أكاديمية كاوست
KAUST ACADEMY

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

FAHIM: Framework for Arabic Human-centered Interaction on Mobile

Nawaf bin Jurayyan, Ammar Abdulrahman, Khaled Alshahry,
Abdulaziz Batis, Muhammad Mubashar

nawaf.binjurayyan@kaust.edu.sa, ammar.abdulrahman@kaust.edu.sa, khaled.alshahry@kaust.edu.sa,
abdulaziz.batis@kaust.edu.sa, muhammad.mubashar@strath.ac.uk

Abstract

Blind and visually impaired users often navigate mobile interfaces one element at a time, making unfamiliar multi-step tasks slow and cognitively demanding. FAHIM (Framework for Arabic Human-centered Interaction on Mobile) is a goal-oriented Android accessibility assistant that accepts high-level Arabic voice commands. It transcribes the request, observes the Android Accessibility Tree, and asks a self-hosted Large Language Model Planner to select the next interface action. The Android client validates and executes that action, observes the updated state, and repeats the cycle until the task completes, clarification is required, or execution stops. Consequential actions require explicit user confirmation.

This report evaluates the submitted FAHIM implementation. An initial eight-scenario baseline completed 2/8 tasks (25%) with an average time of approximately 52.5 seconds. A follow-up tester workbook documented 15 traceable trials across six scenario types, of which 12 completed successfully (80%); the unweighted scenario means were 16.0 seconds and 4.2 Planner steps. The Planner Server also passed all 80 regression tests. Because the two live evaluations used different scenario sets, the figures are descriptive rather than a controlled before-and-after comparison. The results support the feasibility of Arabic goal-oriented interaction for launching, search, and bounded text-entry tasks, while highlighting the need for stronger system-setting execution, reproducible evaluation, safety and privacy validation, and testing with Arabic-speaking blind and visually impaired users.

Date: August 23, 2026

Contents

1 Introduction **1**

1.1 Problem Statement . . . 1
1.2 Key Assumptions and Scope . . . 1
1.3 Proposed Solution . . . 1

2 Related Work **2**

2.1 TalkBack . . . 2
2.2 Insight . . . 2
2.3 LLMs for Mobile GUI Accessibility . . . 2
2.4 AI-Based Assistance for Visually Impaired Users . . . 3
2.5 Arabic Voice-Based Assistive Technologies . . . 3
2.6 Research Gap . . . 4

3 Methodology **4**

3.1 System Overview . . . 4
3.2 Implementation Baseline and Scope . . . 4
3.3 System Architecture . . . 4
3.4 Speech Understanding Module . . . 6
3.5 UI State Analyzer . . . 6
3.6 Task Planning Module . . . 6
3.7 Action Execution Module . . . 6
3.8 Feedback and Safety Module . . . 7
3.9 Evaluation Procedure . . . 7

4 Results **8**

4.1 Baseline Evaluation . . . 8
4.2 Follow-up Live-Device Evaluation . . . 8
4.3 Automated Verification Results . . . 9
4.4 Comparison Boundary . . . 9

5 Discussion **9**

6 Limitations and Future Work **10**

6.1 Evaluation Scale and Comparability . . . 10
6.2 Missing Reproducibility Artifacts . . . 10
6.3 Arabic Fixture Integrity . . . 10
6.4 Device and Interface Variability . . . 10
6.5 Safety Validation . . . 11
6.6 Privacy and Operational Configuration . . . 11
6.7 User-Centered Validation . . . 11

7 Conclusion **11**

References **11**

1 Introduction

In an increasingly digital world, smartphones and mobile applications have become essential tools for daily life, supporting activities ranging from communication to banking and healthcare. However, for blind and visually impaired individuals, navigating complex mobile user interfaces (UIs) remains a major barrier to digital independence. While traditional screen readers such as Apple's VoiceOver and Android's TalkBack provide spoken feedback, they still require users to manually navigate interface elements one step at a time, making complex tasks time-consuming and cognitively demanding.

1.1 Problem Statement

The core problem is that existing mobile accessibility tools are primarily passive; they provide information about what is on the screen but generally require the user to perform manual, step-by-step navigation. For a blind or visually impaired user, completing even a simple task on an unfamiliar mobile application can require considerable time and effort. There is a need for an active, intelligent assistant that can interpret high-level spoken commands, navigate mobile interfaces autonomously on behalf of the user, and verbally report the actions taken and their outcomes.

1.2 Key Assumptions and Scope

In developing this system, we work under the following assumptions:

- The target user interacts with the system primarily using natural Arabic voice commands.
- The system operates on Android devices using the Android Accessibility framework.
- The system relies on a Large Language Model (LLM)-based Planner capable of reasoning over the user's request and the current interface state to determine concrete UI actions, such as clicking, typing, or scrolling.
- UI observation, action validation, and action execution are performed on the Android device, while AI-based planning is performed through a self-hosted Planner Server.

1.3 Proposed Solution

To address this challenge, we introduce FAHIM, an intelligent mobile agent designed to enable voice-driven interaction with Android applications for Arabic-speaking blind and visually impaired users. The system acts as an intermediary between the user and the mobile interface. Instead of manually navigating the screen, users describe the task they wish to accomplish, such as "Call Khalid" or "Turn on Wi-Fi."

FAHIM interprets the spoken request, observes the current interface state, and uses an LLM-based Planner to determine the next appropriate UI action. The Android application validates and executes the selected action using Android Accessibility Services and then captures the updated interface state. This observe–plan–act process continues iteratively until the user's requested goal is achieved, clarification is required, or execution must stop. Throughout the interaction, FAHIM provides spoken feedback and maintains user control over sensitive or consequential actions.

By transforming traditional step-by-step screen navigation into goal-oriented voice interaction, FAHIM aims to improve digital accessibility, task efficiency, and user independence.

2 Related Work

Recent advances in mobile accessibility have demonstrated the potential of conversational interfaces and Large Language Models (LLMs) to improve the interaction experience of blind and visually impaired users. Existing accessibility solutions such as TalkBack provide fundamental screen-reading capabilities, while recent LLM-based systems aim to enable more natural and intelligent interaction. This section reviews conventional screen readers, LLM-based mobile accessibility systems, and related assistive technologies.

2.1 TalkBack

TalkBack is the default Android screen reader that enables blind and visually impaired users to interact with mobile devices through spoken feedback and gesture-based navigation. It provides auditory descriptions of interface elements as users navigate the screen using swipe and touch gestures. In addition, TalkBack supports accessibility features such as voice commands, braille displays, and customizable verbosity settings.

Despite its widespread adoption, TalkBack follows a sequential navigation paradigm in which users must manually traverse interface elements one at a time. This interaction style often increases cognitive load, especially when completing complex multi-step tasks, since users must remember screen layouts and navigate through multiple interface components before reaching the desired action. Furthermore, TalkBack provides limited understanding of the overall screen context, making it difficult to efficiently perform goal-oriented interactions or obtain concise summaries of complex interfaces.

2.2 Insight

Insight is a recently proposed Android accessibility service that leverages Large Language Models (LLMs) to provide conversational interaction for blind and visually impaired users [1]. Unlike traditional screen readers, Insight combines natural language dialogue with real-time screen summarization, allowing users to express high-level goals rather than manually navigating every interface element.

The system analyzes the Android accessibility hierarchy, generates contextual summaries of the current screen, answers user questions, and produces executable UI actions through an LLM-powered planning process.

User studies reported that Insight reduced task completion time and cognitive effort while improving perceived usability compared with TalkBack. Participants preferred the conversational interaction model because it enabled more natural and efficient interaction. However, the study also identified several limitations, including reliance on cloud-hosted LLMs, privacy concerns associated with transmitting screen information, limited interruption during long responses, and the need for more robust conversational interaction.

Compared with Insight, FAHIM focuses specifically on natural Arabic voice interaction and employs a self-hosted planning architecture. UI observation, action validation, and action execution are performed on the Android device, while LLM-based planning is performed through the Planner Server. In addition, FAHIM incorporates explicit user confirmation before performing sensitive or consequential actions, helping users remain in control throughout task execution.

2.3 LLMs for Mobile GUI Accessibility

Recent research has also explored the use of Large Language Models for understanding and improving the accessibility of mobile graphical user interfaces.

Liu et al. proposed HintDroid, an LLM-based approach for automatically generating missing hint text for Android text-input components [2]. Their analysis of thousands of Android applications showed that missing hint text is widespread and can prevent screen readers from adequately communicating the purpose of input fields to visually impaired users.

HintDroid analyzes information extracted from the Android GUI and view hierarchy and uses in-context learning to generate appropriate hint text. Although HintDroid focuses on improving accessibility information rather than autonomously controlling the smartphone, its use of structured Android interface information as contextual input to an LLM is closely related to FAHIM's approach. In FAHIM, interface information is instead provided to the LLM to determine the next action required to complete the user's requested task.

Zhang et al. proposed VisualDroid, a multimodal LLM-based approach for identifying accessibility problems caused by dynamic visual changes in Android applications [3]. The study showed that important graphical interface changes, including warning messages, loading progress, content updates, and automatically changing views, may be visible to sighted users while remaining unavailable through screen readers such as TalkBack.

VisualDroid uses screenshots and multimodal LLM reasoning to detect and classify these inaccessible interface changes. While VisualDroid is primarily designed as an accessibility-testing tool for developers, its findings highlight an important challenge for intelligent accessibility agents: information exposed through the accessibility framework may not always represent the complete state of the graphical interface.

2.4 AI-Based Assistance for Visually Impaired Users

Large vision-language models have also been explored for assisting visually impaired users in understanding their surrounding physical environment.

Chen et al. proposed a Large Vision-Language Model (LVLM)-based environment perception system for visually impaired users [4]. The system combines a vision-language model with image segmentation to provide both global descriptions of a scene and detailed descriptions of individual objects. Segmentation results are incorporated as external knowledge into the LVLM input to reduce hallucination and improve the accuracy of generated descriptions.

Although this system targets the same general user population as FAHIM, its objective is different. It primarily focuses on understanding the user's physical surroundings through visual perception, whereas FAHIM focuses on understanding and interacting with the digital environment represented by Android applications.

Nevertheless, this work demonstrates the potential of large language and multimodal models to provide context-aware assistive technologies for blind and visually impaired users.

2.5 Arabic Voice-Based Assistive Technologies

Arabic voice interaction is particularly relevant to FAHIM because voice provides a natural non-visual communication channel for blind users.

Khasawneh and Khasawneh investigated the availability of voice-recognition technologies for visually impaired students in Saudi Arabian universities [5]. Their study highlighted both the potential value of voice-based assistive technologies and limitations associated with the availability and effectiveness of existing solutions. This provides relevant regional context for the development of Arabic voice-based accessibility systems.

Habbi and Bettayeb developed a voice-assistance system for blind users with explicit Arabic-language support [6]. Their system integrates real-time object detection using YOLOv8, multimodal scene understanding, and Arabic voice interaction on an embedded Raspberry Pi platform.

While their system primarily assists users in perceiving the physical environment, it demonstrates the feasibility of combining Arabic speech interaction with AI-based assistive technologies. FAHIM extends this concept toward smartphone interaction, where spoken Arabic commands are translated into sequences of Android UI actions.

2.6 Research Gap

Existing research demonstrates significant progress in screen reading, conversational accessibility, LLM-based GUI understanding, and AI-assisted environment perception. However, important limitations remain.

Traditional screen readers such as TalkBack generally require users to navigate mobile interfaces manually. Recent LLM-based systems provide more intelligent interface understanding and conversational interaction, but comparatively limited research has focused on autonomous mobile interaction designed specifically around natural Arabic voice commands.

FAHIM addresses this gap by combining Arabic speech interaction, LLM-based reasoning over the Android Accessibility Tree, autonomous multi-step UI action execution, continuous spoken feedback, and explicit confirmation before sensitive operations. Rather than only describing what is present on the screen, FAHIM aims to allow users to express a high-level goal and have the system actively navigate the Android interface toward completing that goal.

3 Methodology

This chapter presents the proposed framework, FAHIM, including its overall architecture, system components, and evaluation methodology.

3.1 System Overview

FAHIM (Framework for Arabic Human-centered Interaction on Mobile) is an intelligent mobile agent designed to assist Arabic-speaking blind and visually impaired users in completing multi-step tasks through natural voice interaction. Rather than requiring users to navigate graphical interfaces manually, the framework allows users to express their desired tasks using spoken Arabic commands.

FAHIM follows an iterative observe–plan–act architecture. The system observes the current mobile interface, uses an LLM-based Planner to determine the next appropriate action, validates and executes that action, and then observes the updated interface state. This process continues until the user's requested goal is achieved, clarification is required, or execution must stop.

Instead of relying on application-specific workflows, the system discovers available applications at runtime and uses the same planning, perception, validation, and execution pipeline across accessible Android applications.

3.2 Implementation Baseline and Scope

This report evaluates the FAHIM implementation maintained in the *feature/robust-agent-pipeline* branch. The submitted system consists of a native Android client implemented in Kotlin and a self-hosted FastAPI Planner Server. The Android client captures and validates the current interface state, while the Planner Server uses a Large Language Model to select the next appropriate action.

3.3 System Architecture

The proposed FAHIM framework consists of the following five main components:

1. Speech Understanding Module

2. UI State Analyzer
3. Task Planning Module
4. Action Execution Module
5. Feedback and Safety Module

These components work together through an iterative process to complete the user's requested task. The overall system workflow is summarized as follows:

1. The user provides a task using natural Arabic speech.
2. The Speech Understanding Module transcribes the spoken command and identifies the user's intended goal.
3. The UI State Analyzer extracts and structures the current Android Accessibility Tree to represent the current interface state.
4. The Task Planning Module reasons over the user's goal and the current UI state to determine the next appropriate action.
5. The planned action is checked to determine whether explicit user confirmation is required.
6. If the action is sensitive or consequential, the system obtains explicit user confirmation before proceeding.
7. The Action Execution Module validates and performs the selected action using Android Accessibility Services, after which the updated interface state is captured.
8. The system determines whether the requested task has been completed. If not, the updated UI state is returned to the planning process and the observe–plan–act cycle continues.

The complete iterative workflow is illustrated in Figure 1.

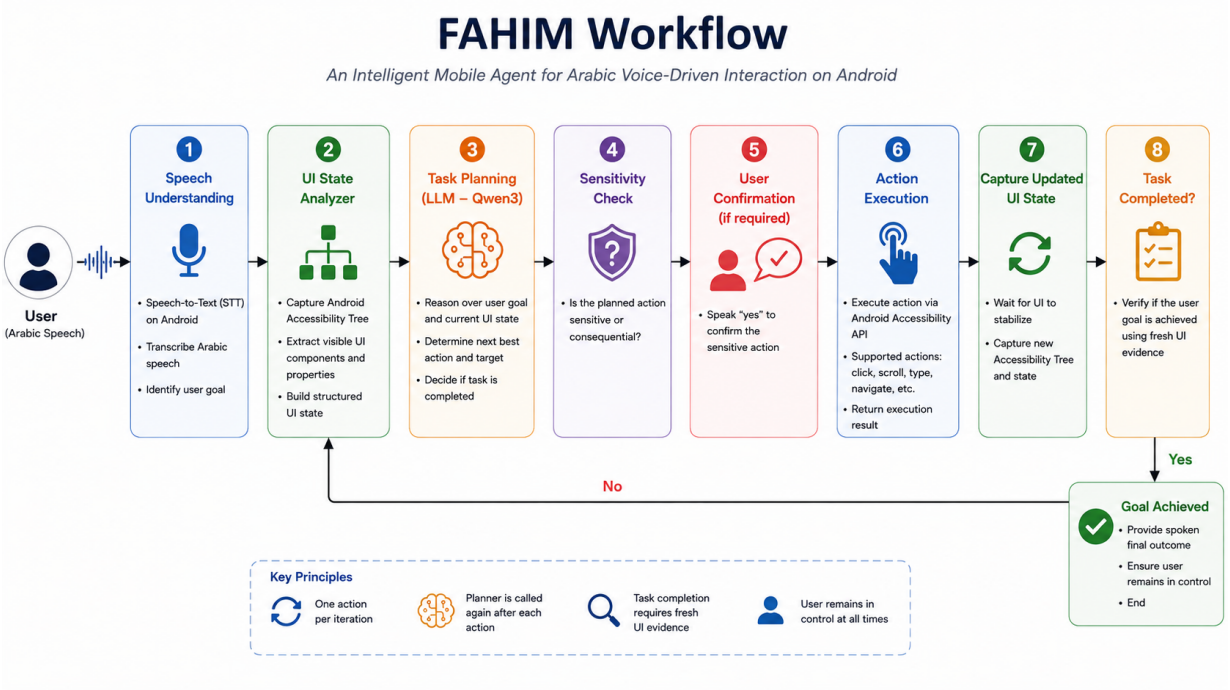


Figure 1 Overall workflow of the FAHIM framework.

3.4 Speech Understanding Module

The Speech Understanding Module serves as the entry point of the FAHIM framework by converting the user's spoken Arabic command into text. To perform speech recognition, the framework utilizes Android's built-in Speech-to-Text (STT) service, enabling integration with the Android platform while supporting natural voice interaction.

The original Arabic command is preserved throughout the planning process rather than being locally reduced to predefined keywords or extracted entities. The transcribed command is provided to the LLM-based Planner as the active user goal, allowing the Planner to reason directly over the user's natural-language request.

3.5 UI State Analyzer

The UI State Analyzer retrieves the Android Accessibility Tree through Android Accessibility Services. The extracted hierarchy contains information about visible interface components, including text, content descriptions, class names, view identifiers, roles, supported actions, bounds, and structural paths.

The raw accessibility hierarchy is transformed into a compact and structured representation of the current interface before being provided to the Planner. Where appropriate, non-clickable child labels are associated with their actionable parent components to form logical interaction targets.

Each actionable component is assigned a screen-scoped target identifier. Before an action is executed, the selected target is re-resolved against the current interface state. This mechanism helps prevent actions from being performed on stale, missing, replaced, or ambiguous interface elements.

3.6 Task Planning Module

The Task Planning Module determines the next action required to progress toward the user's requested goal. Using a Large Language Model (LLM), the module reasons over the user's active goal together with the current Android interface state to determine the most appropriate next accessibility action.

The Planner receives structured information describing the user request, active goal, current application, current UI state, conversation context, result of the previous action, and action history.

Rather than generating the complete action sequence in advance, FAHIM follows an iterative planning approach. Exactly one physical action is selected for each planning iteration. After the action is executed, the updated interface state is retrieved and returned to the Planner. This enables the system to dynamically adapt to interface changes and unexpected states during multi-step tasks.

Task completion is also determined through the planning process. A task is considered complete only when the Planner indicates completion and the current interface provides fresh evidence that the requested goal has been achieved.

3.7 Action Execution Module

The Action Execution Module translates the Planner's decisions into executable Android interactions using Android Accessibility Services. Supported operations include clicking interface elements, scrolling, entering text, navigating between screens, launching applications, and performing other accessibility-supported actions.

Before execution, the requested action and target are validated against the current UI state. Invalid or unsupported actions and stale, missing, or invented targets are rejected.

The framework executes exactly one physical action for each Planner response. Following execution, the system waits for the interface to stabilize before capturing the updated UI state and beginning the next planning iteration.

3.8 Feedback and Safety Module

The Feedback and Safety Module is designed to maintain user awareness and control throughout task execution. FAHIM provides spoken Arabic feedback describing its progress and the outcomes of executed actions.

Actions are classified according to their potential consequences. Reversible interactions such as navigation, scrolling, search, and text entry normally proceed without additional confirmation. In contrast, consequential actions such as placing a phone call, sending content, deleting information, making purchases, posting content, or changing security-related settings require explicit user confirmation before execution.

Each confirmation is associated with the specific goal, action, target, and current interface state. Once used, the confirmation cannot be reused for another action. When the Planner cannot safely determine the appropriate next action, the system may instead request clarification from the user.

3.9 Evaluation Procedure

The system was evaluated using representative end-to-end scenarios that exercise application launching, search, system settings, text entry, multi-step execution, task completion, and dynamic interfaces. The initial baseline used eight scenarios, denoted T01–T08. A later tester workbook documented 15 follow-up trials across six scenario types (T01, T02, T03, T04, T06, and T07). These identifiers are preserved exactly as supplied; no completed T05 record was included. The identifiers in the two evaluations are not assumed to denote identical tasks unless an explicit mapping is available.

The evaluation distinguishes between action-level and task-level success. A trial is counted as successful only when the requested end state is reached; successfully executing one or more Android actions is not sufficient. Results are therefore aggregated primarily at the trial level. The tester also supplied scenario-level average time and Planner-step counts. Because the number of trials differs across scenarios, averages computed from these six scenario summaries are reported as unweighted means.

The primary evaluation metrics available for the follow-up study are:

- **Trial Completion Rate:** the percentage of documented trials in which the requested user goal is successfully achieved.
- **Task Completion Time:** the total time required to complete a user request, summarized by scenario in the tester workbook.
- **Planner Steps:** the number of planning actions required to reach the final outcome.
- **Failure Observation:** the tester's recorded cause or visible symptom when a trial does not complete.

In addition to live-device evaluation, automated regression testing is used to verify Planner Server and Android implementation behavior. Automated checks and end-to-end trials are reported separately because passing unit or regression tests does not by itself demonstrate successful task completion on a physical device.

4 Results

This section separates live-device evidence from automated verification. The baseline results describe the earlier eight-scenario evaluation, while the follow-up live-device results are derived from the tester's evaluation workbook for the submitted system.

4.1 Baseline Evaluation

The baseline implementation successfully completed 2 out of the 8 evaluated scenarios, corresponding to a task completion rate of **25%**. The average completion time observed during the baseline experiments was approximately **52.5 seconds**.

Although individual Android actions could often be executed successfully, the evaluation showed that action-level success did not necessarily correspond to successful completion of the user's overall goal.

Analysis of the failed scenarios revealed several recurring failure modes, including incorrect application selection, incorrect target selection, stale interface state rejection, repeated actions, premature task completion, overshooting the requested goal, and unnecessary confirmation loops.

Metric	Baseline Result
Evaluation scenarios	8
Successfully completed scenarios	2
Failed scenarios	6
Task completion rate	25%
Average completion time	~52.5 s

Table 1 Baseline end-to-end evaluation results.

4.2 Follow-up Live-Device Evaluation

The tester documented 15 traceable trials across six scenario types. Twelve trials completed successfully, giving a primary trial-level completion rate of **80% (12/15)**. Table 2 preserves the scenario-level counts and summary measurements supplied in the workbook.

ID	Scenario	Success	Time	Steps	Observation
T01	Launch YouTube	3/3	10 s	3	Completed
T02	Set a five-minute timer	1/3	15 s	4	Accessibility failure
T03	Search in YouTube	3/3	20 s	5	Completed
T04	Navigate to Wi-Fi settings	2/3	16 s	4	Source criteria conflict
T06	Search WhatsApp for Khalid	1/1	20 s	5	Completed
T07	Type a message without sending	2/2	15 s	4	Completed
	Total / unweighted mean	**12/15**	**16.0 s**	**4.2**	

Table 2 Traceable follow-up live-device results. Time and step values are scenario-level averages supplied in the tester workbook; the final means are unweighted across the six scenario summaries.

The tester categorized five of the six scenario types as passed, corresponding to **83.3%** at the scenario-category level. This value uses a different denominator and success rule from the trial-level rate, so the **12/15 (80%)** result is used as the main completion figure. The scenario identifiers are reproduced exactly as supplied; the workbook contains no completed T05 entry.

The Arabic command recorded for T04 includes turning off the Internet, while the same workbook's expected flow and safety condition stop at the Wi-Fi page and explicitly prohibit

toggling Wi-Fi. To avoid overstating the evidence, T04 is reported here only as navigation to the Wi-Fi settings page; no claim is made that Internet access was disabled.

4.3 Automated Verification Results

Automated checks provide supporting evidence for implementation stability. The Planner Server regression suite was reproduced on the submitted branch and completed **80/80 tests**. The Android test, lint, and build results below were reported in the tester brief; their raw logs or artifacts were not included with the summary document.

Verification	Result	Evidence status
Planner Server tests	80/80 passed	Independently reproduced
Android tests	150 passed, 0 failed	Tester-reported; raw output pending
Android Lint	0 errors, 20 warnings	Tester-reported; raw output pending
Debug APK build	Successful	Tester-reported; artifact or build log pending

Table 3 Automated verification evidence for the submitted implementation.

4.4 Comparison Boundary

The baseline completion rate of 25% and the follow-up trial rate of 80% are encouraging when viewed descriptively. However, the baseline used eight scenarios while the follow-up workbook documents six scenario types with different identifiers and trial counts. The exact Arabic utterances, initial application states, device configuration, and mapping between the two scenario sets are not yet available. Consequently, the difference must not be interpreted as a controlled 55-percentage-point improvement or attributed solely to the revised architecture. A paired rerun of the same scenarios under the same conditions is required for that claim.

5 Discussion

The follow-up evaluation suggests that FAHIM's observe–plan–act pipeline is most reliable when the goal has a clear application target and an easily observable end state. Launching YouTube, searching within YouTube, locating a WhatsApp contact, and entering text without sending all completed in every documented trial. These tasks align well with the system architecture: each step exposes a recognizable accessibility state, and the Planner can verify progress before selecting the next action.

The weaker timer and Wi-Fi results reveal a different class of difficulty. Setting a timer completed in only one of three trials, with an accessibility failure noted by the tester, while opening Wi-Fi settings failed once because of an incorrect action. System interfaces may vary across Android versions, device vendors, locales, and current UI states; targets can also change between observation and execution. These outcomes indicate that robust execution depends not only on language-model planning but also on target freshness, accessibility-node quality, UI stabilization, and recovery after an action fails.

The 80/80 Planner Server regression result is valuable because it shows that the implemented planner contracts and regression cases are stable in the tested environment. Nevertheless, automated tests and live-device trials answer different questions. Regression tests verify expected software behavior under defined fixtures, whereas end-to-end trials measure whether the complete system reaches the user's goal on a real interface. The remaining live failures therefore do not

contradict the automated result; they expose conditions that the current fixtures do not fully represent.

The reduction from the baseline's approximately 52.5-second average to a 16.0-second unweighted follow-up scenario mean is promising, but it cannot yet be treated as a measured speed-up. The two studies used different scenario sets and aggregation procedures. A valid comparison requires identical commands, starting states, devices, Planner configuration, and trial counts. Until that experiment is performed, the strongest supported conclusion is that the submitted system completed a majority of the documented follow-up trials and performed particularly well on common launch, search, and bounded text-entry tasks.

One useful safety-related observation is that T07 required FAHIM to type a message without sending it. Completing this scenario in both trials suggests that the system can respect a bounded goal rather than automatically performing the next consequential action. This is encouraging, but broader negative testing is still required before generalizing the result to sending, calling, deleting, purchasing, or security-related actions.

6 Limitations and Future Work

6.1 Evaluation Scale and Comparability

The follow-up workbook is labelled as a ten-scenario benchmark but contains completed records for six scenario types and 15 aggregate trials, and it is not directly paired with the eight-scenario baseline. The next evaluation should define one versioned protocol, preserve the exact Arabic utterance and initial UI state for every task, and run multiple repetitions under identical conditions. Trial-level records should be exported to a machine-readable file so that completion rates, confidence intervals, and latency statistics can be reproduced.

6.2 Missing Reproducibility Artifacts

The supplied evaluation materials include an aggregate scenario workbook and an automated-test summary, but they do not include the tested commit, APK hash, device and Android version, Planner model and prompt version, server configuration, timestamped per-trial logs, or raw Android test, lint, and build outputs. Future evaluation packages should include this metadata and evidence. Doing so would make the reported results auditable and would separate changes in the Android client, Planner, model, and environment.

6.3 Arabic Fixture Integrity

One committed Planner fixture file intended to contain Arabic examples is corrupted and contains replacement question-mark characters instead of valid Arabic text. This weakens the evidence for Arabic-language regression coverage. The file should be restored from a valid UTF-8 source, reviewed by an Arabic speaker, and protected with automated encoding and content-integrity checks in continuous integration.

6.4 Device and Interface Variability

The timer and Wi-Fi failures show that accessibility actions remain sensitive to device-specific system interfaces and changing UI states. The execution layer should add stronger post-action verification, bounded retries, stale-target recovery, and fallback navigation. Evaluation should then be repeated across multiple Android versions, device vendors, display settings, languages, and application versions.

6.5 Safety Validation

The architecture requires confirmation before consequential actions, but the current evidence covers only a limited non-sending boundary. A dedicated safety suite should test both positive and negative cases for sending messages, placing calls, deleting data, purchasing, posting, and changing security settings. It should verify that confirmation is specific to the current goal, target, action, and interface state and cannot be reused.

6.6 Privacy and Operational Configuration

Debug traces can contain user commands and accessibility-tree content, and temporary Planner endpoints make experiments difficult to reproduce. For production, logged content should be redacted or minimized, retention controls should be defined, authenticated transport should be used, and the tested Planner endpoint, model, prompt, and configuration should be pinned in a versioned release manifest.

6.7 User-Centered Validation

The present evaluation demonstrates technical feasibility but does not establish usability or accessibility benefits for the target population. A future study should involve Arabic-speaking blind and visually impaired participants, with appropriate ethical approval, and measure task completion, workload, perceived control, error recovery, trust, and comparison with standard TalkBack interaction. Arabic dialect coverage and speech-recognition errors should also be evaluated explicitly.

7 Conclusion

FAHIM demonstrates a practical architecture for converting high-level Arabic voice goals into validated Android accessibility actions through an iterative observe–plan–act loop. For the submitted implementation, the follow-up workbook documents 12 successful trials out of 15 (80%) across six scenario types, with an unweighted mean of 16.0 seconds and 4.2 Planner steps, while the Planner Server passed all 80 regression tests. The strongest results were observed for application launching, search, and bounded text entry; timer and Wi-Fi tasks exposed remaining weaknesses in device-level accessibility execution and recovery. Because the baseline and follow-up protocols differ, the results do not yet support a controlled improvement claim. The main takeaway is that FAHIM is a promising technical prototype for goal-oriented Arabic mobile accessibility, but reproducible paired evaluation, stronger safety and privacy evidence, and testing with blind and visually impaired users are required before deployment claims can be made.

References

[1] Yin-Chen N. Chen et al. “Insight: An LLM-Based Conversational Mobile Assistant for Blind Users”. In: *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 10.1 (2026). DOI: 10.1145/3746044.

[2] Zhe Liu et al. “Unblind Text Inputs: Predicting Hint-text of Text Input in Mobile Apps via LLM”. In: *Proceedings of the CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, 2024. DOI: 10.1145/3613904.3642939.

[3] Mengxi Zhang et al. “Towards Testing the Accessibility of Dynamic Visual Changes in Android Mobile GUI with Multi-Modal LLMs”. In: *ACM Transactions on Computer-Human Interaction* 33.2 (2026). DOI: 10.1145/3793673.

[4] Zezhou Chen et al. "A Large Vision-Language Model based Environment Perception System for Visually Impaired People". In: *arXiv preprint arXiv:2504.18027* (2025). arXiv: 2504.18027 [cs.CV].

[5] Yusra Jadallah Abed Khasawneh and Mohamad Ahmad Saleem Khasawneh. "Availability of Voice-Recognition Devices to Support Visually Impaired Students in Saudi Arabian Universities". In: *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications* 14.3 (2023), pp. 186–193. DOI: 10.58346/JOWUA.2023.I3.014.

[6] Ali Habbi and Mohamed Islam Bettayeb. "Voice Assistants for Blind Users". MA thesis. Kasdi Merbah University – Ouargla, 2025.