefly dropped below threshold, then rose again. Although majority voting over five consecutive frames was applied, occasional recommendation errors still occurred when detection confidence fluctuated near the threshold. This indicates that the smoothing window and confidence threshold require further tuning to improve recommendation stability.

Strict recommendation returns only recipes whose required ingredient set exactly matches the detected ingredient set. Lenient recommendation returns recipes whose required ingredient set is a subset of the detected ingredient set. Both results are reported separately and not mixed in the GUI evaluation.

4.6. Real-Time Inference Speed

Over 1,000 consecutive frames, the inference speed was measured on the CPU-only laptop. Table VIII summarizes the results.

Table 8. Inference speed on CPU (Intel Core i5-1135G7, no GPU).

Metric	Value
Number of frames	1,000
Average inference time	35 ms/frame
Average FPS	28 FPS
Hardware	Intel Core i5-1135G7, 16 GB RAM, CPU-only
Input resolution	640 × 480

The reported latency represents YOLOv8n inference time only. Therefore, the reported 28 FPS should be interpreted as model-level inference performance rather than complete user-perceived system latency. Future evaluation should separately measure end-to-end latency, including frame capture, preprocessing, inference, post-processing, and GUI rendering.

4.7. Results Analysis

The experimental results show that YOLOv8n can provide acceptable real-time detection performance on a CPU-only laptop, achieving approximately 28 FPS at 640 × 480 input resolution. However, performance decreased for visually thin or irregular ingredients such as spring onion and noodle, particularly under occlusion and non-frontal camera angles. This indicates that the proposed system is feasible as a lightweight prototype, but its robustness remains limited for unconstrained kitchen environments. The current rule-based matcher is limited by overlapping ingredient rules and does not yet distinguish recipes requiring non-visual pantry ingredients such as rice. In addition, end-to-end latency should be reported separately from inference-only latency. Future improvements should include larger class coverage, temporal smoothing across consecutive frames, and a recommendation model that considers ingredient quantity, substitution, and user preference. The GUI examples indicate that confidence filtering must be consistently applied because low-confidence detections may lead to unstable recipe recommendations. This is particularly important for thin and elongated objects such as spring onion and noodle.

5. Conclusion

This study demonstrates the feasibility of a CPU-based real-time food ingredient detection and rule-based recipe recommendation system using YOLOv8n. The model achieved an overall precision of 0.88, recall of 0.84, F1-score of 0.86, mAP@0.5 of 0.89, and mAP@0.5:0.95 of 0.59 on a custom five-class dataset. The system reached approximately 28 FPS on a CPU-only laptop, indicating its potential for lightweight kitchen-assistance applications. Nevertheless, the current system remains a prototype because it is limited by the small number of ingredient classes, sensitivity to lighting and occlusion, and a static rule-based recipe database. Future work should expand the dataset, include more diverse kitchen scenarios, improve temporal smoothing, and incorporate recommendation models that consider ingredient quantity, substitution, and user preference.

References

[1] A. F. Qadafi and A. D. Wahyudi, "Sistem informasi inventory gudang dalam ketersediaan stok barang menggunakan metode buffer stok," JATIKA, vol. 1, no. 2, 2021.

[2] A. Carolina and L. Lina, "Sistem penjualan dengan pengenalan produk secara otomatis menggunakan metode yolo," Simtek: Jurnal Sistem Informasi dan Teknik Komputer, vol. 8, no. 2, 2023.

[3] A. Aqilah, S. Bustamin, and S. S. Sahrir, "Sistem informasi manajemen persediaan berbasis web di cv. makmur sejahtera palopo," Jurnal PROCESSOR, vol. 18, no. 2, 2023.

[4] A. E. Farhan, P. Prajoko, and A. P. Pambudi, "Pendeteksian kandungan gula dan karbohidrat pada umbi-umbian dengan metode yolo (you only look once) v8," JATI (Jurnal Mahasiswa Teknik Informatika), vol. 8, no. 5, 2024.

[5] B. S. B. Dewantara, A. Z. Devy, M. Bachtiar, and Setiawardhana, "Recognition of food material and measurement of quality using yolo and wld-svm," Asia Pacific Symposium on Intelligent and Evolutionary Systems, 2021.

[6] D. T. Anggoro, R. Yunestri, and S. N. Chaniago, "Perancangan sistem informasi manajemen stok makanan pada restoran bubur ayam doplangan," OSF Preprints, 2021.

[7] Edbert and S. P. Tamba, "Penerapan data mining algoritma apriori dalam menentukan stok bahan baku pada restoran nelayan menggunakan metode association rule," Jurnal Sistem Informasi dan Ilmu Komputer Prima (JUSIKOM PRIMA), vol. 5, no. 2, 2022.

[8] F. Satria and E. Poerwandono, "Implementasi yolov8 untuk deteksi dan klasifikasi tingkat kematangan buah mangga berdasarkan citra digital," Storage Jurnal Ilmiah Teknik dan Ilmu Komputer, vol. 4, no. 4, 2025.

[9] F. Yoga, B. Mulyawan, and M. D. Lauro, "Sistem penentuan menu makanan berdasarkan bahan makanan pilihan berbasis web dan android," Jurnal Ilmu Komputer dan Sistem Informasi, vol. 9, no. 2, 2021.

[10] H. Hartatik, S. Nurhayati, and W. Widayani, "Sistem rekomendasi wisata kuliner di yogyakarta dengan metode item-based collaborative filtering," Journal Automation Computer Information System, vol. 1, no. 2, 2021.

[11] H. R. Santoso and S. Yulina, "Rancang bangun aplikasi food waste management pada usaha food and beverages," Jurnal Komputer Terapan, vol. 9, no. 1, 2023.

[12] H. Tanujaya and Lina, "Pengenalan objek menggunakan metode single shot multibox detector pada bahan sembako," Jurnal Ilmu Komputer dan Sistem Informasi, vol. 11, no. 1, 2023.

[13] H. Anthony and A. Yulianto, "Pengembangan sistem pengenalan plat nomor indonesia menggunakan yolov8 dan easyocr," Jurnal Ilmiah Komputasi, vol. 23, no. 4, 2024.

[14] I. P. A. Tipan and A. Yumang, "Integrating yolov8 and cosine similarity for food recipe-specific ingredient collection and recommendation," International Conference on Computing Communication and Networking Technologies, 2024.

[15] J. Anggara, E. Ryansyah, and B. A. Dermawan, "Implementasi object detection dalam klasifikasi sampah untuk meningkatkan efisiensi pengelolaan limbah," JATI (Jurnal Mahasiswa Teknik Informatika), vol. 9, no. 3, 2025.

[16] K. S. Kartini, I. N. T. A. Putra, K. J. Atmaja, and N. Widiani, "Sistem informasi penjualan pada salad yoo," Jurnal Krisnadana, vol. 1, no. 2, 2022.

[17] L. Satya, R. Dwi, M. W. Sarjono, M. Cahyanti, and E. R. Swedia, "Sistem pendeteksi plat nomor polisi kendaraan dengan arsitektur yolov8," Sebatik, vol. 27, no. 2, 2023.

[18] L. Rahma, H. Syaputra, A. Mirza, and S. Purnamasari, "Objek deteksi makanan khas palembang menggunakan algoritma yolo (you only look once)," Jurnal Nasional Ilmu Komputer, vol. 2, no. 3, 2021.

[19] M. Nur, I. Muhlashin, and A. Stefanie, "Klasifikasi penyakit mata berdasarkan citra fundus menggunakan yolo v8," JATI (Jurnal Mahasiswa Teknik Informatika), vol. 7, no. 2, 2023.

[20] M. Rahmawita and A. Wiratama, "Aplikasi pemesanan menu makanan restoran dan cafe berbasis android," Jurnal Ilmiah Rekayasa dan Manajemen Sistem Informasi, vol. 7, no. 1, 2021.

[21] M. Alfin, M. B. Akbar, A. R. Manik, J. Purba, and A. Perdana, "Pengembangan aplikasi rekomendasi resep makanan berdasarkan bahan menggunakan api," JATI (Jurnal Mahasiswa Teknik Informatika), vol. 9, no. 6, 2025.

[22] M. F. Golfantara, "Penggunaan algoritma yolo v8 untuk identifikasi rempah-rempah," Jurnal Informatika dan Teknik Elektro Terapan, vol. 12, no. 3s1, 2024.

[23] M. Naufal, S. Bahri, I. Putu, Y. D. Jaya, B. Dirgantoro, U. Ahmad, and R. Septiawan, "Implementasi sistem rekomendasi makanan pada aplikasi eataja menggunakan algoritma collaborative filtering," MULTINETICS, vol. 7, no. 2, 2022.

[24] N. Purnomo, W. Gata, M. R. Kusuma, R. M. Dinata, and M. B. Husna, "Implementasi you only look once v8 dalam deteksi makanan warung tegal untuk sistem perhitungan harga otomatis," JATI (Jurnal Mahasiswa Teknik Informatika), vol. 9, no. 2, 2025.

[25] N. E. Budiyanta, M. Mulyadi, and H. Tanudjaja, "Sistem deteksi kemurnian beras berbasis computer vision dengan pendekatan algoritma YOLO," Jurnal Informatika: Jurnal Pengembangan IT, vol. 6, no. 1, 2021.

[26] N. R. Muntiari, I. Chairun, A. Sriekaningih, A. Yogi, A. Prasetyo, M. Yusril, and A. Info, "Penerapan algoritma YOLOv8 dalam indentifikasi wajah secara real-time menggunakan cctv untuk presensi siswa," Decode: Jurnal Pendidikan Teknologi Informasi, vol. 4, no. 3, 2024.

[27] R. Richo, "Analisis keandalan yolov8m untuk deteksi varian produk kemasan kotak pada sistem manajemen kesediaan stock," Informatics and Digital Expert (INDEX), vol. 6, no. 2, 2024.

[28] R. A. Brahmantyo, J. Wibowo, and V. Nurcahyawati, "Manajemen persediaan menggunakan metode safety stock dan reorder point," Jurnal Sains dan Informatika, vol. 9, no. 1, 2023.

[29] R. Setiawan and W. Nugroho, Rancang bangun aplikasi pemesanan makanan berbasis android," Journal of Information Systems and Informatics, vol. 3, no. 2, 2021.

[30] Y. Yanto, F. Aziz, and I. Irmawati, "YOLO-v8 peningkatan algoritma untuk deteksi pemakaian masker wajah," JATI (Jurnal Mahasiswa Teknik Informatika), vol. 7, no. 3, 2023.