

Titre: Food Acquisition Planning with Vision-Language Models for Robot-Assisted Feeding
Title:

Auteur: Félix Auger
Author:

Date: 2026

Type: Mémoire ou thèse / Dissertation or Thesis

Référence: Auger, F. (2026). Food Acquisition Planning with Vision-Language Models for Robot-Assisted Feeding [Mémoire de maîtrise, Polytechnique Montréal].
Citation: PolyPublie. <https://publications.polymtl.ca/75903/>

 **Document en libre accès dans PolyPublie**
Open Access document in PolyPublie

URL de PolyPublie: <https://publications.polymtl.ca/75903/>
PolyPublie URL:

Directeurs de recherche: Sofiane Achiche, & Jérôme Le Ny
Advisors:

Programme: Génie mécanique
Program:

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

**Food Acquisition Planning with Vision-Language Models
for Robot-Assisted Feeding**

FÉLIX AUGER

Département de génie mécanique

Mémoire présenté en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
Génie mécanique

Janvier 2026

POLYTECHNIQUE MONTRÉAL

affiliée à l'Université de Montréal

Ce mémoire intitulé :

**Food Acquisition Planning with Vision-Language Models
for Robot-Assisted Feeding**

présenté par **Félix AUGER**

en vue de l'obtention du diplôme de *Maîtrise ès sciences appliquées*
a été dûment accepté par le jury d'examen constitué de :

Jana PAVLASEK, présidente

Sofiane ACHICHE, membre et directeur de recherche

Jérôme LE NY, membre et codirecteur de recherche

Jean-François GAMACHE, membre

DEDICATION

*To myself—
because at the end of the day,
I'm the one who made this happen.*

“I know that I know nothing.”

— *Socrates*

“Research is what I’m doing when I don’t know what I’m doing.”

— *Wernher von Braun*

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my supervisor, Professor Sofiane Achiche, for his support and guidance throughout this project. I appreciate your help in navigating scholarship applications and for supporting my desire to pursue a research internship abroad. Most importantly, I am grateful for the genuine care and attention you showed me as a student. When I was exploring my options for a master's supervisor a little more than two years ago, I met you in your office and I remember you sensing that I was stressed. You suggested we set aside the decision about choosing a supervisor and instead focus on my well-being. I was indeed going through a stressful moment, and the fact that you noticed showed me how compassionate you are. I'm very glad I had you as my supervisor.

Thank you Professor Jérôme Le Ny for accepting to co-supervise my project. I appreciate your focus and attention to detail during our meetings. Your contributions elevated my project to a new level of rigor.

Thank you Professor Jana Pavlasek for accepting to preside over my jury, and Jean-François Gamache for accepting to be a jury member.

I'd like to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Fonds de Recherche du Québec – Nature et Technologies (FRQNT) for their graduate research scholarships.

Thank you to my friends and family for all your encouragement. A special thanks to Frédérick, Rémi and Yara, my labmates and now close friends, who were always there to listen and support me during my many emotional meltdowns at lunchtime.

Finally, and most importantly, I would like to express my deepest gratitude to my Mom, who is my number one fan and who has supported me selflessly all my life. My interest in pursuing graduate studies stems from our bedtime conversations when I was very young, when you would let me lie in your bed and we would dream about the future. I remember you explaining what a bachelor's degree was, then a master's, then a doctorate, and how each step forward would open up more opportunities for me. You made me aim for the stars and you believed in me before I ever believed in myself. I love you to the moon and back ♥.

I would like to acknowledge the use of generative AI to support the refinement and editing of the written sections of this thesis. Its use was limited to language assistance and did not extend to the generation or analysis of research content. All ideas, results, and conclusions presented in this thesis are my own.

RÉSUMÉ

Ce mémoire étudie l'utilisation de modèles vision-langage (VLMs) pour la planification de tâches en temps réel dans le contexte de l'alimentation assistée par robot, un domaine qui nécessite un couplage étroit entre la perception visuelle, le raisonnement sémantique et l'action physique. Les systèmes actuels d'alimentation assistée par robot séparent souvent le traitement du langage et de la vision en pipelines modulaires, limitant leur capacité à effectuer un raisonnement contextuel ancré dans la scène visuelle. L'objectif de cette recherche est d'évaluer si les VLMs modernes peuvent surmonter cette limitation en permettant une prise de décision multimodale unifiée pour des tâches de manipulation à court horizon.

À cette fin, le travail développe un système complet d'alimentation assistée par robot intégrant un planificateur de tâches basé sur un VLM, un pipeline de perception basé sur SAM 3, une interface de préférences utilisateur pilotée par la voix, et deux primitives de manipulation paramétrées (embrocher et racler) exécutées sur un bras robotique à 7 degrés de liberté équipé d'une fourchette de table standard.

Le système est évalué à travers deux expériences. Premièrement, le planificateur VLM est testé en simulation sur 25 scénarios de repas utilisant un espace d'actions étendu comprenant sept primitives d'acquisition et une consigne ancrée physiquement qui spécifie les contraintes d'exécution de chaque action. Sur 250 essais, le planificateur atteint une précision de 83,2%, suivant de manière fiable les instructions sémantiques et les préférences de l'utilisateur. Cependant, l'analyse révèle de multiples échecs dans le raisonnement physique, particulièrement dans les scénarios nécessitant des séquences d'actions en plusieurs étapes, une compréhension spatiale ou une évaluation des potentialités d'action physiques.

La deuxième expérience déploie le système complet sur un robot réel et l'évalue sur trois repas faits maison comprenant 124 tentatives de bouchées. Le système atteint un taux de succès de bout en bout de 73% avec une latence moyenne par bouchée de 20,6 secondes. La plupart des erreurs ne proviennent pas de la planification sémantique ou de la perception, mais des échecs d'acquisition dus à l'exécution en boucle ouverte de trajectoires paramétrées, qui ne permettent pas d'ajustement en temps réel.

Dans l'ensemble, les résultats montrent que bien que les VLMs excellent dans l'interprétation sémantique et la prise de décision prenant en compte des préférences, ils demeurent fondamentalement limités dans leur capacité à raisonner de manière ancrée physiquement. Ces limitations restreignent leur fiabilité pour la planification de tâches robotiques dans des environnements non structurés et soulignent la nécessité de futurs systèmes combinant perception

multimodale, rétroaction continue et raisonnement physique conditionné par l'action. Malgré ces contraintes, les résultats démontrent que les architectures pilotées par VLM offrent une base prometteuse pour des systèmes d'alimentation assistée par robot adaptatifs et centrés sur l'utilisateur.

ABSTRACT

This thesis investigates the use of vision-language models (VLMs) for real-time task planning in robot-assisted feeding, a domain that requires tight coupling between visual perception, semantic reasoning, and physical action. Current robot-assisted feeding systems often separate language and vision processing into modular pipelines, limiting their ability to perform context-aware reasoning grounded in the visual scene. The objective of this research is to evaluate whether modern VLMs can overcome this limitation by enabling unified, multimodal decision-making for short-horizon manipulation tasks.

To this end, the work develops a complete robot-assisted feeding system integrating a VLM-based task planner, a perception pipeline based on SAM 3, a speech-driven user preference interface, and two parameterized manipulation primitives (skewer and scoop) executed on a 7-DOF robotic arm holding a standard dining fork.

The system is evaluated through two experiments. First, the VLM planner is tested in simulation across 25 meal scenarios using an expanded action space of seven acquisition primitives and a physically grounded prompt that specifies the execution constraints of each action. Across 250 trials, the planner achieves an accuracy of 83.2%, reliably following semantic instructions and user preferences. However, the analysis reveals multiple failures in physical reasoning, particularly in scenarios requiring multi-step action sequences, spatial understanding, or assessment of physical affordances.

The second experiment deploys the full system on a real robot and evaluates it on three home-cooked meals comprising 124 bite attempts. The system achieves an end-to-end success rate of 73% with an average per-bite latency of 20.6 seconds. Most errors arise not from semantic planning or perception, but from acquisition failures resulting from the open-loop execution of parameterized trajectories, which do not support real-time adjustments.

Overall, the results show that while VLMs excel at semantic interpretation and preference-aware decision-making, they remain fundamentally limited in their capacity for physically grounded reasoning. These limitations restrict their reliability for robotic task planning in unstructured environments and highlight the need for future systems that combine multimodal perception, continuous feedback, and action-conditioned physical reasoning. Despite these constraints, the results demonstrate that VLM-driven architectures offer a promising foundation for adaptive and user-centered robot-assisted feeding systems.

TABLE OF CONTENTS

DEDICATION	iii
ACKNOWLEDGEMENTS	iv
RÉSUMÉ	v
ABSTRACT	vii
LIST OF TABLES	x
LIST OF FIGURES	xi
LIST OF SYMBOLS AND ACRONYMS	xiii
LIST OF APPENDICES	xiv
CHAPTER 1 INTRODUCTION	1
CHAPTER 2 LITERATURE REVIEW	4
2.1 User Interfaces and Command Modes	4
2.2 Task Planning for Feeding Assistance	6
2.3 Computer Vision for Food Perception	10
2.4 Robotic Manipulation for Food Acquisition	11
2.5 Human-Centered Design and Other Design Considerations	12
CHAPTER 3 RESEARCH MOTIVATION, OBJECTIVES, AND SCOPE	14
3.1 Research Positioning and Motivation	14
3.2 Project Objectives	14
3.3 Scope and Exclusions	15
CHAPTER 4 SYSTEM DESIGN	16
4.1 System Architecture and Infrastructure	16
4.1.1 System Architecture	16
4.1.2 Hardware Infrastructure	17
4.1.3 Software Infrastructure and Workflow	18
4.2 Robotic Manipulation System	22
4.2.1 Coordinate Frames and Transformations	22
4.2.2 Workspace Configuration and Collision Avoidance	25

4.2.3	Skewer Primitive	25
4.2.4	Scoop Primitive	26
4.2.5	Execution Flow and Motion Constraints	27
4.3	Food Detection, Segmentation & Action Parameterization	28
4.3.1	Food Detection and Segmentation Pipeline	28
4.3.2	Primitive Parameter Extraction	28
4.4	Natural Language Interaction	32
4.4.1	User Input Types and Encoding	32
4.4.2	Speech Recognition Pipeline and Integration	33
4.5	Task Planning Architecture	34
4.5.1	Planning Framework Design and Approach Selection	34
4.5.2	VLM-Based Meal Planning Architecture	34
4.5.3	Prompt Design	36
4.5.4	Expanded Action Space of Primitives for Simulation Experiment . .	38
CHAPTER 5	SYSTEM VALIDATION	40
5.1	VLM-Based Task Planning Evaluation	40
5.1.1	Dataset and Ground Truth	40
5.1.2	Experiment Results	41
5.1.3	Failure Mode Analysis	43
5.2	Full-System Evaluation on Realistic Meal Scenarios	46
5.2.1	Meal 1: Penne with Sausage and Red Peppers	47
5.2.2	Meal 2: Pork with Mashed Potatoes and Green Beans	49
5.2.3	Meal 3: Chicken with Rice, Broccoli and Carrots	51
5.2.4	Global Performance Overview	53
CHAPTER 6	CONCLUSION	57
6.1	Summary of Works	57
6.2	Limitations	58
6.3	Future Research	59
REFERENCES		60
APPENDICES		67

LIST OF TABLES

Table 5.1	VLM-based planner performance across 25 scenarios with 10 runs each	42
Table 5.2	Performance metrics per API call for the VLM-based planner ($n = 250$)	43
Table 5.3	Categorization of 42 VLM planner errors by failure type	44
Table 5.4	Meal 1 performance summary	48
Table 5.5	Meal 2 performance summary	51
Table 5.6	Meal 3 performance summary	53
Table 5.7	Stage-wise success rates across the feeding pipeline	54
Table 5.8	Average bite count and duration per meal session	55

LIST OF FIGURES

Figure 1.1	The Obi robotic feeding system [1]	1
Figure 1.2	Conceptual illustration of a robot-assisted feeding setup. This image was generated using Google’s Gemini image generation tools.	2
Figure 4.1	Overall system architecture, illustrating the key modules along with their corresponding inputs and outputs	17
Figure 4.2	Hardware configuration of the experimental setup	18
Figure 4.3	System architecture illustrating the distributed setup linking the workstation PC, camera, robot controller and GPU server PC	19
Figure 4.4	System workflow. The left panel shows the service mode that runs once at meal start, and the right panel shows the service mode that runs for every bite of the meal	22
Figure 4.5	Reference frames in the robotic system: robot base frame (origin), end-effector frame at the wrist, camera frame mounted on the end-effector, and fork frame at the middle of the fork tip	23
Figure 4.6	RViz setup used for workspace modeling and trajectory validation . .	25
Figure 4.7	Skewer trajectory shown as a 2D projection. The black arrows indicate the orientation of the fork tines along the trajectory.	26
Figure 4.8	Scoop trajectory shown as a 2D projection. The black arrows indicate the orientation of the fork tines along the trajectory.	27
Figure 4.9	Example of detection and segmentation for the query "penne" on a real plate using SAM3	29
Figure 4.10	Skewer primitive parameterization. The red point indicates the geometric centroid, and the yellow line represents the secondary principal axis (minor axis), which is used to define the fork’s roll orientation. .	30
Figure 4.11	Density-based target selection pipeline for the scoop primitive. Starting from (a) an RGB input and corresponding depth map, the system applies a Gaussian blur to the segmentation mask to create (b) a spatial centrality map that assigns higher values to points near the region’s interior. This is then combined with (c) a depth-based weighting map that prioritizes elevated regions. The resulting (d) density heatmap highlights the most suitable scoop target as the peak of the combined spatial and depth maps.	31

Figure 4.12	Scoop primitive parameterization. The red dot marks the target point selected from the density heatmap, the pink dot marks the point within the same segmentation mask that is furthest from it, and the blue arrow indicates the yaw orientation that defines the approach direction for scooping.	32
Figure 5.1	Initial plate configuration for Meal 1: penne, sausage, and red pepper	47
Figure 5.2	Keyframes from a penne skewering sequence during Meal 1	48
Figure 5.3	Initial plate configuration for Meal 2: pork, mashed potatoes, and green beans	49
Figure 5.4	Keyframes from a mashed potato scooping sequence during Meal 2 .	50
Figure 5.5	Initial plate configuration for Meal 3: chicken, rice, broccoli and carrots	52
Figure 5.6	Sankey diagram showing the flow of 124 bite attempts through the three main execution stages and failure points categorized by error type	54
Figure 5.7	Average latency breakdown of the full feeding cycle across 112 complete executions. Each segment represents the mean time spent per operation, including planning, perception, execution, and fixed-duration steps.	56

LIST OF SYMBOLS AND ACRONYMS

ARA	Assistive Robotic Arm
API	Application Programming Interface
ASR	Automatic Speech Recognition
BCI	Brain-Computer Interface
CAD	Computer-Aided Design
DOF	Degree of Freedom
EEG	Electroencephalography
FCI	Franka Control Interface
FSM	Finite State Machine
GUI	Graphical User Interface
GPT	Generative Pre-trained Model
GPU	Graphics Processing Unit
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
JSON	JavaScript Object Notation
LLM	Large Language Model
PC	Personal Computer
PDDL	Planning Domain Definition Language
RAF	Robot Assisted Feeding
REST	Representational State Transfer
ROS	Robot Operating System
RGB-D	Red-Green-Blue + Depth
SAM	Segment Anything Model
SD	Standard Deviation
SO	Specific Objective
TAMP	Task And Motion Planning
TCP	Transmission Control Protocol
USB	Universal Serial Bus
VLM	Vision-Language Model
YAML	YAML Ain't Markup Language

LIST OF APPENDICES

Appendix A	VLM Planning Prompt Used in Simulation-Based Evaluation	67
Appendix B	Meal Scenario Dataset for Simulation-Based Planning Evaluation . .	70

CHAPTER 1 INTRODUCTION

Individuals with severe upper-limb motor impairments, resulting from conditions such as cerebral palsy, amyotrophic lateral sclerosis, or spinal cord injury, often face significant challenges in performing activities of daily living, including self-feeding [2]. The inability to feed oneself compromises nutritional autonomy and significantly impacts personal dignity and social well-being. Furthermore, relying on caregivers for manual feeding can foster feelings of embarrassment or dependence [3]. While human caregivers provide essential support, this assistance is often time-consuming, labor-intensive and difficult to scale [4], particularly in the context of an aging population.

To address this challenge, physically assistive robots have gained attention as a way to support user autonomy [2]. In particular, robot-assisted feeding systems aim not only to support the physical act of feeding, but also to restore a sense of independence and improve quality of life for users with severe motor impairments, including enabling greater participation in social eating contexts and reducing the stigma often associated with assisted feeding [5].

Robot-assisted feeding involves technologies that enable robots to acquire and deliver food items to individuals with upper limb impairments. These systems may operate autonomously or semi-autonomously (i.e., shared control with the user). Among the few commercially available solutions, the Obi feeding robot (Figure 1.1) is among the most widely used assistive devices for self-feeding.



Figure 1.1 The Obi robotic feeding system [1]

Obi is an FDA-registered medical device marketed in North America and Europe. It performs pre-programmed scoop-and-deliver motions, offering four food compartments and a customizable accessibility interface with switch inputs [1]. The robot is known for its ease of use, compactness, and user-friendly design, making it a common choice in clinical and domestic settings. However, Obi operates using fixed trajectories and lacks adaptive perception or control, limiting its ability to handle unstructured or personalized scenarios. It can only perform basic scooping motions, making it unsuitable for varied food types or variable plate configurations.

To overcome such limitations, recent research has focused on developing more flexible, intelligent, and user-responsive approaches to robot-assisted feeding, typically deployed on assistive robotic arms (ARAs) mounted on motorized wheelchairs, as illustrated in Figure 1.2.



Figure 1.2 Conceptual illustration of a robot-assisted feeding setup. This image was generated using Google’s Gemini image generation tools.

Despite promising advances, developing effective robot-assisted feeding systems remains a complex and multifaceted challenge. These systems must operate reliably in dynamic, real-world environments where food items can vary widely in shape, texture, and arrangement. Accurate perception is required to detect and segment food items on a cluttered plate. Based on this input, the system must engage in high-level planning to determine suitable actions for acquiring and delivering each bite. The physical manipulation of food presents further

difficulties, as it involves handling soft or deformable items. Finally, successful interaction with the user, ranging from real-time feedback to preference adaptation, is essential to ensure safety, comfort, and a personalized experience. These interdependent challenges must be addressed to achieve autonomous feeding systems that are both practical and socially acceptable.

The remainder of this thesis is structured as follows. Chapter 2 reviews the literature on robot-assisted feeding, highlighting current challenges in perception, task planning, and manipulation. Chapter 3 defines the specific scope and research objectives of this project. Chapter 4 details the design of the system, describing the system architecture, the integration of Vision-Language Models (VLMs) for autonomous food acquisition planning, and the implementation of the food perception and manipulation modules. Chapter 5 presents the experimental results, including an analysis of VLM-based food acquisition planning performance and a validation of the system on a physical robot. Finally, Chapter 6 summarizes the main contributions, discusses the system’s limitations, and proposes directions for future research.

CHAPTER 2 LITERATURE REVIEW

This literature review provides a comprehensive overview of robot-assisted feeding technologies, with a particular focus on the components of a complete perception–planning–action pipeline. The review is structured to reflect the logical execution sequence of a robot-assisted feeding system: from user input and intent expression, to high-level decision-making, to visual perception and physical execution. Concluding with human-centered considerations, this chapter contextualizes recent advances relevant to the proposed research and identifies opportunities to improve the adaptability, robustness, and inclusivity of robot-assisted feeding systems.

2.1 User Interfaces and Command Modes

Assistive robotic arms are most commonly controlled using joysticks. However, this can be challenging and unintuitive, particularly for users with motor impairments, due to the high number of degrees of freedom (DOF) involved. These robots often require simultaneous coordination of multiple joints for reaching, orienting, and manipulating objects in 3D space, making precise control both physically and cognitively demanding [6]. To address these challenges, a variety of alternative input modalities have been explored, including touch, vision, body motion, voice and neural signals.

Touch-Based Interfaces

Discrete physical interfaces such as physical buttons are widely used. The Obi robot, for instance, allows users to operate pre-programmed feeding motions using simple push buttons [1]. To make continuous manual control of high-DOF arms easier, one approach is to learn a low-dimensional “latent action space” from demonstrations. User joystick inputs are then mapped to these latent variables, which the robot decodes into complex, coordinated actions like scooping or pouring, greatly simplifying control for the user [6]. This greatly reduces the interface complexity while allowing the user to keep full continuous control over the robot’s movements. Additionally, touchscreen-based graphical user interfaces (GUIs) have been employed in multiple systems to enable direct selection of food items or initiation of feeding actions [7–10].

Vision-Based Interfaces

Eye gaze tracking provides a hands-free method for users to communicate their intent by directing their gaze toward a food item and has been demonstrated as a viable input modality

in robot-assisted feeding systems [11, 12]. Infrared cameras or webcam-based tracking are typically used to estimate gaze direction and map it onto regions of the physical plate or a corresponding GUI. By integrating this with real-time object detection, the system can infer which food item the user is looking at and trigger the corresponding acquisition action.

Facial and mouth region analysis has also been used to detect user readiness for bite delivery [11], while enabling more discrete and socially appropriate interaction during feeding [5].

Body Motion-Based Interfaces

Interfaces based on body motion, such as body-machine interfaces (BMI), enable shared autonomy by interpreting movements like head tilt or upper-body gestures to guide robot actions [13]. Furthermore, hybrid systems that integrate head and eye movement expand the expressiveness of motion-based control and increase accessibility for users with limited fine motor control [14].

Voice Interfaces

Speech-based interaction offers a natural and expressive modality for robot control, especially for users with limited physical mobility. VoicePilot exemplifies this approach by leveraging large language models to process vocal commands in real-time [15]. This modality is particularly useful for tasks such as specifying the desired food item or issuing high-level task instructions.

Neural Signal-Based Interfaces

Direct neural interfaces aim to decode user intent by capturing brain activity patterns. For example, non-invasive brain-computer interfaces (BCI) based on electroencephalography (EEG) have been proposed as control methods for users with severe impairments [16]. This modality converts neural signals into robot commands, allowing users to select feeding actions or food items from a screen interface.

Multimodal Integration

Across all interface types, key trade-offs must be considered, including simplicity versus expressiveness, cognitive and physical effort, and adaptability to user capabilities. Adaptive input strategies that enable personalization have been shown to enhance both user comfort and acceptance [17]. For example, FEAST combines GUI, physical buttons, and head gesture inputs into a unified interface [10]. Such multimodal systems improve accessibility by allowing users to select their preferred input method or combine modalities depending on their abilities and context. These considerations are critical for the design of inclusive, user-centered robot-assisted feeding systems [7].

Once user intent is captured through buttons, gaze, voice, or other modalities, the system must interpret that intent and translate it into an actionable plan. This requires a high-level task planner capable of choosing the next appropriate action based on the current plate state, prior interactions, and user preferences.

2.2 Task Planning for Feeding Assistance

The focus of this section is on high-level task planning approaches that leverage language and VLMs for reasoning and decision-making. Classical symbolic planners and low-level motion or control strategies are beyond the scope of this review.

Task planning in robot-assisted feeding involves determining which actions to execute to achieve the goal of feeding an entire plate. Unlike static environments, the feeding context is inherently dynamic: after each bite the plate configuration changes, which requires the system to replan accordingly. Planning scenarios for feeding are typically short-horizon (e.g., selecting the next item to acquire) but may also require long-horizon, multi-step action sequences, such as when an obstructing food item must be moved out of the way before acquiring the target item [18]. These unique characteristics of feeding assistance present distinct challenges that traditional planning approaches struggle to address effectively.

Evolution from Symbolic to Language-Based Planning

Traditional task planning approaches have long relied on frameworks based on symbolic reasoning and formal logic. However, these symbolic planners often depend on discrete state representations and static domain definitions that must be specified in advance. Hence, they often struggle to adapt in dynamic contexts like feeding, where real-time environmental changes and human preferences cannot be fully captured in pre-defined models. To address this, Dai et al. [19] propose a hierarchical control framework integrating a Finite State Machine (FSM) with a shared-autonomy system to manage adaptive feeding sequences and handle unexpected events.

More recently, Large Language Models (LLMs) have increasingly been used as natural language interfaces for robotic planning [20, 21]. These models help both with interpreting user intent and generating high-level symbolic plans. For example, they can translate natural language commands into formal planning representations such as Temporal Logic (TL) or Planning Domain Definition Language (PDDL), allowing users to issue commands without formal training.

AutoTAMP [22] employs LLMs to translate natural language instructions into formal task specifications in TL format. An external planner then generates the corresponding motion

plan. The system includes both syntactic correction—where TL errors are flagged and the LLM is re-prompted with structured feedback—and semantic correction, where an LLM-based checker evaluates whether the generated plan aligns with the original user intent, issuing corrective prompts if misalignment is detected. To ensure plans align with user intent and safety constraints, some approaches combine LLM-based planning with formal verification. For instance, Quartey et al. [23] propose a framework that validates generated plans against logical specifications using Satisfiability Modulo Theories (SMT) solvers, improving reliability in instruction-following tasks.

Nevertheless, text-only LLMs remain limited in their grounding and contextual awareness. In LLM-Planner [24], the LLM generates symbolic action sequences based on few-shot prompts and textual world states. The system relies entirely on structured text inputs and produces formal plans, with no vision-language coupling or grounding in real sensor data. This approach demonstrates significant limitations: it relies heavily on hand-crafted prompts and examples, symbolic plans are not grounded in robot embodiment or physical feasibility, and it has no visual input capability, only textual/environmental state descriptions. To address execution uncertainty, LLM3 [21] introduces a feedback loop where motion feasibility is checked post-plan generation. If an action fails, the failure is described textually and reintroduced into the LLM prompt, enabling dynamic replanning.

VLM-based Planning

More recent approaches integrate visual and linguistic modalities to overcome the limitations of text-only systems. Pre-trained VLMs such as GPT-4V [25] have garnered attention for their strong scene understanding, multimodal reasoning capabilities, and efficiency in parsing complex inputs for code and action generation. VLMs can function as multimodal task planners by translating visual-linguistic prompts into coherent action sequences and supporting zero-shot generalization to previously unseen environments. They are capable of semantic scene parsing and affordance reasoning—identifying objects, their interactions, and spatial relations—which provides high-level contextual awareness for feeding tasks. These visual inputs can then guide LLMs in generating symbolic plans, which may be self-evaluated against visual ground truth using scoring or feedback mechanisms for refinement.

Yang et al. [26] demonstrate how VLMs can be effectively integrated into task and motion planning (TAMP) pipelines by combining the semantic understanding and common-sense reasoning of VLMs with the physical feasibility and logical coherence offered by symbolic planners. In their approach, the VLM is used to generate subgoals (not full action plans), and TAMP (task and motion planner) solves them geometrically, outperforming VLM-alone methods on long-horizon kitchen tasks requiring 30–50 steps.

In the context of robot-assisted feeding, FLAIR [18] leverages a combined VLM-LLM framework to enable long-horizon task planning. The system employs a variety of prompt engineering strategies, including role prompting (e.g., instructing the model to act as a robot-assisted feeding task planner), chain-of-thought prompting (guiding step-by-step reasoning), and few-shot prompting (providing examples and counterexamples). The LLM is provided with the current plate configuration (e.g., food positions and remaining items), bite history, available primitive actions, and user preferences, and then asked to choose the next food item and the corresponding sequence of actions.

Recent work such as MARS [27] demonstrates the integration of VLMs for object perception and LLMs for natural language task planning in mobile assistive robots. While not focused on feeding, their modular architecture and few-shot LLM prompting pipeline provide a relevant reference for structured task planning in unstructured home environments. However, the planning process remains decoupled from visual inputs, leaving open the question of whether end-to-end VLM-based planning could improve robustness in dynamic or ambiguous contexts.

An essential challenge in deploying generative models for robotics, as noted in [28], is grounding high-level reasoning in the physical capabilities (affordances) of real robots. Advancing this goal requires training models not only on language and vision, but also on examples that reflect physical interactions—such as curated video datasets that capture object properties, contact dynamics, and the consequences of manipulation. In this context, LAVA [29] uses VLMs to guide food acquisition but emphasizes learned visual affordances, particularly for semi-solid or deformable foods. The system selects high-level manipulation primitives from visual input—such as direct or wall-guided scooping—then refines and executes precise actions to acquire food items of varying textures and positions in the bowl.

Another promising development is the emergence of vision-language-action (VLA) models such as RT-2 [30] and OpenVLA [31], which extend pre-trained VLMs by fine-tuning on robotic data to enable end-to-end task execution. These models map multimodal inputs directly to structured robot commands, such as joint positions and gripper actions, thus bridging high-level perception and semantic understanding with actionable control outputs.

Robustness Strategies: Error Handling and Multi-Agent Approaches

In this context, robustness refers to a planner’s ability to generate reliable and contextually appropriate plans despite visual ambiguity or dynamic environmental changes, and to adapt these plans when the scene state evolves during task execution.

While multimodal models offer powerful capabilities, they face key limitations affecting their robustness in feeding contexts. They require complex prompt engineering, operate over

predefined action pools rather than generating low-level motor commands, and lack real-time responsiveness and interpretability as black-box systems. Even with carefully crafted prompts [18], they remain susceptible to hallucinations (i.e., factually incorrect outputs that appear plausible), semantic ambiguity, and limited contextual reasoning over longer task sequences.

To mitigate hallucination and increase robustness, several works explore error detection and correction strategies. Dhuliawala et al. [32] propose Chain-of-Verification (COVE), where the LLM drafts a response, formulates and independently answers fact-checking questions, then refines its output. ReplanVLM [33] focuses on replanning in response to execution failures or scene changes by regenerating plans grounded in updated visual input—particularly relevant for feeding’s dynamic context. The system demonstrates robustness to mid-task disturbances through its visual-replanning framework.

Similarly, RoboDexVLM [34] implements a dual-layer recovery mechanism for manipulation tasks, where execution failures trigger VLM-based reflection prompts containing error context and skill attempt history to generate adjusted action sequences. While focused on dexterous manipulation rather than feeding, this validates the efficacy of VLM-driven error recovery in long-horizon scenarios.

Multi-agent architectures represent another approach to improving robustness. Brienza et al. [35] propose a multi-agent system with specialized VLM agents for scene understanding, grounded reasoning, and task planning, with a centralized arbitration module selecting the optimal plan.

While multi-agent architectures raise important questions about performance gains over single-agent pipelines or symbolic approaches, they introduce additional prompt engineering overhead, computational latency, and risks of failure cascades between agents. Collectively, these works suggest that error correction mechanisms and diverse reasoning approaches can enhance VLM-based task planning robustness. However, their applicability to the dynamic, real-world context of robot-assisted feeding, where replanning must occur after each bite and visual ambiguity is common, remains an open research question.

To execute a planned action in the physical world, the robot must first perceive and understand its environment. Bridging the gap between symbolic plans and real-world execution requires a robust perception system capable of identifying and localizing food items with sufficient accuracy to support manipulation.

2.3 Computer Vision for Food Perception

Food identification, detection, segmentation, and localization are essential perception tasks in robot-assisted feeding. These capabilities enable the system to recognize which food items are present on the plate and extract optimal poses for their acquisition. Robust perception is critical, as inaccurate identification or localization can result in failed grasps or inappropriate food manipulation [36].

Several deep learning-based techniques have been proposed. Mask R-CNN [37] has been used to produce instance-level segmentation masks for each food item using RGB-D input, but its training is typically limited to small, curated datasets of predefined food classes [11]. Similarly, Faster R-CNN [38] has been applied to food detection and bite targeting in feeding contexts using a collaborative robotic arm, though this approach was largely limited to simulation environments and predefined scenarios [39]. Similarly, RetinaNet [40] has been employed for food detection and bite targeting, particularly in the context of skewering manipulation [36, 41].

However, several challenges persist. One major difficulty is the complexity of plate configurations: food items often overlap, are partially occluded, or mixed together, making contour extraction and instance-level segmentation difficult. To address these issues, Fan et al. developed a real-time segmentation model optimized for feeding assistance, which improves accuracy and robustness under occlusion and clutter by using a category-level mask mechanism [42].

Zero-shot identification (recognizing food categories not present in the training data) remains a key obstacle, as models must generalize to previously unseen food items in real-world settings. Many perception models still suffer from poor generalization, as they are often trained on curated datasets with neatly separated food items and a limited number of categories, which does not reflect the visual and structural variability of real-world meals [42].

To overcome this, more recent efforts such as FLAIR have explored modular pipelines capable of handling more realistic plate configurations [18]. FLAIR integrates several complementary models: GPT-4V for high-level food identification, Grounding DINO [43] for open-vocabulary object detection, and Segment Anything (SAM) [44] to refine bounding boxes into pixel-level segmentation masks. Although the study was limited to six example meal configurations and a constrained set of food types, the modularity of the pipeline demonstrates how VLMs can improve generalization and support semantic reasoning.

As outlined in Zhang et al.’s survey [45], VLMs learn visual-linguistic associations from large image-text datasets and can perform zero-shot predictions through textual prompts. Without

requiring task-specific retraining, they can recognize food items across diverse categories, which could make them particularly well-suited for the diversity and unpredictability of real-world meals during robot-assisted feeding. Their ability to perform semantic reasoning—that is, to infer object meaning and relationships from contextual cues rather than exact visual matches—enables them to generalize to unfamiliar foods and configurations more effectively. Once the system has accurately identified and localized the target food item, it must physically interact with it to acquire and deliver the bite. This calls for manipulation strategies that are robust to variation in food shape, texture, and placement.

2.4 Robotic Manipulation for Food Acquisition

Robotic manipulation in feeding assistance includes two primary phases: food acquisition and bite transfer. The goal is to reliably acquire a targeted food item from the plate using an appropriate utensil and safely deliver it to the user’s mouth. This task is made difficult by the diversity of food textures and shapes, requiring systems to generalize across a wide range of physical properties and plate arrangements.

One common approach relies on predefined action primitives, where a set of parameterized manipulation strategies is designed to handle different food types. FLAIR [18] exemplifies this strategy with seven primitives, each tailored to a specific type of food interaction, all executable with a single fork: *skewer* for solid items like broccoli, *scoop* for soft or granular foods such as rice, *twirl* for noodles, *dip* for sauce immersion, *cut* to divide soft items, *push* to move obstructing pieces, and *group* to cluster scattered food before acquisition.

This fork-only approach has also been explored in other systems [18, 46], while additional work addresses broader manipulation contexts, including acquisition of liquid or soft foods using a spoon [8, 29, 47], wall-guided scooping using the plate’s edge as support [29], and bimanual scooping with two robot arms for more complex coordination [48, 49]. While most systems rely on a single utensil and switch primitives based on food type, some designs explore changing the utensil itself depending on the item to acquire. For example, Mashrur et al. [50] implemented a prototype that can autonomously switch between a spoon or fork placed on a stand, or use the robot’s gripper directly to pick up a beverage.

To move beyond fixed trajectories, manipulation success is significantly improved by incorporating real-time feedback. For instance, tactile and force-based sensing, such as force and torque measurements at the robot end-effector, help detect contact and regulate the applied force during manipulation [8, 51]. Other systems integrate visual and haptic feedback to adjust motion during interaction with the food, making food acquisition more robust and

reliable. By combining vision with touch sensing, the robot can detect when a food item deforms, slips, or resists penetration and adapt its trajectory in real time. This closed-loop adjustment reduces failed grasps and improves the consistency of acquisition across foods with varying textures and firmness [52–54]. Another approach uses online learning to adapt skewering strategies for previously unseen foods by framing food acquisition as a contextual bandit problem that leverages visual context and binary feedback [51].

Alternatively, learning-based approaches leverage data-driven methods for greater adaptability and generalization. Reinforcement and imitation learning pipelines have been used to replicate expert demonstrations [29, 47, 55]. Similarly, models trained to recognize structural and physical cues have demonstrated that generalization across food types can be achieved through learning-based skewering strategies [41].

The transfer phase introduces additional challenges, including the need for reliable in-mouth bite delivery. Successful transfer requires detecting user readiness, adapting timing to user motion, and maintaining safety and comfort throughout the interaction [56, 57]. Notably, acquisition quality directly affects transfer performance, as unstable skewers compromise handoff, with studies showing that user comfort and success depend heavily on the preceding acquisition action [36].

Despite these advances, many acquisition strategies are still evaluated on curated plates with idealized food placements. Broader deployment in real-world environments requires more robust handling of realistic, unstructured meals and greater generalization across food types and utensil configurations. While robust acquisition remains essential for handling realistic meals, achieving the necessary generalization also requires high-level task planning to strategically select and sequence robotic food acquisition actions.

While technical components like perception and manipulation are essential, the long-term success of a feeding system ultimately depends on user acceptance and trust. This highlights the role of human-centered design in ensuring safety, comfort, and adaptability to individual needs.

2.5 Human-Centered Design and Other Design Considerations

User acceptance and trust are essential factors in the deployment of robotic feeding systems. Studies show that users with mobility impairments have varied preferences regarding levels of robot autonomy and control, which should be carefully considered to foster trust and comfort [3]. Personalization also plays a key role in improving user experience, requiring systems to adapt to individual needs, preferences, and physical capabilities [4, 10, 58]. To

support intuitive and personalized human-robot interaction, recent research has explored voice interfaces powered by large language models to enable natural communication in assistive contexts [15]. Furthermore, inclusive design practices are critical: involving users directly in design and evaluation processes ensures that robotic systems align with real-world expectations and social dynamics [7].

Building on these principles, several technical and design requirements have been emphasized in the literature to ensure that robot-assisted feeding systems are safe, adaptable, and effective in real-world environments. *Safety* is the primary consideration for personal care robots operating in close proximity to users, as defined by ISO 13482:2014 [59]. *Compliance*—the system’s ability to adapt to the user’s posture, preferences, and physical condition—is also essential for comfort and usability. Finally, *portability* plays an important role in enabling use across different environments, such as homes, clinics, or care facilities, which is especially valuable for supporting user independence [9]. Together, these design considerations highlight the importance of aligning assistive technologies with users’ real needs and experiences.

CHAPTER 3 RESEARCH MOTIVATION, OBJECTIVES, AND SCOPE

This chapter presents the motivation behind the proposed research, outlines its specific objectives, and defines the scope and constraints that shaped the system’s development.

3.1 Research Positioning and Motivation

Recent advancements in robot-assisted feeding have demonstrated the feasibility of autonomous systems capable of perceiving, planning, and manipulating food items for individuals with upper limb impairments. These developments include specialized robotic hardware, data-driven manipulation strategies tailored to varied food textures, and high-level task planning driven by vision-language models.

Despite this progress, existing systems face key challenges that limit their applicability in practical settings. Many pipelines adopt a modular design where perception and planning are treated as independent stages, limiting opportunities for end-to-end optimization. This decoupling makes it difficult for the system to reason about visual context during action selection. A unified VLM planner offers a promising alternative by preserving spatial and semantic context throughout the planning process.

Furthermore, prior work on VLM-based systems has predominantly focused on simulation environments or constrained experimental setups. Real-world deployment in complex, unstructured plate scenarios remains underexplored, yet is essential for generalizable, user-ready feeding assistance. By validating the system in real-world settings, this work aims to enhance the contextual reasoning and responsiveness needed for deployable robot-assisted feeding.

3.2 Project Objectives

The overarching goal of this research is to develop a robot-assisted feeding system that enhances autonomy and quality of life for individuals with upper limb mobility impairments. To support this goal, the work addresses foundational technical challenges critical to practical deployment, outlined in the following objectives.

Main Research Objective:

Develop a vision-language model-based system for robot-assisted feeding in realistic meal scenarios.

Specific Objectives:

- **SO1:** Implement a VLM-based task planning architecture that generates executable food acquisition sequences.
- **SO2:** Develop feeding-specific manipulation primitives to handle the acquisition of diverse food categories and spatial arrangements.
- **SO3:** Integrate a computer vision-based action parameterization module to localize food items on the plate and extract optimal poses for robotic acquisition.
- **SO4:** Implement natural language interaction using a speech recognition model to support user food preferences and selection via voice commands.
- **SO5:** Deploy and validate the complete system on a Franka robotic arm through real-world experiments across diverse plate configurations, measuring the task execution success rate and end-to-end system latency.

3.3 Scope and Exclusions

Given the constraints of a master’s thesis, the scope of this project was deliberately limited. The following elements are excluded from the system’s design and evaluation:

- **Bite transfer:** The bite delivery phase is simplified by using a fixed end-effector pose. The system does not include mouth detection, adaptive facial tracking, or user-readiness detection. Handoff dynamics and user-robot coordination are excluded.
- **Human-in-the-loop interaction:** Only initial verbal instructions related to user food preferences or selection are supported. The system does not support mid-meal verbal corrections, real-time speech feedback, emergency-stop voice commands, or adaptive replanning based on user verbal or non-verbal cues.
- **Generalization across utensils and hardware platforms:** The system is implemented using a single utensil (fork) and a single robotic platform. Spoon-based acquisition and cross-platform deployment are not explored.
- **User-centered evaluation:** The system has not been tested with target users, caregivers, or in real-world feeding contexts. All evaluations are conducted in controlled laboratory settings with the experimenter acting as the user.

CHAPTER 4 SYSTEM DESIGN

This chapter presents the design and implementation of the robot-assisted feeding system. The work follows an implementation-driven experimental approach characterized by iterative development, a modular system architecture, and an empirical evaluation. The system is described through five main design blocks that are directly aligned with the specific objectives defined in Section 3.2:

- The hardware and software infrastructure supporting system deployment (deployment phase of SO5).
- The natural language interaction module used to capture user preferences (SO4).
- The VLM-based task planning architecture responsible for bite selection (SO1).
- The vision-based action parameterization module (SO3).
- The implementation of the feeding-specific manipulation primitives (SO2).

This ordering improves narrative clarity by introducing low-level execution modules before addressing perception and high-level reasoning components. The chapter concludes with the system’s scope and exclusions, followed by the experimental validation protocols that assess overall system performance (validation phase of SO5).

4.1 System Architecture and Infrastructure

This section addresses the deployment phase of SO5 and establishes the foundational platform on which all modules (SO1–SO4) are implemented and tested. The overall system architecture is presented, followed by the hardware and software infrastructure.

4.1.1 System Architecture

Figure 4.1 illustrates the high-level system architecture, highlighting the modular organization of the main components and their data flow. Multimodal inputs (visual and audio) are processed through a sequence of perception, task planning, and robotic execution modules, enabling VLM-driven food acquisition.

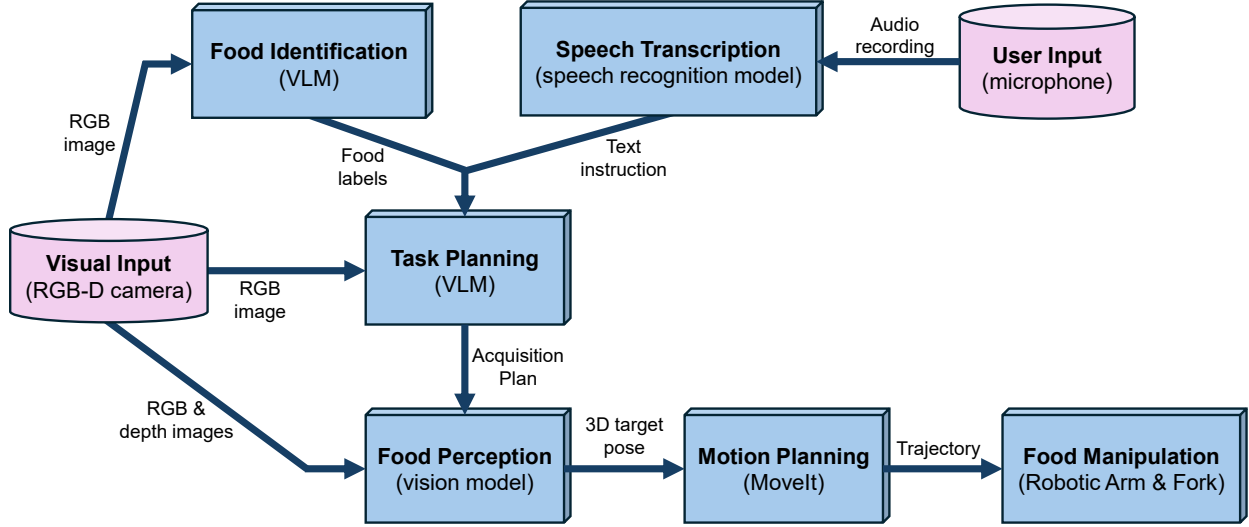
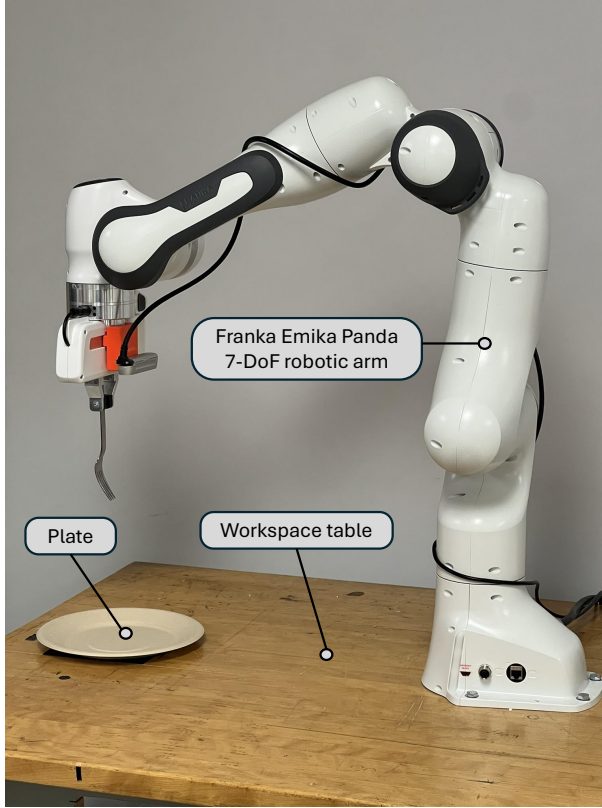


Figure 4.1 Overall system architecture, illustrating the key modules along with their corresponding inputs and outputs

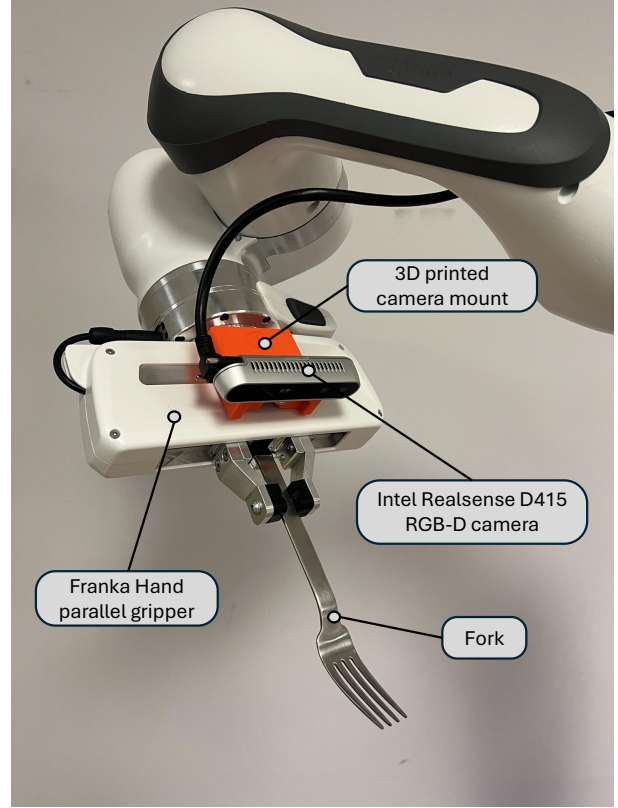
4.1.2 Hardware Infrastructure

Figure 4.2 presents the physical hardware configuration used in the experimental setup. Figure 4.2a shows the overall workspace, while Figure 4.2b details the end-effector assembly. The specific components include:

- **Robotic Manipulator:** Franka Emika Panda, a 7-DOF collaborative robot arm with an 855 mm reach and 3 kg payload capacity. The arm weighs approximately 20 kg and is securely mounted to a stationary table. This robot was chosen due to its availability in the laboratory and widespread adoption in manipulation research, making it easier to compare results with related work. While real-world deployment scenarios would favor a more portable solution such as the Kinova Gen 3 ARA mounted on a motorized wheelchair, the Franka Emika Panda provides a robust research platform with precise control capabilities needed to validate the proposed feeding system.
- **End-Effector:** Franka Hand parallel gripper, used exclusively to grasp a standard dining fork, the only utensil used for food acquisition in this study.
- **Vision System:** Intel RealSense D415 RGB-D camera, mounted on the robot's end-effector via a 3D-printed camera mount. The camera operates at a reduced resolution of 640×480 pixels, which provides sufficient image quality for food perception and localization while reducing processing time.



(a) Full robotic manipulation workspace, including the Franka Emika Panda arm, the table, and a plate



(b) End-effector assembly featuring the camera, the 3D-printed camera mount and the fork gripped by the Franka Hand

Figure 4.2 Hardware configuration of the experimental setup

- **Workspace:** A table serving as both the robot mounting surface and the meal presentation area, with plates positioned within the robot's reach during feeding operations.
- **Computing Infrastructure:** The system includes two distinct computing units: a workstation PC and a GPU server PC (not shown in Figure 4.2).

4.1.3 Software Infrastructure and Workflow

Figure 4.3 shows the system's architecture, which includes two primary computing units connected via Ethernet (TCP/IP): a Workstation PC handling hardware interfacing and execution, and a GPU Server PC. This separation was introduced because workstation PC overloads were found to cause packet loss, which in turn triggered emergency stops through communication errors. Offloading processing-intensive tasks to an external server prevents computational spikes from interfering with real-time communication, ensuring low-latency and stable control of the robot.

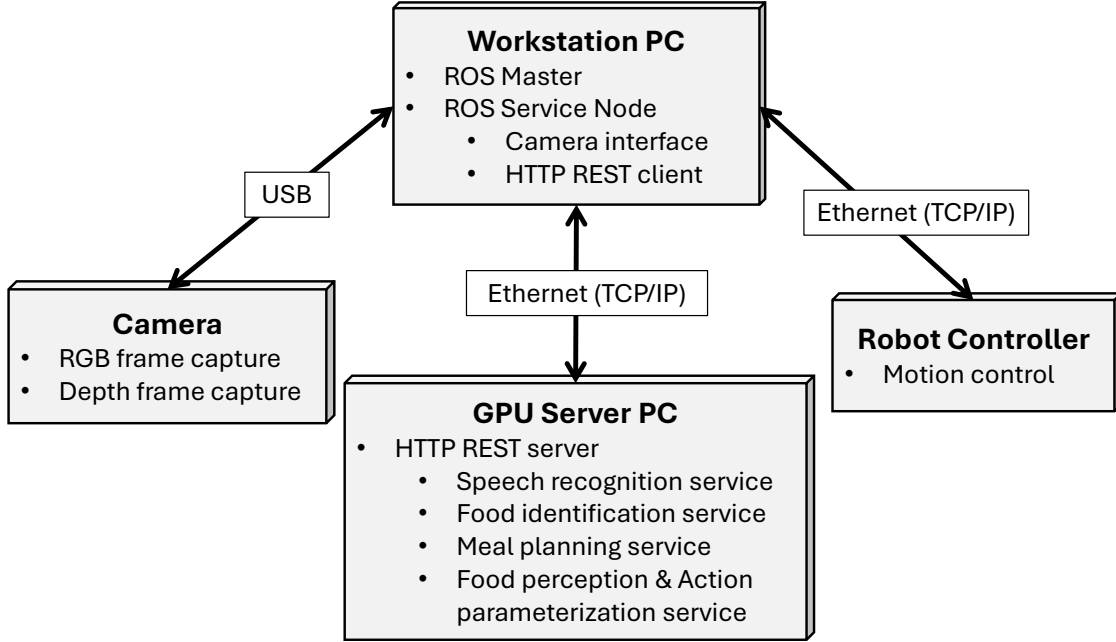


Figure 4.3 System architecture illustrating the distributed setup linking the workstation PC, camera, robot controller and GPU server PC

The specific responsibilities of each of the system’s computing units are defined as follows:

Workstation PC: The workstation PC runs Ubuntu 20.04 LTS (Focal Fossa) and ROS Noetic, and serves as the system coordinator. Two primary software components operate on this machine:

- **ROS Master Node:** Responsible for real-time robot control and trajectory execution. This node interfaces with the Franka Emika Panda controller via Ethernet (TCP/IP) through the Franka Control Interface (FCI), which provides high-rate, closed-loop control. The system uses the `franka_ros` package to bridge FCI into the ROS ecosystem, and MoveIt for motion planning.
- **ROS Service Node:** Manages the RGB-D camera interface and image transfer via USB and maintains a local YAML-based Meal Log file that tracks the food items present on the plate, the user’s input, and the complete bite history for the current meal session. The following example illustrates the Meal Log as it might appear at a given point during a meal session.

Example Meal Log during Meal Session

Bite history:

- skewer(chicken)
- scoop(rice)
- skewer(carrot)
- skewer(chicken)
- scoop(rice)
- skewer(carrot)
- skewer(chicken)
- scoop(rice)

Food items identified at meal start:

- chicken
- broccoli
- carrot
- rice

User input: "I don't like broccolis"

This node also serves as the HTTP REST client, handling all HTTP POST requests sent from the workstation to the GPU server.

GPU Server PC: The GPU server runs a HTTP REST API server that provides three services accessible via HTTP POST requests using JSON payloads and base64-encoded image data:

- **Speech Recognition Service:** Transcribes user vocal input from audio data captured at meal initialization (detailed in Section 4.4).
- **Food Identification Service:** VLM-based module for identifying food items from RGB images. During meal initialization, this service receives an image of the plate and uses the following structured prompt to extract all visible food items:

VLM Food Identification Prompt

Identify all food items visible on the plate. Make sure to:

- Use simple, generic names
- Use singular form only
- Output in this format: [item1, item2, item3, ...]

The resulting list of food labels is stored in the Meal Log and remains fixed throughout the entire meal session. This ensures label consistency across all subsequent VLM planning calls.

- **Meal Planning Service:** VLM-based task planning that determines the next action and food item selection (detailed in Section 4.5).
- **Food Detection, Segmentation & Action Parameterization Service:** Computer vision module that performs food detection and segmentation, then extracts 3D target coordinates for the manipulation primitives (detailed in Section 4.3).

System Workflow: Figure 4.4 illustrates the end-to-end execution pipeline of the robot-assisted feeding system, which is divided into two execution modes of the same ROS service:

- **Meal Initialization Mode:** This sequence executes once at the beginning of the meal. It initializes the Meal Log, captures user vocal input via speech recognition and performs initial food identification, after which both the user preferences and the detected food items are recorded in the Meal Log.
- **Bite Execution Mode:** This loop is executed for each bite. The service captures the current plate image, requests next-bite planning, writes the decision in Meal Log and parameterizes the action based on perception. The system then pauses and waits for user confirmation (in this case via keyboard spacebar press) before triggering the robot execution, allowing the user to verify the planned action and ensure readiness for the next bite.

The system adopts a modular, service-isolated design, enabling individual component testing and integration-level validation across the full pipeline. Identifying all food items at meal start establishes a fixed list of food labels that remains constant throughout the meal, preventing the VLM planner from renaming items across bites. Before entering the feeding loop, the user can verify on the workstation PC that the Meal Log was correctly configured with the food items properly identified and the user input accurately transcribed. If needed, the Meal Log can be manually edited before execution begins.

The system also implements an automatic error recovery procedure: when the system encounters any kind of error (failed meal planning, failed action parameterization, failed path planning, collision detection, communication constraint violation, etc.), the system logs the error in the Meal Log with a timestamp and error description, then automatically returns the robot to the scan pose and continues with the next bite. This recovery mechanism allows the

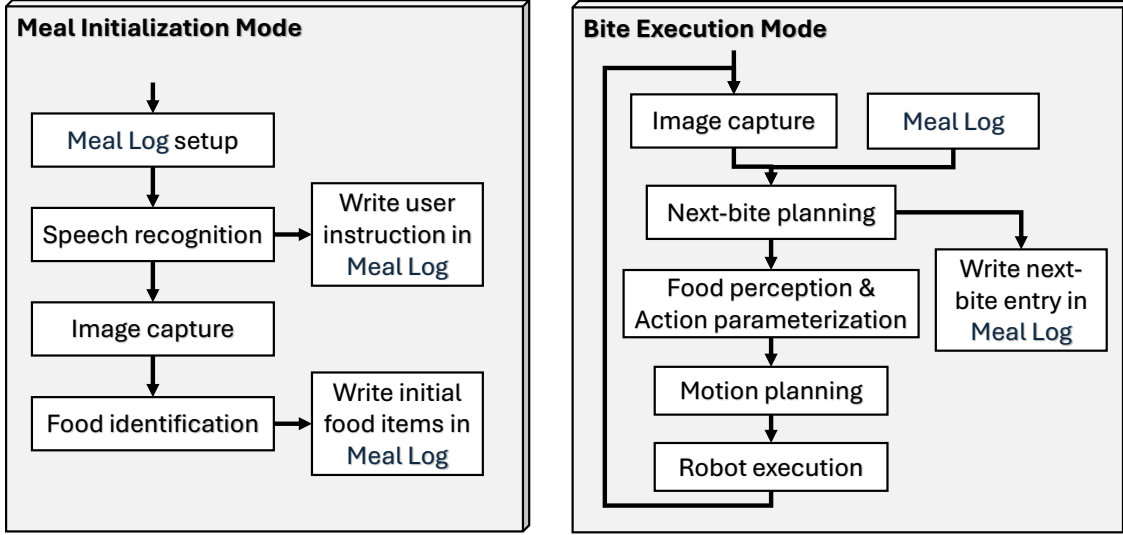


Figure 4.4 System workflow. The left panel shows the service mode that runs once at meal start, and the right panel shows the service mode that runs for every bite of the meal

system to handle failures without requiring a full system restart. In cases where automatic recovery fails or unaccounted-for errors occur, the system performs a controlled shutdown to ensure safety.

4.2 Robotic Manipulation System

This section addresses SO2 and presents the robotic manipulation system, including the coordinate frames used for pose estimation, the workspace configuration and collision-avoidance setup, and the feeding-specific motion primitives implemented for skewering and scooping.

4.2.1 Coordinate Frames and Transformations

Reference Frame Chain

The robotic system uses a chain of coordinate frames to transform visual observations into executable commands. Four primary reference frames are defined in the system: the robot base frame, end-effector frame, camera frame, and fork frame. A gripper frame exists between the two gripper fingers but is not required in the transformation chain. Figure 4.5 illustrates these four reference frames.

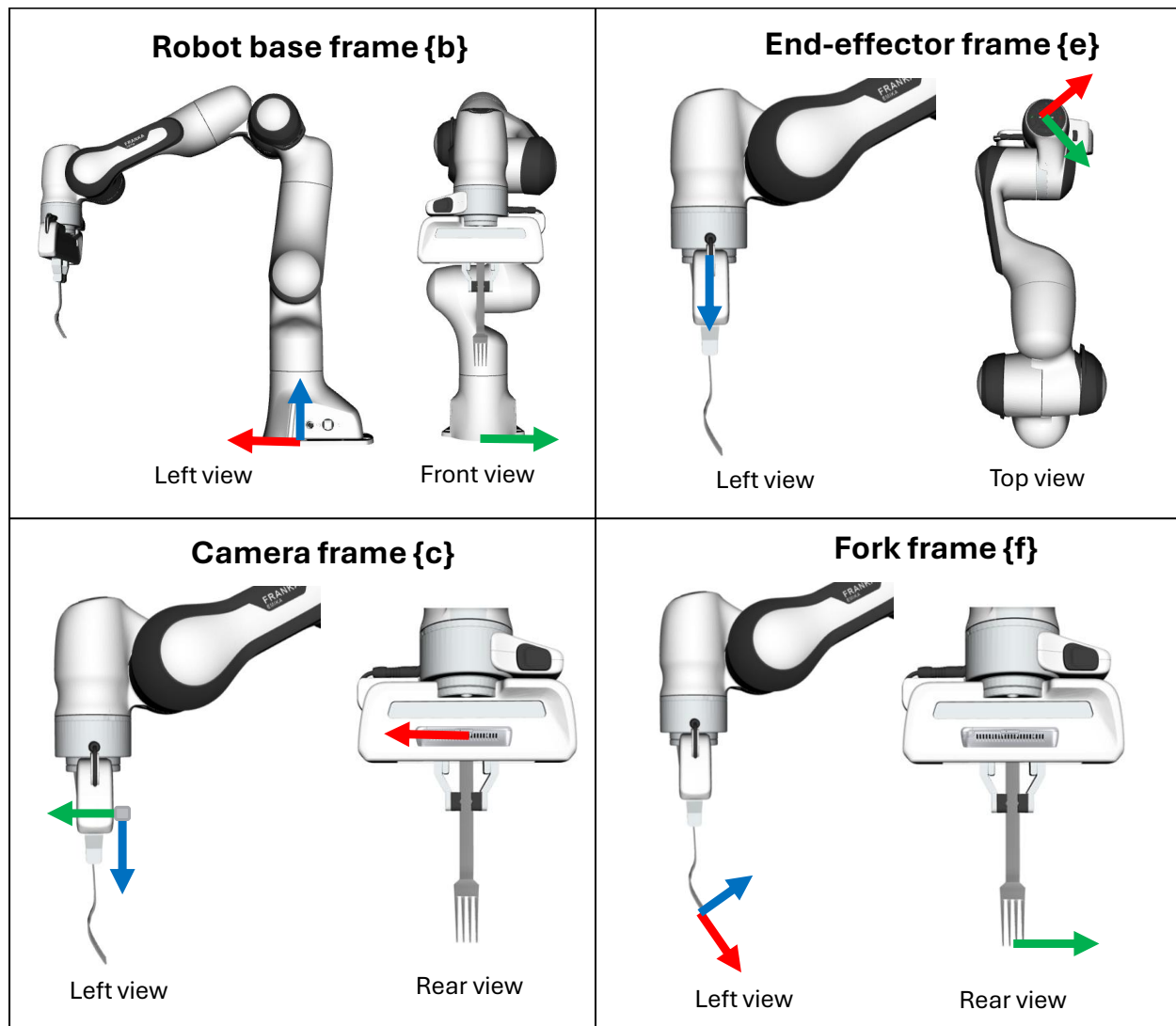


Figure 4.5 Reference frames in the robotic system: robot base frame (origin), end-effector frame at the wrist, camera frame mounted on the end-effector, and fork frame at the middle of the fork tip

Pixel-to-camera Projection

Target localization begins with the pinhole camera model, which converts 2D pixel coordinates to 3D camera coordinates. Given a pixel at position (u, v) with depth Z , the corresponding 3D point in the camera frame is calculated as:

$$\begin{aligned}
X_c &= \frac{Z}{f_x}(u - c_x) \\
Y_c &= \frac{Z}{f_y}(v - c_y) \\
Z_c &= Z
\end{aligned} \tag{4.1}$$

where f_x and f_y are the focal lengths, and (c_x, c_y) is the camera principal point. These intrinsic parameters were obtained from the camera’s factory calibration values. Depth values are extracted from the RGB-D camera’s depth map after applying a smoothing filter to account for pixels with invalid depth due to sensor limitations or surface properties.

Coordinate Transformation Chain

The camera-to-robot transformation chain consists of three sequential transformations. First, the pinhole projection model is used to compute the pixel-to-camera transform, yielding the target food pose $\{t\}$ in the camera frame $\{c\}$, denoted $\mathbf{T}_{t/c}$. Second, the camera frame is related to the end-effector frame $\{e\}$ through a fixed transformation, denoted $\mathbf{T}_{c/e}$, determined from the camera mount’s physical specifications. Third, the end-effector pose relative to the robot base frame $\{b\}$, denoted $\mathbf{T}_{e/b}$, is obtained from forward kinematics at a fixed observation pose, to which the robot returns each cycle to capture the plate image. The complete transformation of the target item pose in the robot base frame, denoted $\mathbf{T}_{t/b}$, from camera coordinates to robot base coordinates is given by:

$$\mathbf{T}_{t/b} = \mathbf{T}_{t/c} \cdot \mathbf{T}_{c/e} \cdot \mathbf{T}_{e/b} \tag{4.2}$$

This chain allows direct mapping from detected food items in camera images to poses in the robot’s workspace.

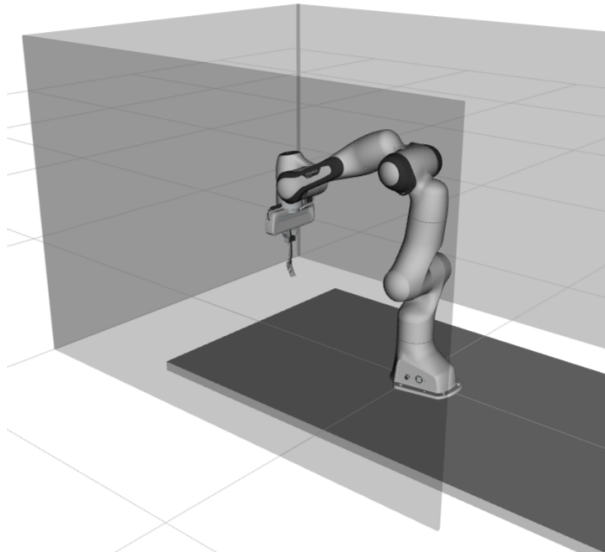
Fork Frame Definition and URDF Integration

The fork attached to the robot’s end-effector was modeled in a CAD environment from physical measurements, and its reference frame $\{f\}$ was defined at the midpoint of the fork tines with orientation aligned to the approach direction (see Figure 4.5). This frame was defined in the robot’s URDF as a fixed transformation from the wrist frame. Defining this fork frame in the URDF allows MoveIt to plan trajectories directly in fork-centered coordinates, so that inverse kinematics can position the fork tines precisely at the target locations while accounting for the fork’s geometry.

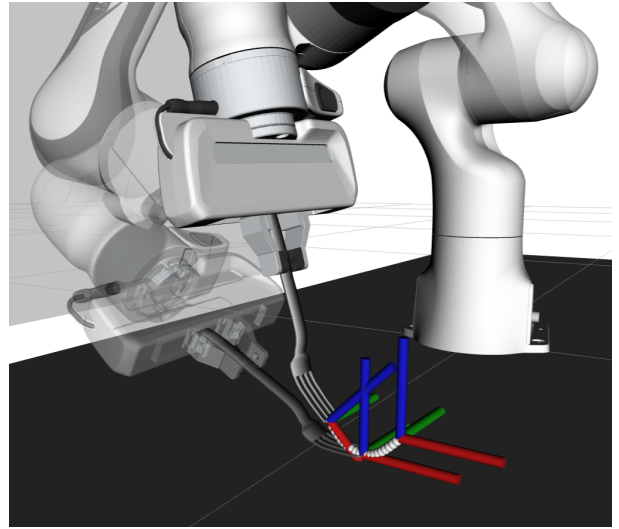
4.2.2 Workspace Configuration and Collision Avoidance

As shown in Figure 4.6a, collision objects representing the physical workspace are integrated into the RViz planning scene: a table surface located directly under the robot base and three virtual safety walls surrounding the feeding area. These structures constrain the motion planner and prevent trajectories that might cause unintended contact with the environment or user. The fork mesh is also attached to the gripper to enable accurate collision checking. MoveIt’s collision-avoidance mechanisms remain active for all planned motions.

During development, RViz was also used to validate planned trajectories before physical execution by pausing the system and inspecting the motion paths. Figure 4.6b illustrates an example trajectory visualization for the scoop primitive. During normal feeding, trajectories are executed automatically without manual confirmation.



(a) RViz environment showing the robot, table, and safety walls



(b) Trajectory visualization of the scoop primitive

Figure 4.6 RViz setup used for workspace modeling and trajectory validation

4.2.3 Skewer Primitive

The skewer primitive targets solid food items that can be pierced by fork tines. The complete skewering trajectory consists of a continuous Cartesian path through three waypoints: pre-skewer $\rightarrow$ skewer-target $\rightarrow$ post-skewer.

For the approach trajectory, the fork starts at a pre-skewer pose offset 3 cm above and 1.5 cm

behind the target, resulting in an approach angle of 30° from the vertical. This offset provides clearance and reduces the risk of premature contact with adjacent food items, while the 30° angle improves item retention during the lift phase.

For the lift trajectory, the fork is raised 3 cm vertically in the robot base frame and simultaneously rotated 45° around the fork's lateral axis (Y-axis). This rotation brings the fork tines closer to horizontal, preventing the skewered item from sliding off during the subsequent motion. Figure 4.7 illustrates the full skewering motion.

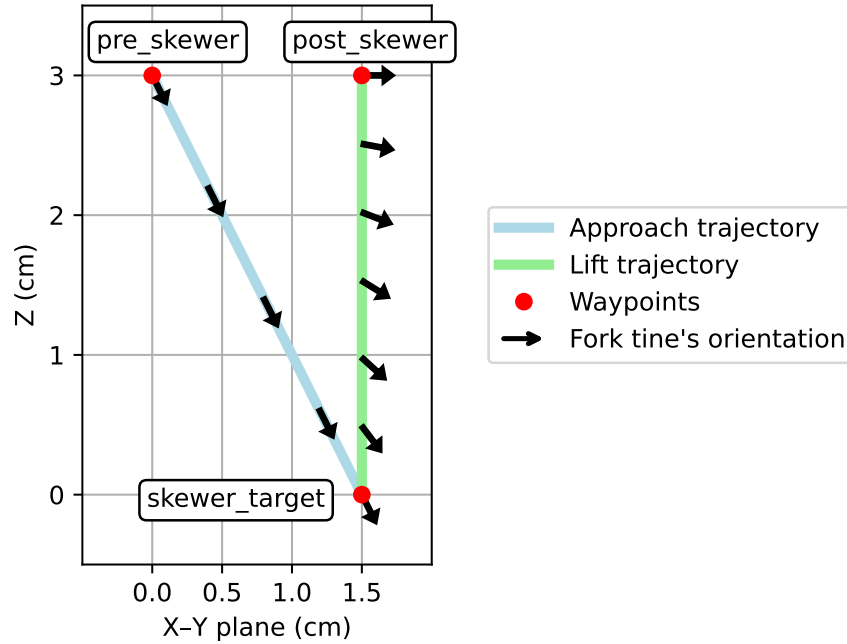


Figure 4.7 Skewer trajectory shown as a 2D projection. The black arrows indicate the orientation of the fork tines along the trajectory.

The robot's Z-axis collision-reflex force threshold was increased from 20 N to 40 N to avoid false collision triggers during skewering.

4.2.4 Scoop Primitive

The scoop primitive is designed for granular, liquid, or soft foods that do not hold their shape. The complete scooping trajectory consists of a continuous Cartesian path through three waypoints: pre-scoop $\rightarrow$ scoop-target $\rightarrow$ post-scoop. The pre-scoop pose is positioned 3 cm behind and 2 cm above the target scoop location. These values were selected empirically for producing trajectories that mimic natural human scooping motion and consistently acquire an adequate amount of food.

The approach trajectory from pre-scoop to scoop-target follows a quadratic Bézier curve, with the fork's pitch angle aligned to the slope of this curve to facilitate a smooth entry into the food.

For the lift trajectory, the fork is raised along a quarter-circle arc with a radius equal to the desired vertical lift of 2 cm (also empirically selected), while maintaining a horizontal fork orientation from the scoop point onward to prevent spillage. Figure 4.8 illustrates this motion.

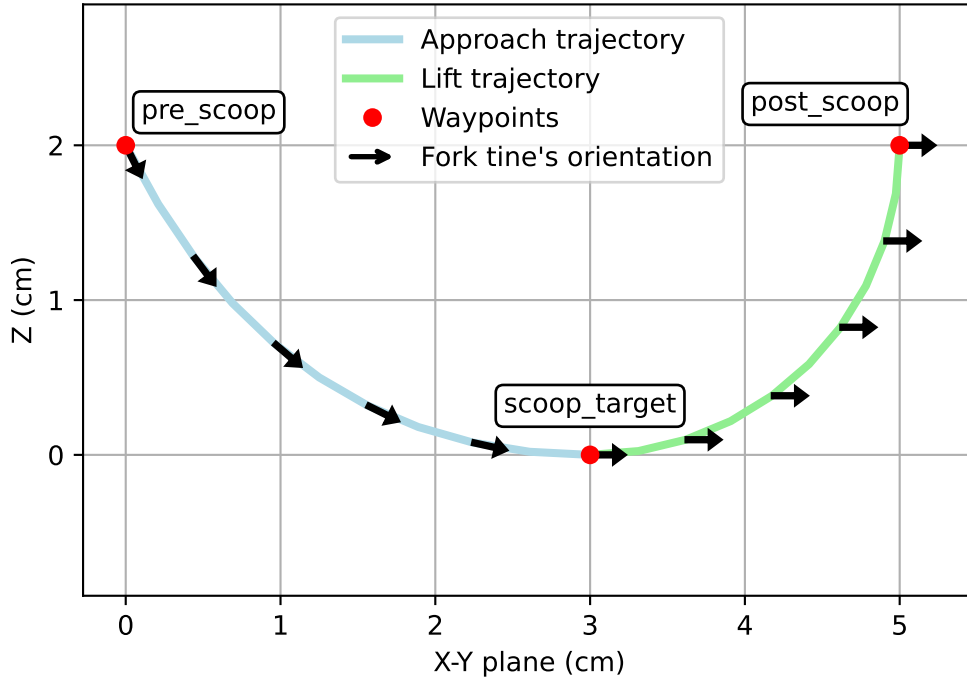


Figure 4.8 Scoop trajectory shown as a 2D projection. The black arrows indicate the orientation of the fork tines along the trajectory.

4.2.5 Execution Flow and Motion Constraints

The system uses a fixed observation pose to position the end-effector-mounted camera directly above the plate with the camera pointing downward, providing a consistent viewpoint for food perception and action parameterization. After completing an acquisition primitive, the robot moves to a predefined bite delivery pose, allowing the user to take the bite from the fork. Finally, after a one second pause at the delivery pose, the robot returns to the observation pose. All Cartesian trajectories are computed through MoveIt. Speed and acceleration are scaled down to ensure safe and controlled motion during feeding tasks. Once the meal session

begins, robot motion cannot be stopped or restarted by the user except via the physical emergency stop.

4.3 Food Detection, Segmentation & Action Parameterization

This section addresses SO3 by introducing the computer-vision pipeline used to detect and segment food items. It also describes how visual data is converted into target poses used to parameterize each manipulation primitive. In this work, a *robotic primitive* refers to a predefined acquisition motion (e.g., *skewer*, *scoop*) implemented as a fixed trajectory template with a small number of free parameters. *Action parameterization* represents the process of computing these free parameters.

4.3.1 Food Detection and Segmentation Pipeline

The system uses the Segment Anything Model 3 (SAM 3) [60], a unified vision foundation model that performs both open-vocabulary object detection and instance segmentation in a single forward pass. Given an RGB image and a text prompt specifying the target food item, SAM 3 directly outputs bounding boxes, confidence scores, and pixel-precise segmentation masks for all detected instances of the specified food item. This single-stage approach eliminates the need for separate detection and segmentation models, reducing inference latency while maintaining robust open-vocabulary capabilities that enable detection and segmentation of any food item without task-specific retraining.

Figure 4.9 presents results for the open-vocabulary query "penne" on a real plate. Only pixel-level segmentation masks are shown, with bounding boxes, labels, and confidence scores omitted for visual clarity.

4.3.2 Primitive Parameter Extraction

Once the target food item has been segmented, the system computes the free parameters required by the robotic primitives. Rather than estimating a full 6-DOF pose end-to-end, the vision system extracts the target pixel coordinates (u, v) , the depth (Z) and a single optimal orientation angle specific to the chosen primitive. These are combined with fixed angular constraints defined for each primitive to construct the full execution pose.

In this context, the fork's roll, pitch, and yaw correspond to rotations about its X-, Y-, and Z-axes, respectively (see Figure 4.5).

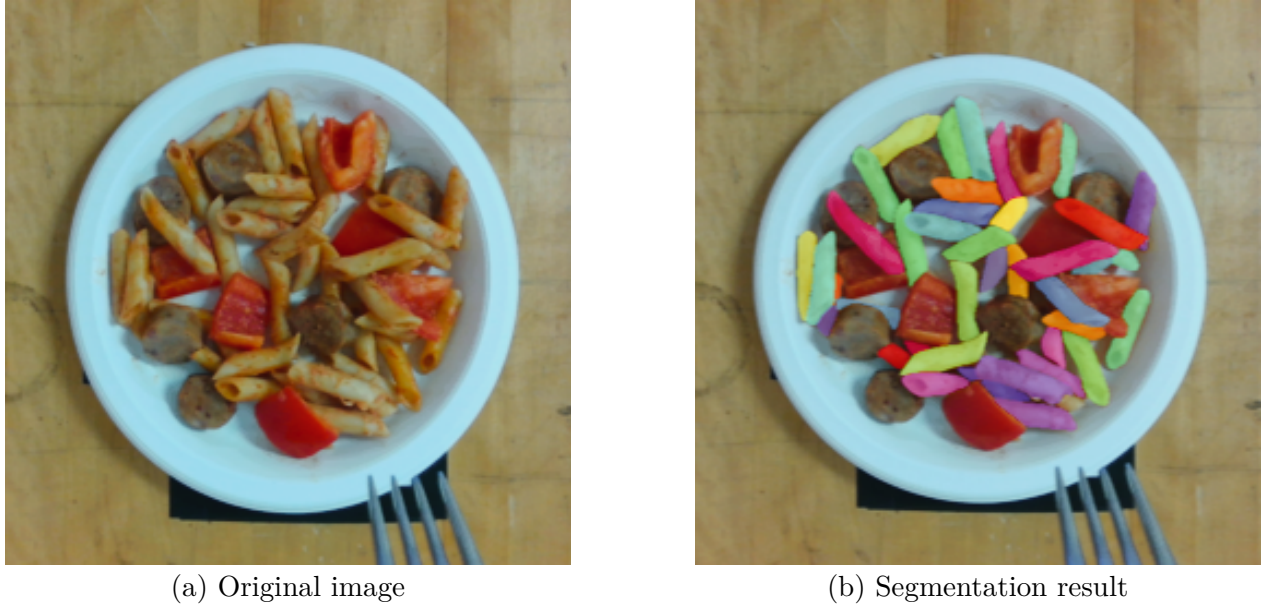


Figure 4.9 Example of detection and segmentation for the query "penne" on a real plate using SAM3

Skewer Parameters

For the skewer primitive, the parameterization process defines the following parameters:

- **Centroid:** The target pixel position (u, v) is defined as the geometric centroid of the segmentation mask.
- **Depth:** The depth Z is extracted from the aligned depth map at the centroid coordinates, with a 2 cm offset applied to enable proper skewering (empirically validated), while ensuring it does not go below the plate surface.
- **Secondary Axis Orientation:** To determine the optimal orientation for the fork, the covariance matrix of the mask is computed to extract its eigenvectors. The system specifically selects the *secondary* eigenvector (minor axis) as the alignment vector. This angle corresponds to the roll of the fork, aligning the fork tines perpendicular to the object's principal axis (e.g., skewering a green bean across its width) to improve insertion success for elongated food items.
- **Fixed orientation constraints:** The pitch angle is set to a fixed 30° , because a vertical 0° approach increases the likelihood that the item slips off during lifting, while a steeper angle increases the risk of pushing the item across the plate surface rather than penetrating it. The yaw angle is fixed at 0° .

If multiple instances of the target food item are detected, the system selects the segmentation mask with the largest area to maximize acquisition success and avoid targeting partially occluded items. Figure 4.10 demonstrates the centroid and principal axis extraction on a real plate, specifically applied to a green bean target to determine the optimal skewering pose.

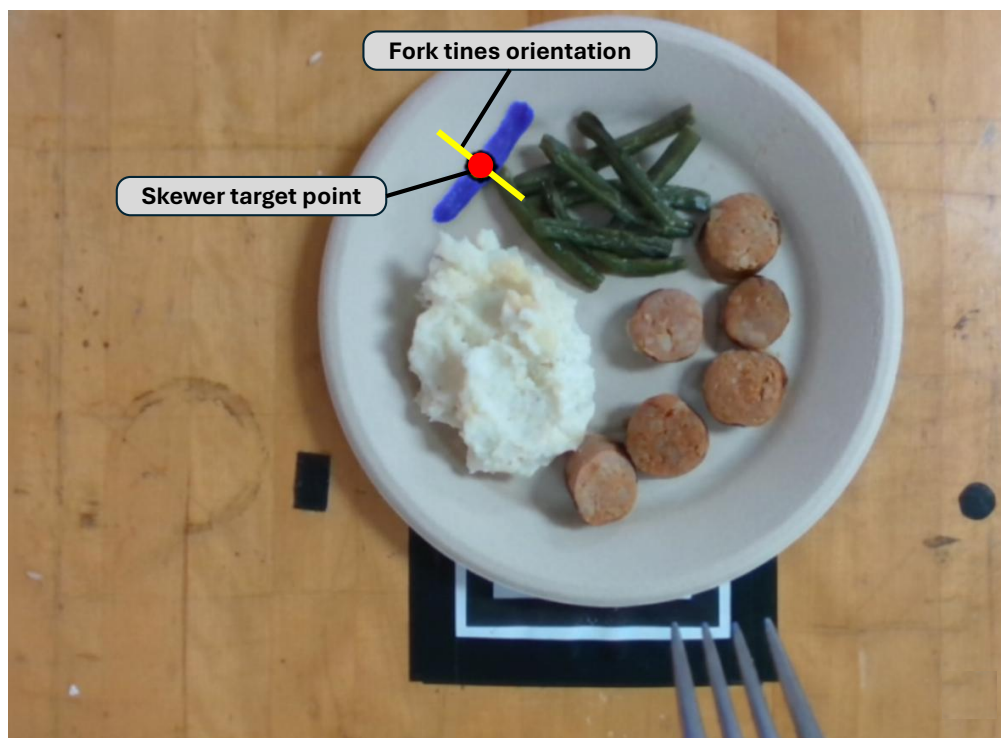


Figure 4.10 Skewer primitive parameterization. The red point indicates the geometric centroid, and the yellow line represents the secondary principal axis (minor axis), which is used to define the fork’s roll orientation.

Scoop Parameters

For the scoop primitive, the parameterization process defines the following parameters:

- **2D coordinates of the densest point:** A custom density heatmap is computed by combining two components: a spatial centrality map, obtained by applying a Gaussian blur (2D Gaussian filter) to the segmentation mask, and a depth-based weighting that assigns higher values to pixels closer to the camera (i.e., higher food piles). The Gaussian filter serves to emphasize the center of the segmented region while down-weighting boundary areas that may lie near the edge of the plate, reducing the risk of unintended contact with the plate rim during scooping. The peak of this combined density

heatmap is selected as the scoop target coordinate (u, v) . Figure 4.11 illustrates this process applied to a pile of mashed potatoes.

- **Depth:** The depth Z is extracted from the depth map at the densest point coordinates, with a 2 cm offset applied to enable a reliable entry depth for acquiring an adequate amount of food (empirically validated), while ensuring it does not go below the plate surface.
- **Approach direction:** As shown in Figure 4.12, the scooping trajectory is defined by a vector starting from the mask pixel most distant from the densest point and ending at the densest target point. This vector determines the yaw angle of the fork, ensuring the robot approaches the food pile from the optimal direction.
- **Fixed orientation constraints:** The roll and pitch angles are fixed at 0° .

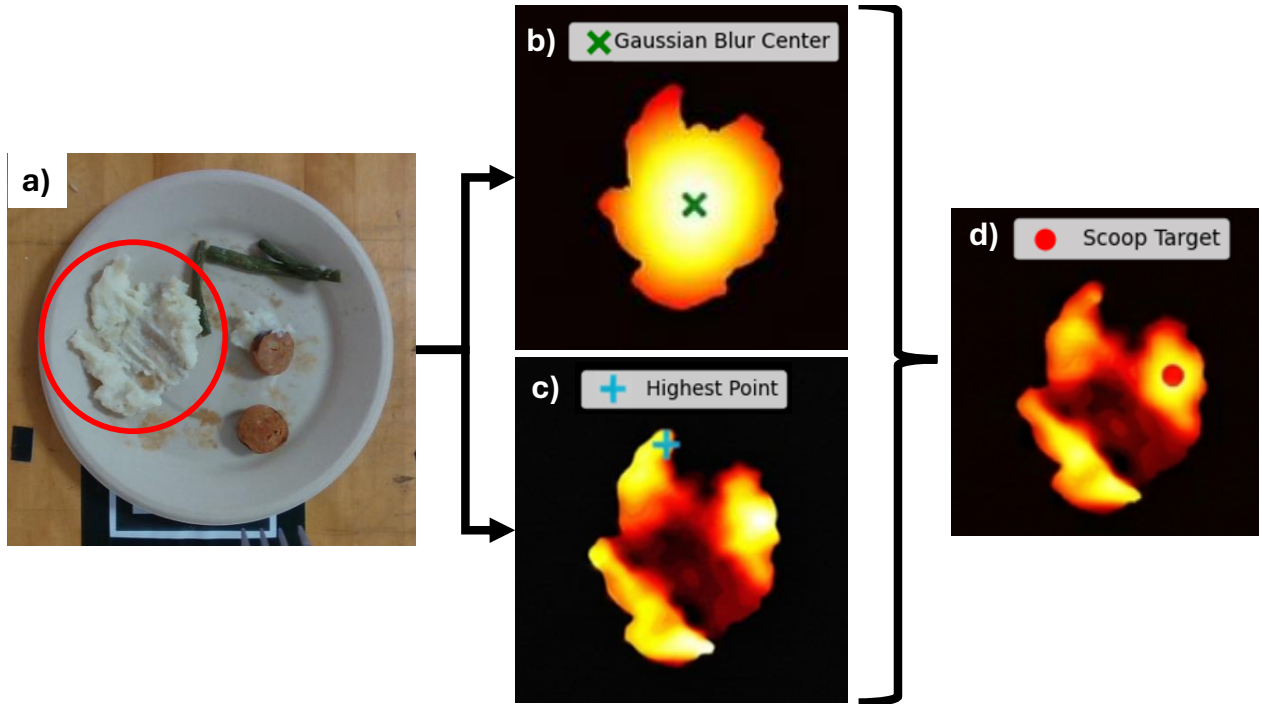


Figure 4.11 Density-based target selection pipeline for the scoop primitive. Starting from (a) an RGB input and corresponding depth map, the system applies a Gaussian blur to the segmentation mask to create (b) a spatial centrality map that assigns higher values to points near the region's interior. This is then combined with (c) a depth-based weighting map that prioritizes elevated regions. The resulting (d) density heatmap highlights the most suitable scoop target as the peak of the combined spatial and depth maps.

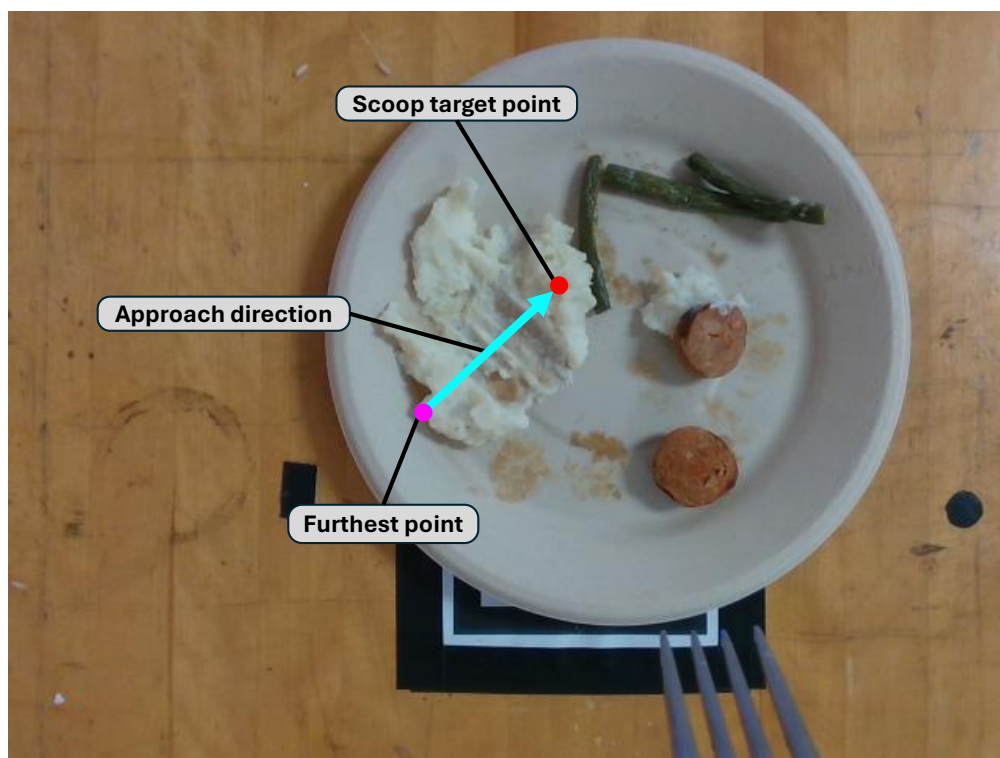


Figure 4.12 Scoop primitive parameterization. The red dot marks the target point selected from the density heatmap, the pink dot marks the point within the same segmentation mask that is furthest from it, and the blue arrow indicates the yaw orientation that defines the approach direction for scooping.

4.4 Natural Language Interaction

This section addresses SO4 by describing the natural language interface through which users express feeding instructions or preferences at the start of each meal session, enabling the robot to personalize its feeding behavior.

4.4.1 User Input Types and Encoding

A voice-based interface captures the user’s feeding instructions at the start of each meal session. This natural language interaction enables users to give feeding instructions that can be anywhere from vague to very specific:

- **General instructions:** Broad directives such as “Feed me the whole plate”. If the user provides no explicit instruction, the planner defaults to autonomous bite selection based on plate configuration and bite history.

- **Exclusions:** Dislikes such as “I don’t like broccoli,” which signal the planner to avoid certain food items.
- **Ordering constraints:** Temporal preferences such as “I want to eat the chicken first,” which influence the action sequence generated by the meal planner.
- **Preferences:** Allow the planner to accommodate specific requests such as “I like chocolate with my bananas, but prefer strawberries on their own,” assuming chocolate dip is available.

4.4.2 Speech Recognition Pipeline and Integration

The speech-based interface is designed to maximize accessibility for users with limited motor functions by eliminating the need for physical interaction with GUI elements. By supporting natural, conversational input, this approach reduces cognitive load and enhances usability for the end user.

The speech recognition module is built using OpenAI’s Whisper model [61], an automatic speech recognition (ASR) system. Whisper was chosen for its robust accuracy in real-world audio conditions and its ability to run locally without requiring external API calls, which reduces latency during meal initialization.

Audio for speech recognition is captured from a webcam’s microphone connected to the workstation PC using PyAudio. The audio data is then sent to the GPU server for transcription. During the meal initialization phase, the system prompts the user via terminal output to press the spacebar to start recording and again to stop recording when finished. The user is provided visual feedback indicating when the system is actively listening and when transcription is in progress. Once recording is complete, the transcribed input is automatically logged to the Meal Log.

The natural language inputs are subsequently passed to the task planning pipeline during each bite cycle as part of the prompt context, enabling the language models to reason about user preferences when selecting the next food item.

The voice interface is only available during meal initialization and cannot be used again mid-meal. However, if a user preference was misinterpreted and needs correction after the meal has already started, it can still be updated by manually editing the Meal Log between bites.

While this work does not include an exhaustive evaluation of the Whisper ASR model, the module was consistently reliable across all real-world feeding trials, each of which begins with a meal initialization step that includes verbal user-preference input. This provides functional

validation sufficient for the scope of this project.

4.5 Task Planning Architecture

This section addresses SO1 by presenting the task-planning architecture that determines the next bite during the feeding process. It outlines the rationale and implementation of the VLM-based approach, details the prompt design strategies used to ensure reliable decision-making, and introduces the expanded action space of manipulation primitives evaluated in simulation.

4.5.1 Planning Framework Design and Approach Selection

As discussed in Section 2.2, symbolic planning frameworks and LLM-based approaches for generating formal specifications (PDDL, TL) have been applied to robotic task planning. However, even with LLM translation layers, these approaches have fundamental limitations for the feeding task: they can’t natively handle open-vocabulary food items, struggle with nuanced user preferences expressed in natural language, and need a separate perception module for visual grounding.

Since the feeding task is short-horizon (just picking the next bite instead of planning long sequences) and needs replanning after each bite (the plate state can change unpredictably), a VLM-based approach was selected. This architecture enables zero-shot generalization to unseen foods, unifies visual perception and semantic reasoning, and naturally integrates user preferences through multimodal prompting.

4.5.2 VLM-Based Meal Planning Architecture

The meal planning system employs a VLM architecture where a single VLM call performs both food identification and next-bite planning simultaneously. Open-source local VLMs such as LLaVA [62] were briefly evaluated during early design iterations, which motivated the decision to offload both the food identification and meal planning services to the GPU server for on-device VLM inference. However, local models were too resource-intensive: even the largest model compatible with the GPU server (32 GB RAM, 12 GB VRAM) was too slow for real-time feeding and had much worse food recognition accuracy and task planning reliability than cloud-based models.

As a result, all VLM inference was migrated to cloud APIs, specifically utilizing GPT-5.1 [63]. This model was selected as a representative state-of-the-art VLM capable of high-performance

semantic reasoning. While comparable models such as Gemini 3 Pro or Claude Opus 4.5 were available, the primary objective of this study is not to benchmark competing architectures, but rather to evaluate the broader potential of VLMs for short-horizon task planning in feeding contexts. GPT-5.1 serves as a robust baseline for these capabilities and aligns with the prevalence of GPT-series models in related literature.

Regarding inference configuration, the model was set to a "Low" reasoning effort (selected from available tiers: None, Low, Medium, High). This specific configuration was chosen to optimize the trade-off between decision reliability and system responsiveness; preliminary testing indicated that while a minimal "thinking budget" is strictly necessary for robust physical reasoning, higher reasoning settings introduced significant latency with no observable improvement in planning accuracy.

At each bite cycle, the VLM receives a structured prompt containing:

- The RGB image of the current plate state,
- The list of food item labels identified at meal start,
- The user's input,
- A preprocessed summary of the bite history,
- The available manipulation robotic primitives (skewer and scoop).

The model is then guided through a constrained four-step reasoning process: identify remaining food items, select the next target by prioritizing variety and avoiding repetition while respecting user preferences, determine the appropriate acquisition primitive based on the item's physical properties, and output the result in a structured format. This prompting strategy encourages consistent labeling, reduce hallucinations, and improve planning reliability across bite cycles. The prompt was deliberately kept minimal to reduce model latency and ensure timely bite decisions during real-world operation. The complete prompt used for real-world deployment is provided below.

VLM Planning Prompt Used in Real-World Deployment

You are a robotic feeding assistant. Decide the next bite.

- Available actions:

- skewer: for items that hold their shape and can be pierced or caught
- scoop: for items that are granular, liquid, or don't hold shape


```
- Food items identified at meal start: {food_items_list}
  (These labels are fixed. Check the image to see which are still present.)
- User input: {user_input}
- Bite history: {bite_history_summary}
```

Think briefly through these 4 steps. No extra commentary.

- 1. State food items still on plate and their history values. Make sure they match labels from "Food items identified at meal start".
- 2. Choose what food item to feed me next. Decision rules:
 - MUST choose ONLY from items you listed as still visible in Step 1.
 - Prioritize user's preference. Never select an item the user dislikes.
 - If no order is provided in user input, pick the item with the highest history value or "not yet fed" if any exist. Prioritize "not yet fed" over highest history value.
 - If multiple valid choices, choose at random.
- 3. State allowed actions, then pick the one matching the item's texture. Decision rule: If an item CAN be pierced, use skewer. Only use scoop if skewering is not feasible.
- 4. Output exactly: `DECISION:action('item')` (e.g., `DECISION:skewer('carrot')`).

4.5.3 Prompt Design

Prompt Engineering Strategy

The VLM prompt employs the following prompting techniques:

- **Role prompting:** The model is explicitly instructed, “You are a robotic feeding assistant. Decide the next bite.” This frames the context and encourages an appropriate behavior for the task.
- **Chain-of-thought prompting:** The prompt breaks the reasoning process into four clearly defined steps (identify remaining foods, select next item, choose action, format output), which reduces hallucinations and ensures the model adheres to the specified constraints.
- **Mistake avoidance through emphasis:** The prompt explicitly requires the model to verify its own outputs against earlier steps, which prevents common reasoning errors. For example, Step 2 instructs the VLM to select a food item *only from the list it*

previously identified as still visible in Step 1, ensuring the model cannot choose or infer items that are not on the plate. This self-verification reduces the risk of the VLM drifting from the provided context.

- **Output format constraints:** The model must return its decision in a parseable format: `DECISION:action('item')`, so the planned action can be reliably extracted.

Prompt Reliability Refinements

Two critical refinements were made to improve planning reliability:

- **Bite history preprocessing:** Rather than providing the raw chronological bite history (e.g., [skewer(“chicken”), scoop(“rice”), skewer(“carrot”), etc.]) as it can become quite lengthy, the history is preprocessed into per-item recency values before being passed to the VLM. For each food item on the plate, a “history value” is computed representing the number of bites since that item was last fed. Items not yet fed are labeled as “not yet fed”. For example:

Preprocessed Bite History

- chicken: history value = 3
- rice: history value = 2
- carrot: history value = 1
- broccoli: not yet fed

This transformation was necessary because when given a raw chronological list and asked to choose the “least recent item”, the VLM often got confused and couldn’t reliably determine which food had been served the longest time ago (even when explicitly told that the end of the list corresponded to the most recent bite). Numerical recency scores provide an unambiguous ordering, enabling more reliable prioritization.

- **Food label consistency enforcement:** The prompt provides the fixed list of food items detected at meal start and instructs the VLM to ensure that any items it reports as still present match these labels exactly. Without this constraint, the model would sometimes relabel foods between bites (e.g., calling an item “chicken” in one step and “meat” in the next), which breaks the alternation logic and corrupts history tracking. Enforcing consistent naming preserves a coherent meal state throughout the meal session.

4.5.4 Expanded Action Space of Primitives for Simulation Experiment

While the physical robot deployment is constrained to a limited subset of actions (skewer and scoop), this simulation-based experiment utilizes an action space of seven manipulation primitives:

- **Pre-acquisition primitives:**
 - **Push:** used to clear an occlusion created by another item.
 - **Group:** used when an item is too scattered or low-density to be acquired.
 - **Cut:** used when a fork-cutable item is not bite-sized and must be resized.
- **Acquisition primitives:**
 - **Skewer:** for solid, pierceable items;
 - **Scoop:** for granular, semi-solid, or non-self-supporting foods.
 - **Twirl:** for long noodles.
- **Post-acquisition primitive:**
 - **Dip:** used when an item can be dipped into a sauce after acquisition.

Exposing the model to an expanded action space along with physical action grounding enables an evaluation of its ability to handle complex physical dependencies (e.g., clearing occlusions, managing food density) that are critical for realistic deployment on in-the-wild food plates.

Physically Grounded Prompting Strategy

A primary challenge in VLM-based task planning is bridging the gap between high-level semantic understanding and the physical constraints of robotic manipulation. For instance, in a robot-assisted feeding context, a planner might detect rice and interpret “give me some rice” as semantically valid, even if the rice is scattered or occluded by another item. Without explicit grounding in the physical realities of the task, such planners may propose actions that are semantically reasonable but physically infeasible.

To address this, this study utilizes a physically grounded prompting strategy. Instead of defining primitives solely by their outcome, the prompt adds details about the trajectory and physical constraints of each primitive. For instance, the skewer primitive is complemented by the following specifications:

Physically Grounded Prompt Segment (Skewer Primitive)

Trajectory: Fork descends vertically to pierce the target.

Constraint: Requires a stable bite-sized target with an exposed top surface area (free of obstacles) at least the width of the fork tines to allow for a vertical strike.

This approach aims to force the model to reason about the physical feasibility of an action (considering factors like density, occlusion, and available workspace) before selecting it. In addition to detailed contextual grounding, the prompt incorporates richer task instructions, as latency is less critical in this setting and the primary objective is to evaluate planning performance. Unlike the real-world deployment prompt, which instructs the model to reason briefly to minimize latency, the simulation prompt requires the VLM to explicitly reason through all four decision steps with fully visible outputs, enabling detailed analysis of its planning logic. The full prompt is provided in Appendix A.

CHAPTER 5 SYSTEM VALIDATION

This chapter presents the experiments conducted to validate the system’s performance and reliability. First, a quantitative and qualitative analysis evaluates the VLM-based planner’s ability to perform physically grounded, short-horizon task planning. Second, a real-world evaluation assesses the performance of the complete end-to-end feeding system using three different home-cooked meals.

5.1 VLM-Based Task Planning Evaluation

The first experiment evaluates the efficacy of VLMs for task planning, with a specific focus on analyzing the impact of physically grounded reasoning.

5.1.1 Dataset and Ground Truth

The VLM planner is evaluated on a dataset of 25 meal scenarios, which was specifically constructed for this work. The dataset covers simple unconstrained plates as well as complex scenarios involving occlusions, low-density items, nuanced user preferences and impossible requests. For example, Scenario 1 represents the most basic scenario, where the VLM must make a bite selection without any user preferences or prior feeding history.

Scenario 1 — No preference and no history

Food items identified at meal start: chicken, rice, broccoli, carrot

User input: none // **Bite history:** none // **Plate image:**



Ground truth response: skewer('chicken') OR scoop('rice') OR skewer('broccoli') OR skewer('carrot')

All plate images used in the 25 evaluation scenarios were generated using OpenAI’s image generation tools. Ground truth for each scenario was generated through a consensus process. Three independent annotators were provided with the same instructions, prioritization rules, and image context as the VLM. They produced action sequences based on physical feasibility, and the consensus output served as the reference label. The full dataset is presented in Appendix B.

5.1.2 Experiment Results

In this experiment, the model executed 10 independent runs for each of the 25 scenarios, yielding a total of 250 trials. By examining repeated executions of the same scenario, this analysis aims to quantify the model’s consistency and isolate scenario-specific limitations in the VLM’s visual grounding capabilities. A discrete score out of 250 (25 scenarios $\times$ 10 runs) was obtained by summing the per-scenario binary outcome. Each scenario contributes a score of 1 if the model’s selected item and chosen primitive sequence exactly match the consensus ground truth, and 0 otherwise. In scenarios where multiple solutions are valid, a score of 1 is awarded if the model outputs any of the acceptable responses. A score of 1 is also given in scenarios with no valid solution if the model correctly outputs an error and the experimenter confirms that the justification aligns with the ground-truth rationale.

The mean standard deviation (SD) of binary outcomes was computed for each scenario over its 10 repeated runs to quantify inter-run consistency. For each scenario i , let p_i denote the proportion of successful runs out of 10. Since each outcome is binary, the variance for scenario i follows a Bernoulli distribution and is given by $Var_i = p_i(1 - p_i)$. Therefore, the mean SD across all 25 scenarios is given as:

$$\bar{\sigma} = \frac{1}{25} \sum_{i=1}^{25} \sqrt{p_i(1 - p_i)} \quad (5.1)$$

Table 5.1 presents the evaluation results.

The planner achieved a total of 208 correct outputs out of 250 trials, corresponding to a success rate of 83.2%. The mean per-scenario standard deviation was 0.088, with 17 of 25 scenarios showing perfect consistency across all runs. Since binary outcomes have a maximum standard deviation of 0.5 (at $p = 0.5$), this low value indicates that the planner showed little fluctuation between success and failure within a given scenario. While this result is based on only 10 runs per scenario and should be interpreted cautiously, it suggests that the planner exhibits stable behavior on the majority of scenarios when provided with a prompt with structured and detailed reasoning steps.

Table 5.1 VLM-based planner performance across 25 scenarios with 10 runs each

Scenario	Score (/10)
1	10
2	10
3	10
4	10
5	10
6	10
7	7
8	10
9	10
10	0
11	10
12	10
13	10
14	10
15	10
16	10
17	1
18	10
19	9
20	9
21	10
22	9
23	3
24	10
25	0
Global Score	208 / 250
Success Rate	83.2%
Mean SD	0.088

Table 5.2 summarizes the API latency, token usage, and cost statistics for the VLM-based task planning experiment, based on 250 independent planner calls. As of the time of writing, the pricing for GPT-5.1 inference is \$1.25 USD per million input tokens and \$10.00 USD per million output tokens.

While eating speed varies significantly between individuals, one clinical study reports an average consumption rate of approximately 3 to 4 bites per minute [64], suggesting a practical cycle time target of 15–20 seconds per bite. In this context, the average API latency of 11.6 seconds is not ideal for real-time deployment, especially when additional delays from

Table 5.2 Performance metrics per API call for the VLM-based planner ($n = 250$)

Metric	Value (Mean $\pm$ SD)
Latency	11.6 ± 2.8 s
Input token usage	1569 ± 16
Output token usage	534 ± 112
Cost	$\$0.0073 \pm 0.0011$ USD

perception and robotic execution are factored in. However, this latency is expected in the simulation experiment, which was deliberately configured with an expanded action space (seven primitives) and a richer, physically grounded prompt to evaluate the planner’s spatial reasoning and affordance understanding. The increased complexity of the planning task naturally leads to longer inference times.

Given the inherently slow motion profiles of collaborative robots used in assistive settings, minimizing processing latency is essential not only for system efficiency but also for maintaining a natural and comfortable user experience. Ideally, the robot should wait for the user to be ready for the next bite, not the other way around.

From a cost perspective, the API usage remains relatively low. A typical meal involves roughly 30 to 60 bites, corresponding to an estimated per-meal cost of \$0.22 to \$0.44 USD. While this cost may accumulate over time (e.g., across three meals daily for a year), it remains modest when compared to the hourly wage of a human caregiver. As such, the approach offers a favorable trade-off between autonomy and cost, particularly in scenarios requiring scalable assistive feeding solutions.

5.1.3 Failure Mode Analysis

To better understand the failure modes of the VLM-based planner, all incorrect outputs (42 out of 250 trials) were manually categorized into three error classes based on their underlying cause:

- **Spatial reasoning errors** occur when the planner misjudges spatial relationships between objects and fails to anticipate the preparatory actions needed to make a target item accessible.
- **Action affordance errors** arise when the planner selects an action primitive that is physically incompatible with physical properties of the target item.

- **Visual quantification errors** involve misestimates of visual attributes such as density, size, or quantity, leading to the selection of inappropriate actions.

Table 5.3 summarizes these categories, the scenarios in which they occurred, the corresponding error counts, and their overall proportion (rounded):

Table 5.3 Categorization of 42 VLM planner errors by failure type

Error Type	Affected Scenarios	Error Count	Proportion
Spatial Reasoning	10, 17	19	45%
Action Affordance	7, 22, 25	14	33%
Visual Quantification	19, 20, 23	9	21%

Spatial Reasoning Errors

These errors reflect the planner’s difficulty in reasoning about spatial constraints and occlusions. For example, in Scenario 10, the system correctly identified that the salmon was on top of the rice and acknowledged the clear runway required by the scoop primitive. However, it overestimated the size of the open regions of rice and omitted the necessary push action to create a clear runway.

VLM Reasoning Segment of Scenario 10 (Run 8/10)

The scoop primitive requires a clear runway along the plate surface with no blocking obstacles. {...} There are open regions of rice around and between salmon cubes where the fork can approach horizontally and slide through a dense patch of rice without colliding with salmon or asparagus. No need to push, the rice is not occluded in all regions.

Similarly, in Scenario 17, the model overlooked the need to displace a piece of chicken that was sitting on top of a noodle strand before attempting to group and twirl noodles.

VLM Reasoning Segment of Scenario 17 (Run 3/10)

group: can scrape scattered items into a dense pile if the scraping path is clear. The plate is mostly empty, so there is ample free space and no obstacles; we can scrape the strands together.

These errors reflect a form of spatial misperception: the VLM often fails to recognize physical constraints such as occlusions, clearances, and available workspace. As a result, it cannot

reliably anticipate the intermediate actions needed to make a target item accessible.

Action Affordance Errors

Here, the planner chose action primitives that were incompatible with the physical properties of the target items. For instance, in one run of Scenario 7 the model correctly rejected the `twirl` primitive after identifying that the remaining noodle portion was too sparse to acquire directly. However, instead of resorting to a `group` action, it incorrectly chose the `skewer` primitive.

VLM Reasoning Segment of Scenario 7 (Run 2/10)

Twirl primitive requires a concentrated, high-density noodle cluster. The remaining strands are sparse and separated, so twirl would not produce a sufficient bite and fails its density constraint. {...} A portion of a strand can be pierced to obtain a bite-sized amount.

In Scenario 25, the VLM consistently chose to scoop soup using a fork, judging the soup to be dense enough.

VLM Reasoning Segment of Scenario 25 (Run 1/10)

The soup appears thick enough for the fork tines to retain some amount (meets density requirement).

This behavior disregards the fundamental mismatch between the tool and the target texture. These mistakes suggest that the VLM sometimes misjudges the manipulability of items based on their visual features, highlighting its limited grasp of implicit physical affordances and tool-use constraints.

Visual Quantification Errors

These errors stemmed from incorrect visual judgments about food density or quantity. In Scenario 23, the model correctly identified that the rice was sparse, but visually overestimated how densely it was distributed. This led it to assume that the rice could be grouped into a sufficiently compact pile for scooping.

VLM Reasoning Segment of Scenario 23 (Run 6/10)

Best acquired via scoop, but density is currently low and scattered. {...} A group action can scrape scattered rice grains into a denser pile, making a subsequent scoop feasible.

These errors reflect challenges in visual quantification, where misjudgments of size, density, or quantity can lead to incorrect assumptions about what actions are physically possible.

Takeaway

In this experiment, the planner was perfectly reliable in following user preferences and ordering rules, with all 42 failures rooted exclusively in invalid physical reasoning. The system never misunderstood instructions or violated priority constraints. Instead, all errors stemmed from misjudging spatial relationships, incorrectly assessing action-item compatibility, or misestimating visual quantities like density and size. Even with detailed execution descriptions specifying trajectories and constraints, the VLM consistently struggled to assess occlusions, workspace clearance, and physical feasibility. These results demonstrate that while VLMs excel at semantic reasoning for task planning, reliable physical grounding remains a fundamental limitation requiring complementary spatial reasoning capabilities.

5.2 Full-System Evaluation on Realistic Meal Scenarios

The second experiment evaluates the performance of the selected vision-language model (VLM) integrated into the complete robot-assisted feeding system. This setup tests end-to-end system behavior on real hardware using three actual, home-cooked meals.

Due to hardware limitations, only two manipulation primitives—*skewer* and *scoop*—are implemented on the real robot. The other five primitives used in simulation are not yet supported. Therefore, each plate contains only bite-sized food items that can be directly acquired using a fork. The plate was secured to the table from the bottom using double-sided tape to prevent any movement during food acquisition by the robot. Each meal trial is documented with: (1) the initial plate image, (2) the Meal Log state after the meal initialization phase (including visual food detection and user voice input), and (3) the success rate along with a detailed description of all failure cases observed during the meal session. These elements are presented in the following three meal-specific evaluations, each corresponding to one real-world feeding session. The section concludes with an overall performance summary and a breakdown of latency statistics, including API response time.

5.2.1 Meal 1: Penne with Sausage and Red Peppers

Meal Initialization

Figure 5.1 shows the initial state of the plate.



Figure 5.1 Initial plate configuration for Meal 1: penne, sausage, and red pepper

The Meal Log shown below reflects the system state after the completion of the Meal Initialization phase, including detected food items and the transcribed user input.

Initial Meal Log for Meal 1

Bite history: []

Food items identified at meal start:

- penne
- red pepper
- sausage

User input: "I don't like peppers"

Performance and Error Analysis

Figure 5.2 illustrates a penne skewering sequence, highlighting the fork trajectory and the acquisition of a solid bite.

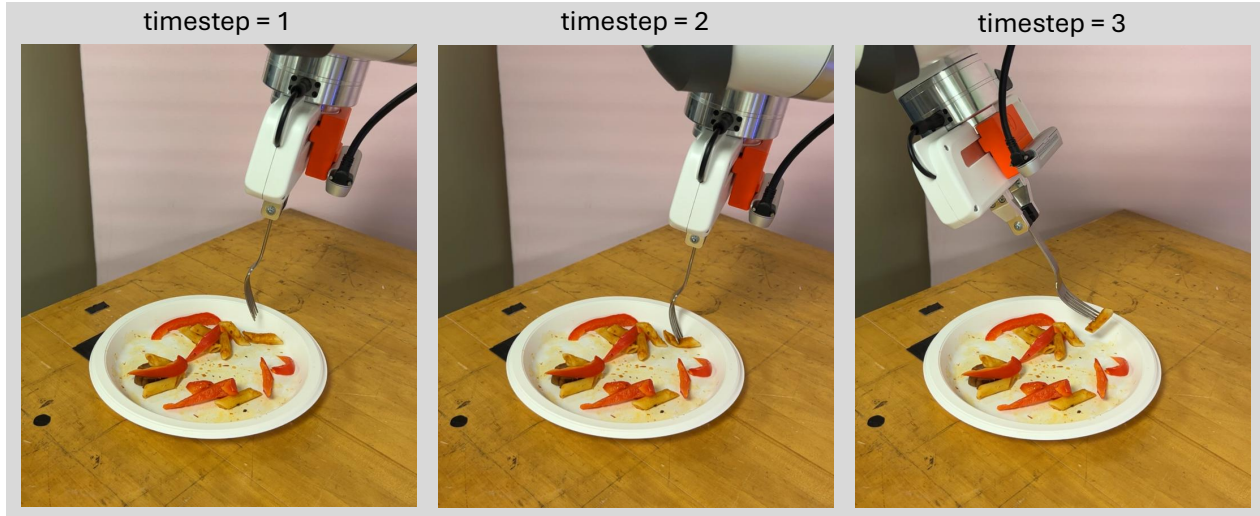


Figure 5.2 Keyframes from a penne skewering sequence during Meal 1

Table 5.4 Meal 1 performance summary

<code>skewer('sausage')</code> Attempts	13
<code>skewer('penne')</code> Attempts	35
Total Bite Attempts	48
Successful Bites	35
Failed Bites	13
Success Rate	73%

Table 5.4 summarizes the performance of the system on Meal 1.

The user preference was followed throughout the session, with the planner never intentionally selecting red pepper as a target item. The 13 failures recorded during Meal 1 are detailed below.

- **Meal Planning Errors (2):**

- In two instances, the VLM selected `skewer('sausage')` even though all sausage pieces had already been consumed and only small plate residue visible.

- **Action Parameterization Errors (6):**

- In all six cases, the SAM 3 segmentation model failed to detect any remaining penne, despite several pieces being clearly visible on the plate.

- **Food Acquisition & Delivery Errors (5):**

- Two skewer attempts failed to properly acquire penne.
- One penne bite was dropped during delivery due to insufficient piercing.
- In one case, the fork accidentally acquired an adjacent red pepper while skewering penne, an item explicitly disliked by the user.
- One execution failure occurred when the robot reached a joint limit while attempting to acquire a sausage located at the edge of the plate, which indicated that the manually defined trajectory constraints were insufficient in this case.

During this meal session, the experimenter intervened a total of six times: once to unlock the robot after it reached a joint limit, once to reposition a penne piece that had failed to be acquired twice, once to return a sausage that was unintentionally picked up while skewering a penne resting on top of it, and three times to reposition penne pieces after failed detections.

5.2.2 Meal 2: Pork with Mashed Potatoes and Green Beans

Meal Initialization

Figure 5.3 shows the initial state of the plate.



Figure 5.3 Initial plate configuration for Meal 2: pork, mashed potatoes, and green beans

The Meal Log shown below reflects the system state after the completion of the Meal Initialization phase, including the food items detected by the perception module and the user input transcribed through speech recognition.

Initial Meal Log for Meal 2

Bite history: []

Food items identified at meal start:

- meat
- green bean
- mashed potato

User input: "Can I get all the pork first, then all the green beans, and then the mashed potatoes."

Performance and Error Analysis

Figure 5.4 illustrates a mashed potato scooping sequence, highlighting the fork trajectory and the acquisition of a semi-solid bite.

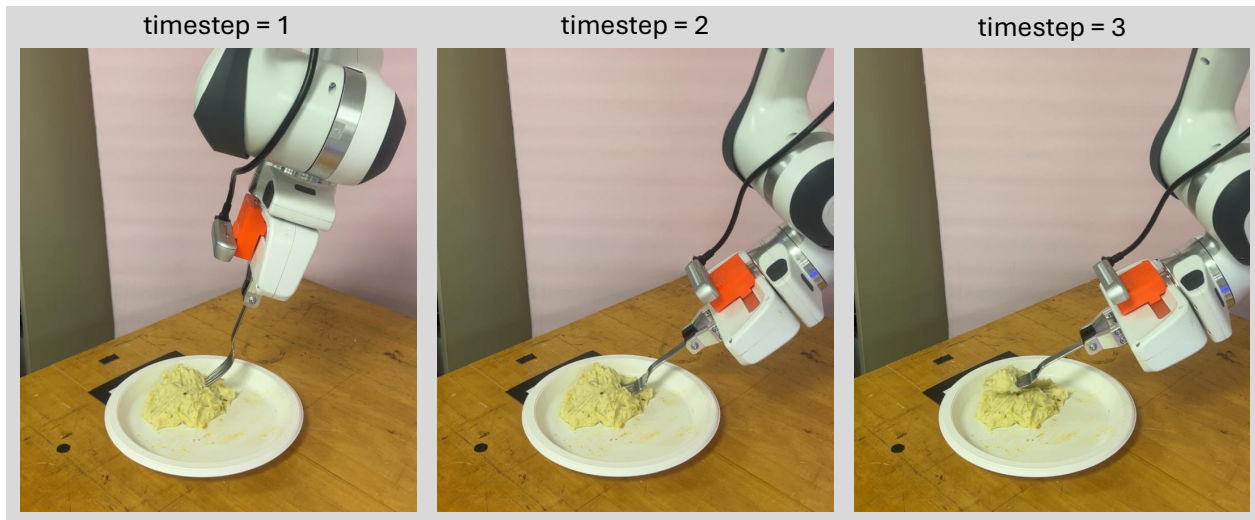


Figure 5.4 Keyframes from a mashed potato scooping sequence during Meal 2

Table 5.5 summarizes the performance of the system on Meal 2.

With the exception of two out-of-sequence selections, the user's requested item order was followed, with pork served first, followed by green beans and then mashed potatoes. A total of 6 failures were recorded during Meal 2, categorized as follows:

Table 5.5 Meal 2 performance summary

<code>skewer('meat')</code> Attempts	9
<code>skewer('green bean')</code> Attempts	13
<code>scoop('mashed potato')</code> Attempts	15
Total Bite Attempts	37
Successful Bites	31
Failed Bites	6
Success Rate	84%

- **Meal Planning Errors (2):**

- In two cases, the VLM selected a food item (once `meat`, once `green bean`) that had already been fully consumed and only small remaining fragments or residue left on the plate.

- **Food Acquisition & Delivery Errors (4):**

- In two instances, mashed potatoes were dropped during delivery due to residual food sticking underneath the fork.
- Two scoop attempts near the end of the meal failed to retrieve a sufficient quantity of mashed potatoes, as the remaining portion was too sparse.

During this meal session, the experimenter intervened once to clean the bottom of the plate with a paper towel after the VLM twice identified the presence of `meat` and `green bean`, despite none remaining on the plate. The session was terminated after repeated scoop attempts failed to retrieve a sufficient quantity of mashed potato to constitute a bite.

5.2.3 Meal 3: Chicken with Rice, Broccoli and Carrots

Meal Initialization

Figure 5.5 shows the initial state of the plate.

The Meal Log shown below reflects the system state after the completion of the Meal Initialization phase, including detected food items and the transcribed user input.



Figure 5.5 Initial plate configuration for Meal 3: chicken, rice, broccoli and carrots

Initial Meal Log for Meal 3

Bite history: []

Food items identified at meal start:

- chicken
- rice
- broccoli
- carrot

User input: "Can you give me all the vegetables first"

Performance and Error Analysis

Table 5.6 summarizes the performance of the system on Meal 3.

The user's requested item order was followed, with all broccoli and carrots served first before engaging with the remaining food items. The 15 failures recorded during Meal 3 are detailed below.

- **Food Acquisition & Delivery Errors (15):**

- In five skewer attempts, the target item could not be acquired because it was resting on top of rice or another food item, causing it to be pushed downward

Table 5.6 Meal 3 performance summary

<code>skewer('chicken')</code> Attempts	9
<code>scoop('rice')</code> Attempts	11
<code>skewer('broccoli')</code> Attempts	8
<code>skewer('carrot')</code> Attempts	11
Total Bite Attempts	39
Successful Bites	24
Failed Bites	15
Success Rate	62%

rather than pierced.

- Three skewer attempts failed due to the irregular shape of the target item, which caused it to shift upon contact rather than being successfully skewered.
- In five scoop attempts, the fork failed to retain a sufficient quantity of rice for the bite to be considered valid.
- One scoop attempt resulted in rice being spilled outside the plate during the lifting motion.
- One attempt resulted in a fork collision with the side of the plate, indicating that the manually defined trajectory constraints were not sufficiently accurate to prevent contact in this case.

The experimenter intervened once to reposition a carrot slice that was resting on its curved side, making it unstable for skewering; the slice was flipped onto its flat base to provide a stable surface for acquisition. The session was terminated after the collision between the fork and the edge of the plate; although some rice was still present, multiple failed attempts properly scoop rice indicated that continuing was no longer meaningful.

5.2.4 Global Performance Overview

End-to-end Success Rates

Figure 5.6 presents a Sankey diagram visualizing the flow of all 124 bite attempts across the real-world feeding sessions. Each stage represents a key component in the pipeline: planning, action parameterization, and execution. Green branches indicate successful transitions

through each stage, culminating in end-to-end success, while red outflows correspond to errors occurring at each stage.

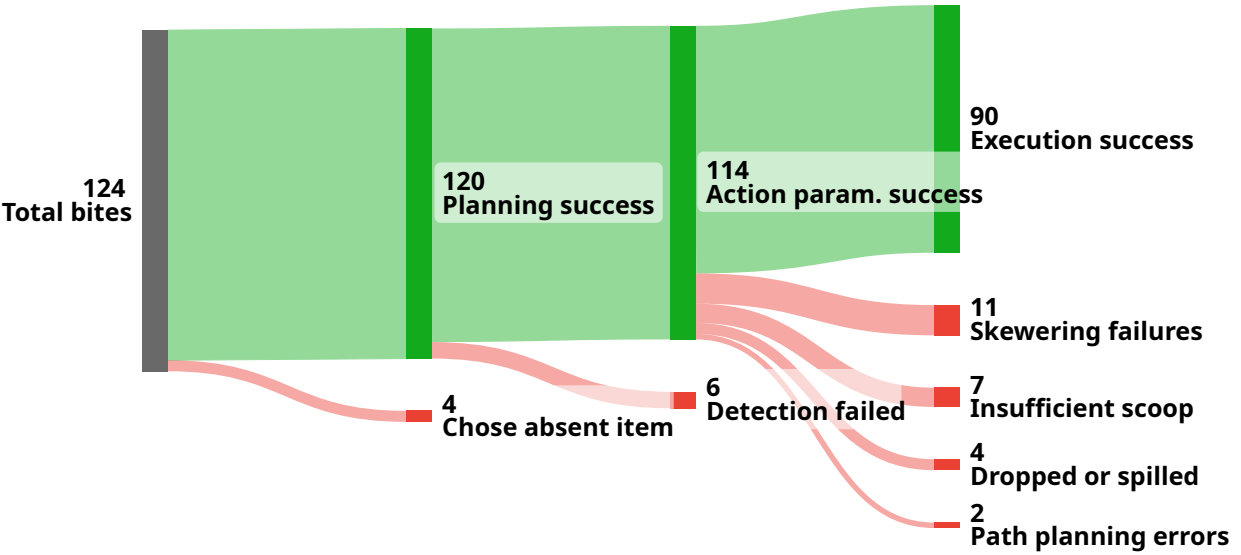


Figure 5.6 Sankey diagram showing the flow of 124 bite attempts through the three main execution stages and failure points categorized by error type

Table 5.7 summarizes the stage-wise success rates of the end-to-end feeding system, highlighting the performance at each major pipeline stage—meal planning, action parameterization, and food acquisition—as well as the overall task success rate.

Table 5.7 Stage-wise success rates across the feeding pipeline

Stage	Success Rate
Meal Planning	97%
Action Parameterization	95%
Food Acquisition	79%
Overall	73%

All meal planning errors stemmed from the vision-language model (VLM) selecting food items that had already been consumed, despite explicit prompt instructions requiring selection to be based on the current plate image, highlighting a key VLM limitation. In the action

parameterization phase, all failures were due to missed detections of penne by the SAM 3 model; overall, this stage demonstrated solid performance and reliability. Execution-phase errors primarily arose from the use of fixed, open-loop motion primitives without any form of real-time feedback or corrective adaptation. In fact, the robot was unable to compensate for dynamic changes in food position or orientation, such as items shifting, slipping, or deforming during contact. Furthermore, although the predefined trajectories were generally well designed, some edge cases produced invalid motions that led to joint-limit or plate-collision events.

Latency Analysis

Figure 5.7 illustrates the average temporal breakdown of a full feeding cycle, based on 112 complete execution cycles. The average total duration per cycle is 20.6 ± 1.7 s, segmented into five stages. The cycle begins with meal planning via the VLM API, followed by the detection, segmentation, and action parameterization stage. Most of the latency in this phase is attributed to the SAM 3 model inference, as the subsequent 3D target extraction relies on a deterministic algorithm with negligible runtime overhead. Next, the food acquisition and delivery stage encompasses robot motion from the observation pose to the bite delivery pose. The final two stages have fixed durations: a 1.0 s pause at the delivery pose, and a predefined joint trajectory that returns the robot to the observation pose.

Table 5.8 reports the average number of bite attempts and total duration per meal session, aggregated over the three real-world feeding trials.

Table 5.8 Average bite count and duration per meal session

Average number of bites per meal	41.3
Average meal duration	13 min 23 s

Planning latency is reasonable at an average of 4.5 seconds as the real-world setup supports only two action primitives (*skewer* and *scoop*) and does not require complex spatial reasoning like in the VLM simulation evaluation where planning time averaged 11.6 seconds. Perception performance is solid, with SAM 3 consistently completing detection and segmentation in under 3 seconds, even when processing plates with over 30 individual penne pieces. The food acquisition and delivery phase accounts for the largest portion of latency, primarily due to conservative robot speed and acceleration settings intended to ensure user safety and minimize the risk of dropping food during transport. Overall, the average cycle time of 20.6 seconds per bite results in a total meal duration of roughly 15 minutes by including the

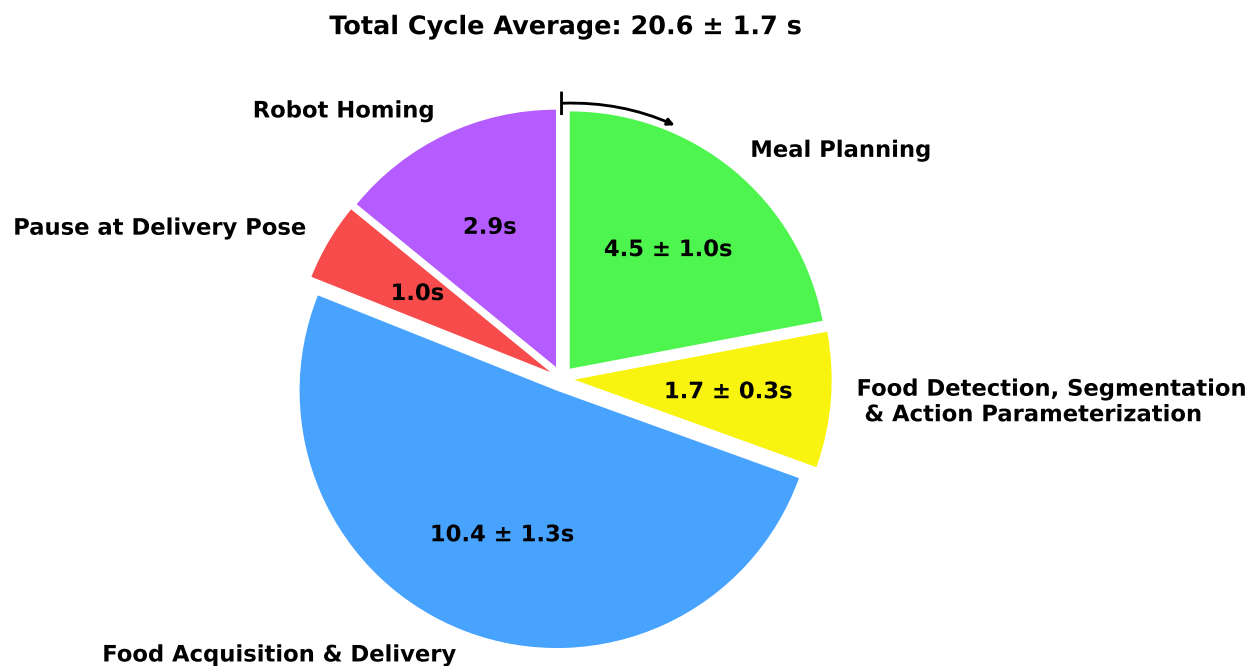


Figure 5.7 Average latency breakdown of the full feeding cycle across 112 complete executions. Each segment represents the mean time spent per operation, including planning, perception, execution, and fixed-duration steps.

initialization phase, which is an acceptable timeframe for real-world deployment in robot-assisted feeding scenarios.

CHAPTER 6 CONCLUSION

This chapter summarizes the project’s main contributions in relation to its specific objectives (Section 3.2), discusses the system’s limitations and scope, and outlines directions for future work.

6.1 Summary of Works

This thesis presents and validates a robot-assisted feeding system driven by vision-language models (VLMs) for real-time task planning in realistic meal settings. The system integrates task planning, manipulation, perception, and natural language interaction to improve the viability of autonomous feeding assistance.

To meet SO1, a VLM-based planning architecture was developed that enables the system to interpret visual plate states, user preferences, and bite history to generate context-aware feeding decisions. The planner performs semantic and physical reasoning within a single inference step, reducing reliance on symbolic representations or manually designed rules.

In support of SO2, the system introduces empirically tuned acquisition strategies for both solid and semi-solid food items, implemented as two parameterized motion primitives: *skewer* and *scoop*. SO3 is addressed through a perception pipeline that integrates the SAM 3 model for open-vocabulary segmentation, followed by custom parameter extraction methods to estimate feasible acquisition poses. This allowed the system to reliably localize food items and generate motion parameters for manipulation.

To personalize feeding behavior and address SO4, a speech-based user interface was implemented, allowing users to express their preferences and constraints in natural language. These instructions are transcribed via a speech recognition module and incorporated directly into the planner’s prompt, enabling adaptive, preference-aware decision-making.

The full system was validated through two experiments. The first evaluated the VLM planner in simulation using seven manipulation primitives and a prompt design grounded in physical constraints, tested across 25 meal scenarios with varying plate layouts and user instructions. Across 250 trials (10 runs per scenario), the planner achieved an overall accuracy of 83.2%. While it reliably followed semantic instructions and user preferences, it showed limitations in visual reasoning, especially in scenarios requiring spatial understanding or assessment of physical feasibility. These issues were especially apparent in tasks that required multi-action sequences.

Finally, SO5 was achieved in the second experiment, where the system was deployed on a 7-DOF Franka Emika Panda robotic arm and evaluated across three complete home-cooked meal sessions using a restricted action space of two acquisition primitives. A total of 124 bite attempts were performed, yielding an end-to-end success rate of 73%. The food acquisition phase proved to be the most challenging, with a success rate of 79% and most failures arising from execution-level constraints rather than perception or reasoning errors. These execution-phase errors were primarily caused by the use of fixed, open-loop motion primitives that lacked real-time feedback or corrective adaptation. As a result, the robot was unable to compensate for dynamic changes in food position or orientation, such as items shifting, slipping, or deforming during contact. The average per-bite cycle duration was 20.6 seconds, and the average meal duration was approximately 13 minutes and 23 seconds for an average of 41 bites per meal. The real-world evaluation highlights the system’s responsiveness and practical feasibility, advancing deployable feeding systems that leverage multimodal AI.

6.2 Limitations

Although the proposed system demonstrates the feasibility of VLM-driven robot-assisted feeding, several limitations restrict its applicability in real settings. First, the action space of manipulation primitives implemented on the physical robot is limited to *skewer* and *scoop*. While these primitives were fully parameterized and validated, they are insufficient for many real-world feeding scenarios that require preparatory actions such as *group* for scattered items or *push* to resolve occlusions. These additional primitives were explored only in simulation and never executed on hardware, limiting the system’s capacity to handle complex plate configurations.

All motion primitives are executed as open-loop trajectories with manually defined offsets and without tactile sensing, force feedback, or slip detection. As a result, the robot cannot adapt when food shifts, slips, or deforms during acquisition. The perception pipeline also has practical shortcomings: while SAM 3 provides robust open-vocabulary segmentation, it occasionally fails to detect small or low-contrast items.

The VLM-based planner exhibits inherent limitations. Although reliable in following user preferences and semantic constraints, it struggles with physically grounded reasoning, especially in scenarios requiring spatial understanding, action affordance estimation, or visual quantification. These limitations become particularly restrictive in scenarios that would require multi-step acquisition sequences. Furthermore, the system relies on OpenAI’s cloud-based VLM inference, creating a dependency on network connectivity and proprietary models, while introducing recurring inference costs that can accumulate significantly with frequent

daily use. Also, larger prompts that include a broader action space of primitives and detailed action specifications can slow inference and reduce real-time responsiveness.

On the interaction side, speech recognition is available only during meal initialization, and the system does not support mid-meal verbal corrections. The system lacks a graphical user interface that would allow users to view system state and launch meals without using a computer terminal. Also, users cannot stop or modify robot motion once execution begins, aside from the hardware emergency stop. The system is not portable or self-contained: it depends on a stationary Franka Emika Panda arm, external computing resources, and a wired multi-PC setup. This limits deployment in assistive contexts, such as wheelchair-mounted robotic arms in home environments.

Finally, real-world validation was conducted only in controlled laboratory settings with the experimenter acting as the user. No trials were performed with end-users or caregivers, and broader user-centered design considerations such as user experience, trust, and long-term autonomy remain unaddressed.

6.3 Future Research

Several directions could advance robot-assisted feeding toward a portable, feedback-driven, and user-centered system capable of operating reliably in diverse real-world environments. Future systems may incorporate continuous perception, enabling the robot to monitor the state of the meal and adapt food-acquisition actions in real time. In addition, always-on audio interfaces would allow for mid-meal verbal corrections and dynamic trajectory adjustments, supporting interactive and personalized feeding behavior that is particularly appealing to users with limited mobility.

A promising direction is the development of physical AI systems that are inherently grounded in the robot’s physical constraints and continuous sensory feedback. Such models could integrate multimodal perception, such as video streams, proprioceptive information, and tactile or force feedback, with real-time inference and action-conditioned reasoning. These systems would maintain an ongoing understanding of the scene and refine decisions as conditions evolve, enabling robust context-adaptive manipulation in unstructured environments.

REFERENCES

- [1] “Meet Obi: the Adaptive Eating Device - Eat Independently!” <https://meetobi.com/>, (accessed 2025-12-12).
- [2] A. Nanavati, V. Ranganeni, and M. Cakmak, “Physically Assistive Robots: A Systematic Review of Mobile and Manipulator Robots That Physically Assist People with Disabilities,” *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 7, pp. 123–147, Jul. 2024.
- [3] T. Bhattacharjee, E. K. Gordon, R. Scalise, M. E. Cabrera, A. Caspi, M. Cakmak, and S. S. Srinivasa, “Is more autonomy always better? Exploring preferences of users with mobility impairments in robot-assisted feeding,” in *15th Annual ACM/IEEE International Conference on Human Robot Interaction, HRI*, 2020, pp. 181–190.
- [4] H. K. Kim *et al.*, “Development of a Comprehensive Design Guideline to Evaluate the User Experiences of Meal-Assistance Robots considering Human-Machine Social Interactions,” *International Journal of Human-Computer Interaction*, vol. 38, no. 17, pp. 1687–1700, 2022.
- [5] A. Nanavati, P. Alves-Oliveira, T. Schrenk, E. K. Gordon, M. Cakmak, and S. S. Srinivasa, “Design Principles for Robot-Assisted Feeding in Social Contexts,” in *Proceedings of the 2023 ACM/IEEE International Conference on Human-Robot Interaction*, Mar. 2023, pp. 24–33.
- [6] D. P. Losey, K. Srinivasan, A. Mandlekar, A. Garg, and D. Sadigh, “Controlling Assistive Robots with Learned Latent Actions,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, May 2020, pp. 378–384.
- [7] A. Nanavati, “Achieving Deployable Autonomy through Customizability and Human-in-the-Loop: A Case Study in Robot-assisted Feeding,” in *19th Annual ACM/IEEE International Conference on Human-Robot Interaction, HRI*, 2024, pp. 136–138.
- [8] D. Park, Y. Hoshi, H. P. Mahajan, H. K. Kim, Z. Erickson, W. A. Rogers, and C. C. Kemp, “Active robot-assisted feeding with a general-purpose mobile manipulator: Design, evaluation, and lessons learned,” *Robotics and Autonomous Systems*, vol. 124, p. 103344, Feb. 2020.

- [9] E. K. Gordon *et al.*, “An Adaptable, Safe, and Portable Robot-Assisted Feeding System,” in *19th Annual ACM/IEEE International Conference on Human-Robot Interaction, HRI*, 2024, pp. 74–76.
- [10] R. K. Jenamani *et al.*, “FEAST: A Flexible Mealtime-Assistance System Towards In-the-Wild Personalization,” arXiv:2506.14968, Jun. 2025.
- [11] J. R. Schultz, A. B. Slifkin, H. Yu, and E. M. Scheerer, “Proof-of-Concept: A Hands-Free Interface for Robot-Assisted Self-Feeding,” in *2022 International Conference on Rehabilitation Robotics (ICORR)*, Jul. 2022, pp. 1–6.
- [12] C. Leblond-Menard and S. Achiche, “Non-Intrusive Real Time Eye Tracking Using Facial Alignment for Assistive Technologies,” *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 31, pp. 954–961, 2023.
- [13] S. Jain, A. Farshchiansadegh, A. Broad, F. Abdollahi, F. Mussa-Ivaldi, and B. Argall, “Assistive robotic manipulation through shared autonomy and a Body-Machine Interface,” in *2015 IEEE International Conference on Rehabilitation Robotics (ICORR)*, Aug. 2015, pp. 526–531.
- [14] S. Stalljann, L. Wöhle, J. Schäfer, and M. Gebhard, “Performance Analysis of a Head and Eye Motion-Based Control Interface for Assistive Robots,” *Sensors*, vol. 20, no. 24, p. 7162, Jan. 2020.
- [15] Z. Erickson, Z. Karachiwalla, C. Majidi, H. Admoni, A. Padmanabha, J. Yuan, and J. Gupta, “VoicePilot: Harnessing LLMs as Speech Interfaces for Physically Assistive Robots,” arXiv.2403.04066, 2024.
- [16] C.-Y. Chiu, A. K. Singh, Y.-K. Wang, J.-T. King, and C.-T. Lin, “A wireless steady state visually evoked potential-based BCI eating assistive system,” in *2017 International Joint Conference on Neural Networks, IJCNN*, vol. 2017-May, 2017, pp. 3003–3007.
- [17] D. Burkolter, B. Weyers, A. Kluge, and W. Luther, “Customization of user interfaces to reduce errors and enhance user acceptance,” *Applied Ergonomics*, vol. 45, no. 2, Part B, pp. 346–353, Mar. 2014.
- [18] R. K. Jenamani, P. Sundaresan, M. Sakr, T. Bhattacharjee, and D. Sadigh, “FLAIR: Feeding via Long-horizon Acquisition of Realistic dishes,” arXiv:2309.07561, Jul. 2024.
- [19] F. Dai, Z. Pei, P. Wang, and W. Chen, “Planning and Control for Robot-Assisted Feeding System Towards the Disabled,” *Journal of Shanghai Jiaotong University (Science)*, Oct. 2024.

- [20] J. Wang *et al.*, “Large language models for robotics: Opportunities, challenges, and perspectives,” *Journal of Automation and Intelligence*, vol. 4, no. 1, pp. 52–64, Mar. 2025.
- [21] S. Wang, M. Han, Z. Jiao, Z. Zhang, Y. N. Wu, S.-C. Zhu, and H. Liu, “LLM3: Large Language Model-based Task and Motion Planning with Motion Failure Reasoning,” in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2024, pp. 12 086–12 092.
- [22] Y. Chen, J. Arkin, C. Dawson, Y. Zhang, N. Roy, and C. Fan, “AutoTAMP: Autoregressive Task and Motion Planning with LLMs as Translators and Checkers,” in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, May 2024, pp. 6695–6702.
- [23] B. Quartey, E. Rosen, S. Tellex, and G. Konidaris, “Verifiably Following Complex Robot Instructions with Foundation Models,” in *2025 IEEE International Conference on Robotics and Automation (ICRA)*, May 2025, pp. 1–8.
- [24] C. H. Song, B. M. Sadler, J. Wu, W.-L. Chao, C. Washington, and Y. Su, “LLM-Planner: Few-Shot Grounded Planning for Embodied Agents with Large Language Models,” in *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2023, pp. 2986–2997.
- [25] “GPT-4V(ision) system card,” <https://openai.com/index/gpt-4v-system-card/>, Feb. 2024, (accessed 2025-12-12).
- [26] Z. Yang, C. Garrett, D. Fox, T. Lozano-Pérez, and L. P. Kaelbling, “Guiding Long-Horizon Task and Motion Planning with Vision Language Models,” in *2025 IEEE International Conference on Robotics and Automation (ICRA)*, May 2025, pp. 16 847–16 853.
- [27] T. M. Bui, Y. Kim, J. Moon, M.-Y. Cho, M. Seo, and D. Shin, “Development of a Mobile Assistive Robot for Daily Living Support,” in *2025 22nd International Conference on Ubiquitous Robots (UR)*, Jun. 2025, pp. 523–528.
- [28] “Will generative AI transform robotics?” *Nature Machine Intelligence*, vol. 6, no. 6, pp. 579–579, Jun. 2024.
- [29] A. Bhaskar, R. Liu, V. D. Sharma, G. Shi, and P. Tokekar, “LAVA: Long-horizon Visual Action based Food Acquisition,” in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2024, pp. 8929–8935.

- [30] B. Zitkovich *et al.*, “RT-2: Vision-Language-Action Models Transfer Web Knowledge to Robotic Control,” in *Proceedings of The 7th Conference on Robot Learning*, Dec. 2023, pp. 2165–2183.
- [31] M. J. Kim *et al.*, “OpenVLA: An Open-Source Vision-Language-Action Model,” arXiv:2406.09246, Jun. 2024.
- [32] S. Dhuliawala, M. Komeili, J. Xu, R. Raileanu, X. Li, A. Celikyilmaz, and J. Weston, “Chain-of-Verification Reduces Hallucination in Large Language Models,” arXiv:2309.11495, Sep. 2023.
- [33] A. Mei, G.-N. Zhu, H. Zhang, and Z. Gan, “ReplanVLM: Replanning Robotic Tasks With Visual Language Models,” *IEEE Robotics and Automation Letters*, vol. 9, no. 11, pp. 10 201–10 208, Nov. 2024.
- [34] H. Liu, S. Guo, P. Mai, J. Cao, H. Li, and J. Ma, “RoboDexVLM: Visual Language Model-Enabled Task Planning and Motion Control for Dexterous Robot Manipulation,” arXiv:2503.01616, Mar. 2025.
- [35] M. Brienza, F. Argenziano, V. Suriani, D. D. Bloisi, and D. Nardi, “Multi-agent Planning using Visual Language Models,” arXiv:2408.05478, 2024.
- [36] D. Gallenberger, T. Bhattacharjee, Y. Kim, and S. S. Srinivasa, “Transfer Depends on Acquisition: Analyzing Manipulation Strategies for Robotic Feeding,” in *2019 14th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, Mar. 2019, pp. 267–276.
- [37] K. He, G. Gkioxari, P. Dollár, and R. Girshick, “Mask R-CNN,” in *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017, pp. 2980–2988.
- [38] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.
- [39] G. Descôteaux, “Système autonome d’assistance au repas pour les personnes avec un handicap du haut du corps,” Master’s thesis, Polytechnique Montréal, Aug. 2020.
- [40] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, “Focal Loss for Dense Object Detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 42, no. 2, pp. 318–327, Feb. 2020.

- [41] R. Feng *et al.*, “Robot-Assisted Feeding: Generalizing Skewering Strategies Across Food Items on a Plate,” in *17th International Symposium of Robotics Research, ISRR*, vol. 20 SPAR, 2019, pp. 427–442.
- [42] Y. Fan, L. Zhang, C. Zheng, Y. Zu, K. Wang, and X. Wang, “Real-time and accurate model of instance segmentation of foods,” *Journal of Real-Time Image Processing*, vol. 21, no. 3, 2024.
- [43] S. Liu *et al.*, “Grounding DINO: Marrying DINO with Grounded Pre-Training for Open-Set Object Detection,” arXiv:2303.05499, Jul. 2024.
- [44] A. Kirillov *et al.*, “Segment Anything,” in *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2023, pp. 3992–4003.
- [45] J. Zhang, J. Huang, S. Jin, and S. Lu, “Vision-Language Models for Vision Tasks: A Survey,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, no. 8, pp. 5625–5644, Aug. 2024.
- [46] E. K. Gordon, A. Nanavati, R. Challa, B. H. Zhu, T. A. K. Faulkner, and S. Srinivasa, “Towards general single-utensil food acquisition with human-informed actions,” in *Proceedings of the Conference on Robot Learning (CoRL)*, Dec. 2023, pp. 2414–2428.
- [47] R. Liu, A. Bhaskar, and P. Tokekar, “Adaptive Visual Imitation Learning for Robotic Assisted Feeding Across Varied Bowl Configurations and Food Types,” arXiv:2403.12891, Mar. 2024.
- [48] J. Grannen, Y. Wu, S. Belkhale, and D. Sadigh, “Learning Bimanual Scooping Policies for Food Acquisition,” in *6th Conference on Robot Learning, CoRL*, vol. 205, 2023, pp. 1510–1519.
- [49] P. Sundaresan, J. Wu, and D. Sadigh, “Learning Sequential Acquisition Policies for Robot-Assisted Feeding,” arXiv:2309.05197, 2023.
- [50] T. Mashrur, Z. Ghulam, G. French, and H. A. Abdullah, “Assistive feeding robot for upper limb impairment—Testing and validation,” *International Journal of Advanced Robotic Systems*, vol. 20, no. 4, Jul. 2023.
- [51] E. K. Gordon, X. Meng, T. Bhattacharjee, M. Barnes, and S. S. Srinivasa, “Adaptive Robot-Assisted Feeding: An Online Learning Framework for Acquiring Previously Unseen Food Items,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2020, pp. 9659–9666.

- [52] P. Sundaresan, S. Belkhale, and D. Sadigh, “Learning Visuo-Haptic Skewering Strategies for Robot-Assisted Feeding,” arXiv:2211.14648, Nov. 2022.
- [53] S. Shrivastava, N. P. Garg, C. Tang, J. AnneYow, W. T. Ang, and W. L. Leong, “Using touch sensor and vision feedback to adapt skewering strategy for robust assistive feeding,” in *16th International Convention on Rehabilitation Engineering and Assistive Technology, i-CREAtE*, 2023, pp. 11–15.
- [54] E. K. Gordon, S. Roychowdhury, T. Bhattacharjee, K. Jamieson, and S. S. Srinivasa, “Leveraging Post Hoc Context for Faster Learning in Bandit Settings with Applications in Robot-Assisted Feeding,” in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, May 2021, pp. 10 528–10 535.
- [55] R. Liu, Z. Mahammad, A. Bhaskar, and P. Tokekar, “IMRL: Integrating Visual, Physical, Temporal, and Geometric Representations for Enhanced Food Acquisition,” arXiv:2409.12092, 2024.
- [56] R. K. Jenamani, D. Stabile, Z. Liu, A. Anwar, K. Dimitropoulou, and T. Bhattacharjee, “Feel the Bite: Robot-Assisted Inside-Mouth Bite Transfer using Robust Mouth Perception and Physical Interaction-Aware Control,” in *2024 19th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, Mar. 2024, pp. 313–322.
- [57] L. Shaikewitz, Y. Wu, S. Belkhale, J. Grannen, P. Sundaresan, and D. Sadigh, “In-Mouth Robotic Bite Transfer with Visual and Haptic Sensing,” in *2023 IEEE International Conference on Robotics and Automation (ICRA)*, Jun. 2023, pp. 9885–9895.
- [58] M. Pascher, K. Zinta, and J. Gerken, “Exploring of Discrete and Continuous Input Control for AI-enhanced Assistive Robotic Arms,” in *19th Annual ACM/IEEE International Conference on Human-Robot Interaction, HRI*, 2024, pp. 828–832.
- [59] *ISO 13482:2014 — Robots and robotic devices — Safety requirements for personal care robots*, Standard, International Organization for Standardization Std., 2014, (accessed 2025-12-12).
- [60] N. Carion *et al.*, “SAM 3: Segment Anything with Concepts,” arXiv:2511.16719, Nov. 2025.
- [61] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. Mcleavey, and I. Sutskever, “Robust speech recognition via large-scale weak supervision,” in *Proceedings of the 40th International Conference on Machine Learning*, vol. 202, 23–29 Jul 2023, pp. 28 492–28 518.

- [62] H. Liu, C. Li, Q. Wu, and Y. J. Lee, “Visual Instruction Tuning,” arXiv:2304.08485, Dec. 2023.
- [63] “GPT-5.1: A smarter, more conversational ChatGPT,” <https://openai.com/index/gpt-5-1/>, Nov. 2025, (accessed 2025-12-12).
- [64] N. Alshurafa, S. Zhang, C. Romano, H. Zhang, A. F. Pfammatter, and A. W. Lin, “Association of number of bites and eating speed with energy intake: Wearable technology results under free-living conditions,” *Appetite*, vol. 167, p. 105653, Dec. 2021.

APPENDIX A VLM PLANNING PROMPT USED IN SIMULATION-BASED EVALUATION

The following prompt is used by the VLM-based meal planner in the 25-scenario simulation evaluation, where all 7 manipulation primitives are available for planning. The variables {food_items_list}, {user_input}, and {bite_history_summary} are populated dynamically for each planning request.

VLM Prompt for Simulation Evaluation

You are a robotic feeding assistant. Your role is to determine the next feeding action sequence for a robotic arm holding a fork, with the goal of feeding the user. Use the step-by-step process outlined below, ensuring that each step includes visible, reasoned logic and that all constraints are strictly followed.

AVAILABLE MOTION PRIMITIVES (7):

- Pre-acquisition:

- push: used to clear an occlusion that blocks access to another item.
 - Trajectory: Fork descends vertically and moves laterally to shove an object.
 - Constraint: Requires free space on the plate to move the item into; cannot push into other items or off the plate.
- group: used to scrape scattered items into a dense pile or cluster suitable for acquisition.
 - Trajectory: Fork descends sideways to the plate bottom and moves laterally to scrape and gather food.
 - Constraint: The scraping path connecting the scattered items must be clear of obstacles.
- cut: used to subdivide an item into a smaller piece.
 - Trajectory: Fork descends sideways to the item and applies vertical force using the side of the tines.
 - Constraint: Target must be soft enough to be cuttable with fork pressure alone.

- Acquisition:

- skewer: used for solid items that hold shape and can be pierced.
 - Trajectory: Fork descends vertically to pierce the target.

- Constraint: Requires a stable bite-sized target with an exposed top surface area (free of obstacles) at least the width of the fork tines to allow for a vertical strike.
- scoop: used for items that are granular, semi-solid, soft, or that do not hold shape.
 - Trajectory: Fork approaches horizontally and executes a forward-sliding scoop motion (descending then ascending arc) along the plate surface.
 - Constraint: Requires a clear runway (no obstacles) extending longer than the fork head to accommodate the forward travel distance. Requires a food density high enough for the fork to retain a full bite (tines cannot retain liquids).
- twirl: used for long noodles only.
 - Trajectory: Fork descends vertically into the densest part of the noodle cluster and rotates around its axis.
 - Constraint: Requires a concentrated high-density cluster of strands within the fork's rotation radius. Sparse or isolated strands will not produce a sufficient bite.
- Post-acquisition:
 - dip: used to dip the acquired item into a sauce. In that case, the item parameter is the SAUCE/DIP (e.g., dip('ketchup')), NOT the food being dipped.
 - Trajectory: Fork descends vertically into the sauce container or pile.
 - Constraint: Requires a reachable sauce or dip container; the path to the sauce must be clear.

INPUTS:

- Food items identified at meal start (fixed labels): `food_items_list`
(You MUST use only these labels. Do NOT invent new item names.)
- Instruction (preferences): `user_input`
- Bite history summary: `bite_history_summary`

TASK: You MUST reason through all 4 steps before giving your final decision.
Produce all reasoning visible text.

STEP 1 - Identify visible items

- List which food items from the meal-start list are STILL visible in the image. Some items from the meal-start list may have all been consumed.
- Assign each visible item its history value ("not yet fed" is highest priority).

STEP 2 - Choose the next item

Follow these decision rules in strict order:

1. MUST choose ONLY from visible items (from Step 1).
2. NEVER select an item the user dislikes.
3. MUST follow explicit user preferences.
4. If no specific ordering is required by user:
 - Choose any "not yet fed" item first.
 - If all items have been fed: choose the item with the highest history value.
5. If multiple valid items remain: choose at random.

STEP 3 - Determine the action sequence

Determine the correct action sequence to acquire the target item by strictly applying the motion primitive definitions and physical constraints provided above. If no sequence of actions can accomplish the task, or if you cannot satisfy the user's request, you MUST output an error with an explanation.

STEP 4 - Output decision

Output your final decision EXACTLY in this format:

DECISION:[action('item'), action('item'), ...] or DECISION:error("reason")

if no valid solution. The DECISION must appear on the last line only, with no extra text after it.

APPENDIX B MEAL SCENARIO DATASET FOR SIMULATION-BASED PLANNING EVALUATION

This appendix presents the complete dataset of 25 meal scenarios used to evaluate VLM-based next-bite planning under the full seven primitive actions (push, group, cut, skewer, scoop, twirl, dip). Each scenario includes the initial food items, the user instruction, the bite history, the plate image, and ground-truth action sequence used for accuracy scoring.

All plate images were generated using OpenAI’s image generation tools, and ground truth was produced by three colleagues (not trained raters) following the same prioritization rules provided to the VLM. The experimenter did not participate in the ground-truth annotation.

Scenario 1 — No preference and no history

Food items identified at meal start: chicken, rice, broccoli, carrot

User instruction: none

Bite history: none



Ground truth: skewer(chicken) OR scoop(rice) OR skewer(broccoli) OR skewer(carrot)

Scenario 2 — One dislike and one item already fed

Food items identified at meal start: tofu, quinoa, bell pepper, mushroom

Instruction: “I don’t like bell pepper.”

Bite history: skewer(tofu)



Ground truth: scoop(quinoa) OR skewer(mushroom)

Scenario 3 — Only risotto remains

Food items identified at meal start: risotto, cherry tomato

Instruction: none

Bite history: skewer(cherry tomato), skewer(cherry tomato), skewer(cherry tomato)



Ground truth: scoop(risotto)

Scenario 4 — Soft dislike

Food items identified at meal start: pork, rice, broccoli, cauliflower

Instruction: “I prefer not to eat the broccoli.”

Bite history: skewer(pork), scoop(rice), skewer(cauliflower)



Ground truth: skewer(pork)

Scenario 5 — Mushrooms first but mushrooms gone

Food items identified at meal start: sausage, penne, mushroom

Instruction: “Mushrooms first.”

Bite history: skewer(mushroom), skewer(mushroom), skewer(sausage), skewer(penne)



Ground truth: skewer(sausage)

Scenario 6 — Disliked item and mistake in history

Food items identified at meal start: chicken, couscous, cauliflower, red pepper

Instruction: “I don’t like cauliflower.”

Bite history: skewer(chicken), scoop(couscous), skewer(chicken), scoop(couscous)



Ground truth: skewer(red pepper)

Scenario 7 — Scattered spaghetti

Food items identified at meal start: spaghetti

Instruction: “Feed me everything.”

Bite history: twirl(spaghetti)



Ground truth: group(spaghetti) AND twirl(spaghetti)

Scenario 8 — Vegetables first

Food items identified at meal start: steak, rice, broccoli, carrot

Instruction: “I want to eat all the vegetables first.”

Bite history: skewer(carrot), skewer(broccoli)



Ground truth: skewer(carrot)

Scenario 9 — Mistake in history

Food items identified at meal start: yogurt, strawberry, blackberry

Instruction: none

Bite history: scoop(yogurt), skewer(strawberry), skewer(blackberry), skewer(strawberry)



Ground truth: scoop(yogurt)

Scenario 10 — Occluded rice

Food items identified at meal start: salmon, rice, asparagus

Instruction: “I like everything.”

Bite history: skewer(asparagus), skewer(salmon)



Ground truth: push(salmon) AND scoop (rice)

Scenario 11 — Priority with exclusion

Food items identified at meal start: zucchini, bell pepper, chicken

Instruction: “Let’s tackle the vegetables first, but don’t feed me bell pepper.”

Bite history: skewer(zucchini)



Ground truth: skewer(zucchini)

Scenario 12 — Chicken with gravy

Food items identified at meal start: chicken, potato, green bean, gravy

Instruction: “I like my chicken with gravy.”

Bite history: skewer(potato), skewer(green bean)



Ground truth: skewer(chicken) AND dip(gravy)

Scenario 13 — Occluded spaghetti

Food items identified at meal start: spaghetti, meatball

Instruction: “I like everything.”

Bite history: skewer(meatball)



Ground truth: twirl(spaghetti)

Scenario 14 — Healthy stuff first

Food items identified at meal start: chicken nugget, cucumber, fries

Instruction: “Give me the healthy stuff first.”

Bite history: none



Ground truth: skewer(cucumber)

Scenario 15 — Fruits and chocolate dip

Food items identified at meal start: strawberry, banana, chocolate dip

Instruction: “I like my banana with chocolate and the strawberries by themselves.”

Bite history: skewer(strawberry)



Ground truth: cut(banana) AND skewer(banana) AND dip(chocolate dip)

Scenario 16 — Scattered fried rice

Food items identified at meal start: fried rice

Instruction: none

Bite history: scoop(fried rice)



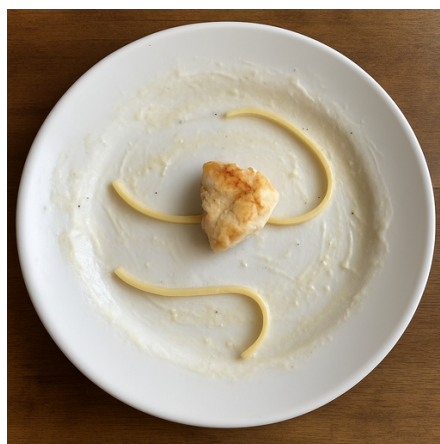
Ground truth: group(fried rice) AND scoop(fried rice)

Scenario 17 — Noodles with occlusion

Food items identified at meal start: noodles, chicken

Instruction: none

Bite history: skewer(chicken)



Ground truth: push(chicken) AND group(noodles) AND twirl(noodles)

Scenario 18 — Rice last

Food items identified at meal start: rice, sausage, zucchini, green bean

Instruction: “I want to eat the rice last.”

Bite history: skewer(zucchini), skewer(sausage), skewer(green bean)



Ground truth: skewer(zucchini)

Scenario 19 — Scattered quinoa

Food items identified at meal start: tofu, brussel sprouts, quinoa

Instruction: “I like everything.”

Bite history: skewer(brussel sprouts), skewer(tofu)



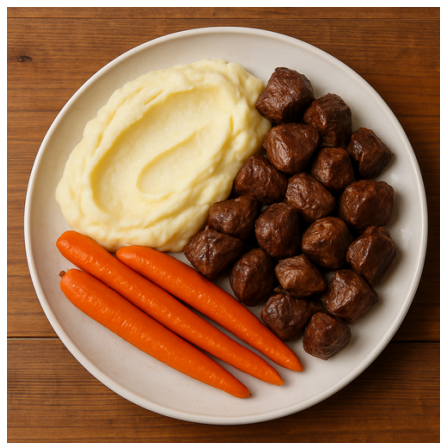
Ground truth: group(quinoa) AND scoop(quinoa)

Scenario 20 — Full carrots

Food items identified at meal start: steak, mashed potatoes, carrots

Instruction: none

Bite history: skewer(steak), scoop(mashed potatoes)



Ground truth: cut(carrot) AND skewer(carrot)

Scenario 21 — Instruction requests nonexistent food

Food items identified at meal start: couscous, roasted chicken, zucchini

Instruction: "I want the lasagna."

Bite history: none



Ground truth: error("requested item not present on the plate")

Scenario 22 — Peas

Food items identified at meal start: rice, turkey, peas

Instruction: none

Bite history: scoop(peas), skewer(turkey), scoop(rice)



Ground truth: scoop(peas)

Scenario 23 — Only disliked item left

Food items identified at meal start: shrimp, rice, green pepper

Instruction: “Feed me everything, I don’t like green peppers.”

Bite history: skewer(shrimp), scoop(rice)



Ground truth: error("only disliked items remain")

Scenario 24 — Empty plate

Food items identified at meal start: salmon, mashed potatoes, mushroom

Instruction: “Feed me the whole plate.”

Bite history: skewer(salmon), skewer(mushroom), scoop(mashed potatoes)



Ground truth: error("empty plate")

Scenario 25 — Soup

Food items identified at meal start: soup

Instruction: “Feed me whatever.”

Bite history: none



Ground truth: error("cannot handle this item with a fork")