AR Cooking Assistant

A Framework and Methods for Automatic Recipe Step Detection

PAUL HAN and ALVIN LE, University of Washington

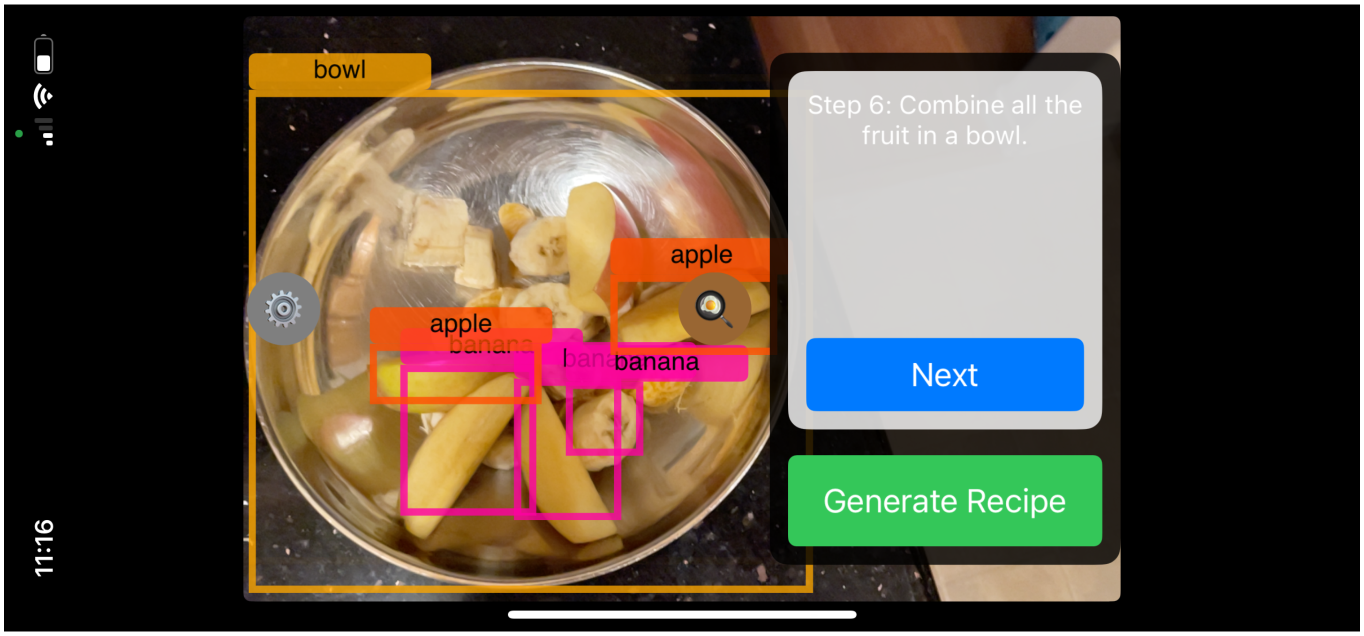


Fig. 1. Screenshot of the AR Recipe Assistant application in action. The system detects food objects in the cooking environment, generates a recipe based on them, and provides real-time step-by-step guidance. In this example, the assistant recognizes a bowl, apple, and banana, and automatically updates the interface to indicate that the user is on Step 6: "Combine all the fruit in a bowl." The AR interface overlays detected objects with bounding boxes and labels while allowing users to navigate through recipe steps interactively.

In this work, we introduce two methods for the task of automatic recipe step detection (ARSD), a fundamental component of augmented reality cooking applications. Despite the increasing interest in AR for assisting users in real-world tasks, there has been limited research into a generalizable, finetuning-free framework for ARSD that is both robust and scalable. We propose a framework that leverages large-scale self-supervised models (LLMs and VLMs) and demonstrate that both methods may be effective in solving the ARSD problem, with distinct trade-offs in accuracy, latency, and cost.

1 INTRODUCTION

Augmented reality (AR), in which virtual content is integrated with real-world environments, is a growing area for industry and research [1]. With advancements in AR hardware such as Meta's Orion, Microsoft's HoloLens, and Apple's Vision Pro, the potential applications of AR have expanded significantly [2][3][4]. A key area of interest is AR-based assistant applications for everyday tasks.

1.1 AR Recipe Assistance

In this paper, we examine a specific category of AR-assistance applications: AR-based cooking and recipe assistance. Given the central role of food in daily life, developing intelligent and interactive cooking support systems has substantial practical implications.

The premise of an AR recipe assistance system is simple. The assistant analyzes visual input—typically a photo—to identify ingredients and tools, then generates a suitable recipe. After, it augments the cooking process by setting timers, providing step-by-step guidance, detecting errors, and adapting the instructions in real time. All of this is done in an AR interface, so users can engage with the application in real-time without disrupting their cooking workflow.

1.2 Automatic Recipe Step Detection

To further narrow the scope, we focus on a key task in AR recipe assistance: automatic recipe step detection (ARSD). The task is defined as follows. Given an enumerable recipe and visual input from a camera positioned over the cooking area, can we identify the exact step that is currently executing?

We believe ARSD is important for several reasons. First, during cooking, users often have both hands occupied, making manual step transitions inconvenient. Second, it serves as a foundation for other AR-based features, such as setting timers at appropriate steps or detecting mistakes in real time. This paper primarily examines

Authors' address: Paul Han, paulh27@cs.washington.edu; Alvin Le, lealv000@cs.washington.edu, University of Washington.

ARSD and explores methods to implement it within a generalizable framework.

1.3 Contributions

Our contributions are as follows:

(1) We introduce a general framework for automated recipe step detection (ARSD) and propose two distinct methods with different trade-offs.
(2) We define the ARSD task and establish a framework for its evaluation, although we do not provide a comprehensive dataset.
(3) We develop a prototype application implementing our framework and qualitatively demonstrate its effectiveness in recipe generation and ARSD.

2 RELATED WORK

Although not extensive, there has been some prior research on automated recipe step sensing.

Decades ago, older methods used sensors and probabilistic models to infer activities from sensor data. One notable system, PROACT, tracks object movement by tagging items with radio-frequency identification (RFID) tags [5]. A Bayesian network then estimates the probability of specific actions, such as making tea with a kettle, by monitoring object movements. However, these approaches are brittle and heavily dependent on human insight and annotations. PROACT, for example, requires all objects to be tagged and each action to be labeled with the probability of object involvement.

More recently, supervised learning-based methods are gaining traction in recipe step detection. Datasets like YouCook2, which contains more than 2,000 cooking videos segmented into discrete annotated steps, enable training and fine-tuning of models for this task [6]. In particular, the authors of YouCook2 trained an LSTM-based model to segment and describe cooking videos into steps, which works relatively well on new recipe videos. However, supervised methods have limitations. They require high cost and time for annotation, training, and fine-tuning.

In contrast, we propose a framework based on generative machine learning. Unlike prior task-specific supervised models, our approach uses self-supervised models like LLMs and VLMs. It does not require any sensors, instead relying purely on vision. This has several key advantages: (1) it works out of the box without need for fine-tuning or new hardware, (2) it is still robust, avoiding the brittleness of sensor-based probabilistic methods, and (3) it is more cost-effective. While we use some supervised models (e.g., YOLO) for the first method, we also have an alternative method that relies only on a self-supervised VLM.

3 METHODS

Below, we outline the high-level design of our framework and present two specific methods for implementing ARSD.

3.1 High Level Design

Figure 2a illustrates the prototype design at a high level. The AR Recipe Assistant runs on a device (phone, headset, etc.) and requires only a readable camera feed. It uses multiple machine learning models, such as YOLO and GPT-4o, which can run on-device or in the cloud. The application passes images from the camera stream to the models to retrieve detected objects, recipes, and recipe steps. These are then packaged into the AR UI for the user.

3.2 Recipe Generation

Before enabling recipe step detection, we first need a way to generate a recipe from visual input. Since this is not our focus, we use a simple approach: capturing a camera frame on demand and querying the VLM to generate a recipe. Figure 2b illustrates this process, including the exact prompt template used. This method reliably produces well-formatted, enumerable recipes suitable for ARSD. Given the VLM's strong performance, we do not further analyze the recipe generation task in this paper.

3.3 Method 1: Object Detection w/ Language Models

Our first method for automatic recipe step detection consists of three phases: (1) object detection, (2) conversion into text-based world state trajectories, and (3) querying an LLM to determine the current step.

The intuition behind this method is that LLMs, despite being text-only, have strong spatial and temporal reasoning abilities. Research suggests that they develop an internal "world model" during pretraining, enabling them to learn rich spatio-temporal representations of the real world [7]. This allows LLMs to take spatial and temporal coordinates encoded in text and analyze them. For the purposes of ARSD, we use this capability to detect the current recipe step using only textual input. The key idea is to convert visual information into textual world state information. Our approach extracts object positions from camera frames, then provides this data—along with the recipe—to an LLM to infer the most likely step.

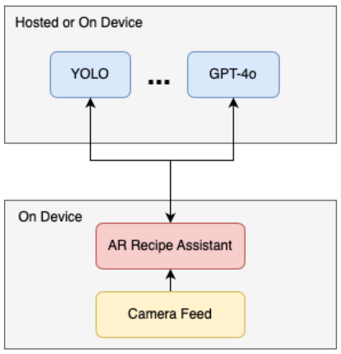
In the first phase, we use an object detection model like YOLO to identify all objects in a camera frame, filtering for food and kitchen-related items. As shown in Figure 3, YOLO is a convolutional neural network (CNN) model that detects objects in an image with bounding boxes, class labels, and estimated confidence scores [8].

In the second phase, we extract spatial information about the detected objects. For many recipes, cooking actions can be inferred from the movement of key objects like hands, utensils, and food items. By tracking these movements over time, we create spatio-temporal context for LLM analysis. Our transformation scheme is simple: we record object positions in camera space over time and present this as a textual list. Given a specific bounding box $B = \{(x_{min}, y_{min}), (x_{max}, y_{max})\}$ we compute the object's position in camera space as:

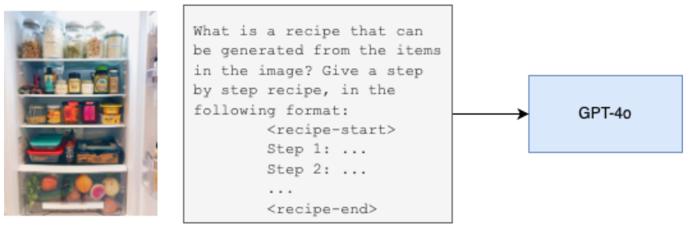
$$x_c = \frac{x_{min} + x_{max}}{2}, y_c = \frac{y_{min} + y_{max}}{2}$$

We continuously sample frames at a fixed rate, building a history of detected objects and their camera space positions.

In the final phase, we provide the LLM with the recorded spatio-temporal data and the recipe text, then ask it to infer the most likely step. Figure 4 illustrates this process: initially, YOLO detects a whole banana and its camera coordinates are recorded. Later, the banana is sliced into pieces with a knife. Based on these coordinate transitions



(a) High Level Design



```
What is a recipe that can
be generated from the items
in the image? Give a step
by step recipe, in the
following format:
        <recipe-start>
        Step 1: ...
        Step 2: ...
        ...
        <recipe-end>
```

(b) Recipe Generation

Fig. 2. Figure (a) illustrates the high-level design of the AR Recipe Assistant prototype. The main application runs on-device and can communicate with pluggable machine learning models, either locally or in the cloud. Figure (b) depicts the recipe generation process: when the user clicks a button, the current camera frame is sent to a VLM, which generates a structured, step-by-step recipe.

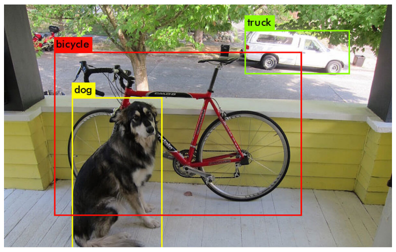
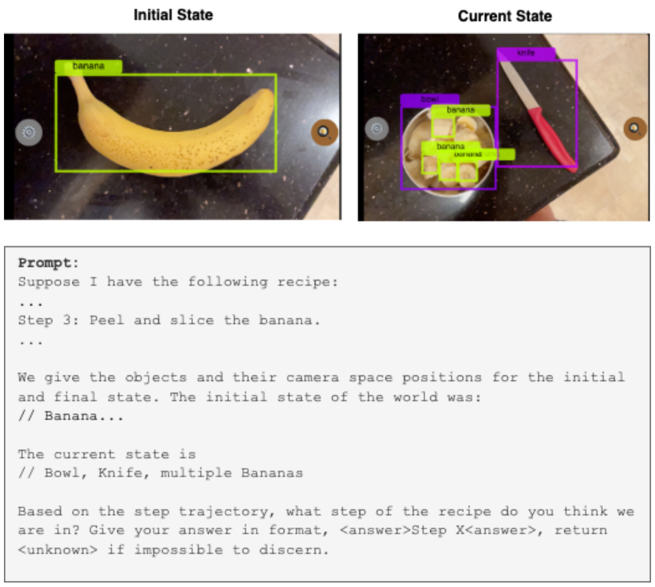


Fig. 3. Example of YOLO object detection in action. The model processes an image in real time, identifying objects such as a dog by drawing bounding boxes around them. Each detected object is assigned a class label (e.g., "dog") and a confidence score (not shown) indicating the model's certainty.



```
Prompt:
Suppose I have the following recipe:
...
Step 3: Peel and slice the banana.
...

We give the objects and their camera space positions for the initial
and final state. The initial state of the world was:
// Banana...

The current state is
// Bowl, Knife, multiple Bananas

Based on the step trajectory, what step of the recipe do you think we
are in? Give your answer in format, <answer>Step X<answer>, return
<unknown> if impossible to discern.
```

Fig. 4. Example of automatic recipe step detection using object state transitions. The initial state (left) shows a whole banana, while the current state (right) includes a knife and sliced banana pieces. The system tracks object state changes and spatial positions, transforming visual data into a structured text prompt for an LLM. The model then infers the most likely recipe step based on the detected state transitions.

in text form, the LLM infers that the user is on Step 3: slicing and peeling the banana.

3.4 Method 2: Vision-Language Model Approach

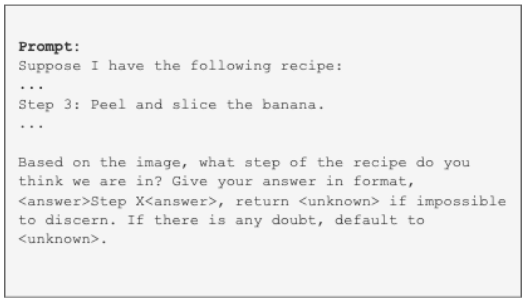
Our second method for ARSD is simpler and relies directly on a unified vision-language model (VLM). Unlike LLMs, VLMs are pre-trained on image-text pairs from the Internet using contrastive learning, allowing them to learn rich vision-language correlations [9]. They can natively tokenize and process images alongside text prompts.

Our approach is straightforward: to determine the current step, we capture the current frame and prompt the VLM to infer the most likely step. Figure 5 illustrates an example where the VLM correctly identifies sliced bananas and determines that the user is on Step 3.

Similar to Method 1, we can provide a historical sequence of images, starting with the initial image used to generate the recipe. This sequence can be passed to the VLM to give additional context about the user's actions. However, in practice, we do not use the full image history and instead provide only the current image (and optionally the first image from recipe generation). There are two main reasons for this choice. First, images are significantly more expensive to tokenize than text. Moderate to high quality images can require anywhere from a few hundred to a thousand tokens per image, compared to just a few dozen for text-based coordinates. This can quickly fill the VLM's context window, increasing the risk of hallucination. Second, for the simple recipes we evaluate, the current camera image by itself is sufficient for accurate step inference. A full visual history of kitchen states is usually unnecessary.

4 IMPLEMENTATION DETAILS

In this section, we go over specific implementation details for each part of the high level design, as shown in Figure 2a.



```
Prompt:
Suppose I have the following recipe:
...
Step 3: Peel and slice the banana.
...

Based on the image, what step of the recipe do you
think we are in? Give your answer in format,
<answer>Step X<answer>, return <unknown> if impossible
to discern. If there is any doubt, default to
<unknown>.
```

Fig. 5. Example of recipe step detection using a vision-language model (VLM). Given an image of sliced bananas and the corresponding recipe, the VLM is prompted to infer the current step.

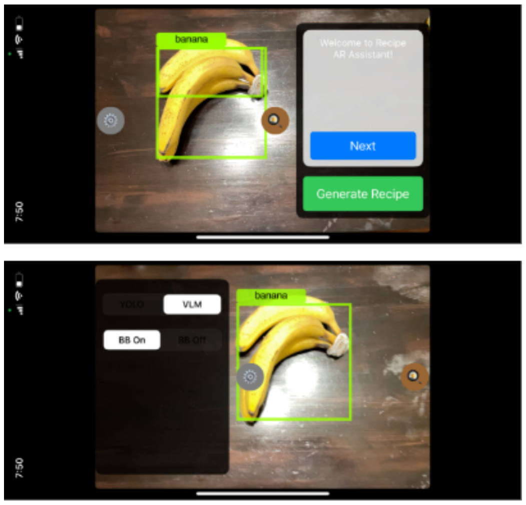


Fig. 6. UI of the AR Recipe Assistant app. The interface displays detected objects with bounding boxes (e.g., "banana") using YOLO. Users can generate a recipe by capturing a camera frame (Generate Recipe button) and navigate through detected steps (Next button). The settings panel (bottom image) allows users to toggle between YOLO and VLM-based processing methods and enable or disable bounding box visualization.

4.1 AR Recipe Assistant

The core application is built for iOS using Apple's UIKit for the user interface. As a prototype, it features a simple layout, as shown in Figure 6. The interface includes a text label displaying recipe steps and a *Generate Recipe* button. When pressed, the app captures a camera frame, sends it to a VLM (GPT-4o), parses the returned recipe, and updates the text label accordingly. For camera stream access, the app uses Apple's ARKit API.

By default, YOLO runs in the background, displaying bounding boxes for detected objects, though this feature can be disabled. Users can select between Method 1 (object detection with an LLM) and Method 2 (direct VLM inference) for processing. Once a recipe is generated, the app continuously captures and processes camera frames at fixed intervals (we use every 3 seconds), querying for the current step based on the chosen method. The UI updates automatically as new steps are detected, and a manual navigation button is available for corrections if needed.

4.2 YOLO (Object Detection)

4.2.1 Hosting the Model. A key consideration in our system design is where to run the YOLO model. Initially, we experimented with hosting YOLO on a Google Cloud VM with an A100 40GB GPU, implementing a basic server that received an image, processed it, and returned detected objects with labels and bounding boxes. However, we found that the network latency was too high for real-time bounding box rendering.

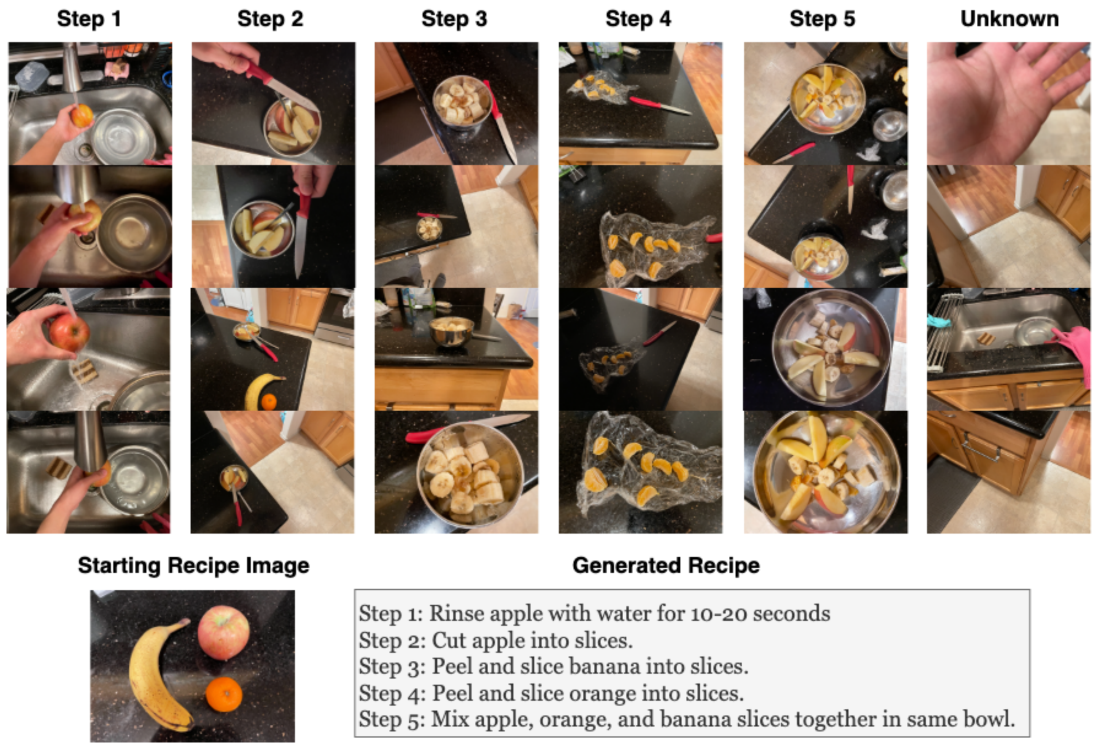
To address this, we run YOLO on-device with a CPU instead. We use YOLOv8 from Ultralytics, which offers five model sizes [10]. We found that YOLOv8n, the smallest model with 3.2M parameters, achieves acceptable latency even on a CPU (80.4 ms). Additionally, we observed no significant qualitative difference between YOLOv8n and larger models (up to 68.2M parameters), suggesting minimal performance degradation. As a result, we opted to run the model locally on devices such as phones or headsets.

4.2.2 Confidence Threshold. YOLO assigns a confidence score to each detected object, indicating the model's certainty in the classification. We apply a static confidence threshold of 0.5 to filter out low-confidence detections, reducing transient errors. This threshold ensures that detected objects remain stable, as high-confidence items persist regardless of variations in lighting or viewing angles. As a result, we did not need to implement additional smoothing techniques like Kalman filtering, as qualitative testing showed that the bounding boxes remained highly stable.

4.2.3 Occlusions. One challenge is occlusion—instances where the user unintentionally obstructs objects, such as covering the lens with their hand or pointing it in an unintended direction. In most cases, however, the model detects these disruptions and responds appropriately by returning an <unknown> token, staying robust in uncertain conditions.

4.2.4 Food Category Filter. YOLO is trained on the COCO dataset, which includes 80 object categories, many of which are unrelated to food or kitchen environments. To ensure relevance, we manually filter out non-food items from the detected results.

As an aside, this approach limits the recognized food categories to primarily fruits and vegetables, restricting the types of recipes the system can generate. Although not discussed further in this paper, we believe that YOLO could be fine-tuned on food-specific



Step 1: Rinse apple with water for 10-20 seconds
Step 2: Cut apple into slices.
Step 3: Peel and slice banana into slices.
Step 4: Peel and slice orange into slices.
Step 5: Mix apple, orange, and banana slices together in same bowl.

Fig. 7. Evaluation task for ARSD, illustrating a step-by-step fruit salad recipe. Each step is represented by multiple images under varying conditions. The starting recipe image (bottom left) is used to generate the structured recipe (bottom right). The unknown category includes occluded or misleading images that should not correspond to any valid step.

datasets to recognize more ingredients. This would allow it to detect a broader range of recipe steps.

4.3 GPT-4o (LLM/VLM)

For both the LLM and VLM components, we use OpenAI's GPT-4o via the OpenAI API [11]. GPT-4o is a multimodal model capable of processing text, images, and audio. Since OpenAI has deprecated its pure text-based LLM models, we use GPT-4o for both functionalities. To simulate an LLM-only setup, we restrict inputs to text prompts, while for VLM functionality, we include images as input.

5 EVALUATION

Our evaluation focuses on two key questions:

(1) What are the accuracy, cost, and latency tradeoffs between Method 1 and Method 2?
(2) What are the main qualitative observations that can be made about both in terms of ARSD?

5.1 Experimental Setup

5.1.1 Hardware. All experiments are run on an iPhone 12, using on-device YOLO and server API calls for GPT-4o. The camera frame rate is 60 fps.

5.1.2 Parameters. For both Method 1 and Method 2, the application samples a camera frame every 3 seconds for ARSD detection. YOLO uses a confidence threshold of 0.5, and GPT-4o is queried once per sample.

5.1.3 Evaluation. To quantitatively compare Method 1 and Method 2 on ARSD, we construct an evaluation task for detecting recipe steps. Figure 7 provides an overview of the exact recipe steps and corresponding images used for evaluation.

Given the limited food categories in YOLO's COCO dataset, we use a simple fruit salad recipe to ensure fair detection across both methods. For each recipe step, we capture multiple images that visually indicate the step in progress. These images vary in angles, lighting conditions, and environmental factors to test model robustness. Additionally, we include a starting anchor image, which is required for Method 1 to establish the initial world state. We also include several auxiliary images containing occlusions or misleading objects, which should be classified as <unknown>.

We assess latency, cost, and accuracy for both methods, defined as follows:

Accuracy: Formulated as a classification task, where each method must correctly identify the current recipe step given an input image. Accuracy is measured as the percentage of correctly classified images out of a total of 24 test images.

Latency: Measured as the end-to-end processing time, from when an image starts processing to when GPT-4o returns the predicted step token.

Method	Avg. Latency (s)	Accuracy (%)	Cost per Request ($)
Method 1 (YOLO + LLM)	3.34	88%	0.0007875
Method 2 (VLM)	6.91	100%	0.0028575

Table 1. Performance comparison between Method 1 and Method 2 for ARSD.

Cost: Calculated as the average request cost per sample when querying GPT-4o. Since GPT-4o is the primary cost factor, we do not include additional computational expenses.

5.2 Quantitative Results

Table 1 presents the quantitative comparison between Method 1 (YOLO + LLM) and Method 2 (VLM-based inference) in terms of accuracy, latency, and cost per request.

Both methods demonstrate strong performance in detecting the correct recipe step. Method 2 achieves perfect accuracy (100%), correctly identifying every step in the evaluation set. In contrast, Method 1 achieves 88% accuracy, indicating some misclassifications but still performing well overall. This suggests that direct vision-language modeling is better overall for ARSD in terms of raw visual understanding and accuracy.

However, although Method 2 is more accurate, it comes at the cost of significantly higher latency. Method 1 has a 48% lower average latency, processing each request in 3.34 seconds, compared to 6.91 seconds for Method 2. This reduction in latency is critical for a real-time AR cooking assistant, where fast step detection is necessary to provide users with immediate feedback. In a live cooking scenario, delays in recognizing and updating steps could result in a frustrating user experience, making low-latency solutions vital for interactive AR applications.

Cost is another key factor in assessing the feasibility of deploying these models at scale. Method 1 is 3.62 times more cost-efficient than Method 2, with an average request cost of 0.0007875 compared to 0.0028575 per request for Method 2. This difference is largely due to the high computational cost of processing images in Method 2, where each image incurs significant tokenization overhead when passed to GPT-4o. Method 1, being purely text-based, avoids this bottleneck.

Overall, the results highlight a trade-off between accuracy and efficiency. If accuracy is the highest priority, Method 2 is the clear choice. If real-time responsiveness and cost-efficiency are more important, Method 1 is preferable, as it provides quicker updates and significantly lower costs while maintaining decent accuracy. Moreover, Method 1's performance can likely be improved by adjusting several architectural parameters, such as: (1) using a larger YOLO model for more precise object detection, (2) adding more context to the LLM with world-space state tokens, and (3) fine-tuning the confidence threshold to reduce false positives. These optimizations would shift the balance between accuracy and efficiency, allowing for a more flexible trade-off depending on the specific needs of the application.

5.3 Qualitative Observations

Beyond the quantitative results, we note several qualitative differences between Method 1 and Method 2 that highlight their respective strengths and limitations.

One key observation is that Method 1 is best suited for shorter recipes with a limited number of steps. This is due to the fact that Method 1 relies on tracking an unbounded list of objects and their position histories over time. As a recipe grows longer, the accumulated history increases in size, which can significantly expand the token count required for the LLM prompt. In extreme cases, the prompt length for Method 1 could exceed that of Method 2, potentially making it more expensive to run despite its initial cost efficiency. In contrast, Method 2 operates with a constant number of images, typically using one anchor image and one current image. While Method 2 has a high startup cost per request, this cost remains stable as the recipe progresses. In scenarios where a recipe involves dozens of steps over an extended period, Method 2 could become more scalable and cost-effective than Method 1.

Another important difference is category generalization. Method 1 is inherently constrained by YOLO's object detection categories, which are based on the COCO dataset. While COCO contains 80 object classes, only about 20 are related to food and kitchen objects. This means that Method 1 cannot detect certain ingredients, tools, or niche cooking items unless YOLO is fine-tuned on additional food datasets. However, fine-tuning YOLO on custom datasets introduces additional costs for data annotation and training, making it a non-trivial improvement. Method 2, on the other hand, is likely trained on a significant chunk of Internet images, allowing it to generalize across nearly any recipe image. This flexibility means that Method 2 can handle a far wider variety of ingredients and cooking tools, whereas Method 1 may fail when encountering objects outside of its pre-trained detection set.

These observations further reinforce the trade-offs between the two methods. A practical AR recipe assistant could combine both approaches, using Method 1 for short, simple recipes to minimize costs, while switching to Method 2 for longer, more intricate recipes, where accuracy and generalization are more critical.

6 LIMITATIONS AND FUTURE WORK

A significant limitation of our work is the limited scope of our evaluation dataset. Due to time and cost constraints, we were unable to construct a large-scale, diverse dataset that would better reflect real-world ARSD scenarios. Ideally, future work should incorporate a larger, more diverse dataset of annotated recipe images or even cooking videos, each labeled with ground-truth step annotations. Such a dataset would provide a more rigorous benchmark for evaluating ARSD methods and enable further improvements in model performance.

Looking ahead, there are several promising directions for future work, spanning both algorithmic improvements and system-level optimizations. On the algorithmic side, our current methods are relatively simple compared to what is possible with modern LLMs and VLMs. Future research could explore more sophisticated techniques, such as prompt engineering strategies, retrieval-augmented generation (RAG) with structured recipe templates, or multi-step reasoning approaches that dynamically adjust model queries.

On the systems side, there is considerable room for optimization in terms of real-time performance. Our current implementations exhibit latencies measured in seconds, which is suboptimal for real-time AR applications. A major bottleneck is network overhead, as we rely on cloud-based inference for model processing. One potential solution is to distill a lightweight, on-device model specifically optimized for recipe step detection, enabling inference with low latency while maintaining acceptable accuracy. Additionally, application-level optimizations could improve performance, such as batch processing of frames, caching mechanisms, and more efficient scheduling of API requests.

Beyond ARSD, our framework highlights the broader potential of LLMs and VLMs for AR-based assistance. These models are highly generalizable and can be adapted to various AR-guided tasks beyond cooking. Even within the cooking domain, LLMs could be leveraged for setting timers, providing cooking tips, suggesting ingredient substitutions, and offering context-aware recommendations based on real-time observations. Furthermore, the emergence of AI agents introduces new possibilities for assistants that actively engage with users, guiding them through complex real-world tasks. Future research could explore the intersection of AR assistance and AI agent frameworks, combining vision, language, and real-time interactivity to create more intelligent, context-aware applications.

7 CONCLUSION

In this paper, we introduced the task of automatic recipe step detection (ARSD) for augmented reality (AR) cooking assistants. We proposed a general framework for ARSD and presented two distinct approaches: (1) a method combining object detection with an LLM-based inference system, and (2) a direct vision-language model (VLM) approach. Through our evaluation, we demonstrated that while the VLM-based approach achieves superior accuracy, it incurs higher latency and cost, whereas the object detection–based approach offers a more cost-effective and efficient alternative, albeit with slightly lower accuracy. Our findings highlight the trade-offs between accuracy, cost, and responsiveness in AR-assisted cooking applications.

REFERENCES

[1] Mehdi Mekni and André Lemieux. Augmented reality : Applications , challenges and future trends. 2014.
[2] Microsoft. Hololens documentation, 2025. Accessed: 2025-03-19.
[3] Meta. Meta reality labs - orion, 2025. Accessed: 2025-03-19.
[4] Apple Inc. Apple vision pro, 2025. Accessed: 2025-03-19.
[5] M. Philipose, K.P. Fishkin, M. Perkowitz, D.J. Patterson, D. Fox, H. Kautz, and D. Hahnel. Inferring activities from interactions with objects. *IEEE Pervasive Computing*, 3(4):50–57, 2004.
[6] Luowei Zhou, Chenliang Xu, and Jason J. Corso. Procnets: Learning to segment procedures in untrimmed and unconstrained videos. *CoRR*, abs/1703.09788, 2017.
[7] Wes Gurnee and Max Tegmark. Language models represent space and time, 2024.
[8] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2016.
[9] Jingyi Zhang, Jiaxing Huang, Sheng Jin, and Shijian Lu. Vision-language models for vision tasks: A survey, 2024.
[10] Ultralytics. Yolov8 documentation, 2025. Accessed: 2025-03-19.
[11] OpenAI. Hello gpt-4o, 2024. Accessed: 2025-03-19.