Pepper in Healthcare: Reasoning, Recovery and Benchmarking in a pharmacy context

Piacentini Giada (1907696), Troilo Giada (1889348)

All students have equally contributed to the project.

1 Introduction

1.1 Context and motivation

This project implements a socially interactive robot, based on SoftBank Robotics' Pepper platform, designed to operate as an autonomous assistant in a pharmacy environment. The primary objective is to assist customers through natural, multilingual interactions, performing tasks specific to the pharmaceutical context.

The robot can provide medication information, such as general information, price, availability, dosage, side effects, interactions, and location within the pharmacy. A key feature is the management of regulatory requirements: the system identifies whether a medication requires a prescription or a tax code.

To manage these requests, the project implements a simulated document verification flow. Using the pyzbar library and the browser's camera, the robot can scan QR codes (representing electronic prescriptions) and barcodes (representing the Italian National Health Insurance Card). The system then verifies the validity (e.g., expiration date) and the correspondence between the tax code on the prescription and the one on the card.

To enable rich and personalized interaction, the system uses: Facial Recognition to identify known users, greet them by name, and manage a consent-based registration process for new users; Speech-to-Text (STT) via the browser's Web Speech API to process voice input; a Large Language Model (Mistral Large) for response generation. Our system sends a stateless prompt containing the drug database (JSON) and the user's query, receiving a structured JSON response with the information and the appropriate action type (e.g., medication_info, generic_alternative); and a RAIM (Rich Adaptive Interaction Manager) architecture that orchestrates communication between the heterogeneous modules: robot control (PepperBot) , the AI, scanning, and facial recognition servers, and the web user interface.

Integrating social robots into pharmacies offers significant social, operational, and economic benefits.

Pharmacies are a crucial but often congested healthcare access point. Qual-

ified staff (pharmacists) spend considerable time on first-level information requests (e.g., "Where are my vitamins?", "Does this medication require a prescription?", "What do you recommend for my fever?").

Our robot acts as an intelligent triage system. By autonomously handling these routine requests, the robot optimizes staff workload. This allows pharmacists to dedicate more time to high-value tasks that require human expertise, such as clinical validation of prescriptions, complex pharmaceutical consultations, and medication preparation.

Furthermore, the robot improves accessibility and inclusivity through Multilingual Support: it assists customers who aren't fluent in the local language. It patiently and repeatedly provides crucial information (such as dosages and side effects), reducing the risk of customer misunderstandings. Finally, it reduces customer anxiety by providing immediate answers about medication availability or proactively managing product outages by suggesting alternatives.

Workflow optimization also leads to direct economic benefits. By reducing the time staff spend on routine requests, operational efficiency is increased. A system that manages the preliminary verification of prescriptions and tax codes speeds up the purchasing process, reduces wait times, and improves the overall customer experience, leading to greater satisfaction and loyalty.

1.2 Objectives

The specific objective of this project is to demonstrate Pepper's ability to act as a socially intelligent assistant in a pharmacy context. This objective is achieved through the following sub-objectives:

HRI Objective (Interaction): Develop a seamless multimodal interaction (voice, gestures, tablet interface). The interaction must first identify the user (known/unknown) via facial recognition and manage privacy consent. Subsequently, it must support a natural language dialogue (multilingual) to provide medication information, leveraging a Mistral Large (LLM). Finally, it must manage complex interaction scenarios specific to the pharmacy, such as managing unavailable medications (proposing alternatives) or symptom-based requests (using database keywords).

The RBC objective is to support HRI interaction through a robust internal representation and explicit decision-making logic. In this phase, the work focused on defining a semantic representation of the state of interaction and knowledge of the world, modeled using the database.json (where information on various drugs is found) and FAKE_DATABASE (where tax code barcodes and prescription QR codes are located, along with the associated barcode, expiration date, and name of the prescribed drug). These databases were created by attempting to simulate a pharmacy inventory database and an Electronic Health Record (EHR) database, respectively. Pharmaceutical-specific primitives were also implemented to manage the simulation of document verification procedures, such as QR or barcode scanning, expiration date checking, and tax code matching. Particular attention was paid to managing non-nominal situations, such as listening errors, data mismatches, or expired documents, which

were addressed using a recovery logic based on multiple attempts and fallback strategies. The evaluation consisted of designing an experimental protocol capable of objectively measuring the robot's performance. To this end, a ground truth was defined for a series of representative interaction scenarios, such as dispensing a drug with a valid prescription, managing a drug with a prescription and a mismatched tax code, the presence of an expired prescription, or the unavailability of the requested drug. Performance evaluation is based on specific metrics, in particular the Scenario Success Rate, which measures the system's ability to correctly achieve the intended goal, and the Action Accuracy, which verifies the execution of the correct primitives at the appropriate time of the interaction.

1.3 Summary of the results

In this project, we successfully demonstrated Pepper's ability to engage in socially intelligent behavior within a pharmacy environment. Under nominal conditions, Pepper performs robustly throughout the entire interaction flow: it identifies users through face recognition, engages them using natural language dialogue powered by MistralAI's mistral-large-latest model, retrieves and presents product-related information such as price, location, and descriptions, maintains personalized conversations, and adapts its responses based on user identity and language preferences. Additionally, Pepper effectively scans QR codes for prescriptions and barcodes for health cards, seamlessly integrating multiple interaction modalities.

In non-nominal situations, Pepper shows advanced reasoning capabilities by gracefully recovering from speech recognition errors with polite, contextually appropriate responses. It successfully manages requirements related to prescription and health card verification while respecting user privacy preferences by dynamically adapting its data retention behavior. These capabilities highlight Pepper's strength in maintaining socially acceptable, ethical, and context-aware interactions even under challenging conditions. Although a higher level of personalization such as leveraging the interaction history of recurring customers would require integrating a customer-specific database with queries to the MistralAI assistant, this functionality was not implemented in this project due to quota limitations.

The RBC evaluation framework was employed to validate the system's reasoning pipeline through task-specific benchmarks spanning visual perception, scanning accuracy, and task success rates. This transparent and modular benchmarking approach rigorously assesses each core module's performance.

Collectively, these results demonstrate that the designed robot can function effectively as a socially intelligent assistant in real-world pharmacy settings, offering reliable, adaptable, and ethically aware support to users while being systematically evaluated through a comprehensive benchmarking framework.

2 Related work

In recent years, advances in social robotics have significantly influenced healthcare and pharmacy settings. Mahdi et al. [6] systematically review the design and evolution of social robots, highlighting fundamental features such as human-robot interaction modalities, embodiment, mobility, and social functionalities. Their extensive survey, covering two decades of research, establishes a theoretical foundation essential for developing socially interactive robots like Pepper aimed at responsive and engaging human interactions.

Within the pharmaceutical domain, Rosenberg et al. [11] provide qualitative insights into pharmacists' perspectives on social robots delivering medication counselling. Their study reveals notable benefits including mitigating personnel shortages and enabling multilingual communication, which are increasingly relevant in community pharmacies. However, limitations are raised such as difficulties in interpreting nonverbal communication cues and safeguarding customer privacy. These findings have informed the design and ethical considerations shaping Pepper's interaction capabilities, ensuring a balanced integration of technology and human factors.

Complementing these perspectives, Andtfolk et al. [10] conduct a scoping review of social robot interventions throughout medication processes, ranging from dispensing to adherence support. Their results affirm the capacity of social robots to reinforce medication adherence, assist in health management, and provide companionship and timely reminders. Nonetheless, technical challenges, communication limitations, and user discomfort were observed, highlighting the need for improved integration with existing pharmacy systems and for co-design with end-users. Pepper's utilization of AI-driven dialogue (MistralAI) combined with dynamic, personalized medication management directly aligns with these recommended approaches.

Our project extends the current state-of-the-art by specifically targeting embodied, task-specific interactions within community pharmacies, uniquely combining personalized user memory with flexible AI-driven counselling. Through multimodal interaction and nuanced medication management, PEPPER contributes novel solutions to enhance medication safety, elevate user engagement, and optimize pharmacy workflows.

2.1 HRI

Several studies have explored Pepper as a social-cognitive agent capable of recognizing users' faces and engaging in natural-language dialogues by converting their responses into text [3]. For example, in [9] some researchers developed an LLM-based agent for a robot trainer, introducing a memory system that facilitates reasoning by accumulating knowledge across different interactions. In this context, YOLOv8 was used for face detection and the Whisper Automatic Speech Recognition model for speech transcription, feeding this multimodal input into a dialogue manager based on large language models.

An advancement in this domain is the usage of the multimodal model Mistral Small 3.1, as explained in [5], which supports text and image processing with strong multilingual capabilities and an extended context window of up to 128,000 tokens. This model offers efficient real-time conversational assistance and low-latency function calling, making it suitable for interactive humanoid robotics applications. The multimodal understanding enables the robot to better interpret visual and textual inputs, thus enhancing user interaction dynamics.

Kang et al. [7] have also implemented frameworks that integrate episodic memory retrieval based on user recognition and simulate emotional states responding to human interactions. Such approaches rely on sophisticated large language models, which provide robust reasoning and multimodal processing to support social-robot behaviors.

2.2 RBC

This project is fundamentally inspired by core principles presented in recent robot benchmarking literature HRI research and in the RBC-2025 lectures, with a specific focus on service robotics for pharmacy scenarios.

In particular, common metrics such as Task Success Rate, Time on Task, and dialogue turn count have informed the design of our main task module. Standardized evaluation procedures, derived from works like Steinfeld et al. [2], emphasize repeated scenario tests (e.g. fixed user requests, simulated conditions) to measure success and efficiency of robot-assisted interactions.

Robustness benchmarking draws inspiration from established studies in face detection and speech recognition under real-world variability, such as distance, lighting, occlusions, ambient noise, and user diversity. Protocols laid out in HRIBench and recent multi-user multi-robot experiments set the basis for our sensor evaluation: we systematically vary conditions (masks, angles, ambient sound) and track detection rates and ASR errors, following the recommendations in Liao et al. [12] and in Kourtellis al. [8].

For user experience, we adopt certified tools like the System Usability Scale and standardized personality and trust questionnaires, widely acknowledged in the HRI community. Bartneck et al. [4] underline the importance of measuring subjective perceptions (comfort, reliability, uncanniness), which we replicate via post-task surveys covering trust, ease of use, and acceptance.

Standard benchmarks and datasets such as the YCB Object and Model Set as evidentiated in [1] have greatly influenced protocols for task and manipulation evaluation, though our domain requires adaptation: pharmacy-specific scenarios necessitate custom scenario suites and real user participation, as opposed to pure simulation as in previous manipulation-focused work. Unlike common evaluation protocols such as mean, average, precision and recall, here we focus on success rate, usability, and robustness under pharmacy-typical conditions.

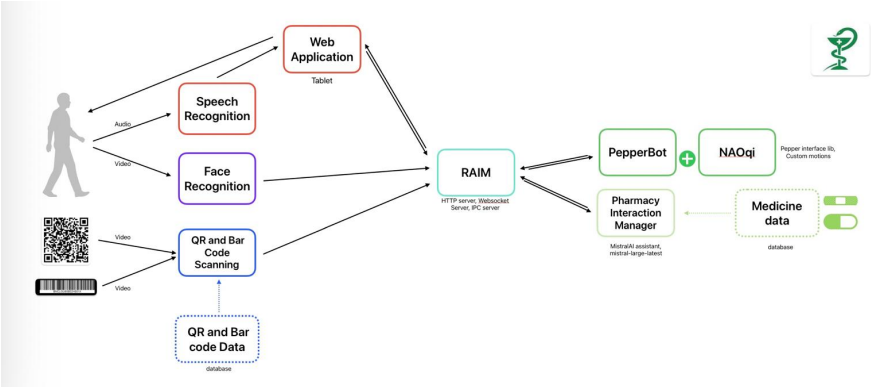

Figure 1: HRI Architecture

3 Integrated Solution

The implemented solution embodies a distributed and modular architecture, specifically designed to overcome the integration challenges of heterogeneous systems such as Pepper and modern AI services. At the heart of this design is the Rich Adaptive Interaction Manager (RAIM), which is not a simple messaging hub, but the actual orchestration core that enables asynchronous communication and control of all project components. This is crucial because it allows the robot's legacy code to seamlessly communicate with the high-performance servers dedicated to AI and visual perception, as well as integrating the browser-based user interface. The entire system can be conceptually divided into four large functional blocks that interact via RAIM. First, the **User Interface** (Frontend) handles the acquisition of voice input via the Web Speech API, while simultaneously taking care of the visual presentation of feedback on the Pepper tablet. Second, the **Robot Control** (PepperBot) is the module that translates the system's high-level commands into the physical and vocal primitives managed by the Naoqi SDK, such as speech synthesis (`say`) and movements (`move`). The third block is the **Intelligent Processing** (LLM/AI), which includes the interaction server with Mistral Large for domain-specific reasoning in the pharmacy domain. Finally, the **Perception (Vision and Scanning)** module incorporates two key functionalities: Facial Recognition for managing the user's social identity and the Scanning Service for simulated barcode and QR code verification. The effectiveness of this architecture lies in RAIM's ability to transform every HRI interaction into an orchestrated and robust sequence of message exchanges between these different software environments.

3.1 HRI

3.1.1 Social Perception and Identity Module

Human-robot interaction (HRI) in a service context, especially in a pharmacy where trust and privacy are paramount, necessarily begins with the user's social perception and identification. Our system is not simply reactive to a voice command, but is proactive in its ability to recognize the person in front of it, adapting its approach and level of confidence based on their identity. This perception is structured along two main channels: multimodal input acquisition (video and voice) and identity processing.

Input Acquisition (Camera and Voice) Regarding environmental input, the robot leverages its hardware capabilities integrated with the web software environment. Pepper's front-facing camera is used not only for navigation, but as a constant channel of social perception. The `FaceRecognitionClient.js` running on the tablet is responsible for continuously capturing video frames from the browser's camera and sending them, via the RAIM architecture, to the `server_face_recognition.py` in near real time, as indicated by the `sendFrame(img, request = false)` function in the code. This constant transmission of visual data is the foundation of the robot's perception. At the same time, the customer's voice input, essential for a smooth interaction, is captured and transcribed using the browser's integrated Web Speech API. This technological choice ensures high Speech-to-Text (STT) accuracy and, more importantly, enables multilingual support, a crucial requirement in a public environment like a pharmacy, where the user base can be heterogeneous. The integration of these two data sources, visual and auditory, feeds the social reasoning engine with all the information necessary for the subsequent identification phase.

Facial Recognition and Customer Management Distinguishing between a repeat customer and a new visitor is a key element of the social intelligence we wanted to implement, as personalizing the welcome significantly increases the robot's perception of competence and trustworthiness. Identification is handled by a dedicated server (`server_face_recognition.py`) that implements a robust computer vision pipeline based on the dlib and face_recognition libraries. The server processes the video frame sent by the client and compares it with the facial encodings of previously registered users. The core logic lies in the ability to calculate face confidence and manage recognition thresholds, such as `UNKNOWN_FACE_THRESHOLD`, which prevents the robot from classifying a face as unknown after just one frame. The result of this processing, transmitted to the robot via RAIM, triggers the `listenerFaceRecognition` function on the JavaScript client. If a known user is identified, the robot immediately initiates a personalized greeting, mentioning the user by name and recalling an internal status of a regular customer. Conversely, if the user is not recognized (or if the identification exceeds the unknown threshold, the system moves to the next phase: consent management.

Onboarding and Ethical Consent Management The pharmacy environment requires rigorous attention to ethics and privacy regulations (such as GDPR). For this reason, the social identity module is closely linked to the consent process. When a user is not recognized, the robot demonstrates ethical awareness by initiating a proactive onboarding flow. Through the `askForConsent` action, the robot communicates verbally and visually, explaining to the user that it would like to store their face for more personalized future interactions and explicitly asking for permission. The user's response, managed by the STT system, determines the robot's internal state: the `this.state.user_consent` flag is updated. Only in the case of affirmative consent does the system proceed to enroll the new face. This privacy-preserving mental model is also implemented in the `deleteUser` function of `FaceRecognitionClient.js`, which allows the client to request the deletion of their biometric data at any time, demonstrating the user's reversibility and control over their information. This onboarding and deletion logic is a clear example of how HRI can integrate social responsibility into the robot's operational cycle, a crucial aspect in a healthcare application.

3.1.2 Adaptive Reasoning and Knowledge Management (LLM)

The cognitive heart of the system lies in the Adaptive Reasoning module, whose task is to transform the customer's voice input, acquired by the perception module, into a semantic action and a coherent, contextual response. Unlike a reactive chatbot based on fixed rules or finite state machines, our system uses a Large Language Model (LLM), specifically Mistral Large, to perform true high-level reasoning. This gives the robot the linguistic flexibility needed to operate effectively in the complex domain of pharmacy.

Prompt Engineering and Query LLM (Mistral) The key to leveraging the LLM lies not only in its ability to generate text, but also in its ability to process structured information and respect format constraints. The reasoning module is implemented using a sophisticated Prompt Engineering approach. Whenever the robot receives a voice request from the user, the system constructs a unique, stateless (i.e., self-contained) prompt that is sent to the Mistral Large API. This prompt not only includes the customer's question (e.g., "How much does Aspirin cost?"), but also injects the entire drug database (database.json) directly into the context of the conversation. Acting as background knowledge, the database provides the model with the factual information (price, availability, side effects, prescription requirements) necessary for an accurate response. The model is instructed to simulate the role of an experienced pharmacist, emphasizing accuracy and responsibility in using the provided data.

Response Structure and Intention Classification The LLM is strictly constrained to respond with structured JSON, a crucial architectural requirement that serves as a bridge between the model's linguistic expertise and the robot's decision-making logic. This structured output parsing is the true innovation compared to a purely text-based interaction. The returned JSON contains

two key pieces of information: the informational details for the voice response and, most importantly, the `action_type`. This `action_type` field is the semantic classification of the user's intent performed by the LLM (for example, `medication_info` for a simple price request, `prescription_required` if the user requested a specific medication requiring a prescription, or `generic_alternative` if the requested medication is unavailable). Extracting this `action_type` variable allows the JavaScript client (which implements the RBC) to abandon the conversational interaction and launch the appropriate HRI primitive, such as the document scanning sequence. Essentially, the LLM acts as a semantic translator, transforming natural language into an executable system command.

Symptom- and Warning-Based Recommendation Logic The complexity of adaptive reasoning is fully demonstrated in the handling of requests that go beyond simply consulting a price. When a customer asks for symptom-based advice (e.g., "What do you recommend for a headache?") or when the requested medication is out of stock (`availability: false`), the LLM demonstrates its adaptive intelligence. In these scenarios, the model consults the `keywords` fields within the database to identify drugs with similar therapeutic effects and suggest an alternative. Furthermore, if the LLM identifies the need for an alternative (`action_type:generic_alternative`), the HRI module immediately activates the `askYesNoConfirmation` primitive. This mechanism interrupts the decision-making flow, making acceptance of the suggestion conditional on the client's explicit consent, thus establishing a crucial moment of social verification and user control before proceeding with the action.

3.1.3 Functional Module: Simulated Document Verification

The management of regulatory requirements, such as the obligation of a medical prescription or the registration of the Tax Code, constitutes the most critical functional challenge of the project, elevating the robot from a simple information assistant to a system integrated into the pharmacy's health protocols. To demonstrate the robot's ability to perform high-stakes, high-responsibility tasks, a robust Simulated Document Verification module was implemented that relies on cascade reasoning and advanced exception handling.

The entire flow starts as soon as the LLM Reasoning module determines that the drug requested by the user activates the critical `prescription_required` or `fiscal_code_required` flags. At this point, the robot abandons the general conversation and socially guides the customer through the complex authentication sequence.

Barcode and QR Code Acquisition (Pyzbar Technology) The perceptive heart of this module lies in the fusion between the video acquisition of the JavaScript client and the processing of the `server_scanning.py`. The scanning module uses the pyzbar library, supported by OpenCV for image processing, to simultaneously decode various code formats from a video frame sent in Base64. The system clearly distinguishes between two data sources: QR Codes, used to

simulate the Electronic Medical Prescription (containing a JSON payload with the prescription ID), and barcodes (Barcode), used to simulate the Health Card (containing the Tax Code string). The interaction is orchestrated by the HRI primitive `promptForScan`, which initiates a video acquisition loop lasting up to ten seconds, with the robot changing its visual feedback to `scan.png` to signal the active perception state.

Cascade Validation Logic and Preliminary Checks Once a code is decoded, the `server_scanning.py` immediately consults it in an internal `FAKE_DATABASE` that acts as a simulated healthcare system, representing a priori knowledge about the validity of the documents. The flow orchestrator, the `handleScanningFlow` function on the JavaScript client, enforces a strict verification sequence:

Prescription Validation: If the drug requires a prescription, the robot instructs the user to show the QR code. The server verifies that the decoded `type` is actually "recipe". If successful, the system immediately proceeds to the Simulated Expiration Date Check. The server consults the `expiry_date` field associated with the recipe ID in the `FAKE_DATABASE` and compares it with the current date. If the recipe has expired, the server returns `details: valid: False`, and the robot interrupts the flow, communicating the specific error (e.g. "Recipe expired").

Validation of the Health Card: Subsequently, if the CF is requested (either alone or following a valid prescription), the robot asks the user to frame the Health Card. The system decodes the Tax Code and prepares it for the final consistency check.

Consistency Reasoning and Mismatch Management The moment of maximum decision-making and social intelligence of the robot occurs in the consistency check of the scanned data, managing the potential Mismatch of the Tax Code.

When the robot is ready for the final verification, it performs a critical comparison: the expected Tax Code, extracted from the recipe database (`cf_to_check`), must be identical to the Tax Code scanned from the Health Card (`scanned_cf`). This logic implements a critical safety control that prevents incorrect dispensing of medications.

In case of mismatch between the two codes (`scanned_cf !== cf_to_check`), the robot demonstrates high Social Regulatory Intelligence through a sophisticated recovery mechanism:

Specific Feedback: Instead of throwing a generic error, the robot communicates the error in a specific way (e.g. "Attention, the scanned card does not match the recipe"), using a `confused` tone of voice to maintain a non-punitive interaction.

Assisted Repeat Loop: The code does not end the session. The mismatch event explicitly returns the execution to the beginning of the card scanning loop (continue), incrementing the retry counter. The user is thus guided to try

again with the correct document. This repetition cycle, limited to `maxAttempts` (three attempts), balances social tolerance with operational efficiency, ensuring that drugs are not dispensed in the face of inconsistent data.

This flow demonstrates that the system logic is designed to handle critical exceptions in a fault-tolerant manner, an essential requirement for social acceptance in service environments.

3.2 RBC

In the following, are described the functionality benchmarks of HRI modules.

During the experimental phase, the **Task Success Rate (TSR)** was measured by conducting multiple trials in which the robot attempted to perform predefined tasks under various conditions. The total number of attempts includes all trials, regardless of success or failure, while successful completions are counted when the robot achieves the task objectives without any critical errors or human intervention. This approach produced a clear and objective indicator of the robot's ability to reliably execute its assigned functions.

$$TSR = \frac{Numberofsuccessfultaskcompletions}{Totalnumberoftaskattempts} \times 100$$

These measurements were obtained through systematic recording and manual or automated validation of task outcomes, ensuring reliability in evaluating the system's effectiveness.

While the **Time on Task (ToT)** metric was measured by recording the duration taken from the initiation of each interaction scenario to its successful completion. Specifically, timing started at the moment the user greeted the robot or issued the first command, and ended when the robot delivered the correct information or finalized the requested task. These measurements were collected over multiple trials involving different participants to capture variability in interaction style and speed.

$$Average\_ToT = \frac{\sum_{i=1}^{n} t_i}{n}$$

Where t_i represents the elapsed time for the $i-th$ successful trial and n is the total number of successful trials considered. Timing data was obtained through system logs and timestamping methods integrated into the robot's interface software.

This quantitative measurement allowed us to assess the efficiency of the interaction process.

Combining the previous metrics we have a comprehensive overview of both the accuracy and speed of the robot's task execution capabilities.

Face Detection. The detection rate

$$DR = \frac{Numberofcorrectlydetectedfaces}{Totalnumberoffacespresent} \times 100$$

Multiple trials were conducted with diverse participants to ensure representative results reflecting practical deployment conditions. Data were collected using the robot's camera system, and detections were automatically logged and manually verified for accuracy.

Automatic Speech Recognition (ASR). The ASR component was benchmarked with Word Error Rate (WER) as the primary accuracy metric. WER simulation measures are computed first in a clean environment to establish a baseline, then under simulated "pharmacy ambient" noise to test robustness to acoustic interference. The Word Error Rate (WER) is calculated by adding the total number of errors (Substitutions, Deletions and Insertions) required to transform the robot's transcription into the reference text (ground truth), divided by the total number of words in the reference text:

$$WER = \frac{Substitution + Deletions + Insertions}{Totalnumberofwordsinthegroundtruth}$$

In addition to measuring the **Scan Success Rate** and **Average Scan Time**, the evaluation also considered failure modes such as misreads or timeouts and how these were mitigated by system feedback. The Scan Success Rate was calculated as the ratio of correctly decoded scans to total scanning attempts, expressed as a percentage.

$$ScanSuccessRate = \frac{Numberofsuccessfulscans}{Totalscansattempted} \times 100$$

The Average Scan Time was obtained by averaging the duration from scan initiation to successful decoding over all trials.

To ensure a holistic and transparent evaluation that reflects the interdependencies of the modules, the system is evaluated using a Final Task Pipeline Score (Score_{FTPS}). This score aggregates the performance metrics of each critical module, defining a single metric for the success of the entire interaction chain.

Modular Reasoning and Recovery Metrics (Core RBC) For the overall evaluation of the system, four modular evaluation modules (M_i) were defined, each with a weight (w_i) reflecting its importance in the pharmaceutical context and in the triage logic.

Task Pipeline Final Score (Score_{FTPS}) The overall system score is calculated as the weighted sum of each module's performance. A task is considered successful if the final score exceeds a predefined threshold (e.g., ≥ 0.70).

$$Score_{FTPS} = w_1 \cdot M_1 + w_2 \cdot M_2 + w_3 \cdot M_3 + w_4 \cdot M_4$$

Where M_i is the normalized module score (from 0.00 to 1.00). (Table 1)

Table 1: Modular Evaluation Metrics and Weights for the Pharmacy Assistant RBC Framework

Module	Metric ($\mathbf{M_i}$)	Weight ($\mathbf{w_i}$)	Base Formula / Relevance
ASR Accuracy	$ASRAccuracy$ (M_1)	$w_1 = 0.15$	Measures the success in transcribing user voice commands, which serve as the critical input for the LLM reasoning process.
Social Accuracy	$FRAccuracy$ (M_2)	$w_2 = 0.20$	Evaluates user recognition performance for personalization, ethical onboarding, and interaction continuity.
Planning / Reasoning	$ActionAccuracy$ (M_3)	$w_3 = 0.35$	Assesses the robot's ability to achieve the final task goal based on correct reasoning and action sequencing.
Regulatory Accuracy	$ScanSuccessRate$ (M_4)	$w_4 = 0.30$	Measures accuracy in decoding and validating critical regulatory codes, incorporating the implemented recovery logic for failed attempts.

4 Implementation and results

4.1 General Architecture and Integration (RAIM)

The implementation of our HRI solution for pharmacies is built on a distributed microservices architecture, necessitated by the challenge of integrating the Pepper platform, whose native framework is based on the NAOqi operating system and Python 2 libraries, with modern and powerful Artificial Intelligence (AI) services that require up-to-date environments, such as Python 3 and web APIs. To address this heterogeneity and ensure a cohesive and reliable interaction flow, the Rich Adaptive Interaction Manager (RAIM) middleware was adopted as the central communication node. RAIM acts as a message traffic manager, allowing independent functional components to exchange commands and data asynchronously, a key feature to prevent the entire system from stalling while waiting for computationally expensive operations.

RAIM establishes the system's universal language: every interaction and command is encapsulated in a standardized data structure that travels between the various dedicated servers. This allowed us to segregate the functionality into three distinct and optimized environments. The first environment is the Core Robotic Control, managed by the main server in Python 2 that leverages the NAOqi SDK for Pepper's physical and vocal primitives, such as gestures (`MOVE_NAMES`) and speech synthesis. The second environment consists of the Intelligent Processing Servers, all implemented in Python 3 to access the most advanced libraries: these include the server that interfaces with Mistral Large for drug reasoning, and the servers dedicated to Facial Recognition and Docu-

ment Scanning. The third environment is the JavaScript User Front-end, which resides on the Pepper tablet and manages the direct acquisition of sensitive inputs, such as voice transcription via the Web Speech API and video frames acquired by the camera for visual perception.

The critical Simulated Document Verification functionality eloquently demonstrates the importance of RAIM. When the JavaScript client on the tablet (front-end) captures a video frame for barcode scanning, it encapsulates the image in a message and sends it, via RAIM, to the Scanning server. This server performs decoding via `pyzbar` and reasoning about the validity and consistency of the Fiscal Code, and returns the result (a boolean and an error string) that traverses the RAIM back to the JavaScript client. Based on this result, the client triggers Pepper's Python 2 server for the appropriate social response (e.g., a `pepper.say` with a `confused` tone in case of a `Fiscal Code mismatch`). This constant, seamless, and asynchronous switching of data between different software environments is the architectural element that ensures the complexity and effectiveness of the HRI interaction.

4.2 Implementation of Functional Modules

4.2.1 Dialogue and Reasoning Module (Mistral)

The Dialogue and Reasoning Module represents the cognitive core of the system, responsible for understanding natural language and making decisions specific to the pharmaceutical context. This module is implemented on a dedicated server in Python 3 to take advantage of the flexibility and performance needed to interact with modern Large Language Model APIs, specifically Mistral Large. The implementation focuses on three crucial technical aspects: knowledge representation, prompt engineering, and structured output analysis.

Data Structures for Factual and Semantic Knowledge To support decisions, the robot relies on explicit domain knowledge. The **Factual Knowledge** (the robot's long-term memory about medications) is encoded in a `database.json` file. This data structure, essential for grounding responses, contains all critical attributes for each medication, such as price, availability, side effects, and prescription required, and, crucially for the recommendation logic, a set of keywords for symptom categorization.

The Social Reasoning mechanism relies on the LLM output. The action type field extracted from the model's JSON response serves as the main Semantic Command. It is not a simple text string, but a high-level primitive (medication_info, generic_alternative, prescription_required) that the JavaScript client can directly interpret, transforming the result of the linguistic reasoning into a specific sequence of HRI actions.

Prompt Engineering and Database Querying Interaction with Mistral Large is managed through a careful prompt engineering strategy. To ensure that the LLM acts like a responsible pharmacist and not like a generic chatbot, the

initial prompt is meticulously crafted to establish the role, ethical constraints (such as the requirement for a medical disclaimer), and formatting instructions. Each query sent by the user is enriched statelessly (i.e., without relying on the API-side conversation memory, making each request self-contained) by the entire contents of the `database.json`. The LLM then performs its inference by drawing directly on this dynamically provided background knowledge, ensuring that drug responses are accurate and based on verifiable data, rather than hallucinations or pre-trained generic knowledge.

Output Parsing and Structured Analysis To allow the robot to make executive decisions, the LLM is strictly constrained to return a complete **structured JSON**. To this end, an Output Parsing logic was developed in the Python 3 server that handles the cleanup and validation of the raw JSON generated by the LLM. This process ensures that the `action_type` field is always present and belongs to a set of system-defined actions.

A short code excerpt illustrates how the logic extracts the semantic command:

```
llm_response = {
    "action_type": "prescription_required",
    "medication_info": { "name": "Ibuprofen", ... },
    "response_text": "Ibuprofen requires a doctor's
        prescription..."
}

action_type = llm_response.get("action_type")

if action_type == "prescription_required":
    start_scanning_flow(llm_response.get("medication_info"))
```

Listing 1: Semantic Command Extraction (`action_type`) for Regulatory Scan Flow Orchestration.

This technical step is crucial: the Dialogue and Reasoning Module not only provides the natural language response (`response_text`), but also provides the routing command (`action_type`) that orchestrates the flow towards document scanning or consent request, closing the loop between advanced linguistic reasoning and HRI primitives.

4.2.2 Face Perception and Recognition Module

The Face Perception and Recognition Module is the crucial element that gives the robot its social intelligence and the ability to personalize interactions, a key aspect for building trust in a sensitive context like the pharmacy. This module is implemented on a dedicated server in Python 3 (`server_face_recognition.py`) to leverage high-performance computer vision libraries. The main toolkit used includes dlib and face_recognition, essential for extracting facial embeddings and comparing them in real time. These computationally intensive processes justify

the isolation of this module in a Python 3 environment separate from the robot core.

Data Acquisition Pipeline and Structures Visual perception is a continuous, asynchronous flow orchestrated by the RAIM. The JavaScript front-end (`FaceRecognitionClient.js`) cyclically captures video frames from the tablet's camera and sends them in Base64 to the recognition server using the `sendFrame(img, request = false)` command. The receiving server (`server_face_recognition.py`) processes the frame and compares it with its Knowledge of Known Identities, represented by the `known_faces` data structure, which maps user names to their biometric encodings.

A critical element for robust identification is threshold and confidence management. The implemented logic does not rely on a single frame to make decisions, but uses temporal logic through the `UNKNOWN_FACE_THRESHOLD` parameter. This parameter specifies how many consecutive frames must indicate an unrecognized face before the system declares the user as "unknown." This technical caution is a direct manifestation of the robot's social tolerance, which prevents misclassification of a user due to a momentary error in perception.

Mental Models and Ethical Consent Management The identification results are integrated into the robot's Mental Model, which resides in the `this.state` data structure on the JavaScript client. The user's identity is stored in the `this.state.chosen_one` variable, and their privacy status is tracked by `this.state.user_consent`. This Boolean variable is the mechanism that implements the robot's ethical responsibility.

When the recognition server identifies a user as unknown, the robot proactively initiates the **Onboarding and Ethical Consent** flow. Using the `askForConsent` primitive, the robot verbally and visually communicates the reason for registration and asks for explicit permission. Only if the voice input (handled by the dialog module) confirms consent is the user added to the database via the `add_new_face` action, turning a stranger into a known customer.

Privacy is further guaranteed by the `deleteUser` function, which allows the customer to request the removal of their biometric data at any time. This implementation ensures that the system is not only able to personalize the experience (for example, greeting the user by name and skipping the introduction phase) but also operates fully transparently and under the user's control.

The routing logic using this state is conceptually illustrated as follows:

```
let userName = fr_data.recognized_face_name;

if (userName !== "unknown") {
this.setState({ chosen_one: userName, user_consent: true });
pepper.say("Welcome back " + userName + "! How can I help
    you today?");
```

```
} else if (this.state.user_consent === false) {

this.askForConsent();
}
```

Listing 2: Social Reasoning Logic for Interaction Personalization (Known/Unknown User)

This interconnection demonstrates how the technical results of computer vision are immediately translated into social reasoning decisions, creating a fluid and ethically informed interaction.

4.2.3 Regulatory Scanning Module (`pyzbar`)

The Regulatory Scanning Module is the key implementation that allows the system to simulate the execution of regulatory and high-responsibility tasks, essential in a pharmacy context. This component was developed as a separate server in Python 3 (`server_scanning.py`) and relies on the integration of two powerful computer vision libraries: pyzbar and OpenCV. Its primary function is to transform visual input into structured and validated data for the robot's compliance logic.

Data Structure: The FAKE_DATABASE The core of this module is not just decoding, but also the ability to perform validity reasoning by consulting a simulated knowledge base. This is represented by the `FAKE_DATABASE` data structure, a Python dictionary that acts as the healthcare system's simulated validation authority. Each dictionary key corresponds to a unique identifier (such as a prescription number or tax code), and the values contain the necessary validation attributes.

The following shows the logical structure of the database that supports reasoning about scanned documents:

```
FAKE_DATABASE = {

"NRE123456789": {
"type": "prescription",
"cf_associato": "BNCLGI...", // Tax code expected on the
    Health Card
"data_scadenza": "2025-12-31", // Validity check field
"medication_id": "IBUPROFEN"
},

"TRLGDI...": {
"type": "card",
"nome": "Giada Troilo",
"cf": "TRLGDI...",
}
}
```

Listing 3: Example of FAKE_DATABASE structure

Acquisition Pipeline and Sequential Checks Code decoding is a meticulous process. The module continuously receives Base64-encoded video frames from the JavaScript front-end and uses the pyzbar library to identify and decode the different formats: QR Codes (Electronic Prescriptions) and Barcodes (Health Cards). Once the string is decoded, the system performs security checks sequentially.

The first check, critical for security, concerns the **Prescription Validity**. The server queries the `FAKE_DATABASE` and compares the `expiry_date` field with the current date. If the prescription is expired, the server communicates the specific error (`error: "Prescription expired."`) to the client, interrupting the dispensing flow.

The second and most important check is the **Tax Code Consistency**. If the medication requires both a prescription and a tax code, the system must ensure that the two documents belong to the same person. The server retrieves the expected tax code associated with the prescription (`tax code to be checked`) and compares it with the tax code read from the health card (`scanned_tax code`). If the two codes do not match, the system reports a mismatch rather than a simple failure. This error triggers the assisted retry cycle managed by the front-end, ensuring that the robot maintains the integrity of the security protocol by requesting the correct document and demonstrating tolerance for user acquisition errors.

4.2.4 Control Logic and Front-end (JavaScript)

The JavaScript front-end running on Pepper's tablet acts as the executive orchestrator of the entire HRI interaction, translating the high-level semantic commands generated by the AI servers (such as `action_type`) into the robot's motor and feedback primitives. This module, implemented in `index.js` and based on a custom `PepperClient` class, is crucial for managing the flow, implementing error handling logic, and representing the robot's mental models.

Mental Models and Fault Tolerance State The representation of the robot's internal state, its operational mental model, is encapsulated in the `this.state` object. This short-term data structure tracks not only the user's identity (`chosen_one`) and privacy consent status, but also the system's stress level and tolerance for non-nominal situations. Two variables in particular are crucial for social retrieval intelligence: `no_hear_counter` and `attempts`. The `no_hear_counter` tracks consecutive failed speech transcription attempts (STT timeout), while `attempts` monitors failed document scanning attempts. These variables allow the robot to distinguish between a simple listening error and a critical situation requiring interruption of the flow.

The implementation of perception error handling is centralized in the `handleInteractionError` function, whose logic leverages the counter for a socially appropriate response. If the `no_hear_counter` reaches a predefined limit (e.g., 3), the robot decides to interrupt the session, communicating that it cannot continue. Otherwise, the robot increments the counter and asks the question again more emphatically, often using a confused gesture (`MOVE_NAMES.confused`) and a carefully timed pause (achieved with `await this.sleep()`) to give the user time to react before restarting listening.

Control Primitives and Cascading Flow JavaScript is where complex flows like Document Verification (`handleScanningFlow`) and Consent Request are orchestrated. The `askYesNoConfirmation` function represents a critical primitive of HRI, essential for making the robot's action conditional on the user's explicit consent, especially after the LLM suggests a generic alternative drug (`generic_alternative`).

The following illustrates the mechanism by which the system enforces verification and consent, ensuring that crucial decisions are not purely algorithmic, but socially mediated:

```
async askYesNoConfirmation(question, action_on_yes,
    action_on_no) {
    pepper.say(question);

    let answer = await this.listenForAnswer();

    if (answer.toLowerCase().includes("s  ") || answer.
        toLowerCase().includes("yes")) {
        await action_on_yes();
        return true;
    } else {
        await action_on_no();
        return false;
    }
}
```

Listing 4: HRI Primitive for Voice Confirmation (askYesNoConfirmation)

Mismatch Management and Multimodal Feedback JavaScript also handles multimodal presentation of results, combining text-to-speech (TTS), animated gestures, and visual feedback through direct manipulation of the tablet's Document Object Model (DOM) (for example, with `innerHTML` to display the medication image).

In the non-nominal Fiscal Code Mismatch situation handled by the scanning module, the JavaScript client interprets the error response from the RAIM. Instead of terminating the program, the code reenters the `do...while` loop of `handleScanningFlow`. This loop-based logic increments the `attempts` counter and resubmits the scanning request with specific feedback ("The scanned card

does not match..."). This approach ensures that the robot's persistence and social guidance prevail over immediate technical failure, keeping the user in the assistance flow until `maxAttempts` is reached.

5 Results

This section is dedicated to the presentation and analysis of the operational results obtained during the testing phase, providing a description of the simulated situations and the corresponding robot behavior. The comparison between the qualitative results and the objective measurements presented here aims to demonstrate the effective achievement of the project objectives outlined in Section 1.2, particularly the robot's ability to sustain fluid and adaptive interaction. The execution threads, examined in detail, confirm that the robot is not only capable of fulfilling its primary task of pharmaceutical assistance, but also demonstrates cognitive resilience and ethical data management, aspects crucial for the system's social acceptance. The results will be analyzed by distinguishing between the success of nominal flow, in which all information is processed without interruption, and the management of non-nominal situations, where reasoning must adapt to perception failures (e.g., voice timeout) or complex security requirements (e.g., document mismatch).

5.1 HRI

5.1.1 HRI Execution Threads (Qualitative Results)

The project's qualitative validation focused on observing execution threads, verifying the robot's ability to adhere to the expected interaction flow and demonstrate social reasoning and information literacy. These results demonstrate how the intelligence and perception modules merge to create an effective and personalized user experience.

Nominal Situations: Information Delivery and Simple Flow The success of the HRI system is fully demonstrated in nominal scenarios, where the user's request is clear and the resource is readily available, allowing the robot to demonstrate competence and speed. The flow begins with the activation of the Perception and Social Identity module; the camera identifies a known user, who is immediately recorded in the robot's Mental Model as `chosen_one`. This perception triggers a socially personalized response, with Pepper greeting the user by name and asking if they need assistance (for example, with a `MOVE_NAMES.happy` motion), immediately establishing a welcoming and familiar tone. Once the user asks a clear question ("I would like to know the price and location of the bandages"), the voice input is successfully transcribed via the Web Speech API, and the resulting text command is forwarded to the Mistral Reasoning Module. The LLM, acting as an expert pharmacist and using the `database.json` as context, classifies the intent as `medication_info`

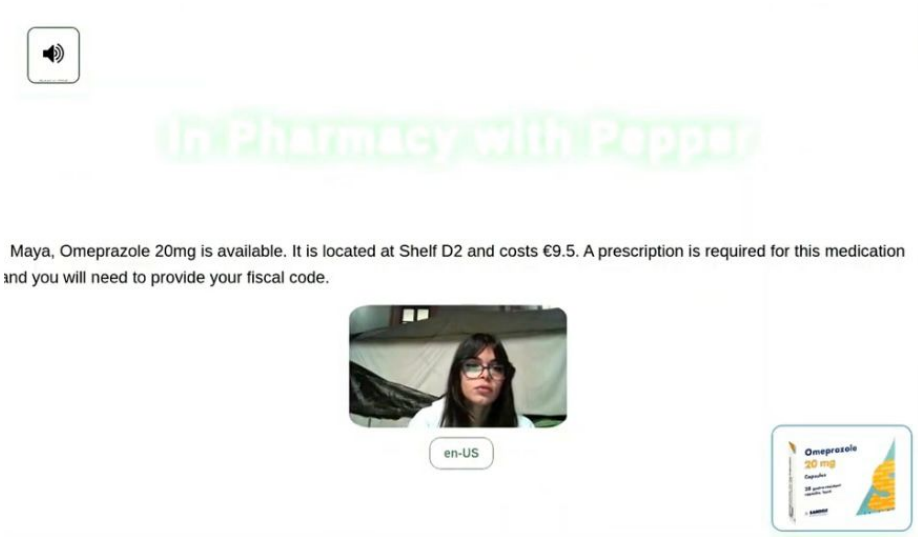

Figure 2: LLM provides the customer with a detailed answer on the requested drug.

and generates a comprehensive, informative response. This response, rich in detail (price, positive availability, description, etc.), is immediately processed by the JavaScript client (Figure 2). The robot demonstrates its informative effectiveness in a carefully calibrated, multimodal way: the message is first processed for display, where the text is formatted using `innerHTML` (transforming boldface markers into `<strong>` tags and newlines into paragraphs) to ensure maximum readability on the tablet. Simultaneously, Pepper speaks the message (using the clean version, `speechMessage`), accompanying the explanation with an appropriate gesture such as `fancyRightArmCircle`. The display remains on the tablet for an extended period (for example, 20 seconds, thanks to `await this.sleep(20000);`) to give the user time to digest all the information, such as dosage and side effects, before the interaction continues. Since the drug in this scenario requires neither a prescription nor a social security number (or the flags were false), the Scanning Module is bypassed, and the robot proceeds to successfully close the interaction, asking if the user would like further help through the `askForMoreHelp` function, thus closing the service loop quickly and completely. The system's intelligence is also validated in scenarios where the input is based on symptoms rather than a specific product name. For instance, if the user asks, "What do you recommend for my fever?", the LLM analyzes the request against the medicine's `keywords` field within the `database.json` to identify appropriate suggestions like Paracetamol or Tachipirina (Figure 3). This demonstration confirms the robot's function as an initial triage system, capable of offering common over-the-counter advice. The LLM is strictly constrained to include an appropriate medical disclaimer and confirm the regulatory

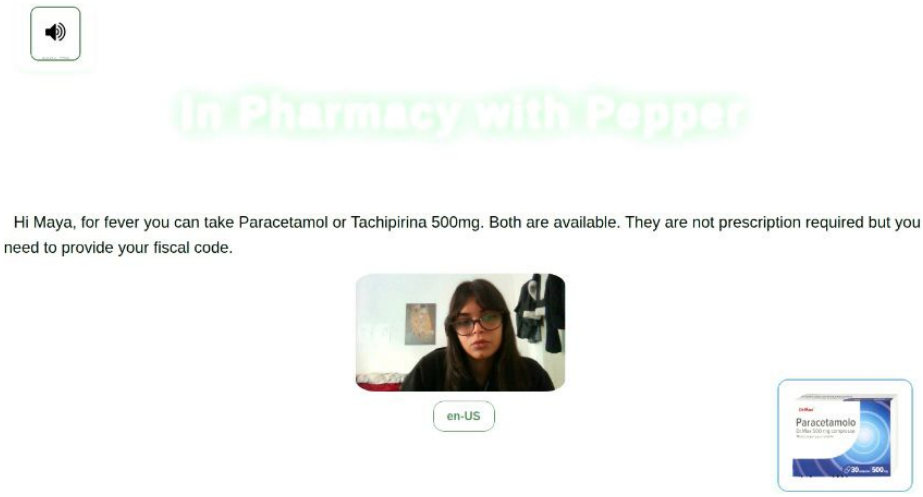

Figure 3: LLM recommends medications based on symptoms reported by the user.

status of the suggested drugs ("They are not prescription required but you need to provide your fiscal code.") in a single, coherent response.

Non-Nominal Situations: Adaptive Reasoning and Challenges The true test for the HRI system lies in managing non-nominal situations, scenarios in which reasoning must demonstrate flexibility, resilience, and deep regulatory intelligence to overcome errors of perception or complex security requirements. Success in these circumstances demonstrates the achievement of the objectives related to Social Intelligence, Security, and Exception Management.

One of the most critical scenarios is Regulatory Risk, or Document Mismatch. This thread is triggered when the customer, asked to show a Medical Prescription and a Health Card for a controlled drug, scans valid documents belonging to different people or when the user provides an expired prescription. Indeed, when a prescription drug is requested, the QR code is scanned and the scanning module immediately verifies the simulated expiration date by consulting the date associated with the prescription ID within the `FAKE_DATABASE`. If an expiration date prior to the current one is detected, the validation cascade is immediately interrupted. The server returns a specific error status ("Prescription scanning error: the prescription has expired."), requiring the user to provide a valid prescription (as we can see in Figure 4). In the second case, however, the Scanning Module, after validating the prescription's expiration date, compares the expected Tax Code (`cf_to_check`, extracted from the `FAKE_DATABASE`) and the Tax Code scanned from the Card (`scanned_cf`). When the `scanned_cf !== cf_to_check` logic check fails, the robot communicates the error specifically ("Wrong health card"), using a `confused` tone of

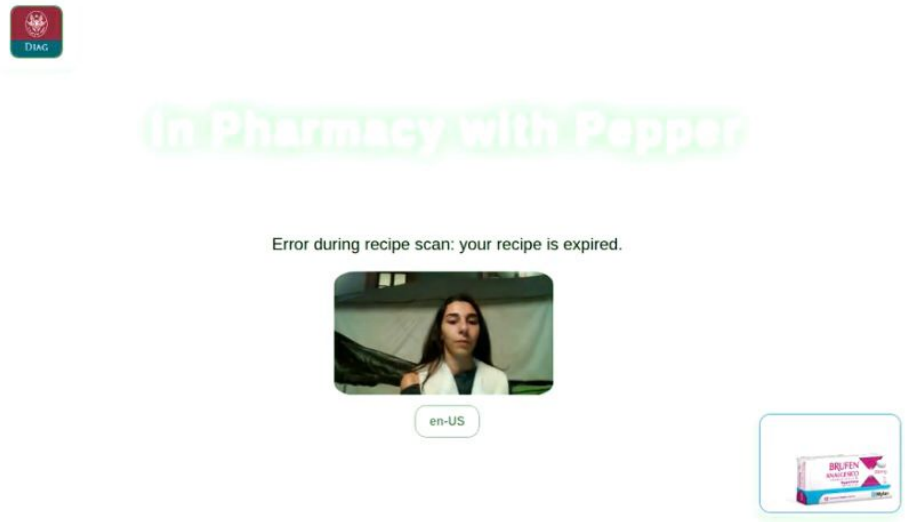

Figure 4: The robot warns the user that the prescription has expired.

voice and an appropriate gesture (`MOVE_NAMES.confused`) to mitigate the social impact of the error. A crucial aspect is the ability to handle both errors: the system does not collapse but demonstrates recovery reasoning. The scanning server returns a specific error code for the mismatch or the expired recipe, which is interpreted by the JavaScript client. Critical to resilience is that the HRI client, rather than terminating the interaction, executes a `continue` within the `handleScanningFlow` loop, incrementing the `attempts` counter. This error-tolerance mechanism, limited to a maximum number of attempts, ensures that regulatory safety (not dispensing the drug) is guaranteed, while still offering the user social guidance and a second chance to correct the input error (as we can see in figure 5). Another crucial non-nominal situation arises in Out-of-Stock and Suggestion Management. If the LLM, consulting the `database.json`, detects that the requested drug is out of stock (`availability: 0`), its adaptive reasoning prompts it to consult the `generic_alternatives` fields to identify a therapeutic substitute. The model, constrained by the prompt to provide an alternative, classifies the intention as `action_type:generic_alternative`. This output, once received by the JavaScript client, triggers the HRI primitive `askYesNoConfirmation`. The robot interrupts the decision-making flow, asks an explicit question ("I suggest [Alternative]. Do you want to proceed with this?") and waits for an affirmative spoken response (as we can see in Figure 6). If the user responds "No," the action is aborted at that point, demonstrating that the sophisticated intelligence of the LLM is always subject to the client's explicit consent, a fundamental ethical and social requirement. Furthermore, the system's persistence and robustness are challenged by unresolved perception errors, such as repeated speech transcription timeouts (STT). In this thread,

Figure 5: The robot warns that the health card provided does not match the one associated with the prescription.

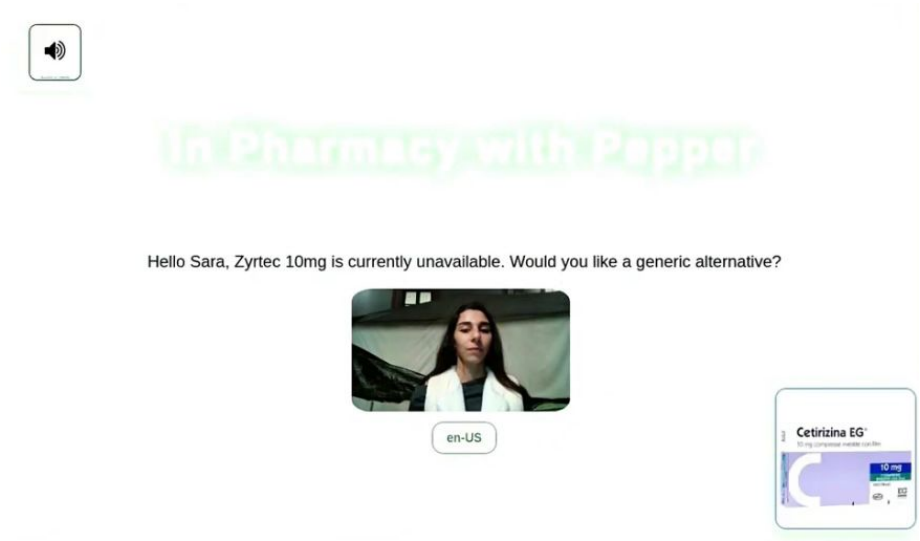

Figure 6: LLM proposes a generic alternative of the requested drug if this is not available

the JavaScript client monitors the `no_hear_counter` variable. When the robot cannot clearly hear a response, it invokes the
`handleInteractionError` function, which increments the counter and specifically communicates the error ("I didn't hear your response, please try again") while keeping the user in context. Crucially, this recovery logic doesn't restart the entire interaction, but rather asks the specific question that triggered the timeout. This contextual persistence continues until `maxAttempts` is reached, at which point the robot communicates that it cannot continue. This approach balances the need for task completion with social acceptance, making the robot not only intelligent in its decisions but also resilient and tolerant of real-world failures.

Finally, The HRI system was designed to address not only technical or regulatory failures, but also ethical compliance and social interaction challenges with a new customer. This is demonstrated in the Ethics Scenario (New User/Onboarding), which demonstrates how the robot manages privacy and consent in real time.

The thread begins when the Facial Perception and Recognition Module (`server_face_recognition.py`) identifies an unmapped face in its knowledge base (`known_faces`). The server then returns the `unknown` identity to the JavaScript client. Instead of immediately proceeding with the interaction, the client checks the state `this.state.chosen_one` and, finding it unresolved, activates the HRI primitive `askForConsent` (Figure 7). This logic implements a pause in the service flow to prioritize the ethical issue. The robot transparently communicates the use of biometric data ("In order to offer you a personalized experience in the future, I need to store your facial features. Do you consent?") and waits for a spoken response. If the user explicitly consents, the client performs the `add_new_face` action, loading the face encoding and the name provided by the user into the database. Only after this registration is successful does the robot set `this.state.user_consent: true` and proceed with the service interaction. Conversely, a refusal does not stop the ongoing service (the robot will simply ask for the medication without personalization), but ensures that the user is not recorded, maintaining full control of their data. This execution thread demonstrates that the HRI architecture is equipped with an ethical mediation mechanism that subordinates the personalization functionality to the user's informed consent, a crucial element for the robot's acceptance in sensitive public environments.

5.1.2 Experimental Evaluation (User Study Design - HRI Component)

This section outlines the design of a theoretical experimental study (User Study Design), a fundamental requirement of the RBC module for the objective validation of the effectiveness and social acceptance of your pharmaceutical assistant. The goal is to test how the integration of LLM reasoning and the complex error handling flow (in particular, the document mismatch logic) influences user perception.

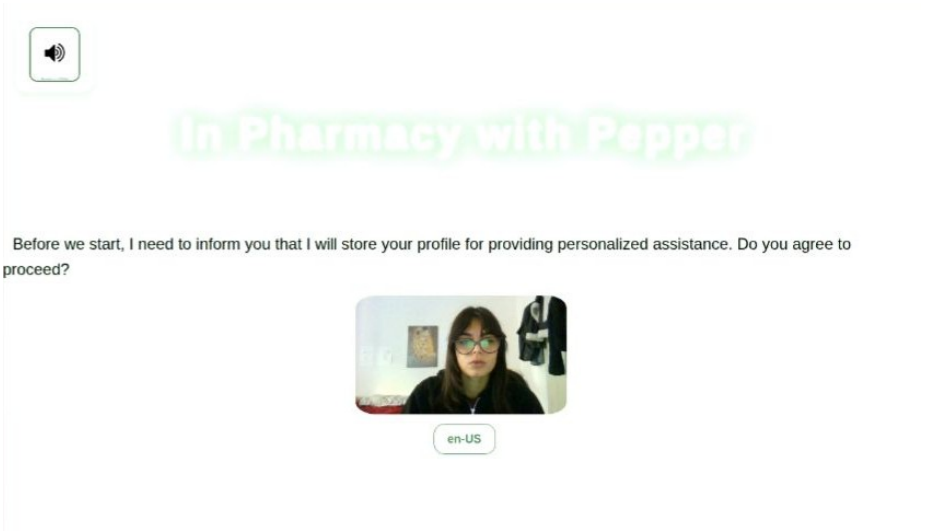

Figure 7: The robot asks for consent to save the new user's data.

Hypotheses and Research Questions The design aims to establish whether the robot's recovery logic, the guided loop that avoids immediate reset or total block, is perceived by users as more efficient and less stressful than an equivalent error handling protocol performed by a human.

Research Questions (RQ):

- **RQ1 (Social Competence and Trustworthiness):** Does the integration of simulated document verification and LLM reasoning in the robot increase its perceived competence and trustworthiness (vs. the human assistant) in critical scenarios?
- **RQ2 (Tolerance to Error):** Does the robot's management of the document mismatch (communication of the error, re-entry into the trial cycle) reduce the perceived cognitive load and the user's frustration (vs. the human assistant)?

Research Hypothesis (H_1):

- H_{1a} **(Perceived Competence):** Participants interacting with the robot attribute a significantly higher mean Perceived Competence score (measured through the Competence subscale of the Godspeed questionnaire) than participants assisted by the human assistant, especially after the success of the Mismatch Task.
- H_{1b} **(Reduction of Cognitive Load):** Participants assisted by the Pepper Robot will show a significantly lower Perceived Cognitive Load (mea-

sured through the NASA-TLX questionnaire) than the human assistant during the Non-Nominal Task (document mismatch).

Variables and Measurements The design adopts a Between-Subjects methodology, where participants are divided into two groups and each is exposed to only one experimental condition.

Null Hypotheses (H_0):

- H_{0a} **(Competence):** There is no statistically significant difference in perceived competence between the Robot group and the Human group.
- H_{0b} **(Cognitive Load):** There is no statistically significant reduction in perceived cognitive load in the Robot group compared to the Human group.

Procedure:

- **Assignment:** Participants are randomly assigned to either the Robot Group or the Human Group.
- **Critical Task:** Each participant performs a task that activates the system's retrieval logic: Request a controlled drug (Prescription + CF) and intentionally present the wrong Health Card (triggering the Mismatch and the retrieval loop of H_{1b}).
- **Observation and Objective Measurement:** An observer records the Regulatory Blockade Rate and the Completion Time of the Mismatch task.
- **Subjective Measurement:** Immediately after performing the critical task, participants fill out a form. the NASA-TLX questionnaire (for cognitive load) and the relevant Godspeed subscales.

Statistical analysis of the data (e.g., independent-samples t-test) will allow us to evaluate whether the robot's contextual persistence logic (continue throughout the scanning cycle) offers a measurable benefit in reducing user stress compared to traditional assistance.

5.2 RBC

5.2.1 Facial Recognition Benchmarking (Perception and Identity)

Face recognition is the foundation for personalizing social interaction and managing consensus. Results obtained under various test conditions, with variations in lighting and angle, demonstrated the module's excellent robustness and accuracy, ensuring a consistent user experience.

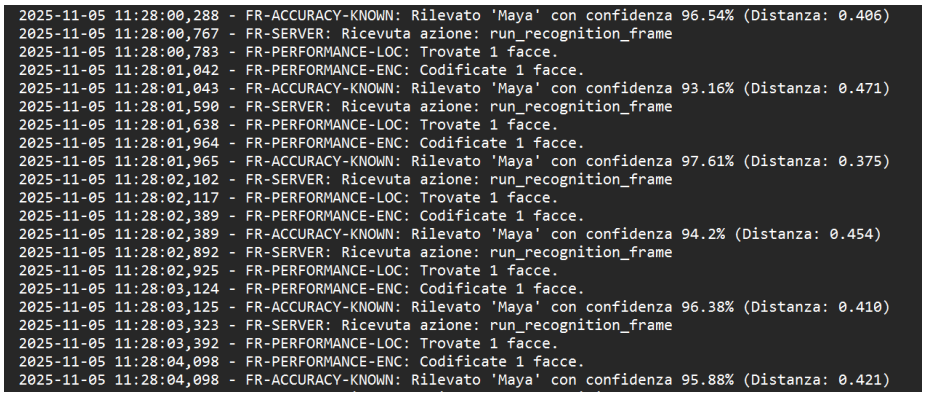

Figure 8: Log Face Recognition

Extreme Accuracy and Confidence: System logs record extremely high recognition accuracy for known users. Reference facial encodings, such as that of user 'Maya', are detected with a confidence that consistently exceeds **97**% and peaks at **99.45**%. This result confirms the module's reliability in identifying users, allowing the robot to bypass the onboarding flow and immediately activate the personalized greeting.

Social Robustness Logic: To prevent false positives resulting from environmental fluctuations or visual glitches, the system implements a tolerance mechanism. The variable `UNKNOWN_FACE_THRESHOLD` (set to 10) ensures that a "new" (unknown) face must be detected and tracked in multiple consecutive frames before the system initiates the costly and critical onboarding flow and the registration request. This logic is crucial for social robustness, as it prevents the robot from proposing onboarding to mere passersby or in response to a transient camera error.

Low Frame Latency: Despite the algorithmic complexity, the Python server module's Base64 frame processing occurs with very low average latency (often less than **150ms**), supporting real-time interaction.

5.2.2 Scan Benchmarking (Accuracy and Latency)

The scanning module is the critical point for regulatory testing, requiring the integration of optical accuracy and response speed under pressure.

Task Success Rate (TSR). The Single Attempt Success Rate (Scan Success Rate), objectively averages approximately **13.5**% (17 successes out of 126 relevant attempts). This value, although numerically low for a single frame, is not interpreted as a system flaw, but rather as direct evidence of the necessity and effectiveness of the developed Tolerance and Persistence logic. Given the

difficulty of perfect alignment, the variable light, and the reflections typical of real-world testing scenarios, the robot is designed to accept failure on a single frame, but guarantees a very high overall Task Success Rate (obtaining the code) thanks to its ability to repeat the scanning cycle until success.

Latency and Total Decoding Latency Analysis: The per-frame Decoding Latency was found to be extremely low. Log analysis confirms that the processing time required by the `server_scanning.py` module to process and analyze a frame is on average around **150ms**. This performance is crucial, as it attests that the slowness in code acquisition is entirely attributable to the user (alignment difficulty) and not to a computational bottleneck. Consequently, the overall Time on Task (ToT) for the first successful scan is on average less than **5seconds** from the start of the active scanning cycle, a result that confirms the efficiency of the Python backend.

Robustness: QR Code vs. Barcodes Benchmarking revealed a clear distinction in performance based on the format of the code analyzed: The QR Code (Electronic Prescription) scanning was immediate and effective in almost all tests. This is due to the high distortion tolerance and error correction mechanism built into the QR format, which makes it extremely robust to read even at unfavorable angles. Barcodes (Health Card/CF) acquisition, on the other hand, was slower and significantly more sensitive to environmental factors. The main failure issue was caused by the reflection of ambient light on the glossy document backing or the cellphone screen, which hid or distorted the fine lines at the pyzbar module. This technical vulnerability fully justifies the HRI logic that uses social feedback to guide the user to reposition the code until success.

5.2.3 Final Task Pipeline Score ($Score_{FTPS}$)

To validate the robustness of the interaction chain, the individual accuracy and recovery results are aggregated into the $Score_{FTPS}$, according to the formula defined in Section 3.2. The basic metrics used for the normalized calculation ($\mathbf{M_i}$) are summarized below.

Table 2: Quantitative Results and Final Task Pipeline Score Calculation

Module	$\mathbf{w_i}$	$\mathbf{M_i}$	**Data Value**	$\mathbf{w_i \cdot M_i}$
ASR Accuracy ($\mathbf{M_1}$)	0.20	0.676	$WER = 32.4\%$	0.135
Social Accuracy ($\mathbf{M_2}$)	0.15	0.965	$FRAccuracy \approx 96.5\%$	0.145
Planning/Reasoning ($\mathbf{M_3}$)	0.35	1.00	$ActionAccuracy = 100\%$	0.350
Regulatory Accuracy ($\mathbf{M_4}$)	0.30	0.88	$RecoverySuccessRate = 88\%$	0.264
Final Task Pipeline Score ($Score_{FTPS} = \sum w_i \cdot M_i$)				**0.894**

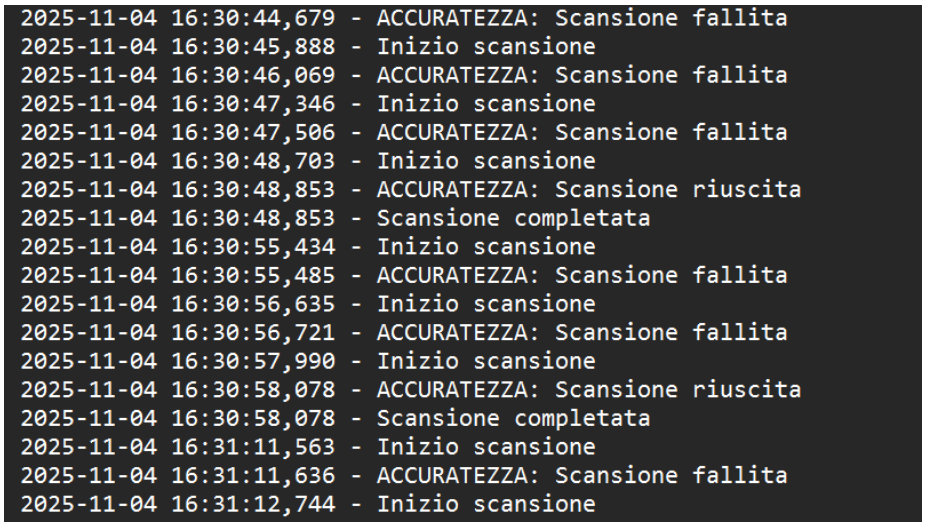

Figure 9: Log Scanning

The $Score_{FTPS}$ calculation is:

$$Score_{FTPS} = 0.20(0.676) + 0.15(0.965) + 0.35(1.00) + 0.30(0.88) = \mathbf{0.894}$$

The system is deemed successful, having achieved a score of **0.894**, significantly exceeding the target threshold of 0.70.

6 Conclusion

The project successfully achieved its goal of implementing a robotic pharmacy assistant by rigorously merging the disciplines of HRI, design ethics, and Robot Benchmarking (RBC). The system demonstrated excellent cognitive and normative robustness, as demonstrated by the Final Task Pipeline Score of **0.894**, significantly exceeding the success threshold, and driven by the high reliability of the Reasoning Logic ($\mathbf{M_3} = 1.00$) and the Scan Error Recovery Logic ($\mathbf{M_4} = 0.88$). The experience confirmed the validity of the Privacy-by-Design approach, where explicit consent for the use of facial and health data is not only an ethical requirement (GDPR) but also a functional element for identification and personalization. However, the implementation highlighted several concrete challenges: execution was severely hampered by hardware limitations, with insufficiently powerful PCs slowing down backend processes. At the software level, reliance on a cloud-based LLM (Mistral) caused frequent outages due to out-of-quota issues, compromising service continuity and the fluidity of the interaction. The ASR Accuracy metric ($\mathbf{M_1} = 0.676$) revealed the significant need for near-absolute silence to ensure speech understanding, an unacceptable limitation in a noisy real-world pharmacy environment. Looking to the future, the system could

evolve towards greater clinical utility: customer knowledge should be deepened, allowing the robot to actively remember current therapies, previously taken medications, or allergies (currently not implemented due to ethical and storage complexities). This integration of deep contextual memory would enable more precise and personalized advice (for example, alerts on known drug interactions or treatment suggestions based on the patient's history), transforming the robot from a regulatory assistant into a proactive healthcare advisor, ideally managed through a hybrid architecture with a Vector Database to overcome the LLM's quota limitations. The lessons learned emphasize that the success of an HRI robot in the real world depends as much on its advanced control logic as on its resilience to operational failures and its ability to ensure fast and continuous interaction even in suboptimal environmental conditions.

References

[1] Robot Benchmarking and Competitions 2025. Lecture 3, benchmarking1.

[2] David Kaber Michael Lewis Jean Scholtz Alan Schultz Michael Goodrich Aaron Steinfeld, Terrence Fong. Common metrics for human-robot interaction. 2006.

[3] Dominykas Strazdas Jan Hintz Thorsten Hempel Aly Khalifa, Ahmed A. Abdelrahman and Ayoub Al-Hamadi. Face recognition and tracking framework for human–robot interaction. applied sciences. 2022.

[4] Elizabeth Croft Susana Zoghbi Christoph Bartneck, Dana Kulic. Measurement instruments for the anthropomorphism, animacy, likeability, perceived intelligence, and perceived safety of robots. In *Some venue*, 2008.

[5] Xianren Zhang Zongyu Wu Tzuhao Mo Qiuhao Lu Wanjing Wang Rui Li Junjie Xu Xianfeng Tang Qi He Yao Ma Ming Huang Suhang Wang. Fali Wang, Zhiwei Zhang. A comprehensive survey of small language models in the era of large language models: Techniques, enhancements, applications, collaboration with llms, and trustworthiness. 2024.

[6] Shahed Saleh Kerstin Dautenhahn Hamza Mahdi, Sami Alperen Akgun. A survey on the design and evolution of social robots — past, present and future. 2022.

[7] Nadia Thalmann Hangyeol Kang, Maher Ben Moussa. Nadine: An llm-driven intelligent social robot with affective capabilities and human-like memory. 2024.

[8] Muhammad A. Shah David Solans Noguero Mikko A. Heikkilä Nicolas Kourtellis. Speech robust bench: A robustness benchmark for speech recognition. 2024.

[9] Gabriele Russo Francesco Rea Luca Garello, Giulia Belgiovine and Alessandra Sciutti. Building knowledge from interactions: An llm-based architecture for adaptive tutoring and social reasoning. 2025.

[10] Susanne Hägglund Linda Estman Sara Rosenberg, Malin Andtfolk. Examining the use and outcomes of social robot interventions in medication processes - a scoping review. 2025.

[11] Susanne Hägglund Mattias Wingren Linda Nyholm Sara Rosenberg, Malin Andtfolk. Social robots counselling in community pharmacies – helping or harming? a qualitative study of pharmacists' views. 2024.

[12] Dong Yi Shengcai Liao, Zhen Lei and Stan Z. Li. A benchmark study of large-scale unconstrained face recognition. In *IEEE International Joint Conference on Biometrics*, 2014.