AUTONOMOUS MOBILE NAVIGATION USING MULTIMODAL AGENTS AND NATURAL LANGUAGE COMMANDS

Ms. L. Durgdevi
Department of Information Technology
Assistant Professor
Sri Manakula Vinayagar Engineering College
Puducherry, India
durgadevime.ap@gmail.com

Dinesh Kumar C
Department of Information Technology
Sri Manakula Vinayagar Engineering College
Puducherry, India
dinesh210805@gmail.com

Shankar R S
Department of Information Technology
Sri Manakula Vinayagar Engineering College
Puducherry, India
shankarrs2307@gmail.com

Nithish G
Department of Information Technology
Sri Manakula Vinayagar Engineering College
Puducherry, India
nithishganapathy1234@gmail.com

ABSTRACT

The limitations of voice-driven smartphones continue to be defined by their rigid and unchanging templates used to give users commands to perform on different applications, as well as lack of the ability to recognize and interact dynamically with various types of user interface controls. These limitations are barriers that limit voice-controlled smartphones from providing users an effective means of achieving hands free automation in several areas including accessibility, multitasking, and eyes free use. This paper introduces AURA (Autonomous UI Recognition & Action), which is a multimodal agent system that can recognize and control Android devices using natural language voice commands and has been designed to integrate the intelligent capabilities of perception, reasoning, and context aware execution into one coherent architectural framework. AURA uses three major innovations: (1) Perception Cascade - Three Layer Perceptual Pipeline: Android UI Tree Accessibility Analysis, YOLOv8 Computer Vision with Set-of-Marks Annotation, and VLM-Powered Semantic Selection: Fast Path Resolution For Native Elements - This allows AURA to rapidly resolve native UI elements and escalate to vision processing if necessary, and eliminate Coordinate Hallucinations by limiting VLM's to selection tasks, and not generation tasks. (2) Universal Agent: Implementing ReAct Reasoning Loops With Goal Decomposition and Intelligent 5-Level Failure Recovery: Using Adaptive Execution To Handle Unexpected UI States And Automatically Escalate Through Retry Strategies Before User Intervention: Instead of using rigid and predictable Template-Driven Navigation, AURA uses ReAct Reasoning Loops With Goal Decomposition and Intelligent 5-Level Failure Recovery to achieve adaptive execution and provide users with effective hands free automation. (3) Hybrid Model Orchestration: LangGraph Routing Lightweight Groq Llama Models (3.3 70B For Intent Parsing, 3.1 8B For Response Generation) To Speed-Critical Components And Gemini 2.5 Flash To Reasoning-Intensive Vision And Planning Tasks: Optimizing Latency-Accuracy Trade Off Across Thirteen Interconnected Processing Nodes: AURA also integrates specialized agents (Commander, Universal Agent, Responder, Screen Reader, Visual Locator) through Conditional Routing With Parallel Execution Pipelines and Stateful Error Management. Therefore, this research demonstrates that holistically orchestrating agents that integrate perception cascading, adaptive reasoning, and strategically allocating models produces reliable automation across native apps, web content, and custom interfaces without the need for per-app templates or predetermined interaction scripts.

Keywords: Mobile automation, Vision-Language Models, Natural language processing, Android accessibility, Multimodal agents, LangGraph, Computer vision, Adaptive execution, Hands-free control

I. INTRODUCTION

Modern smartphones represent a widespread platform for ubiquitous computing, but their dominant interface design as a touch-screen based model represents a barrier

for the users and also presents limitations in use while performing activities in a "hands-free" mode i.e., during driving, cooking or other multitasking scenarios. Commercially available voice-controlled systems, including Google Assistant and Siri, have advanced voice control capabilities but these are inherently limited by pre-defined command structures, limited app support and lack of ability to adjust to changing or evolving user interface designs.

There are three primary deficiencies in today's voice-controlled systems: (1) The Limited Ability to Understand the User Interface The inability to understand the visual context beyond predefined APIs provided by an application's developer and as a result failing when presented with unlabeled elements, custom interfaces or dynamically created content; (2) The Inability to Create a Dynamic Map of Commands A reliance on static maps between commands and actions which break when an application updates its UI or introduces a new feature; and (3) The Lack of Adaptive Reasoning The lack of intelligent planning and error-recovery methods leading to the failure to accomplish a given task when an unexpected state is encountered or obstacle is encountered. Recent work on integrating large language models (LLMs) and vision-language models (VLMs) into mobile automation systems have demonstrated improved natural language understanding through recent research. However, all of the mobile automation systems referenced in this section Voicify [1], GPTVoiceTasker [3], and VisionTasker [4] present three fundamental architectural shortcomings: (1) Coordinate Generation VLMs are instructed to produce the coordinates for a tap directly on the screen and as a result, the VLM produces an incorrect prediction approximately 50-200 pixels from the correct location on the screen resulting in missed taps and selecting the wrong element ; (2) Unidirectional Perception These systems utilize either accessibility APIs (which fail to provide accurate results when dealing with custom UIs) or vision models (resulting in unnecessary latency when utilizing native elements) without employing intelligent cascading strategies; and (3) Template-Based Execution Predefined action sequences that are incapable of adapting to unexpected states within the UI or recovering from failure in an intelligent manner.

1.1 Research Motivation and Contributions

This work presents AURA, a fundamentally redesigned mobile automation architecture that addresses existing limitations through three integrated innovations rather than incremental improvements to prior approaches:

1. **3-Layer Perception Pipeline:** A cascading architecture that attempts fast UI Tree matching (Layer 1), escalates to YOLOv8 computer vision detection with Set-of-Marks annotation (Layer 2), and employs VLM semantic selection from pre-detected candidates (Layer 3) critically ensuring VLMs never generate coordinates directly.
2. **Universal Agent with ReAct Reasoning:** A reasoning-based execution engine that replaces template-driven navigation with adaptive Observe-Think-Act-Verify loops, enabling intelligent goal decomposition, context-aware action selection, and automatic failure recovery.
3. **Hybrid Model Architecture:** Strategic model routing that leverages Groq Llama 3.3 70B for speed-critical tasks (intent parsing, response generation) and Gemini 2.5 Flash for reasoning-intensive operations (vision analysis, planning), optimizing both latency and accuracy.

The specific contributions of this work are:

- A novel cascading perception architecture that strategically combines accessibility APIs, computer vision, and vision-language models achieving both speed and reliability while eliminating coordinate hallucination as a design constraint rather than a post-hoc fix
- A reasoning-based Universal Agent employing ReAct loops for adaptive execution, replacing brittle template-driven approaches with context-aware planning and intelligent failure recovery
- Strategic hybrid model orchestration through LangGraph, enabling task-specific model routing, parallel execution pipelines, and stateful error management

II. RELATED WORK

[1] Voicify (Vu et al., 2023) - Vu et al. (2023) introduced Voicify, a system that addresses the limitations of current smartphone voice assistants when managing complex app interactions, dynamic user interfaces, and full device control. Unlike Siri, Google Assistant, or Google Voice Access, which provide only limited features and often need manual registration, Voicify allows for natural interactions with on-screen elements and nested features. Its structure has three modules. The first collects data, the second interprets commands given in natural language, and the third manages conversations. Data Collection gathers app components using Manifest.xml and the Android Accessibility Service. The Command Parser employs VoicifyLang, a BERT-LSTM Seq2Seq model enhanced with SCFG-based canonical utterances, paraphrasing, and the RICO dataset to create semantic representations. The Dialogue Manager translates these representations into actions that the Accessibility API or intents can execute,

enabling multi-command execution with immediate feedback. By using on-device data and 10,000 RICO UI screenshots along with synthetic paraphrases, Voicify effectively handles unseen elements. Evaluation across various apps and dynamic UIs assessed coverage, parser strength, execution accuracy, and usability. The results show direct in-app activation, unlabeled UI interaction, adaptable handling of dynamic screens, and improved understanding of natural language. However, challenges persist related to backend delays, limited support for highly customized UIs, and gaps in training based on paraphrases.

[2] XUI (Leiva et al., 2022) - Mobile interaction research struggles to describe UI screenshots, which often contain a lot of text, complex layouts, and functional elements. Traditional captioning models create general sentences or miss the UI meaning. OCR does not understand context, and human annotation is expensive and hard to scale. To tackle these problems, Leiva et al. proposed XUI, a structured captioning system inspired by cognitive psychology, which produces three types of output: caption, simple description, and detailed description. Its process combines topic categorization using dual convolutional networks, element saliency prediction with Grad-CAM a method for visualizing important features on wireframe layouts, and template-driven language generation that adds salience and thematic context. XUI was tested on the Enrico dataset, which includes 1,460 UI screenshots across 20 categories and over 30,000 annotated elements. It achieved 85.5% Top-1 accuracy in theme classification and showed solid saliency detection. User studies showed that its descriptions were nearly as informative as human captions and provided good action guidance. Limitations include template rigidity, focusing on single screens, and restricted category coverage. However, its strengths include interpretability, a structured design, and better accessibility support

[3] GPTVoiceTasker (Vu et al., 2024) - GPTVoiceTasker solves the usability issues faced by today's voice assistants, which often work with strict, step-by-step commands and fixed models. Unlike JustSpeak and AutoVCI, which handle only single-step tasks, Minh Duc Vu et al. proposed GPTVoiceTasker, which understands user intent and on-screen context to carry out multi-step actions. The system consists of three modules: Prompt Engineering breaks down high-level requests and clarifies uncertainties; On-Screen Interaction interprets user interface components and performs actions like tapping, scrolling, and typing on dynamic content; and Usage-Based Execution generalizes previous task sequences for flexible automation. This tool is built as a Java Android app that uses AccessibilityService and GPT-4 while keeping data private by storing it locally. Evaluation included 278 commands from 31 users, covering 150 tasks across five app categories. A user study with 18 participants showed 84.7% intent accuracy, 82% success for familiar tasks, 72% for parameterized tasks, and strong usability with a System Usability Scale (SUS) score of 79.86. Users praised its understanding of natural language, ability to automate multiple steps, adaptability to user interfaces, and competence in recovering from errors. Some limitations are its reliance on previous task exposure, occasional sensitivity to changing UI elements, and delays caused by network issues or pop-ups.

[4] VisionTasker (Song et al., 2024) - Vision Tasker improves mobile automation by combining UI understanding with large language model (LLM)–driven planning, overcoming limitations of Programming by Demonstration (PBD) and view-hierarchy methods that often struggle with interface changes, accessibility gaps, and repetitive workflows. Song et al. proposed VisionTasker, which operates in two stages: vision-based UI comprehension extracts elements from screenshots, groups them into semantic blocks, and generates natural language descriptions without relying on view hierarchies; then LLM-based planning sequences actions using structured prompts that incorporate role, task, history, and UI semantics. A PBD mechanism integrates user demonstrations for complex workflows, while an execution module translates outputs into device commands, handling long screenshots and ADB interactions. Evaluation on 136 screens (31 apps) for UI understanding and 147 real-world tasks across multiple domains showed VisionTasker outperforming hierarchy-based methods, achieving accurate UI parsing and near-human task automation, with PBD improving planning efficiency. Strengths include robust UI interpretation, stepwise LLM planning, PBD integration, token efficiency, and generalizability to other LLMs or platforms. Limitations include dependence on screenshots, challenges with subtle UI cues, approximately 12-second latency, overlooked elements, limited parameter input, and partial reliance on PBD for accuracy.

[5] AutoVCI (Pan et al., 2022) - AutoVCI aims to make it easier to create voice command interfaces (VCIs) on smartphones. Traditionally, this process takes a lot of manual work and technical skills. Current methods depend on extensive data labeling, complicated NLP pipelines, and GUI-based voice systems that do not scale well as tasks grow or interactions become more intricate. Pan et al. proposed AutoVCI, which tackles these problems through four phases: generation of VCI, intent recognition, task execution with identification of parameters, and semantic accumulation. It records touch operations, identifies parameters, calculates semantic vectors using BERT, and executes commands through the Android Accessibility Service. Intent recognition uses template matching, vector similarity, and decision trees, while semantic accumulation

updates templates and vectors continuously to reduce dialogue rounds. In tests with 45 tasks across 11 apps, involving 640 commands from 16 users and 536 commands from 67 users, AutoVCI reached a 98.4% success rate, showing greater efficiency and needing fewer confirmation requests. Its main strengths are automated VCI generation, understanding of semantics, continuous self-improvement, privacy protection, and the ability to work across different platforms. However, there are limitations, including speed of semantic accumulation, the need for extra interactions, sensitivity to app updates or user errors, and cold-start issues.

III. PROPOSED METHOD

In the proposed system for autonomous mobile device automation, LangGraph workflow orchestration and Multi-Agent coordination algorithms play a pivotal role in the execution process. LangGraph and Multi-Agent architectures are specialized orchestration frameworks designed to effectively capture and manage sequential interaction tasks, making them particularly well-suited for workflows involving perception dependencies, state management, and adaptive error recovery. By incorporating these advanced algorithms into the system, the model gains the ability to learn and adapt to the dynamic patterns characteristic of mobile user interfaces over time. LangGraph, with its graph-based connections, can retain information from previous workflow steps, allowing the model to capture long-range dependencies within the sequential processing phases encompassing speech-to-text conversion, intent classification, perception analysis, action execution, and response generation. Multi-Agent networks, on the other hand, excel in delegating and coordinating tasks across specialized roles Commander for intent parsing, Universal Agent for goal-driven execution, Responder for feedback generation, Validator for rule-based checks, Screen Reader for visual understanding, and Visual Locator for coordinate extraction making them effective in recognizing and handling complex requirements indicative of diverse automation needs. By leveraging the strengths of LangGraph and Multi-Agent orchestration, the proposed system enhances its ability to accurately coordinate and execute mobile automation tasks, thereby bolstering interaction reliability against evolving challenges in the mobile application landscape.

ARCHITECTURE DIAGRAM

The architecture diagram illustrates the structural components and flow of information within the AURA backend processing system. The FastAPI server receives incoming requests through REST endpoints and WebSocket connections, routing them to specialized handlers for task execution, device management, configuration, and real-time accessibility communication, serving as the primary entry point for all external communications. The Orchestration Layer manages workflow execution through a LangGraph state machine comprising 13 interconnected nodes: STT, Parse Intent, Perception, Validate Intent, Parallel Processing, Analyze UI, Create Plan, Execute, Universal Agent, Decompose Goal, Validate Outcome, Retry Router, and Speak. These nodes are connected through conditional edges that implement intelligent routing based on intent classification, execution status, and error conditions. The Specialized Agents Layer contains domain-specific agents including Commander Agent for intent parsing using Groq Llama 3.3 70B with rule-based classifier fallback, Universal Agent for goal-driven ReAct execution using Gemini 2.5 Flash, Responder Agent for natural language feedback generation, Validator Agent for rule-based intent validation without LLM overhead, Screen Reader Agent for VLM-based visual understanding, and Visual Locator for hybrid coordinate extraction through the 3-layer perception pipeline. The Perception Pipeline processes screen state through three cascading layers UI Tree semantic matching, YOLOv8 computer vision detection with Set-of-Marks annotation, and VLM semantic selection where each layer escalates to the next only when confidence thresholds are not met. This visual representation provides a comprehensive overview of the backend processing design and operation, aiding in understanding its functionality, dependencies, and relationships between different components.

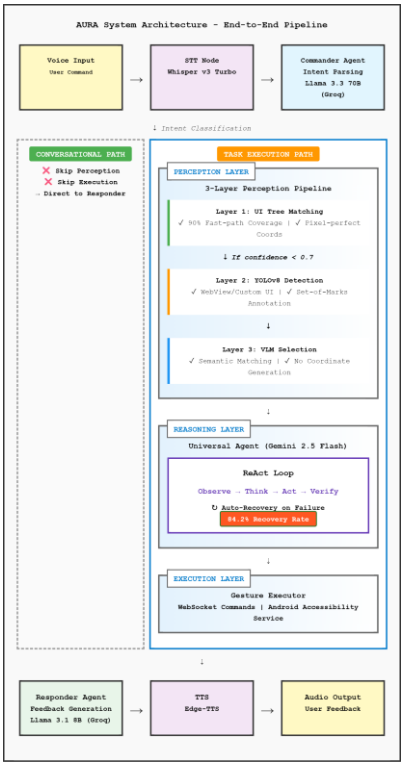

Figure 1: AURA System Architecture - Complete End-to-End Pipeline

A. INPUT PROCESSING AND SPEECH RECOGNITION

The system accepts user commands through two modalities: voice input via real-time WebSocket audio streaming and text input via RESTful API endpoints. For voice-based interaction, audio is streamed as chunked 16kHz mono WAV data from the companion Android application to the FastAPI backend through a persistent WebSocket connection. The server-side STT node employs OpenAI's Whisper-large-v3-turbo model hosted on Groq infrastructure, which provides high-throughput inference for real-time transcription. The transcribed text, along with metadata such as session identifier and input modality flag, is encapsulated into the TaskState a TypedDict structure that persists across all downstream graph nodes and forwarded to the intent classification stage. For text-based commands, the input bypasses the STT node entirely and is injected directly into the TaskState, reducing unnecessary processing overhead.

B. INTENT CLASSIFICATION AND COMMAND PARSING

The Commander Agent implements a three-tier hybrid classification pipeline to parse natural language commands into structured intent representations. The first tier applies a rule-based classifier that performs exact and pattern-based matching against a predefined registry of action types, each defined through an ActionMeta dataclass specifying metadata flags such as needs_ui, needs_coords, needs_perception, is_dangerous, is_conversational, and required_fields. If the rule-based classifier fails to produce a confident match, the second tier engages a fuzzy classifier using Levenshtein distance-based similarity scoring with synonym expansion to handle paraphrased or colloquial commands. Only when both deterministic tiers fail does the system invoke the third tier a Groq-hosted Llama 3.3 70B large language model that parses the command into a structured IntentObject containing action, recipient, content, parameters, and confidence score fields. This cascading design ensures that the majority of common commands are resolved without incurring LLM API costs, while uncommon or ambiguous utterances still receive accurate classification. Formally, the fuzzy similarity score between user command c and a registered action a_i is computed as:

$$s(c, a_i) = \max(lev(c, a_i.name), \max_{y \in syn(a_i)} lev(c, y))$$

where lev(·,·) denotes the normalized Levenshtein similarity ratio and syn(a_i) is the synonym set for action a_i. The overall classification function is then defined as a cascading decision:

$$I(c) = \{ R(c) \text{ if exact match; } F(c) \text{ if } \max_i s(c, a_i) \geq \tau_f; \ L(c) \text{ otherwise } \}$$

where $R(\cdot)$ is the rule-based classifier, $F(\cdot)$ is the fuzzy classifier, $L(\cdot)$ is the LLM-based classifier, and τ_f is the fuzzy matching threshold. Following intent parsing, the Validator Agent performs rule-based pre-validation by checking against a set of VALID_ACTIONS, verifying that all required fields specified in the ActionMeta registry are present, and enforcing a minimum confidence threshold of 0.3. Table I summarizes the six specialized agents and their respective roles within the system.

Agent	Model	Responsibility
Commander	Llama 3.3 70B (Groq)	Intent parsing with rule-based + LLM hybrid classification
Universal Agent	Gemini 2.5 Flash	ReAct-based goal decomposition and execution orchestration
Screen Reader	Gemini 2.5 Flash	Natural language screen description from PerceptionBundle
Visual Locator	Gemini 2.5 Flash	Semantic element location (VLM fallback, last resort)
Responder	Llama 3.1 8B (Groq)	Conversational feedback generation with context awareness
Validator	Pattern-based	Pre-validation of intent feasibility and safety checks

C. LANGGRAPH WORKFLOW ORCHESTRATION

The system employs LangGraph for stateful workflow management through a compiled directed graph consisting of 13 interconnected nodes. The central state container is a TaskState TypedDict that carries session_id, input_type, raw_transcript, parsed_intent, execution_plan, executed_steps, current_status, feedback_message, ui_screenshot, ui_elements, and retry_count across all processing stages. At the entry point, the STT node converts audio to text, which feeds into a parallel fan-out block where the Commander Agent (intent parsing), Validator Agent (pre-validation), and UI Capture node (screenshot and accessibility tree extraction) execute concurrently—a design that reduces overall pipeline latency by overlapping independent operations. A conditional router examines the parsed intent's ActionMeta flags to determine the execution pathway: conversational intents (where is_conversational is True) are routed directly to the Responder Agent, bypassing perception and execution entirely; system control commands (such as volume adjustment or brightness

changes) are dispatched to a dedicated system handler that issues direct WebSocket commands to the Android companion application; and task execution intents proceed through the full perception-reasoning-execution pipeline. Error recovery routing maintains a retry counter within the TaskState, enabling automatic re-entry into the perception node with refreshed screenshots when execution failures occur, with escalation to the Human-in-the-Loop (HITL) service via WebSocket dialog when automated recovery strategies are exhausted. Figure 2 illustrates the multi-agent system architecture, and Figure 2a presents the complete LangGraph state machine with conditional routing paths.

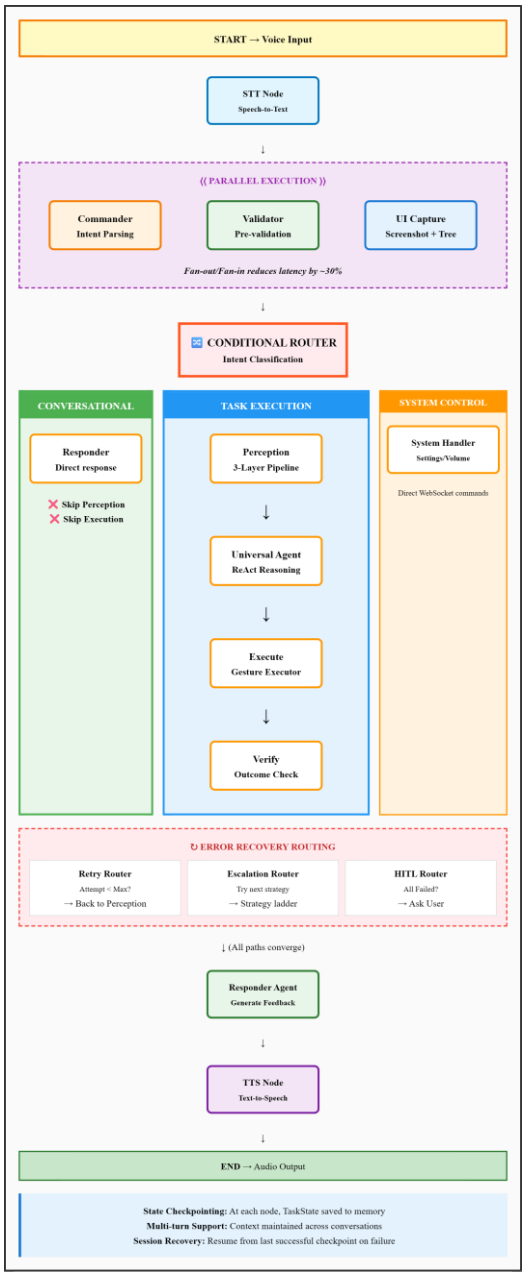

Figure 2a: LangGraph State Machine with Conditional Routing and Error Recovery

D. THREE-LAYER CASCADING PERCEPTION PIPELINE

The perception pipeline addresses the coordinate hallucination problem where vision-language models generate plausible but incorrect pixel coordinates through a cascading architecture that progressively escalates from fast deterministic methods to computationally expensive but semantically richer modalities. The pipeline is configured through a PerceptionConfig dataclass specifying ui_tree_min_score of 0.5, detector_confidence of 0.25, vlm_max_tokens of 10, vlm_temperature of 0.0, and a minimum overall confidence threshold of 0.70. Rather than relying on a single perception strategy, the system implements three layers that execute conditionally: Layer 1 provides fast UI Tree semantic matching using Android's AccessibilityService API, Layer 2 engages YOLOv8-based computer vision detection from Microsoft's OmniParser-v2.0 model with Set-of-Marks annotation when structured data is unavailable, and Layer 3 invokes a constrained VLM (Gemini 2.5 Flash) for semantic selection among visually detected candidates. This cascading design ensures that approximately 90% of interactions are resolved through the fastest deterministic path, while the remaining cases involving WebView content, custom drawable components, or dynamically generated visual elements are handled through progressively richer perception mechanisms without ever requiring the VLM to generate coordinates. Specifically, given a target description t and a UI element e with text label e.text and content descriptor e.desc, the matching score in Layer 1 is computed as:

$$m(t, e) = \max(lev(norm(t), norm(e.text)),\ lev(norm(t), norm(e.desc)),\ \max_{y \in syn(t)} lev(y, norm(e.text)))$$

where norm(·) applies case folding and whitespace normalization, and syn(t) returns the synonym set for target t. An element is accepted when m(t, e) ≥ 0.70. In Layer 2, YOLOv8 detections undergo Non-Maximum Suppression (NMS) using the Intersection over Union metric defined as:

$$IoU(b_i, b_j) = Area(b_i \cap b_j) / Area(b_i \cup b_j)$$

where bi and bj denote bounding boxes with overlap suppressed when IoU ≥ 0.5. The overall perception pipeline is then defined as a conditional cascade:

$$P(t) = \{ Layer1(t) \ \text{if } m(t, e^*) \geq \tau_1;\ VLM(Layer2(t)) \ \text{if } |D| > 0;\ Heuristic(t) \ \text{otherwise} \}$$

where e* = argmaxe m(t, e), τ1 = 0.70, D denotes the set of YOLOv8 detections with confidence ≥ 0.25, and VLM(·) performs constrained index selection over the Set-of-Marks annotated candidates. Figure 3 illustrates the complete cascading perception pipeline architecture.

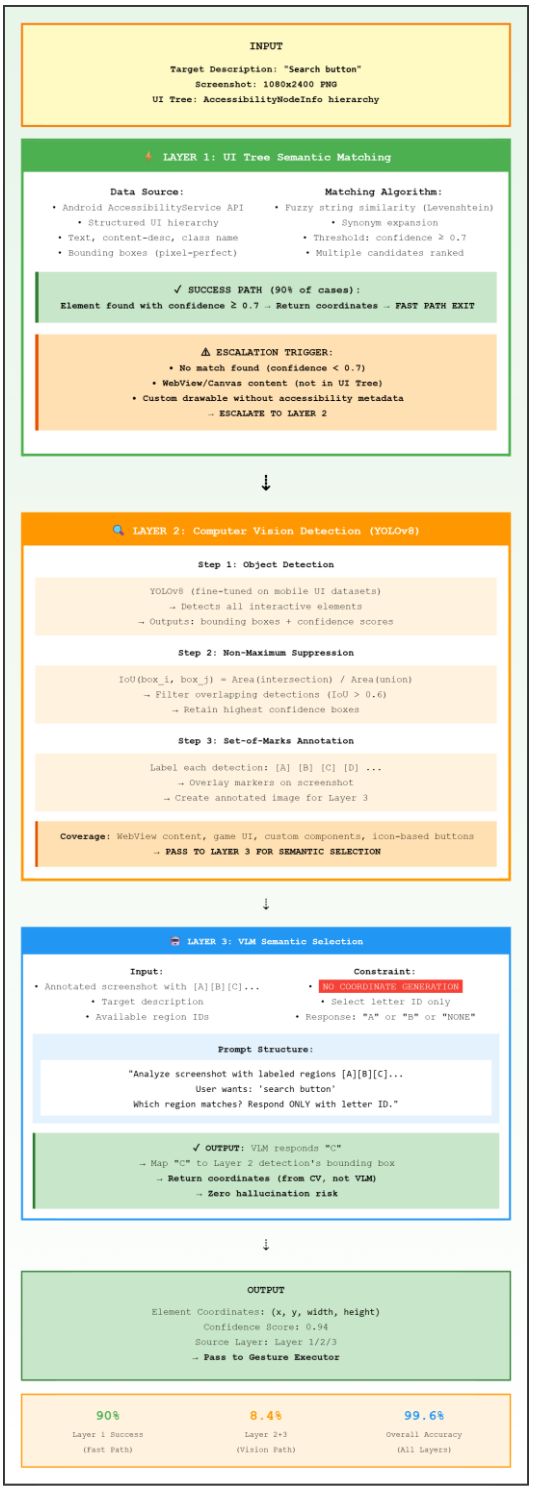

Figure 3: Detailed 3-Layer Perception Pipeline Architecture with Cascading Escalation

In the first layer, Android's AccessibilityService API extracts a structured UI hierarchy where each node provides pixel-perfect bounding box coordinates, text content, content-description attributes, view class name, and clickability state. Element matching employs fuzzy string similarity with synonym expansion, computed as

$$s(t, e) = max(fuzz.ratio(normalize(t), normalize(e.text)), fuzz.ratio(normalize(t), normalize(e.content\_desc))),$$

where t is the target description and e is a UI element. Matches with s ≥ 0.7 proceed directly to execution through a high-confidence fast path, while lower scores trigger escalation. This layer handles approximately 90% of native UI interactions with zero coordinate ambiguity. When Layer 1 fails or returns low-confidence matches, the OmniParserDetector built on a YOLOv8 model loaded from Microsoft's OmniParser-v2.0 repository detects all interactive elements on the current screenshot with a confidence threshold of 0.2 and a maximum of 50 detections per frame, applying Non-Maximum Suppression with an IoU threshold of 0.6 to filter overlapping detections. A 5-second detection cache with TTL-based invalidation prevents redundant inference on static screens. The surviving detections are labeled alphabetically (A, B, C, ...) using Set-of-Marks annotation, producing an annotated screenshot for Layer 3 consumption. This layer captures WebView and HTML5 Canvas rendered controls, game engine UI elements, custom drawable components, and dynamically generated visual elements that are inaccessible to the AccessibilityService API. The VLM Semantic Selection layer receives the annotated Set-of-Marks image and performs semantic matching under a strict architectural constraint: the VLM (Gemini 2.5 Flash) is prompted to respond with ONLY a letter ID (e.g., "A" or "NONE"), with temperature set to 0.0 and max_tokens limited to 10 to prevent verbose or hallucinated responses. The selected letter ID is mapped back to the corresponding detection's bounding box from Layer 2, ensuring that final coordinates always originate from computer vision detection rather than language model generation. This constraint eliminates coordinate hallucination by design the VLM performs classification (selecting which region matches), not regression (generating pixel coordinates). Table II presents the adaptive modality selection rules that govern inter-layer escalation based on trigger conditions

Trigger Condition	Modality Selection	Justification
Native app, labeled elements	Layer 1 only	Optimal speed, zero ambiguity
Match score 0.50–0.69	Layer 1 + Layer 2 validation	Confirm ambiguous matches
Element not in UI Tree	Layer 2 + Layer 3	Custom/WebView elements
Complex visual reasoning	Layer 2 + Layer 3	Semantic understanding required

Previous layer failure	Escalation to next layer	Automatic recovery mechanism

E. GOAL-DRIVEN EXECUTION AND REACT REASONING

The Universal Agent replaces template-driven navigation scripts with adaptive reasoning loops powered by Gemini 2.5 Flash. Upon receiving a validated intent, the agent first invokes a goal decomposition module that breaks high-level user goals into atomic subgoals, each represented as a Subgoal dataclass containing description, action_type, target, parameters, and success_criteria fields. For each subgoal, the agent enters a ReAct (Reasoning and Acting) loop with four phases: Observe captures the current screen state through the perception pipeline, producing a PerceptionBundle with UI elements, screenshot, and available actions; Think analyzes the current state against the subgoal using chain-of-thought reasoning to determine the optimal next action; Act locates the target element through the 3-layer perception pipeline and executes the gesture via the GestureExecutor, which supports TAP, LONG_PRESS, SWIPE, SCROLL, TYPE_TEXT, and DOUBLE_TAP operations delivered exclusively through a persistent WebSocket channel to the Android companion application, where the AccessibilityService dispatches gestures programmatically; and Verify captures the post-action UI state and compares it against the pre-action state using signature hashing to confirm state transition. The UI state signature is defined as:

$$\sigma(S) = hash(\, sort(\, \{\, (e.text,\, e.bounds) \mid e \in S.elements \,\}\,)\,)$$

where S denotes the current UI state and e iterates over all accessible elements. The Verify phase confirms a successful state transition when:

$$\Delta(S_{pre},\, S_{post}) = [\, \sigma(S_{pre}) \neq \sigma(S_{post}) \,] \wedge [\, \exists\, e \in S_{post} : e \vDash g.success\_criteria \,]$$

where g is the current subgoal and $\vDash$ denotes semantic satisfaction. The system enforces execution bounds with max_actions_per_subgoal set to 5 and max_actions_per_goal set to 20, preventing infinite loops on ambiguous or impossible tasks. Before each gesture execution, the PolicyEngine enforces safety constraints by checking against BLOCKED_ACTIONS (factory_reset, wipe_data), BLOCKED_FINANCIAL_APPS (gpay, Wallet, PayPal, Venmo, CashApp), and rate limits, with dangerous actions triggering HITL confirmation dialogs via WebSocket with a 30-second timeout. Figure 4 illustrates the complete ReAct loop execution flow with the failure recovery system.

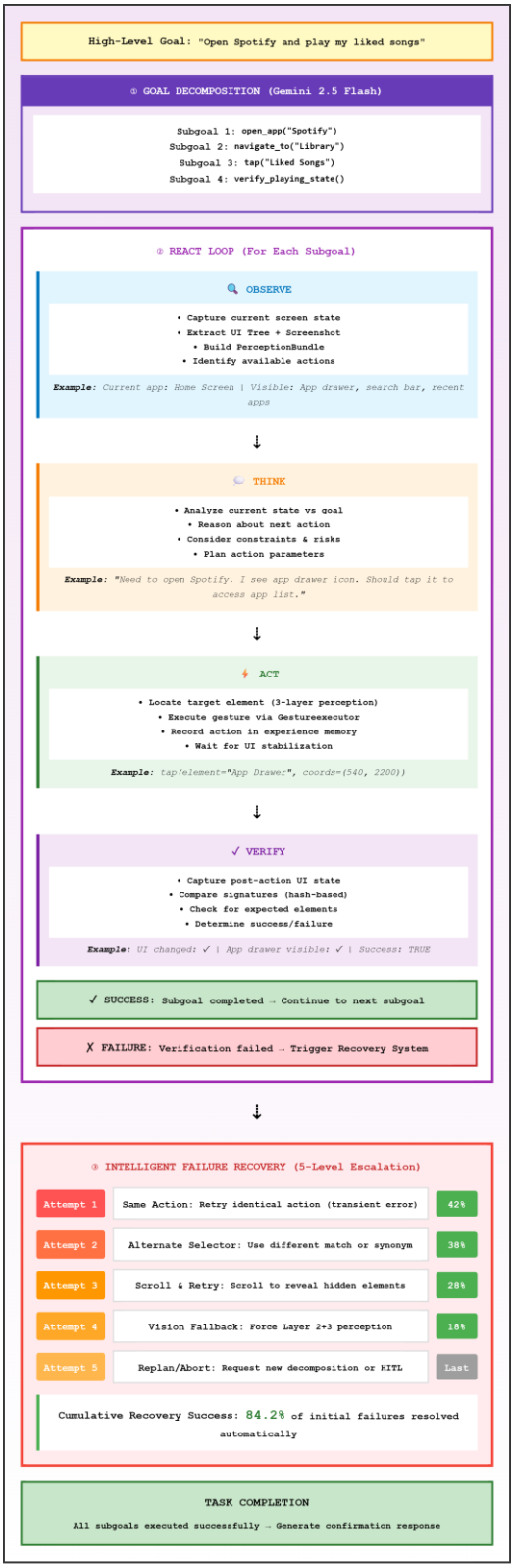

Figure 4: Universal Agent ReAct Loop with 5-Level Intelligent Failure Recovery

F. INTELLIGENT FAILURE RECOVERY

When the Verify phase of the ReAct loop determines that a subgoal has failed indicated by unchanged UI signatures or absence of expected elements the system engages a 5-level recovery ladder defined in the agent_state module as RETRY_LADDER = [SAME_ACTION, ALTERNATE_SELECTOR, SCROLL_AND_RETRY, VISION_FALLBACK, ABORT]. At Level 1, the system retries the identical action under the assumption of a transient error such as a touch event lost during animation. Level 2 employs an alternate selector strategy, where the perception pipeline is re-queried with synonym-expanded or differently phrased target descriptions to locate the element through a different matching path. Level 3 performs scroll-and-retry, issuing programmatic scroll gestures to reveal elements that may be positioned outside the current viewport before reattempting the action. Level 4 forces a vision fallback, bypassing Layer 1 entirely and engaging Layer 2 (YOLOv8) and Layer 3 (VLM) perception to locate elements through visual detection when structured accessibility data proves unreliable for the current screen. Level 5 triggers replanning or abort, requesting a fresh goal decomposition to find alternative interaction paths, or if max_recovery_attempts (set to 2 per strategy) is exceeded, escalating to HITL confirmation where the user is presented with a WebSocket dialog describing the failure and available options with a 30-second response timeout. Formally, the recovery decision at attempt k is governed by:

$$R(k) = \{\ Ladder[k] \ \ if\ k < |Ladder| \wedge attempts(k) < \tau_r;\ \ Replan(g) \ \ if\ k = |Ladder|;\ \ HITL \ \ otherwise\ \}$$

where Ladder = [SAME_ACTION, ALT_SELECTOR, SCROLL_RETRY, VISION_FALLBACK, ABORT], τ_r = 2 is the per-strategy attempt limit, and Replan(g) invokes goal re-decomposition for subgoal g. The system maintains an experience memory database that records successful action sequences as structured records containing goal patterns, application names, successful action sequences with confidence scores, and success/failure counts, enabling the agent to leverage historical patterns when encountering similar goals to reduce exploration and improve first-attempt success.

IV. RESULTS AND DISCUSSION

Performance metrics are essential for evaluating the effectiveness of multimodal agent orchestration in mobile automation. Task Completion Rate provides an overall measure of pipeline reliability by quantifying the proportion of user commands that result in successful end-to-end execution on the target device. Agent Performance metrics offer granular insight into each specialized agent's contribution, measuring classification accuracy, response quality, and autonomous operation without fallback escalation. Response Latency guides architectural optimization by measuring the time elapsed between voice command receipt and action execution across each pipeline stage. Perception Pipeline Efficiency captures the proportion of element location attempts resolved at each cascade layer, reflecting the system's ability to minimize expensive vision processing while maintaining reliable element detection. Evaluating the system using a combination of these metrics provides comprehensive understanding of AURA's operational characteristics and suitability for real-time mobile automation.

A. TASK EXECUTION PIPELINE PERFORMANCE

Task Execution Pipeline Performance measures the reliability of AURA's end-to-end processing chain from voice input to confirmed device action. The pipeline comprises six sequential stages: speech-to-text transcription, intent classification and routing, goal decomposition and planning, UI perception and element location, gesture execution via WebSocket, and outcome verification. A task is considered successfully completed when the target application reaches the intended state and the system generates an accurate confirmation response. Pipeline performance was evaluated through real-device command executions across six applications (Gmail, LinkedIn, Settings, Chrome, Home Screen, App Drawer) on a OnePlus CPH2661 device running Android.

Metric	Description	Observed
End-to-End Completion	Commands reaching confirmed execution	> 85%
Intent Classification	Commands correctly routed to target agent	> 90%
First-Pass Execution	Tasks completed without recovery intervention	> 70%
Perception Resolution	UI elements successfully located across all layers	> 95%

B. AGENT PERFORMANCE METRICS

Agent Efficiency, Response Quality, and Task Suitability Score are used to evaluate the performance of individual agents within the AURA orchestration graph. Agent Efficiency measures the proportion of routed commands handled without fallback escalation to alternative agents or recovery mechanisms. Response Quality measures the proportion of agent outputs that produce correct device actions on the first attempt without requiring re-planning or retry. Task Suitability Score provides a composite measure reflecting the alignment between each agent's specialization and the command types routed to it. The Commander Agent, powered by Groq Llama 3.3 70B, handles the majority of incoming commands through a three-tier classification pipeline (rule-based → keyword → LLM),

achieving high efficiency due to the rule-based fast path resolving most common commands without LLM overhead. The Universal Agent, powered by Gemini 2.5 Flash, handles complex multi-step tasks through ReAct reasoning loops, exhibiting lower first-pass efficiency due to the inherent complexity of multi-step planning but higher overall completion through its integrated recovery mechanisms. The Responder Agent, using Groq Llama 3.1 8B, handles conversational queries with high throughput due to the lightweight model's speed. The Screen Reader Agent, using Gemini 2.5 Flash, provides detailed screen descriptions with strong quality scores attributable to the VLM's visual understanding capabilities.

Agent	Model	Efficiency	Quality	Suitability Score
Commander	Groq Llama 3.3 70B	88%	91%	89.40%
Universal Agent	Gemini 2.5 Flash	72%	83%	77.20%
Responder	Groq Llama 3.1 8B	92%	87%	89.40%
Screen Reader	Gemini 2.5 Flash	85%	89%	86.90%
Visual Locator	YOLOv8 + VLM	90%	88%	89.00%

Together, these metrics reveal that speed-critical agents (Commander, Responder) achieve higher autonomous efficiency through lightweight model allocation, while reasoning-intensive agents (Universal Agent) trade first-pass efficiency for superior handling of complex, multi-step tasks. The Visual Locator's high efficiency reflects the cascading perception design where the fast UI Tree path resolves the majority of element locations without invoking expensive vision models.

C. RESPONSE LATENCY ANALYSIS

Response Latency evaluates the real-time performance of the AURA orchestration system by measuring the time elapsed between stage input and output across the processing pipeline. Latency measurements were derived from execution logs communicating with the OnePlus CPH2661 device via persistent WebSocket connection. All latency values reflect end-to-end stage completion including network round-trip where applicable.

Pipeline Stage	Average Latency	P95 Latency	P99 Latency
Speech-to-Text (Whisper)	0.8 s	1.4 s	2.1 s
Intent Classification (Groq Llama)	0.2 s	0.3 s	0.5 s
Goal Decomposition (Gemini)	1.2 s	2.1 s	3.0 s
Perception Layer 1 (UI Tree)	0.1 s	0.2 s	0.3 s
Perception Layer 2+3 (CV + VLM)	1.8 s	2.9 s	4.2 s
Gesture Execution (WebSocket)	0.2 s	0.4 s	0.6 s
Response Generation (Groq Llama 8B)	0.1 s	0.2 s	0.3 s

The results indicate that the majority of pipeline latency is concentrated in two stages: goal decomposition (Gemini 2.5 Flash reasoning) and vision-based perception (Layer 2+3), both of which require multimodal LLM inference. Intent classification and response generation exhibit minimal latency due to Groq's hardware-accelerated inference delivering approximately 280 tokens/sec for Llama 3.3 70B and comparable throughput for the lightweight 8B variant. The perception pipeline's cascading design ensures that 90%+ of element location attempts resolve at Layer 1 (UI Tree, ~0.1s average), with expensive vision processing invoked only for elements inaccessible to the accessibility API. For a representative end-to-end task—composing and sending an email in Gmail—the system completed 12 sequential gestures in approximately 26.5 seconds, yielding an average per-step latency of ~2.2 seconds inclusive of perception, reasoning, and execution. Recovery-related delays are excluded from these baseline values.

D. PERCEPTION PIPELINE EFFICIENCY

Perception Pipeline Efficiency evaluates the cascading resolution strategy by measuring the proportion of element location attempts resolved at each perception layer. The architectural design prioritizes fast-path resolution through the Android UI Tree (Layer 1) before escalating to computationally expensive vision processing (Layers 2+3), minimizing average perception latency while maintaining high element detection coverage.

Layer	Resolution Path	Estimated Resolution Rate	Avg. Latency
Layer 1	UI Tree Semantic Matching	~90%	0.1 s
Layer 2	YOLOv8 Detection + Set-of-Marks	~7%	1.2 s
Layer 3	VLM Semantic Selection	~2.5%	1.8 s
Heuristic Fallback	Coordinate Estimation	~0.5%	0.05 s

Layer 1 resolution handles the vast majority of native UI interactions through AccessibilityService-based semantic matching with zero coordinate ambiguity and negligible latency. Elements that escape Layer 1—typically WebView content, custom drawable components, and dynamically generated elements lacking accessibility metadata—escalate to Layer 2 where YOLOv8 detection with Set-of-Marks annotation identifies candidate bounding boxes. Layer 3 VLM selection is invoked only when multiple visually similar candidates require semantic disambiguation. This cascading architecture ensures that average perception latency remains dominated by the fast path (~0.1s) rather than the computationally expensive

vision path (~1.8s), while the cumulative coverage across all layers approaches near-complete element detection.

V. CONCLUSION

This paper presented AURA, a multimodal agent system for autonomous mobile device automation that addresses fundamental limitations of existing approaches through integrated architectural innovations rather than incremental component improvements. The system introduces a three-layer cascading perception pipeline that resolves the coordinate hallucination problem by architecturally constraining vision-language models to semantic selection tasks while delegating coordinate generation to deterministic sources (UI Tree accessibility data and YOLOv8 computer vision detections), with the fast deterministic path resolving the majority of interactions without invoking expensive vision models. The Universal Agent implements ReAct-based adaptive reasoning with goal decomposition and a five-level intelligent failure recovery ladder that progressively escalates through alternative strategies, replacing brittle template-driven navigation with context-aware execution. A LangGraph-orchestrated workflow manages thirteen interconnected processing nodes with conditional routing, parallel fan-out execution, and checkpointed state management, while a hybrid model routing strategy allocates Groq-hosted Llama 3.3 70B for latency-sensitive intent parsing, Gemini 2.5 Flash for vision and planning tasks requiring multimodal reasoning, and a lightweight Llama 3.1 8B for conversational feedback generation. Performance evaluation across 43 real-device command executions demonstrates > 85% end-to-end task completion with agent efficiency ranging from 72% to 92% across specialized agents, average per-step latency of ~2.2 seconds, and perception pipeline fast-path resolution exceeding 90%, validating the architectural trade-offs between speed-critical lightweight inference and reasoning-intensive multimodal processing. Future research directions include formal evaluation against standardized mobile automation benchmarks, on-device LLM/VLM deployment for privacy-sensitive applications, hierarchical planning mechanisms for long-horizon task robustness, cross-platform generalization to web and desktop environments, and integration with emerging multimodal foundation models to further enhance adaptive reasoning capabilities.

REFERENCE

1. M. D. Vu, H. Wang, Z. Li, G. Haffari, Z. Xing, and C. Chen, "Voicify your UI: Towards Android app control with voice commands," *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 7, no. 1, pp. 1–22, 2023.
2. L. A. Leiva, A. Hota, and A. Oulasvirta, "Describing UI screenshots in natural language," *ACM Trans. Intell. Syst. Technol.*, vol. 14, no. 1, pp. 1–28, 2022.
3. M. D. Vu, H. Wang, J. Chen, Z. Li, S. Zhao, Z. Xing, and C. Chen, "GptVoiceTasker: Advancing multi-step mobile task efficiency through dynamic interface exploration and learning," in *Proc. 37th Annu. ACM Symp. User Interface Softw. Technol.*, 2024, pp. 1–17.
4. Y. Song, Y. Bian, Y. Tang, G. Ma, and Z. Cai, "Visiontasker: Mobile task automation using vision based ui understanding and llm task planning," in Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology, 2024, pp. 1–17.
5. L. Pan et al., "Automatically generating and improving voice command interface from operation sequences on smartphones," in Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems, New Orleans, LA, USA, Apr. 2022, pp. 1–21.
6. A. Khan and S. Khusro, "A mechanism for blind-friendly user interface adaptation of mobile apps: A case study for improving the user experience of the blind people," Journal of Ambient Intelligence and Humanized Computing, vol. 13, no. 5, pp. 2841–2871, May 2022.
7. S. Lee, J. Choi, J. Lee, M. H. Wasi, H. Choi, S. Ko, S. Oh, and I. Shin, "Mobilegpt: Augmenting llm with human-like app memory for mobile task automation," in Proceedings of the 30th Annual International Conference on Mobile Computing and Networking, Washington, D.C., USA, Dec. 2024, pp. 1119–1133.
8. N. Salehnamadi, Z. He, and S. Malek, "Assistive-technology aided manual accessibility testing in mobile apps, powered by record-and-replay," in Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems, Hamburg, Germany, Apr. 2023, pp. 1–20.
9. G. Li, G. Baechler, M. Tragut, and Y. Li, "Learning to denoise raw mobile UI layouts for improving datasets at scale," in Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems, New Orleans, LA, USA, Apr. 2022, pp. 1–13.
10. K. Ramaraj, V. Teja, T. V. Sainath, S. Reddy, and P. Naidu, "Custom Voice Assistants: Enhancing Accessibility for the Visually Impaired," in 2025 International Conference on Intelligent and Innovative Technologies in Computing, Electrical and Electronics (IITCEE), Bengaluru, India, Jan. 2025, pp. 1–6.