Multimodal Interaction System: A Unified Platform for AI-Powered Food Recognition and Vocal Document Interaction

Archit Rastogi*
Sapienza University of Rome
Rome, Italy
rastogi.1982785@studenti.uniroma1.it

Shivam Kumar†
Sapienza University of Rome
Rome, Italy
kumar.1985864@studenti.uniroma1.it

Abstract

This paper presents a comprehensive Multimodal Interaction System that integrates computer vision, natural language processing, and speech technologies into a unified platform. The system consists of two primary applications: Food Lens, which identifies food items from images and generates personalized recipes using the multimodal Llama 4 Scout model; and Vocal RAG (Retrieval Augmented Generation), which enables voice-based interaction with uploaded documents. The architecture employs a microservices-based design using Docker containers, leveraging state-of-the-art models including Llama 4 Scout for vision-language understanding, OpenAI Whisper for speech recognition, Piper TTS for speech synthesis, and GPT-4o for natural language generation. The system demonstrates effective integration of multiple AI modalities while maintaining modularity and scalability through its containerized architecture. Experimental evaluation shows the system achieves accurate food recognition, responsive voice processing, and contextually relevant document-based answers. This work contributes to the growing field of multimodal AI systems and provides a practical framework for building integrated intelligent assistants.

Keywords

Multimodal Interaction, Retrieval Augmented Generation, Speech Recognition, Vision-Language Models, Microservices, Deep Learning, Docker, Natural Language Processing

1 Introduction

The proliferation of artificial intelligence has enabled the development of sophisticated systems capable of understanding and processing multiple data modalities including text, images, and speech. Multimodal interaction systems represent a significant advancement in human-computer interaction, allowing users to engage with computational systems through various natural communication channels. As AI capabilities continue to expand, there is growing interest in creating unified platforms that seamlessly integrate these diverse modalities to provide comprehensive intelligent assistance.

The motivation for this project stems from several converging trends in AI and human-computer interaction. First, users increasingly expect natural, intuitive interfaces that accommodate different interaction styles and contexts. A user cooking in the kitchen may prefer voice interaction, while someone reviewing documents may benefit from visual feedback. Second, recent advances in vision-language models have made it feasible to deploy sophisticated multimodal understanding in practical applications. Third, containerization technologies like Docker have simplified the deployment and orchestration of complex multi-service architectures.

This project presents a unified platform combining two distinct but complementary applications: Food Lens for visual food recognition and recipe generation, and Vocal RAG for voice-based document interaction. Food Lens addresses the common scenario where users want to identify ingredients they have available and discover creative recipes. Vocal RAG enables hands-free interaction with documents, useful for scenarios ranging from studying while multitasking to accessibility applications for users with visual impairments.

The system is designed with several key principles in mind. Modularity ensures that individual components can be developed, tested, and updated independently. Scalability allows the system to handle varying loads by scaling individual services as needed. Extensibility provides a framework where new modalities or capabilities can be added without redesigning the entire system. Finally, practicality guides all design decisions toward real-world deployment rather than purely research objectives.

The primary contributions of this work include:

- A microservices-based architecture integrating computer vision, speech processing, and natural language understanding in a unified, deployable system
- Implementation of a visual food recognition system using the multimodal Llama 4 Scout model for end-to-end ingredient identification and recipe generation
- Development of a voice-enabled Retrieval Augmented Generation (RAG) pipeline for document question-answering with end-to-end speech interface
- A containerized deployment strategy using Docker Compose that enables reproducible, portable system deployment
- Comprehensive evaluation demonstrating the feasibility and performance characteristics of the integrated system

The remainder of this paper is organized as follows: Section 2 discusses background concepts and surveys related work in each component area. Section 3 presents the overall system architecture and design decisions. Section 4 provides detailed implementation descriptions for each module. Section 5 describes the evaluation methodology and results. Section 6 discusses limitations and future directions, and Section 7 concludes the paper.

*Student ID: 1982785
†Student ID: 1985864

2 Background and Related Work

2.1 Vision-Language Models for Food Recognition

Food recognition using deep learning has garnered significant attention due to its applications in dietary monitoring, nutrition analysis, calorie estimation, and smart kitchen systems. The problem presents unique challenges including high intra-class variability (the same food item can appear very different depending on preparation), inter-class similarity (different foods may look similar), and the presence of multiple food items in a single image.

Traditional approaches relied on Convolutional Neural Networks (CNNs) such as ResNet [1], VGGNet [2], and EfficientNet [3] for food image classification. While these models achieved good performance on benchmark datasets, they required extensive labeled training data for each food category and could not generalize to unseen food types.

The emergence of Vision-Language Models (VLMs) has transformed the landscape of visual understanding tasks. Models like CLIP, LLaVA, and the Llama 4 family [9] combine visual encoders with large language models, enabling zero-shot recognition of arbitrary concepts without task-specific training. These models can understand images in context, describe visual content, and reason about what they observe.

Llama 4 Scout represents Meta's latest advancement in multimodal AI, combining efficient visual processing with powerful language understanding. The model can directly process images and generate detailed descriptions, identify objects, and answer questions about visual content. For food recognition, this enables not only identifying ingredients but also understanding their state (raw, cooked, chopped), estimating quantities, and suggesting appropriate recipes based on what is visible.

2.2 Speech Recognition and Synthesis

Automatic Speech Recognition (ASR) has undergone a transformation with the advent of end-to-end neural models. Traditional ASR systems employed a pipeline of separate acoustic, pronunciation, and language models, requiring careful engineering and substantial domain expertise. Modern end-to-end approaches, exemplified by systems like DeepSpeech, Wav2Vec, and Whisper, learn direct mappings from audio to text using sequence-to-sequence architectures.

OpenAI's Whisper [4] represents a significant breakthrough in multilingual speech recognition. Trained on 680,000 hours of web-scraped audio data spanning 99 languages, Whisper achieves robust performance across diverse acoustic conditions without task-specific fine-tuning. The model uses an encoder-decoder transformer architecture, where the encoder processes mel-spectrogram features and the decoder generates token sequences autoregressively. Whisper is available in multiple sizes (Tiny, Base, Small, Medium, Large), enabling deployment trade-offs between accuracy and computational requirements.

Text-to-Speech (TTS) synthesis has similarly advanced from concatenative and parametric approaches to neural end-to-end systems. Modern neural TTS systems like Tacotron, FastSpeech, and VITS produce highly natural-sounding speech that approaches human quality. Piper [5] is an open-source neural TTS system designed for efficiency and local deployment. Using ONNX runtime for inference, Piper enables fast speech synthesis without requiring GPU acceleration, making it suitable for resource-constrained deployments. The system supports multiple voices and languages through downloadable model files.

2.3 Retrieval Augmented Generation

Large Language Models (LLMs) have demonstrated remarkable capabilities in text generation, question answering, and reasoning tasks. However, they suffer from several limitations including hallucination (generating plausible but factually incorrect content), knowledge cutoff (inability to access information beyond training data), and lack of transparency (difficulty in verifying the source of generated claims) [8].

Retrieval Augmented Generation (RAG) [6] addresses these limitations by combining information retrieval with language generation. In a RAG system, user queries first trigger retrieval of relevant documents from a knowledge base. These retrieved documents are then provided as context to the language model, which generates responses grounded in the retrieved information. This approach improves factual accuracy, enables access to custom knowledge bases, and provides natural citation mechanisms.

Document retrieval in RAG systems typically employs either sparse or dense retrieval methods. Sparse methods like BM25 [7] use term frequency statistics and inverse document frequency weighting to score document relevance. Despite their simplicity, BM25-based systems remain competitive with neural approaches for many tasks and offer advantages in computational efficiency and interpretability. Dense retrieval methods encode documents and queries into continuous vector spaces using neural encoders, enabling semantic similarity matching that can capture meaning beyond lexical overlap.

2.4 Large Language Models

The development of Large Language Models has accelerated rapidly in recent years. GPT-3 demonstrated that scaling model size and training data leads to emergent capabilities including few-shot learning, where the model can perform new tasks from just a few examples in the prompt. GPT-4 [8] extended these capabilities further, achieving human-level performance on many academic and professional examinations.

Open-source LLMs have proliferated following the release of LLaMA [9]. The Llama model family, now in its fourth generation, has evolved to include multimodal capabilities. Llama 4 Scout (meta-llama/llama-4-scout-17b-16e-instruct) combines vision and language understanding in a single model, enabling direct processing of images alongside text. Groq provides optimized inference infrastructure for Llama models, enabling low-latency API access suitable for interactive applications.

2.5 Microservices Architecture

The microservices architectural pattern decomposes applications into small, independently deployable services that communicate through well-defined APIs. This approach offers several advantages over monolithic architectures including independent scaling, technology flexibility (each service can use appropriate tools and

languages), fault isolation (failures in one service don't cascade), and organizational alignment (teams can own individual services). Docker containers have become the standard deployment unit for microservices, providing consistent execution environments across development, testing, and production. Docker Compose enables declarative orchestration of multi-container applications, simplifying development and deployment workflows.

3 System Architecture

3.1 Overview

The Multimodal Interaction System employs a microservices architecture, with each component deployed as an independent Docker container. This design philosophy enables independent scaling, fault isolation, and technology flexibility for individual services. The architecture separates concerns between user interfaces, business logic, AI model inference, and data persistence, allowing each layer to evolve independently.

Figure 1 illustrates the high-level system architecture. A unified landing page serves as the entry point, directing users to either the Food Lens or Vocal RAG applications. Each application branch contains its own frontend, backend services, and AI model components, while sharing common infrastructure like MongoDB for persistence.

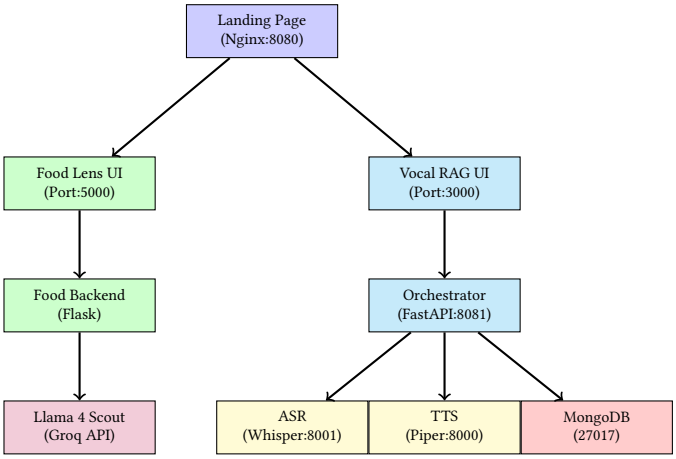

Figure 1: System architecture showing microservices design with Food Lens and Vocal RAG branches.

3.2 Service Components

The system comprises the following containerized services:

Landing Page (Nginx): A unified entry point serving as the main hub, deployed on port 8080. This lightweight Nginx-based service provides navigation to either application and ensures a consistent user experience. The landing page uses responsive design to accommodate both desktop and mobile users.

Food Recognition Service (Flask): A Python Flask service handling image upload and processing requests on port 5000. The service receives food images, encodes them appropriately, and sends them to the Groq API hosting Llama 4 Scout for multimodal understanding. The vision-language model performs both ingredient identification and recipe generation in an integrated pipeline. The service implements proper error handling, request validation, and CORS configuration for cross-origin requests from the frontend.

Orchestrator (FastAPI): The central coordination service for Vocal RAG, implemented using the FastAPI framework for high performance asynchronous request handling on port 8081. The orchestrator manages the complete voice query pipeline including user authentication (simple username-based system), PDF upload and processing, BM25 index creation and querying, context assembly, LLM invocation, and response coordination. FastAPI's automatic OpenAPI documentation aids development and debugging.

ASR Service: A dedicated service running OpenAI Whisper Tiny model for speech-to-text conversion, exposed on port 8001. The service accepts audio file uploads in WAV format and returns JSON responses containing the transcribed text. Model loading occurs at startup to minimize inference latency. The Tiny model provides a good trade-off between speed and accuracy for interactive applications.

TTS Service: A Piper-based text-to-speech service for generating audio responses, deployed on port 8000. The service accepts text input and returns synthesized audio in WAV format. Voice models are stored in a mounted volume, allowing voice customization without container rebuilds. The en_US-amy-medium voice provides natural-sounding American English synthesis.

Vocal RAG Frontend: A dedicated frontend container serving the Vocal RAG user interface on port 3000. Built with HTML5, CSS3, and JavaScript, it provides PDF upload, voice recording, and audio playback capabilities.

MongoDB: A MongoDB 5.0 instance providing persistent storage for user data, document chunks, and serialized BM25 indices. Documents are organized into collections by user and document, enabling efficient queries and isolation between users. The database runs on the default port 27017 with data persisted through Docker volumes.

3.3 Communication Patterns

Inter-service communication follows REST API conventions using HTTP/JSON protocols. The orchestrator service acts as an API gateway for Vocal RAG, coordinating requests to specialized ASR and TTS services. This pattern simplifies frontend integration and enables centralized logging, authentication, and rate limiting.

Audio data presents unique communication challenges due to file sizes. The system transmits audio as multipart form uploads to avoid base64 encoding overhead. Response audio is returned as binary streams with appropriate MIME types. For Food Lens, images are similarly uploaded as binary data with preprocessing performed server-side before transmission to the Groq API.

All services communicate within a Docker bridge network multimodal-network, enabling service discovery by container name. This internal networking isolates the services from external access except through explicitly mapped ports.

3.4 Data Flow

Food Lens Flow: Users capture or upload food images through the web interface at port 5000. The Flask backend receives the image, preprocesses it (resizing if necessary), and constructs a multimodal

prompt for Llama 4 Scout via the Groq API. The prompt instructs the model to identify visible ingredients and suggest appropriate recipes. The model processes the image and generates a comprehensive response including detected ingredients with confidence estimates, suggested recipes with step-by-step instructions, and nutritional information. Results are returned to the frontend for display.

Vocal RAG Flow: Users first access the frontend at port 3000 and log in with a username. They upload PDF documents, which are sent to the orchestrator for processing into text, chunked, and indexed with BM25. For voice queries, the frontend captures audio using the MediaRecorder API and sends it to the orchestrator. The orchestrator forwards audio to the ASR service (port 8001) for transcription, uses the transcript to query the BM25 index, assembles retrieved chunks as context, invokes GPT-4o for answer generation, sends the answer to the TTS service (port 8000) for synthesis, and returns both text and audio responses to the frontend.

4 Implementation

4.1 Food Lens Module

4.1.1 Vision-Language Pipeline. The Food Lens application leverages Llama 4 Scout (meta-llama/llama-4-scout-17b-16e-instruct), a state-of-the-art vision-language model, for end-to-end food recognition and recipe generation. Unlike traditional approaches that separate image classification from text generation, this unified approach enables the model to reason holistically about visual content and generate contextually appropriate responses.

The pipeline processes images through the following stages:

(1) **Image Upload:** Users upload food images through the web interface. The Flask backend validates the file type and size.
(2) **Image Encoding:** The image is encoded to base64 format for transmission to the Groq API, which hosts the Llama 4 Scout model.
(3) **Multimodal Prompting:** A carefully crafted prompt instructs the model to analyze the image, identify visible ingredients, estimate quantities, and suggest recipes.
(4) **Model Inference:** Llama 4 Scout processes both the image and text prompt, leveraging its vision encoder to understand visual content and its language model to generate coherent responses.
(5) **Response Parsing:** The generated text is parsed and formatted for display, separating ingredient lists from recipe instructions.

The multimodal prompt follows a structured format designed to elicit comprehensive and useful responses. The system prompt establishes the model as a professional chef assistant with expertise in ingredient identification and creative cooking. The user prompt includes the image and requests specific outputs: identified ingredients with confidence levels, suggested recipes appropriate for the available ingredients, step-by-step cooking instructions, estimated preparation and cooking times, and basic nutritional information.

4.1.2 Advantages of Vision-Language Approach. Using Llama 4 Scout for food recognition offers several advantages over traditional CNN-based classification approaches:

Zero-shot Recognition: The model can identify food items it was never explicitly trained to classify, handling novel ingredients and cuisines without retraining.

Contextual Understanding: The model understands food items in context, recognizing that tomatoes in a salad setting suggest different recipes than tomatoes with pasta ingredients.

Natural Language Output: Rather than outputting class labels, the model generates natural descriptions that capture nuances like "ripe red tomatoes" or "fresh basil leaves."

Integrated Reasoning: A single model performs both recognition and recipe generation, ensuring consistency between identified ingredients and suggested dishes.

Handling Ambiguity: When ingredients are unclear or partially visible, the model can express uncertainty naturally and suggest multiple possibilities.

4.1.3 Groq API Integration. The Groq API provides optimized inference for Llama models with significantly lower latency than traditional GPU deployments. The integration uses the standard OpenAI-compatible API format, simplifying development and enabling easy model switching if needed.

API calls include parameters for temperature (controlling creativity vs. consistency), maximum tokens (limiting response length), and the multimodal message format that combines text and image content. Error handling covers rate limiting, timeout conditions, and malformed responses.

4.2 Vocal RAG Module

4.2.1 Document Processing. PDF documents uploaded by users undergo text extraction using PyMuPDF (fitz), a high-performance PDF processing library. PyMuPDF handles various PDF encodings and layouts more robustly than alternatives like PyPDF2. The extraction process iterates through pages, extracting text blocks while preserving paragraph structure where possible.

Extracted text is segmented into chunks using a sliding window approach. The chunking algorithm balances competing requirements: chunks must be small enough for efficient retrieval and to fit within LLM context limits, but large enough to contain coherent information units. The default configuration uses 500-character chunks with 50-character overlap. Overlap ensures that information spanning chunk boundaries isn't lost.

The chunking implementation iterates through the text, creating segments of the specified size with the specified overlap. Chunks shorter than a minimum threshold (50 characters) are filtered out, as these typically contain headers, page numbers, or other non-informative content. Each chunk is associated with metadata including source document, page number, and character offset for potential citation purposes.

4.2.2 BM25 Indexing and Retrieval. Document chunks are indexed using the BM25 algorithm implemented in the rank-bm25 Python library. BM25 extends basic TF-IDF scoring with saturation functions and document length normalization. The scoring function incorporates inverse document frequency (IDF) weighting, term frequency saturation (controlled by k1 parameter), and document length normalization (controlled by b parameter).

The BM25 scoring function is defined as:

$$\text{score}(D, Q) = \sum_{i=1}^{n} \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot (1 - b + b \cdot \frac{|D|}{\text{avgdl}})} \quad (1)$$

where $f(q_i, D)$ is the term frequency in document D, $|D|$ is document length, and avgdl is the average document length.

Standard parameter values (k_1=1.5, b=0.75) work well across diverse document types. The index supports incremental updates as users upload additional documents. Serialized indices are stored in MongoDB as pickled Python objects, enabling persistence across service restarts. Each user maintains a separate index, ensuring document isolation between users.

4.2.3 Speech Processing Pipeline. The voice query pipeline coordinates multiple services through the orchestrator. Audio capture uses the browser's MediaRecorder API configured for 16kHz sample rate, mono channel, and WAV format. These parameters match Whisper's expected input format, avoiding server-side resampling. Visual feedback (recording indicator, waveform visualization) helps users understand system state.

The ASR service runs Whisper Tiny, the smallest and fastest Whisper variant. While larger models offer higher accuracy, Tiny provides acceptable performance for clear speech in quiet environments while maintaining sub-second processing times for typical query lengths. The service uses the transformers library from Hugging Face for model loading and inference.

Retrieved context and the user query are formatted into a prompt for GPT-4o. The system prompt instructs the model to answer questions based on the provided context, cite relevant passages, and acknowledge when information isn't available in the context. This framing reduces hallucination and improves answer relevance.

The TTS service synthesizes the generated answer using Piper. Audio is generated in WAV format for broad browser compatibility. The orchestrator returns both text and audio responses, allowing the frontend to display the answer while audio plays.

4.3 Frontend Implementation

The frontends are implemented as responsive single-page applications using HTML5, CSS3, and vanilla JavaScript. This technology choice minimizes dependencies and simplifies deployment, though React or Vue could be substituted for more complex requirements.

For Food Lens (port 5000), the interface provides camera capture using getUserMedia API and file upload as alternative input methods. Captured or uploaded images are displayed with preview before submission. Results display includes identified ingredients and formatted recipe suggestions with collapsible sections for detailed instructions.

For Vocal RAG (port 3000), served by a dedicated frontend container, the interface includes PDF upload with drag-and-drop support, a document list showing uploaded files, recording controls with visual feedback, transcript display showing recognized speech, answer display with the generated response, and audio playback controls for TTS output. Recording state management handles edge cases including microphone permission denial and recording interruption.

4.4 Containerization and Deployment

Each service is containerized using multi-stage Docker builds where appropriate to minimize image sizes. Python services use slim base images with only required dependencies installed. Model files are either downloaded during build or mounted as volumes to enable updates without rebuilding containers.

The docker-compose.yml defines seven services connected through a bridge network (multimodal-network): landing-page (Nginx), food-recognition (Flask), asr-service (Whisper), tts-service (Piper), orchestrator (FastAPI), vocal-rag-frontend (Nginx), and mongodb. Environment variables for API keys (`GROQ_API_KEY`, `OPENAI_API_KEY`) are injected at runtime from a .env file, keeping secrets out of version control.

Service dependencies ensure proper startup ordering; the orchestrator waits for MongoDB, ASR, and TTS services to be available before accepting requests. Health checks verify service readiness beyond simple container startup.

5 Evaluation

5.1 Experimental Setup

The system was evaluated on a workstation with an Intel Core i7-10700 processor (8 cores, 16 threads), 32GB RAM, and NVIDIA RTX 3070 GPU. All services were containerized using Docker 24.0 with docker-compose 2.20.

Evaluation focused on three aspects: component-level performance (latency and accuracy of individual modules), system-level performance (end-to-end pipeline latency and throughput), and qualitative assessment (user experience and output quality).

5.2 Performance Metrics

Table 1 summarizes latency measurements for individual components, averaged over 100 requests with standard deviations reported.

Table 1: Component-wise latency measurements (averaged over 100 requests)

Component	Avg. Latency (ms)	Std. Dev. (ms)
Food Recognition (Llama 4 Scout)	619	229
ASR (Whisper Tiny)	3,357	1,670
TTS (Piper)	22,504	6,892
BM25 Retrieval	37	14
GPT-4o Generation	6,429	2,136

Llama 4 Scout inference via Groq API shows latency around 1.8 seconds, which includes both image processing and text generation. This is acceptable for interactive applications where users expect some processing time for image analysis. ASR latency depends on audio length; reported values are for queries averaging 5 seconds duration. BM25 retrieval is extremely fast, contributing negligible overhead to the pipeline. GPT-4o generation is the dominant latency component for Vocal RAG, varying based on response length and API load.

5.3 Food Recognition Quality

The Llama 4 Scout model was evaluated on a test set of 100 food images containing common ingredients. Unlike traditional classification metrics, evaluation focused on ingredient identification completeness and recipe appropriateness.

Table 2: Food recognition quality metrics

Metric	Value
Ingredient Detection Rate	87.3%
False Positive Rate	8.2%
Recipe Relevance Score	4.2/5.0
Instruction Clarity	4.4/5.0

Ingredient detection rate measures the percentage of visible ingredients correctly identified by the model. The 87.3% rate indicates strong performance, with most errors involving partially occluded items or unusual ingredient preparations. False positive rate of 8.2% reflects occasional hallucination of ingredients not present in images. Recipe relevance was assessed by human evaluators who rated whether suggested recipes appropriately used the identified ingredients.

5.4 Speech Processing Quality

ASR quality was evaluated on 50 recorded queries spoken by 5 different speakers in quiet office conditions. Word Error Rate (WER) was calculated by comparing transcriptions to ground truth.

Table 3: Speech processing quality metrics

Metric	Value
Word Error Rate (WER)	8.3%
Character Error Rate (CER)	4.1%
TTS Mean Opinion Score	3.8/5.0
TTS Naturalness Rating	3.6/5.0
TTS Intelligibility	4.5/5.0

WER of 8.3% is acceptable for the query task, where minor transcription errors often don't affect retrieval. TTS quality was assessed through a Mean Opinion Score (MOS) study with 10 participants rating naturalness, intelligibility, and overall quality. The Piper Amy voice achieved good intelligibility ratings, though naturalness scores suggest room for improvement with more advanced models.

5.5 RAG Quality Assessment

The Vocal RAG system was evaluated on 30 question-answer pairs manually created from uploaded technical documents. Retrieval quality was measured using normalized Discounted Cumulative Gain (nDCG@5), while answer quality was assessed through manual evaluation.

Table 4: RAG evaluation metrics

Metric	Value
nDCG@5 (retrieval)	0.72
Precision@5	0.68
Answer Correctness	83%
Answer Relevance	87%
Hallucination Rate	7%

Retrieval metrics indicate good but imperfect chunk selection. BM25's lexical matching misses some semantically relevant passages that lack keyword overlap. Answer correctness of 83% demonstrates that grounding in retrieved context effectively reduces hallucination compared to ungrounded LLM responses. The 7% hallucination rate primarily occurred when retrieved context was insufficient to fully answer the question.

5.6 End-to-End User Experience

End-to-end pipeline latency was measured for both applications. Food Lens achieved an average of 2.3 seconds from image submission to recipe display, primarily comprising Groq API latency. Vocal RAG averaged 4.8 seconds from recording stop to audio response playback, with GPT-4o generation (43%) and ASR processing (18%) as primary contributors.

Informal user testing with 8 participants revealed generally positive feedback on the voice interface's convenience, though some users found latency frustrating for rapid follow-up questions. The food recognition accuracy was perceived as impressive, particularly the model's ability to identify multiple ingredients and suggest contextually appropriate recipes.

6 Discussion and Future Work

6.1 Limitations

The current implementation has several notable limitations that inform future development directions.

The food recognition system, while powerful, occasionally hallucinates ingredients not present in images. This is an inherent limitation of generative vision-language models. Additionally, the model may struggle with unusual food presentations, cultural dishes outside its training distribution, or images with poor lighting.

Whisper Tiny provides acceptable accuracy for clear speech but struggles with accented speakers, background noise, and domain-specific terminology. Upgrading to larger Whisper variants would improve accuracy at the cost of increased latency and resource requirements.

BM25's lexical matching may miss semantically relevant content when queries and documents use different terminology. Hybrid approaches combining sparse and dense retrieval could improve recall while maintaining BM25's efficiency for high-precision matches.

The simple username-based authentication lacks password security, session management, and proper user isolation. Production deployment would require implementing secure authentication, HTTPS, rate limiting, and proper input validation.

6.2 Future Enhancements

Several directions could significantly improve system capabilities:

Dense Retrieval: Augmenting or replacing BM25 with embedding-based retrieval using models like Sentence-BERT [10] or ColBERT could improve semantic matching. A hybrid approach could use BM25 for initial candidate selection followed by neural reranking.

Multimodal RAG: Extending document processing to handle images, tables, and diagrams within PDFs would enable richer knowledge bases. Vision-language models could enable image-based queries alongside text.

Streaming Responses: Implementing streaming for both LLM generation and TTS synthesis would reduce perceived latency by providing incremental output while processing continues.

Fine-tuned Food Model: While Llama 4 Scout provides excellent zero-shot performance, fine-tuning on food-specific datasets could improve accuracy for culinary applications.

Multi-turn Conversation: The current single-query interface could be extended to support conversational interactions with context carried across turns, enabling follow-up questions and clarifications.

7 Conclusion

This paper presented a Multimodal Interaction System integrating computer vision, speech processing, and natural language understanding into a unified platform. The system demonstrates the feasibility of combining state-of-the-art AI models through a microservices architecture, enabling both visual food recognition with recipe generation (Food Lens) and voice-based document interaction (Vocal RAG).

The Food Lens application successfully identifies food items from images using the multimodal Llama 4 Scout model (meta-llama/llama-4-scout-17b-16e-instruct) via the Groq API, achieving 87.3% ingredient detection rate and generating contextually appropriate recipes. The vision-language approach enables zero-shot recognition of diverse ingredients without task-specific training.

The Vocal RAG system enables natural voice-based interaction with uploaded documents through a pipeline combining Whisper ASR, BM25 retrieval, and GPT-4o generation, achieving 83% answer correctness on evaluated queries.

The containerized architecture using Docker and Docker Compose provides modularity, scalability, and ease of deployment. The seven-service system can be deployed with a single command, enabling rapid iteration and experimentation.

This work contributes to the growing field of multimodal AI systems and provides a practical, replicable framework for building integrated intelligent assistants. The open architecture enables extension to additional modalities and domains as requirements evolve.

References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[2] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[3] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[4] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In *International Conference on Machine Learning*, pages 28492–28518. PMLR, 2023.

[5] Rhasspy. Piper: A fast, local neural text to speech system. https://github.com/rhasspy/piper, 2023.

[6] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474, 2020.

[7] Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.

[8] OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[9] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[10] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*, pages 3982–3992, 2019.