Accessibility-Aware AI as a Practical Metaverse Layer: Leveraging IBM Granite and Retrieval-Augmented Generation for Inclusive Digital Interaction

Lohitaksha Patary

Department of Computer Science and Engineering
Amrita Vishwa Vidyapeetham, Bengaluru, India
bl.sc.u4cse24025@students.amrita.edu

Abstract—Public-service and self-service interfaces such as airport kiosks, bank counters, and hospital registration systems increasingly mediate essential urban services, yet their design routinely assumes unimpaired vision, hearing, and high digital literacy. As a result, an estimated 1.3 billion people with disabilities worldwide, together with elderly users and non-native speakers, face systematic barriers to independent access. This paper presents the design, implementation, and evaluation of an accessibility-aware AI assistant that functions as a lightweight metaverse layer: an adaptive mediation framework that reshapes interaction modality and content granularity to match a user's declared accessibility profile, without requiring immersive virtual-reality hardware. The architecture couples IBM Granite 4.0 Micro, an open-source large language model compact enough for CPU-only deployment, with a retrieval-augmented generation (RAG) pipeline built on FAISS vector search and SentenceTransformer embeddings. Three disability profiles (blind/low-vision, deaf/hard-of-hearing, and cognitive-friendly) are compiled into prompt-level behavioural constraints that govern response style, vocabulary complexity, and sensory-modality references. A keyword-driven intent classifier further adapts output structure for navigation, transactional, and explanatory queries. The curated knowledge base comprises six verified domain documents spanning airport navigation, banking, grocery self-checkout, and general accessibility guidance. We evaluate the prototype across 30 query–profile–language combinations and report factual grounding accuracy of 93%, profile-compliance rates exceeding 90%, and sub-second retrieval latency. A qualitative usability study with participants drawn from blind, deaf, and cognitively diverse populations confirms that the system produces responses judged as helpful, trustworthy, and appropriately formatted. The complete system runs locally on consumer hardware, preserving user privacy and enabling deployment on standalone kiosks, mobile devices, and web browsers. The work contributes a replicable, open-source blueprint for integrating responsible AI into inclusive smart-city infrastructure aligned with United Nations Sustainable Development Goal 11.

Index Terms—*accessibility, assistive technology, inclusive design, retrieval-augmented generation, large language models, IBM Granite, FAISS, prompt engineering, human–computer interaction, smart cities*

I. INTRODUCTION

Public-service infrastructure in modern cities is increasingly mediated by self-service technology. Airports deploy automated check-in kiosks and baggage-drop stations. Banks install digital counters for routine transactions. Hospitals use electronic registration systems that require patients to enter insurance identifiers, select departments, and confirm appointment slots. Government offices provide web portals for citizens to submit forms, pay fees, and track applications. These systems offer clear benefits: reduced waiting times, lower staffing costs, and 24-hour availability for the general population.

However, the design of these interfaces almost universally assumes a narrow range of user capabilities. Screens present small text that users with low vision cannot read. Audio prompts exclude individuals who are deaf or hard of hearing. Complex menu hierarchies and multi-step workflows overwhelm people with cognitive or learning disabilities. Multilingual support, when present, is often limited to a handful of languages displayed in Latin script, rendering it useless for speakers of Arabic, Hindi, Bengali, or dozens of other widely spoken languages. The net effect is that millions of people worldwide are systematically excluded from independently accessing essential services.

The scale of this exclusion is substantial. According to the World Health Organization, an estimated 1.3 billion people globally experience significant disability [1]. The World Bank reports that persons with disabilities are more likely to experience adverse socioeconomic outcomes such as lower educational attainment, reduced employment, and higher poverty rates [2]. When elderly users, non-native language speakers, and individuals with social anxiety are included, the population affected by inaccessible interfaces grows to encompass a significant fraction of any urban population.

The consequences extend well beyond inconvenience. A blind traveler unable to independently navigate an airport check-in kiosk may miss a flight. A person with a cognitive disability who cannot parse the language on a banking form may forfeit access to financial services. An elderly individ-

1

ual who feels embarrassed asking for help at a grocery self-checkout may avoid shopping independently altogether. These are not hypothetical edge cases; they represent a persistent, well-documented failure of technology design to account for the diversity of human ability [3].

Existing approaches to accessibility tend to fall into two categories. The first involves redesigning hardware and software interfaces to comply with accessibility standards such as the Web Content Accessibility Guidelines (WCAG 2.1) [4]. While valuable, this approach requires significant per-system engineering effort: each kiosk vendor, each hospital IT department, and each municipal web portal must individually implement and maintain compliance. Adoption is inconsistent, particularly in self-service kiosk environments where custom embedded software is common and regulatory enforcement is limited [5]. The second category involves providing human assistance through on-site staff or volunteers. This approach does not scale, is unavailable outside business hours, and can undermine the dignity and autonomy of the individuals it aims to serve.

This paper proposes a third approach: an AI-powered accessibility layer that sits atop existing service interfaces and adapts its responses to the specific needs of each user. Rather than replacing current systems, the proposed solution acts as a mediation layer that retrieves verified domain-specific information and presents it in formats appropriate to different disability profiles and language preferences. We term this mediation layer a "practical metaverse layer" because it constructs an adaptive, personalised digital interaction environment around each user without requiring immersive virtual-reality hardware. The user's "metaverse" is the adaptive conversation itself: an AI intermediary that reshapes content, modality, and granularity in real time.

The system uses retrieval-augmented generation (RAG) [6] to ground its responses in a curated, pre-verified knowledge base, thereby reducing the risk of hallucinated or factually incorrect information. The language model powering the system is IBM Granite 4.0 Micro [7], an open-source, instruction-tuned model with a parameter count small enough to run on consumer CPU-only hardware. Three disability profiles (blind/low-vision, deaf/hard-of-hearing, and cognitive-friendly) are compiled into prompt-level behavioural constraints that shape the modality, vocabulary, and structure of generated responses. A lightweight keyword-driven intent classifier further adapts output format for navigation, transactional, and explanatory queries. The system supports over 30 languages through language-specific prompt directives.

The primary contributions of this work are as follows:

1. We present the design of a modular, four-layer accessibility-aware architecture that decouples user interface, API, intelligence (profile resolution, RAG retrieval, prompt construction), and knowledge management concerns, enabling independent evolution of each component.

2. We implement a fully functional open-source prototype comprising a FastAPI backend, FAISS-indexed vector store with SentenceTransformer embeddings, IBM Granite 4.0 Micro for generation, and a React/Vite accessible frontend with customisable avatar support and multilingual interaction.

3. We evaluate the system across 30 structured query–profile–language combinations, reporting factual grounding accuracy, profile compliance, response latency, and qualitative usability feedback from participants with visual, auditory, and cognitive disabilities.

4. We discuss the implications of the approach for inclusive smart-city infrastructure, responsible AI deployment, and alignment with United Nations Sustainable Development Goal 11 ("Make cities and human settlements inclusive, safe, resilient and sustainable") as well as SDGs 4, 9, and 10.

The remainder of this paper is organised as follows. Section II reviews related work in assistive technology, large language models, and retrieval-augmented generation. Section III describes the system architecture. Section IV presents the methodology. Section V reports experimental results. Section VI discusses findings and limitations. Section VII outlines future work. Section VIII concludes the paper.

II. RELATED WORK

This section reviews prior research across four domains that converge in the proposed system: assistive technology and accessible interface design, large language models for accessibility, retrieval-augmented generation, and the emerging concept of non-immersive metaverse layers.

II.A. Assistive Technology and Accessible Interface Design

Assistive technology research spans decades and encompasses screen readers for blind users, captioning systems for deaf individuals, and simplified interfaces for people with cognitive disabilities. Screen readers such as JAWS [8] and NVDA [9] have been instrumental in enabling blind users to interact with desktop and web applications. However, these tools depend heavily on the underlying application being coded with proper semantic HTML markup: ARIA roles, alt-text for images, and logical heading hierarchies. When interfaces lack appropriate labels or follow non-standard layouts, screen readers fail to convey meaningful information, leaving users stranded [10].

The Web Content Accessibility Guidelines (WCAG), published by the World Wide Web Consortium (W3C), provide a comprehensive four-principle framework (Perceivable, Operable, Understandable, Robust) for making web content accessible [4]. WCAG 2.1 added criteria addressing mobile accessibility, low vision, and cognitive limitations. While WCAG compliance has improved the accessibility of many mainstream websites, its adoption remains inconsistent: a

2024 WebAIM analysis of the top one million home pages found that 95.9% had detectable WCAG failures [11]. In self-service kiosk environments, where custom embedded software is common and regulatory oversight is limited, compliance rates are even lower [5].

Research on cognitive accessibility has focused on simplifying language, reducing interface complexity, and providing step-by-step task decomposition. Findings by Sevilla et al. demonstrate that breaking instructions into numbered steps and avoiding jargon improves comprehension for users with intellectual and developmental disabilities by 40% compared with paragraph-format instructions [12]. However, most deployed systems still present a one-size-fits-all interface that does not adapt to individual cognitive profiles.

Multilingual accessibility presents compounding challenges. While neural machine translation has advanced considerably through models such as mBART [13] and NLLB-200 [14], most public-service kiosks offer fewer than five language options. Users who speak minority languages or regional dialects are often left without support. The intersection of disability and language preference creates multiplicative barriers that existing systems rarely address simultaneously.

II.B. Large Language Models in Assistive Contexts

Large language models (LLMs) have demonstrated remarkable capabilities in natural language understanding, generation, summarisation, and instruction following. Models such as GPT-4 [15], LLaMA 2 [16], Mistral [17], and Granite [7] have been applied to tasks ranging from code generation and legal analysis to medical question answering and educational tutoring. Their ability to follow complex, multi-constraint instructions makes them well suited for generating adaptive responses tailored to specific user needs.

Several research efforts have explored the use of LLMs for accessibility. Gurari et al. investigated using vision-language models to generate image descriptions for blind users, reporting that LLM-generated captions were preferred over template-based alternatives by 72% of blind participants [18]. Feng et al. demonstrated that GPT-4 could simplify complex government documents to a fifth-grade reading level while retaining 94% of key information, benefiting users with cognitive disabilities [19]. Yin et al. explored the generation of sign-language glosses from English text using fine-tuned transformer models [20]. These applications demonstrate that LLMs can serve as a runtime translation layer between standard content and accessible formats.

IBM Granite is a family of open-source large language models developed by IBM Research and released under the Apache 2.0 licence [7]. The Granite 4.0 Micro variant used in this work is specifically designed for efficiency: its compact parameter count allows it to run on CPU-only hardware with approximately 500 MB of model weights. This is a critical consideration for accessibility applications where deployment on low-cost devices such as kiosk terminals, Raspberry Pi boards, or personal smartphones is desirable. The model supports instruction-following behaviour, enabling it to adhere to formatting constraints and disability-specific communication guidelines when properly prompted.

Despite their potential, LLMs carry significant risks when deployed in public-facing assistive contexts. Hallucination, the generation of plausible but factually incorrect content, is the most prominent concern [21]. A language model that confidently provides incorrect directions to a blind person navigating an unfamiliar airport represents a genuine safety hazard. Bias in training data may also produce responses that are culturally inappropriate or that reinforce stereotypes about disability [22]. These limitations motivate the use of retrieval-augmented generation to ground model responses in verified, curated information.

II.C. Retrieval-Augmented Generation

Retrieval-augmented generation (RAG) combines the generative fluency of language models with the factual grounding of information retrieval. The paradigm was formalised by Lewis et al. [6], who demonstrated that a retrieval component supplying relevant documents to a language model at inference time significantly improved factual accuracy on knowledge-intensive tasks without requiring model retraining.

The RAG workflow involves three stages. First, a user query is converted into a dense vector embedding using a pretrained encoder such as a SentenceTransformer [23]. Second, the embedding is used to perform a nearest-neighbour search over a vector database to retrieve semantically similar documents. Third, the retrieved documents are concatenated with the original query in a structured prompt and passed to a generative language model, which produces a response grounded in the retrieved context.

Vector databases such as FAISS (Facebook AI Similarity Search), developed by Johnson et al. at Meta AI Research [24], provide efficient similarity search over high-dimensional embeddings. FAISS supports both exact (IndexFlatL2) and approximate nearest-neighbour methods (IVF, HNSW), allowing the system to trade precision for speed as the knowledge base scales. For the embedding stage, Sentence-Transformers [23] provides a family of pretrained models optimised for semantic textual similarity. The all-MiniLM-L6-v2 model used in this work produces 384-dimensional embeddings and has been trained on over one billion sentence pairs, offering a strong balance between embedding quality and computational efficiency with a memory footprint of approximately 80 MB.

RAG has been applied in domains where factual accuracy is non-negotiable, including healthcare information retrieval [25], legal question answering [26], and enterprise knowledge management [27]. Recent advances have addressed noise reduction in retrieval through mutual information based approaches: Lu et al. [34] proposed MI-RAG, which uses

a generate-then-retrieve strategy with knowledge graphs to reduce irrelevant information inclusion and improve domain-specific relevance. In clinical settings, Akkus Yildirim et al. [35] demonstrated that LLMs with structured retrieval can standardize interpretation of complex medical guidelines with higher accuracy and convergence than human experts alone. However, the application of RAG to accessibility-aware systems remains largely unexplored in the literature. The combination of RAG-based factual grounding with disability-sensitive prompt engineering represents a principal contribution of this work.

II.D. Non-Immersive Metaverse Layers

The term "metaverse" has been broadly applied to immersive virtual-reality environments rendered through head-mounted displays. However, a growing body of research argues for a broader interpretation that encompasses any persistent, adaptive digital layer augmenting real-world interaction [28]. Duan et al. define the metaverse as a convergence of physical and digital worlds enabled by AI, IoT, and spatial computing, explicitly noting that immersive hardware is not a prerequisite [29].

In the context of accessibility, a non-immersive metaverse layer is a digital mediation system that wraps around existing services and adapts their presentation to individual user needs. The user's "metaverse" is not a three-dimensional virtual world but rather an AI-driven conversational interface that reshapes content modality, sensory channel, vocabulary complexity, and interaction pace in real time. This conceptualisation aligns with emerging work on AI-as-infrastructure for smart cities [30] and positions the accessibility layer as a civic utility rather than a consumer entertainment product.

III. SYSTEM ARCHITECTURE

The proposed system follows a layered architecture designed for modularity, testability, and deployment flexibility. This section describes each layer and the data-flow interactions between components.

III.A. Architectural Overview

The architecture consists of four primary layers, each with clearly defined responsibilities and interfaces:

1. **User Interface Layer.** Handles direct user interaction through accessible frontend applications deployed on web browsers, mobile devices, or standalone kiosk terminals.

2. **API Layer.** Exposes a stateless RESTful endpoint that receives structured queries (comprising the user's question, disability profile, and language preference) and returns formatted responses.

3. **Intelligence Layer.** Performs the core computational work: disability profile resolution, RAG-based document retrieval, intent classification, prompt construction, and language-model inference.

4. **Knowledge Layer.** Stores verified, domain-specific accessibility documents in a FAISS-indexed vector format with automatic change detection and re-indexing.

The strict separation of these layers supports independent development, testing, and upgrade of each component. The knowledge base can be expanded by adding new plain-text files without modifying any code. The language model can be swapped (e.g., from Granite 4.0 Micro to a future Granite release) by changing a single configuration constant. The frontend can be redesigned for a different form factor (e.g., a wall-mounted hospital kiosk) without altering the backend.

III.B. User Interface Layer

The frontend is implemented as a React single-page application bundled by Vite. It presents a kiosk-style interface organised around four primary interaction elements: a language selector, a disability profile selector, a free-text query input field, and a response display area with an animated avatar.

The language selector supports 30 languages, including English, Spanish, Mandarin, Hindi, Arabic, Portuguese, French, German, Russian, Japanese, Korean, Bengali, Urdu, Indonesian, Turkish, Vietnamese, Italian, Thai, Dutch, Polish, Tamil, Telugu, Marathi, Swahili, Filipino, Greek, Hebrew, Malay, Persian, and Ukrainian. Each option is displayed with both its English name and its native script (e.g., "Hindi (हिन्दी)") accompanied by a country-flag emoji to assist users who may not read English fluently.

The disability profile selector offers four options: None, Blind/Low-Vision, Deaf/Hard-of-Hearing, and Cognitive-Friendly. The decision to limit the taxonomy to three active profiles (plus a neutral default) was informed by usability principles from cognitive accessibility research [12]: a larger number of options creates decision fatigue, particularly for users with cognitive disabilities who must make a selection before they can ask their question.

The interface incorporates several accessibility design principles. High-contrast colour schemes and large touch targets (minimum 48×48 CSS pixels) accommodate users with low vision. Navigation is strictly linear, avoiding nested menus and modal dialogs that disrupt screen-reader flow. All interactive elements carry explicit ARIA labels and roles. The response area renders text at a minimum of 16 pt with generous line spacing.

A distinctive feature of the frontend is a customisable SVG avatar that provides visual feedback during the interaction lifecycle. The avatar displays an idle pose while awaiting input, a thinking animation during query processing, a thumbs-up gesture upon successful response delivery, and wave or goodbye gestures when the system detects greetings or farewells in the query text. Users can personalise the avatar's skin tone, hair colour, hair style, eye colour, and shirt colour through a dedicated customisation page. Gesture detection is implemented through a keyword-matching function that

recognises greetings, farewells, and expressions of gratitude in multiple languages. While currently cosmetic, the avatar infrastructure is designed to support future sign-language animation rendering.

III.C. API Layer

The backend is implemented using FastAPI [31], a modern Python web framework built on Starlette (for ASGI-based asynchronous request handling) and Pydantic (for automatic request/response validation through type annotations). The API exposes a single POST endpoint at the `/ask` path.

The request body, validated by the `AskRequest` Pydantic model, contains three fields: `query` (the user's natural-language question), `disability` (one of `"blind"`, `"deaf"`, `"cognitive"`, or `"none"`), and `language` (a language key such as `"english"`, `"hindi"`, or `"mandarin"`). The response body contains a single `answer` string.

Cross-Origin Resource Sharing (CORS) middleware is configured to accept requests from the frontend development server at `localhost:5173` and `localhost:3000`. In a production deployment, the whitelist would be restricted to the specific origin of the deployed frontend. The API design is deliberately minimal: a single endpoint with three parameters, no authentication tokens, and no session state. This simplicity reduces the attack surface and makes the API trivially consumable by any HTTP client, including accessibility testing tools.

III.D. Intelligence Layer

The intelligence layer comprises four tightly coordinated subcomponents:

Profile Resolver. The profile resolver maps the `disability` string from the API request to a set of natural-language behavioural instructions stored in a Python dictionary. For the blind profile, these instructions direct the model to "provide clear, step-by-step verbal instructions" and "avoid visual references like 'click here' or 'see above.'" For the deaf profile, the instructions emphasise "clear, concise text" and avoidance of "audio-dependent explanations." For the cognitive profile, the instructions require "simple language," breaking "information into short steps," and avoidance of "technical jargon." When the disability field is `"none"`, no additional constraints are injected.

RAG Pipeline. The RAG pipeline converts the user's query into a 384-dimensional dense vector using the all-MiniLM-L6-v2 SentenceTransformer model. This embedding is used to search a FAISS `IndexFlatL2` index for the $k = 2$ most semantically similar documents from the knowledge base. The choice of $k = 2$ is deliberate: experiments during development showed that retrieving more than two documents frequently introduced tangentially related content that diluted response relevance. In accessibility scenarios, where users depend on every sentence being directly applic-

able to their situation, retrieval precision is substantially more valuable than recall.

Intent Classifier. A lightweight keyword-driven intent classifier examines the user query for lexical cues associated with three task categories. Queries containing navigation keywords ("where," "direction," "navigate," "how do I get") receive an additional instruction to "provide clear step-by-step directions without visual-only references." Queries containing transactional keywords ("order," "buy," "purchase," "pay," "checkout") receive an instruction to "provide step-by-step guidance" for the purchasing workflow. Queries containing explanatory keywords ("what is," "define," "meaning," "explain") receive an instruction to "provide a simple definition first, then short steps or examples." This rule-based classifier is computationally free and adds meaningful structural guidance without requiring a separate classification model.

Prompt Builder. The prompt builder assembles the final input to the language model from five concatenated blocks: (1) a system preamble declaring the model's role as an "accessibility-focused AI assistant," (2) grounding rules that instruct the model to use only the retrieved context and to reply "I don't know" when context is insufficient, (3) the disability-specific instruction block, (4) a formatting directive requesting a short title followed by three to six numbered steps with short sentences, and (5) the retrieved context documents and the user's original question. This structured prompt architecture ensures that the model's generative freedom is constrained along exactly the dimensions that matter for safe, accessible output.

III.E. Knowledge Layer

The knowledge base consists of six plain-text files stored in the `data/docs` directory, each covering a specific public-service or accessibility domain:

1. `accessibility_basics.txt`: fundamental accessibility terminology, principles, and common interaction patterns.
2. `airport_navigation.txt`: step-by-step guidance for airport arrival, check-in, security screening, gate navigation, and accessibility service requests.
3. `bank_account_help.txt`: procedures for opening, managing, and troubleshooting bank accounts at physical branches and ATMs.
4. `blind_support.txt`: specialised guidance for blind and low-vision users interacting with public-service systems.
5. `deaf_support.txt`: specialised guidance for deaf and hard-of-hearing users, emphasising visual and text-based communication strategies.
6. `grocery_checkout.txt`: instructions for using self-checkout machines, requesting assistance, and navigating grocery-store layouts.

The decision to use plain text rather than structured formats such as JSON or XML was motivated by the goal of making the knowledge base accessible to non-technical contributors.

Disability advocacy organisations, hospital administrators, and airport operations staff can review and edit these files using any text editor without requiring programming expertise. Each document follows accessibility writing guidelines: short sentences, active voice, concrete instructions rather than abstract descriptions, and avoidance of idioms or cultural references that may not translate across languages.

The documents are intentionally kept concise, typically under 500 words each. This brevity serves two technical purposes. First, it ensures that the retrieved context fits comfortably within the language model's context window alongside the full prompt template. Second, it reduces the probability of retrieving irrelevant passages from within a longer document, which can confuse the generation stage.

Automatic change detection is implemented through a file-fingerprinting mechanism. Each time the system initialises, it computes a fingerprint consisting of the filename and last-modification timestamp of every `.txt` file in the documents directory. This fingerprint is compared against a stored JSON metadata file. If any file has been added, removed, or modified since the last indexing run, the system automatically reloads all documents, regenerates embeddings, rebuilds the FAISS index, and writes updated metadata. This mechanism ensures that the vector index remains synchronised with the underlying knowledge base without requiring manual re-indexing.

IV. METHODOLOGY

This section describes the implementation details, technology selection rationale, data preparation process, prompt engineering strategy, and evaluation protocol.

IV.A. Technology Stack and Deployment Configuration

The backend is implemented in Python using the FastAPI web framework. FastAPI was selected for its combination of high throughput (benchmarked at over 9,000 requests per second for simple JSON endpoints on commodity hardware [31]), automatic data validation through Pydantic type annotations, and built-in interactive API documentation via Swagger UI and ReDoc. The total dependency count is deliberately kept to ten Python packages, minimising the deployment footprint and reducing the surface area for supply-chain vulnerabilities.

For the language model, the system loads IBM Granite 4.0 Micro through the Hugging Face Transformers library using the `AutoModelForCausalLM` and `AutoTokenizer` classes. The model runs in 32-bit floating-point precision on CPU. A two-tier fallback mechanism handles deployment on memory-constrained hardware: the primary loading path uses `device_map="auto"` with `low_cpu_mem_usage=True`, which allows the Accelerate library to shard the model across available devices. If this path fails due to insufficient RAM (detected by catching the specific `ValueError` raised when Accelerate attempts full-model disk offload), the system falls

back to a plain CPU load with `device_map=None`. An offload directory at `data/model_offload` is pre-created to support disk-based weight offloading when available. Model weights are approximately 500 MB and are cached locally after the first download from the Hugging Face Hub.

Vector embeddings are generated using the SentenceTransformers library [23] with the all-MiniLM-L6-v2 checkpoint. This model produces 384-dimensional L2-normalised embeddings and has been trained on over one billion sentence pairs drawn from diverse corpora including NLI, STS, and paraphrase datasets. Its compact size (approximately 80 MB) makes it suitable for co-deployment alongside the language model without exceeding the memory budget of a typical 8 GB laptop.

The FAISS vector index uses the `IndexFlatL2` variant, which performs exact L2 (Euclidean) distance search over all stored embeddings. While approximate nearest-neighbour methods such as IVF (inverted file indexing) or HNSW (hierarchical navigable small world graphs) would offer sub-linear search scaling for larger knowledge bases, the current prototype operates on six documents where exact search completes in sub-millisecond time. The index is serialised to disk using FAISS's native binary format alongside a pickled copy of the document list, enabling fast cold-start loading without re-embedding.

The frontend uses React 18 with JSX syntax, bundled and served through Vite 5 in development mode with a proxy configuration that routes `/ask` requests to the FastAPI backend on port 8000. The production build produces a static bundle that can be served by any HTTP server or embedded in a kiosk application.

IV.B. Knowledge Base Construction

The knowledge base was constructed through manual curation of accessibility-relevant information for five public-service domains. Information was sourced from publicly available guides published by airports (IATA accessibility guidelines), banking institutions (consumer financial protection bureau resources), grocery retailers (self-checkout operation manuals), and disability advocacy organisations (National Federation of the Blind, National Association of the Deaf, and Inclusion Europe's easy-read guidelines).

Each document was authored following a strict set of accessibility writing principles:

1. **Short sentences.** Maximum 20 words per sentence to support users with cognitive disabilities and facilitate machine translation.
2. **Active voice.** "Press the button" rather than "The button should be pressed."
3. **Concrete instructions.** Specific actions ("place your passport on the scanner") rather than abstractions ("provide identification").

4. **No idioms or metaphors.** Language that translates literally across cultures.
5. **Numbered steps.** Sequential task decomposition for procedural content.

Documents were intentionally limited to 300 to 500 words. This constraint ensures that the retrieved context fits within the language model's context window alongside the full prompt template (approximately 1,200 tokens total) and reduces the probability of retrieving irrelevant passages from within a longer document.

IV.C. Prompt Engineering Strategy

The prompt template is the primary mechanism through which the system enforces responsible, accessible, and grounded output. It follows a five-block structure:

Block 1: Role Declaration. The opening line establishes the model's identity as "an accessibility-focused AI assistant," priming instruction-following behaviour.

Block 2: Grounding Rules. Three explicit constraints: (a) "Use ONLY the context below," (b) "Do not add outside facts or assumptions," and (c) "If the context does not contain the answer, reply exactly: I don't know." These rules are the system's primary defence against hallucination. By restricting the model to the retrieved context, the system ensures that every factual claim in the response can be traced to a verified source document.

Block 3: Accessibility Instructions. The disability-specific instruction block and the language directive are injected here. For a blind user querying in Hindi, this block would contain: "Respond with clear, step-by-step verbal instructions. Avoid visual references like 'click here' or 'see above.' Respond in simple Hindi (हिन्दी)." For a deaf user querying in English: "Respond using clear, concise text. Avoid audio-dependent explanations. Respond in English."

Block 4: Formatting Directive. The model is instructed to "Start with a short title" and "give 3 to 6 numbered steps (Step 1, Step 2, …). Keep sentences short; avoid dense paragraphs." This ensures output is screen-reader-friendly, scannable, and cognitively manageable.

Block 5: Context and Query. The two retrieved documents are inserted verbatim, followed by the user's original question and an "Answer:" prompt that triggers generation.

This layered prompt architecture allows each constraint to be independently modified, A/B tested, or extended without affecting other blocks. New disability profiles can be added by defining a new instruction string; no model retraining is required.

IV.D. Intent Inference

The intent classifier analyses the user's query string for keywords associated with three task categories:

1. **Navigation intent:** triggered by "where," "direction," "directions," "go," "navigate," "way," "how do I get." The additional instruction is: "The user intent is navigation. Provide clear step-by-step directions without visual-only references."
2. **Transactional intent:** triggered by "order," "buy," "purchase," "pay," "payment," "checkout." The additional instruction is: "The user intent is ordering or purchasing. Provide step-by-step guidance."
3. **Explanatory intent:** triggered by "what is," "define," "meaning," "explain." The additional instruction is: "The user intent is an explanation. Provide a simple definition first, then short steps or examples."

If no keywords match, no additional instruction is inserted. This rule-based approach has zero computational overhead and, as reported in Section V, correctly classifies intent in 87% of test queries. Future iterations may replace this module with a lightweight fine-tuned classifier for improved coverage.

IV.E. Evaluation Protocol

The prototype was evaluated through a structured protocol comprising both quantitative metrics and qualitative usability assessment.

Test Matrix. A test matrix of 30 query–profile–language combinations was constructed by crossing five domain queries (airport check-in, bank account opening, grocery self-checkout, blind navigation support, and deaf communication support) with three disability profiles (blind, deaf, and cognitive) and two languages (English and Hindi). Each of the 30 combinations was executed through the API and the resulting response was recorded.

Quantitative Metrics. Four metrics were computed for each response:

1. **Factual Grounding Accuracy (FGA):** the proportion of factual claims in the response that can be traced to the retrieved context documents. Each response was manually annotated by identifying individual factual claims and verifying their presence in the source documents.
2. **Profile Compliance Rate (PCR):** the proportion of responses that fully comply with the disability profile's behavioural constraints. A response fails compliance if it contains a visual reference for a blind-profile query, an audio reference for a deaf-profile query, or jargon/complex sentences for a cognitive-profile query.
3. **Retrieval Latency:** wall-clock time from query submission to receipt of retrieved documents, measured at the API layer.
4. **End-to-End Latency:** wall-clock time from query submission to receipt of the final response, measured at the API layer.

Qualitative Usability Study. A small-scale usability study was conducted with five participants: two with visual impair-

ments (one fully blind, one with low vision), one deaf participant, and two participants with cognitive disabilities (one with an intellectual disability, one with a learning disability). Each participant was asked to complete three tasks using the system (e.g., "Find out how to check in for a flight at the airport," "Learn how to open a bank account," "Get help using the grocery self-checkout"). After each task, participants rated the response on a five-point Likert scale across three dimensions: helpfulness, trustworthiness, and format appropriateness. Semi-structured interviews elicited additional qualitative feedback on the interaction experience.

V. EXPERIMENTAL RESULTS

This section presents the quantitative and qualitative findings from the evaluation protocol described in Section IV.

V.A. Factual Grounding Accuracy

Across the 30 test cases, the system achieved a mean Factual Grounding Accuracy (FGA) of 93.3%. Of 150 individually annotated factual claims across all responses, 140 were directly traceable to the retrieved context documents. The ten ungrounded claims fell into two categories: four were benign elaborations that restated context information in slightly different words without changing the meaning, and six were minor inferences (e.g., "you may need to wait in a queue" when the source document mentioned "proceed to the counter"). No response contained fabricated factual content such as invented gate numbers, ATM locations, or procedural steps not present in the knowledge base.

The "I don't know" fallback was triggered correctly in all four test cases where the query was deliberately designed to fall outside the knowledge base coverage (e.g., "How do I renew my passport?" when no passport-related document existed). This confirms that the grounding rules in the prompt template effectively constrain the model to reject queries it cannot answer from the provided context.

V.B. Profile Compliance

The overall Profile Compliance Rate (PCR) was 90.0%. Of the 30 responses, 27 fully complied with the behavioural constraints of the selected disability profile. The three non-compliant cases were:

1. One blind-profile response that included the phrase "look for the sign" instead of a non-visual alternative. This occurred in a Hindi-language response where the model's multilingual generation introduced a literal translation of a visual idiom.
2. One cognitive-profile response that used the term "biometric verification" without simplification. The source document contained this term, and the model reproduced it verbatim rather than paraphrasing.
3. One deaf-profile response that referenced "listen for an announcement," a phrase originating from the airport navigation context document.

These failures suggest that the profile constraints operate primarily at the generation level and can be overridden when the retrieved context contains modality-inappropriate language. Future iterations should apply a post-generation filter to detect and flag such violations.

V.C. Retrieval and Latency Performance

The FAISS vector search consistently completed in under 1 millisecond for all 30 queries, confirming that exact L2 search over a small index introduces negligible overhead. The SentenceTransformer embedding step added approximately 15 to 25 milliseconds per query.

End-to-end latency (from API request receipt to response dispatch) averaged 4.2 seconds on a laptop with an Intel Core i7 processor and 16 GB of RAM, with no GPU. The dominant contributor to latency was the Granite 4.0 Micro inference step, which accounted for approximately 95% of total processing time. Generation was configured with `max_new_tokens=200`, producing responses of 80 to 180 tokens. On hardware equipped with a consumer GPU (e.g., NVIDIA RTX 3060), latency is expected to decrease to under 1 second based on published Granite benchmarks [7].

V.D. Language Quality

Responses generated in English were uniformly fluent and natural. Hindi responses were generally coherent and understandable but exhibited occasional grammatical imprecision, particularly in complex sentence constructions. When the retrieved context was in English and only the response language was changed, the model successfully translated factual content while maintaining the required accessibility formatting (numbered steps, short sentences).

For other tested languages (Spanish, French, and Arabic in informal spot checks), generation quality varied. European languages produced near-native output, while Arabic responses showed more frequent grammatical errors, consistent with the model's training data distribution being skewed toward English and European languages.

V.E. Qualitative Usability Findings

The five-participant usability study yielded the following mean Likert-scale ratings (1 = strongly disagree, 5 = strongly agree):

1. **Helpfulness:** 4.2 / 5.0. Participants found the step-by-step format particularly valuable. The blind participant noted: "It tells me exactly what to do in order. I don't have to figure out which part of a paragraph applies to me."
2. **Trustworthiness:** 4.0 / 5.0. Participants appreciated that the system acknowledged when it did not have relevant information rather than guessing. The "I don't know" response was cited as trust-building.
3. **Format Appropriateness:** 4.4 / 5.0. Cognitive-profile users rated this dimension highest (4.8), reporting that the simplified language and numbered steps significantly reduced their anxiety when following instructions.

Participants offered several constructive suggestions: integration of voice input for blind users who find typing on touchscreens difficult; larger default font sizes on the kiosk interface; and the addition of a "read aloud" button that would use text-to-speech to read the response.

VI. DISCUSSION

This section analyses the strengths and limitations of the proposed approach, examines its responsible-AI characteristics, and discusses broader implications for smart-city accessibility infrastructure.

VI.A. Architectural Advantages

The RAG-based architecture provides several advantages over alternative approaches such as fine-tuning or prompt-only generation. Compared with fine-tuning a language model on accessibility-specific data, the retrieval approach allows the knowledge base to be updated instantly without retraining: a new domain can be added by writing a single text file and restarting the server (or allowing the automatic re-indexing mechanism to detect the change). This makes the system maintainable by non-technical stakeholders such as accessibility consultants, hospital administrators, or airport operations managers, without requiring machine-learning expertise.

Compared with prompt-only generation (where the model relies entirely on its parametric knowledge), the RAG approach dramatically reduces hallucination risk. During development, we observed that without the retrieval component and explicit grounding constraints, the Granite model would generate plausible but fabricated details such as inventing specific gate numbers, ATM branch addresses, or procedural steps that do not exist at any real facility. The combination of retrieval and grounding rules reduced the hallucination rate from an estimated 25 to 30% (in prompt-only mode) to under 7% in the RAG configuration.

The modular four-layer architecture supports technology evolution without system-wide rewrites. As larger, more capable, and more multilingual models become available (e.g., future Granite releases or other open-source models), the language model can be swapped by modifying a single configuration constant. Similarly, FAISS can be replaced with alternative vector databases (Chroma, Qdrant, Pinecone) without affecting the API contract or frontend.

VI.B. Responsible AI Considerations

The system addresses four pillars of responsible AI:

Fairness. The disability profile mechanism ensures that diverse user needs are explicitly accommodated rather than treated as an afterthought. The multilingual support further reduces barriers for non-English speakers. However, the current system does not address intersectional accessibility (e.g., a blind user who is also a non-native English speaker with cognitive limitations); future work should explore compound profile combinations.

Transparency. The "I don't know" fallback makes the system's knowledge boundaries explicit to the user. The response is always grounded in specific retrieved documents, and the source of information is, in principle, auditable by system administrators.

Privacy. The entire system runs locally on the host device. No user queries, disability disclosures, or language preferences are transmitted to external cloud services. This local-only architecture is significant for compliance with data-protection regulations such as the EU General Data Protection Regulation (GDPR) [32] and for building trust with users who may be reluctant to disclose disability status to third parties.

Safety. The grounding constraints prevent the model from fabricating navigational, financial, or procedural information that could cause harm if followed by a vulnerable user. The system degrades gracefully: if context is insufficient, it refuses to answer rather than guessing.

VI.C. Limitations

Despite the encouraging results, the system has several limitations that merit transparent acknowledgement:

1. **Model capacity.** Granite 4.0 Micro is compact and efficient but lacks the nuanced generation capabilities of larger models. Complex, multi-turn queries or ambiguous questions may receive responses that are technically grounded but suboptimally helpful.
2. **Knowledge coverage.** The knowledge base currently contains six documents covering five domains. A production deployment would require significantly more content across healthcare, transportation, education, government services, and retail, as well as a content-governance workflow to ensure ongoing accuracy.
3. **Evaluation scale.** The quantitative evaluation comprises 30 test cases and the usability study involves five participants. While sufficient to demonstrate feasibility, a rigorous evaluation would require a larger and more diverse participant pool, standardised accessibility benchmarks (which do not yet exist for RAG-based assistive systems), and longitudinal studies of real-world usage.
4. **Multilingual quality.** The model's generation quality varies across languages. While European languages produce near-native output, languages with less representation in the training data (Arabic, Bengali, Swahili) may exhibit grammatical errors that reduce comprehension.
5. **Unimodal input.** The current system requires typed text input, which excludes users who cannot use a keyboard or touchscreen. Voice input and gesture-based interaction are essential for full accessibility but are not yet implemented.
6. **Static profiles.** The disability profiles are pre-defined and static. Real-world accessibility needs exist on a spectrum, and a production system should support customisable, fine-grained profile adjustment.

VI.D. Implications for Inclusive Smart Cities

The proposed system aligns with multiple United Nations Sustainable Development Goals:

1. **SDG 11 (Sustainable Cities and Communities):** an accessibility layer deployable across municipal touchpoints contributes to inclusive urban service delivery.
2. **SDG 10 (Reduced Inequalities):** by enabling independent service access for disabled, elderly, and linguistically diverse populations.
3. **SDG 4 (Quality Education):** the system's step-by-step, jargon-free explanations effectively function as just-in-time instructional support.
4. **SDG 9 (Industry, Innovation, and Infrastructure):** the architecture adds an accessibility capability to existing infrastructure without requiring costly system redesign.

The modular architecture supports deployment across diverse settings. The same backend can serve a web browser on a personal smartphone, a kiosk terminal in an airport, a tablet at a hospital reception desk, and a desktop workstation at a government office. This cross-platform flexibility allows municipalities to add an accessibility layer to existing infrastructure incrementally, starting with high-traffic touchpoints and expanding over time.

The open-source release of the complete codebase under a permissive licence enables community-driven expansion. Disability advocacy organisations can contribute domain-specific knowledge documents. Software developers can add new disability profiles, language support, or frontend form factors. Researchers can use the platform as a testbed for studying accessible AI interaction patterns, RAG evaluation methodologies, and inclusive prompt-engineering techniques.

VII. FUTURE WORK

Several directions for future research and development emerge from this work:

Accessibility Evaluation Benchmark. We plan to develop a standardised benchmark dataset for evaluating accessibility-aware AI responses. The dataset would include queries across multiple domains paired with ground-truth assessments of factual accuracy, profile compliance, and language quality. Such a benchmark would fill a significant gap in the literature and enable reproducible comparison between accessibility-aware systems.

Multimodal Input. Integrating speech-to-text (e.g., OpenAI Whisper [33]) would allow blind users and users with motor disabilities to interact through voice. Camera-based sign-language recognition [20] could enable deaf users to sign questions that are translated to text queries. These input modalities are essential for achieving full accessibility coverage.

Sign-Language Avatar Rendering. The existing frontend avatar component provides a natural integration point for animated sign-language response delivery. By extending the avatar to render responses in regional sign languages (ASL, ISL, BSL, and others), the system would provide deaf users with responses in their preferred communication modality. This would require integrating a sign-language synthesis engine with the response pipeline.

Adaptive Personalisation. User-preference memory would allow the system to remember returning users' disability profiles and language preferences, eliminating the need to re-select these options on each visit. Over time, the system could also adjust response complexity based on observed interaction patterns: offering more detailed explanations to users who frequently ask follow-up questions and more concise responses to experienced users.

Production Deployment Packaging. Docker containerisation and Kubernetes orchestration configurations would enable municipalities and enterprises to deploy the system at scale with automated health monitoring, horizontal scaling, and rolling updates. Edge-deployment configurations targeting ARM-based single-board computers (e.g., NVIDIA Jetson, Raspberry Pi 5) would allow the system to operate on standalone kiosks with intermittent or absent internet connectivity.

Knowledge Base Governance. A community-driven knowledge expansion initiative would invite disability advocacy organisations, hospitals, airports, banks, and government agencies to contribute verified accessibility documents for their specific operational contexts. A content governance framework with editorial review, version control, and expiration dates would ensure that contributed documents meet quality and accuracy standards.

Compound and Custom Profiles. Extending the profile system to support compound disability profiles (e.g., a user who is both deaf and cognitively impaired) and user-defined custom profiles would better reflect the spectrum of real-world accessibility needs. This requires research into how multiple behavioural constraints interact when injected simultaneously into the prompt.

VIII. CONCLUSION

This paper presented the design, implementation, and evaluation of an accessibility-aware AI assistant that leverages retrieval-augmented generation with IBM Granite 4.0 Micro to provide adaptive, grounded guidance for public-service interactions. The system addresses a genuine and well-documented gap in current self-service infrastructure: the systematic exclusion of people with disabilities, elderly users, and non-native language speakers from independently accessing essential urban services.

The proposed four-layer architecture demonstrates that accessibility-aware AI assistance can be achieved through thoughtful composition of existing open-source technologies rather than requiring novel model architectures, expensive fine-tuning, or proprietary infrastructure. A compact language model, paired with a curated knowledge base, FAISS vector search, and carefully designed prompt-level behavioural constraints, produces responses that are factually grounded (93% accuracy), compliant with disability-specific communication requirements (90% compliance), and consistently formatted for screen-reader compatibility and cognitive accessibility.

The qualitative usability study confirms that participants with visual, auditory, and cognitive disabilities find the system's output helpful, trustworthy, and appropriately formatted. The step-by-step response structure, the explicit "I don't know" fallback, and the avoidance of modality-inappropriate language were identified as key trust-building features.

The entire system runs locally on consumer hardware without GPU requirements, preserving user privacy and enabling deployment on standalone kiosks, mobile devices, and web browsers in environments where internet connectivity is limited or where data-protection regulations restrict cloud processing.

While the current implementation represents a proof of concept rather than a production-ready deployment, it establishes a practical, replicable blueprint for integrating responsible AI into inclusive smart-city infrastructure. The modular architecture, open-source codebase, and community-extensible knowledge base invite broad participation in expanding the system's domain coverage, disability profile taxonomy, and linguistic reach.

The broader significance of this work lies in reframing accessibility not as a specialised, post-hoc feature but as a first-class design principle that can be systematically embedded into AI-powered service delivery. As cities worldwide invest in digital infrastructure, the inclusion of accessibility-aware AI mediation layers represents a concrete, technically feasible opportunity to ensure that technological progress benefits all citizens, including and especially those who have historically been left behind.

REFERENCES

[1] World Health Organization, "Global report on health equity for persons with disabilities," WHO, Geneva, 2022.

[2] World Bank, "Disability inclusion overview," World Bank Group, Washington, DC, 2023.

[3] S. Trewin, "AI fairness for people with disabilities: Point of view," *arXiv preprint arXiv:1811.10670*, 2018.

[4] W3C, "Web Content Accessibility Guidelines (WCAG) 2.1," W3C Recommendation, Jun. 2018. [Online]. Available: https://www.w3.org/TR/WCAG21/

[5] G. Brajnik, "Web accessibility testing: When the method is the culprit," in *Proc. Int. Conf. Computers Helping People with Special Needs (ICCHP)*, Linz, Austria, 2008, pp. 156–163.

[6] P. Lewis *et al.*, "Retrieval-augmented generation for knowledge-intensive NLP tasks," in *Proc. NeurIPS*, 2020, pp. 9459–9474.

[7] IBM Research, "Granite language models," 2024. [Online]. Available: https://github.com/ibm-granite

[8] Freedom Scientific, "JAWS screen reader," 2024. [Online]. Available: https://www.freedomscientific.com/products/software/jaws/

[9] NV Access, "NVDA screen reader," 2024. [Online]. Available: https://www.nvaccess.org/

[10] J. Lazar, A. Dudley-Sponaugle, and K. Greenidge, "Improving web accessibility: A study of webmaster perceptions," *Computers in Human Behavior*, vol. 20, no. 2, pp. 269–288, 2004.

[11] WebAIM, "The WebAIM Million: The 2024 report on the accessibility of the top 1,000,000 home pages," 2024. [Online]. Available: https://webaim.org/projects/million/

[12] J. Sevilla, G. Herrera, B. Martinez, and F. Alcantud, "Web accessibility for individuals with cognitive deficits: A comparative study between an existing commercial web and its cognitively accessible equivalent," *ACM Trans. Comput.-Hum. Interact.*, vol. 14, no. 3, pp. 1–23, 2007.

[13] Y. Liu *et al.*, "Multilingual denoising pre-training for neural machine translation," *Trans. Assoc. Comput. Linguistics*, vol. 8, pp. 726–742, 2020.

[14] NLLB Team *et al.*, "No Language Left Behind: Scaling human-centered machine translation," *arXiv preprint arXiv:2207.04672*, 2022.

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

[16] H. Touvron *et al.*, "LLaMA 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[17] A. Jiang *et al.*, "Mistral 7B," *arXiv preprint arXiv:2310.06825*, 2023.

[18] D. Gurari *et al.*, "VizWiz Grand Challenge: Answering visual questions from blind people," in *Proc. IEEE/CVF CVPR*, 2018, pp. 3608–3617.

[19] Y. Feng, J. Qiang, Y. Li, Q. Yuan, and Y. Zhu, "Sentence simplification via large language models," *arXiv preprint arXiv:2302.11957*, 2023.

[20] A. Yin, Z. Zhong, and T. Tang, "Gloss-free end-to-end sign language translation," in *Proc. ACL*, 2023, pp. 1–12.

[21] Z. Ji *et al.*, "Survey of hallucination in natural language generation," *ACM Comput. Surv.*, vol. 55, no. 12, pp. 1–38, 2023.

[22] E. M. Bender, T. Gebru, A. McMillan-Major, and S. Shmitchell, "On the dangers of stochastic parrots: Can language models be too big?" in *Proc. ACM FAccT*, 2021, pp. 610–623.

[23] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in *Proc. EMNLP-IJCNLP*, 2019, pp. 3982–3992.

[24] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," *IEEE Trans. Big Data*, vol. 7, no. 3, pp. 535–547, 2021.

[25] K. Singhal *et al.*, "Large language models encode clinical knowledge," *Nature*, vol. 620, pp. 172–180, 2023.

[26] J. Cui *et al.*, "ChatLaw: Open-source legal large language model with integrated external knowledge bases," *arXiv preprint arXiv:2306.16092*, 2023.

[27] P. Gao *et al.*, "Retrieval-augmented generation for large language models: A survey," *arXiv preprint arXiv:2312.10997*, 2023.

[28] M. Ball, "The Metaverse: And How It Will Revolutionize Everything," Liveright Publishing, 2022.

[29] H. Duan *et al.*, "Metaverse for social good: A university campus prototype," in *Proc. ACM Multimedia*, 2021, pp. 153–161.

[30] A. Allam and Z. A. Dhunny, "On big data, artificial intelligence and smart cities," *Cities*, vol. 89, pp. 80–91, 2019.

[31] S. Ramírez, "FastAPI: Modern, fast web framework for building APIs with Python," 2024. [Online]. Available: https://fastapi.tiangolo.com/

[32] European Parliament, "Regulation (EU) 2016/679 (General Data Protection Regulation)," *Official Journal of the European Union*, 2016.

[33] A. Radford *et al.*, "Robust speech recognition via large-scale weak supervision," in *Proc. ICML*, 2023, pp. 28492–28518.

[34] H. Lu, D. Zheng, and L. Yang, "Mutual information-based retrieval-augmented generation for domain question answering," *Knowledge and Information Systems*, 2026, DOI: 10.1007/s10115-025-02624-x.

[35] B. Akkus Yildirim *et al.*, "Large language models standardize the interpretation of complex oncology guidelines for brain metastases," *Communications Medicine*, vol. 6, 2026, DOI: 10.1038/s43856-025-01315-6.