UPANZI
DIGITAL PUBLIC INFRASTRUCTURE
NETWORK

Pepper Robot Tour for Digital Experience Center

D4.3 Conversation Manager

Due date: **28/02/2026**
Submission Date: **06/03/2026**
Revision Date: **15/08/2026**

Start date of project: **01/07/2026**

Duration: **12 months**

Responsible Person: **Yohannes Haile**

Revision: **1.2**

Project funded by Upanzi Network Dissemination Level		
PU	Public	**PU**
PP	Restricted to other programme participants	
RE	Restricted to a group specified by the consortium	
CO	Confidential, only for members of the consortium	

Executive Summary

Deliverable D4.3 presents the Conversation Manager module for the Pepper robot at the Digital Experience Center (DEC) of the Upanzi Network at Carnegie Mellon University Africa. The module is implemented as a ROS2 Lifecycle Node named `conversation_manager` that uses Retrieval-Augmented Generation (RAG) to produce contextually grounded responses about the Upanzi Network's Digital Public Infrastructure (DPI) research. When a visitor utterance arrives via a ROS2 action goal, the node performs a semantic vector search over a ChromaDB knowledge base populated from the Upanzi Network data file (`upanzi_data.json`), submits the retrieved context together with the conversation history to an OpenAI-compatible LLM, and streams the response internally sentence-by-sentence as it is generated, accumulating it into the full answer text. The full response text, classified intent, and confidence score are returned as the action result; downstream playback is handled by the BehaviorTree `SpeechWithFeedback` node, which is the sole consumer responsible for speech and gesture output.

Contents

1 Introduction

Natural conversation capability is central to the DEC tour experience. Visitors interact with Pepper using speech, asking questions about exhibits and giving tour instructions. The Conversation Manager translates transcribed visitor speech into contextually relevant, knowledge-grounded responses using a RAG pipeline built on top of an OpenAI-compatible LLM.

The module is implemented as a ROS2 Lifecycle Node named `conversation_manager`, part of the `conversation_manager` package. The node:

- Exposes a ROS2 action server (`dec_interfaces/action/ConversationManager`) that accepts a visitor utterance as a goal prompt.
- Performs a semantic vector search over a ChromaDB knowledge base populated from the Upanzi Network data file (`upanzi_data.json`).
- Constructs a message sequence comprising the system prompt, recent conversation history, and retrieved context, then calls the configured LLM.
- Streams the LLM response internally, sentence-by-sentence, as tokens arrive, accumulating them into the final answer text returned in the action result.
- Returns the complete response text, classified intent, and confidence score as the action result.
- Maintains a rolling conversation history across turns for multi-turn dialogue.

2 Requirements Definition

Functional Requirements

- The module must accept visitor utterances via a ROS2 action interface and return a generated response, intent classification, and confidence score.
- Responses must be grounded in the Upanzi Network knowledge base, covering Digital Public Infrastructure (DPI), Digital Public Goods (DPGs), cybersecurity, and Carnegie Mellon University Africa topics.
- Each utterance must be classified into one of eight intent categories: `ASK_EXHIBIT_QUESTION`, `ASK_TOUR_META`, `NAVIGATION_REQUEST`, `SOCIAL_SMALL_TALK`, `OFF_TOPIC`, `STOP`, `AFFIRMATIVE`, or `NEGATIVE`.
- Responses must be limited to a maximum of one sentence and returned strictly as a JSON object with `intent`, `confidence`, and `answer` fields.
- The module must stream the LLM response incrementally, sentence-by-sentence, as tokens arrive rather than waiting for the full response to complete, so downstream consumers see minimal added latency.
- Multi-turn context must be maintained across consecutive action goals via a rolling conversation history.

RAG and LLM Requirements

- The knowledge base must be stored in a ChromaDB persistent collection, populated from `upanzi_data.json` on first launch and reused on subsequent launches.
- Semantic search must use the `all-MiniLM-L6-v2` SentenceTransformer embedding model.
- The LLM endpoint must follow the OpenAI Chat Completions API. The `LLM_API_KEY` must be supplied as an environment variable; no API key may be stored in the configuration file.
- The LLM model, base URL, similarity threshold, history window, and all other tunable parameters must be configurable via the YAML configuration file.
- Response latency must be acceptable for real-time interaction; streaming is used to minimise perceived latency.

Lifecycle Requirements

- The node must implement the ROS2 Managed Node lifecycle (`configure` → `activate` → `deactivate` → `cleanup`).
- ChromaDB initialisation and configuration loading must occur in `on_configure`; the action server must only be registered, and goals only accepted, once configuration succeeds and the node reaches the ACTIVE state.

3 Module Specification

3.1 Functional Description

The `conversation_manager` node exposes a `dec_interfaces/action/ConversationManager` action server. When a goal is received, the node executes the following pipeline:

1. **Vector search.** The visitor utterance is embedded using `all-MiniLM-L6-v2` and queried against the ChromaDB collection. The top-*k* results above the similarity threshold are retrieved as context. Feedback status `"searching"` is published.

2. **Prompt construction.** A message list is assembled: the system prompt (loaded from `data/system_prompt.txt`), the most recent `context_turns` of conversation history as alternating `user`/`assistant` messages, and the current user message with the retrieved context appended.

3. **Streaming LLM call.** The message list is submitted to the LLM via a streaming Chat Completions request. Feedback status `"generating"` is published. As each sentence of the `answer` JSON field is decoded it is yielded internally and accumulated into the full response text; no per-sentence topic is published — the accumulated text is returned in full once streaming completes.

4. **Result extraction.** After streaming completes, the full raw LLM response is parsed to extract the `answer`, `intent`, and `confidence` fields. Thinking-model chain-of-thought blocks (`<think>...</think>`) are stripped before parsing.

5. **History update.** The query–response pair is appended to the rolling conversation history, capped at `max_history_turns`.

6. **Action result.** `success`, `response`, `intent`, and `confidence` are returned to the action client.

4 Module Design

4.1 Node Architecture

The package follows the same two-file application/implementation split used throughout this work package. `conversation_manager_application.py` is the ROS2 entry point: it initialises `rclpy`, instantiates the `ConversationManagerNode` class, and spins it. All RAG logic lives in `conversation_manager_implementation.py`. Shared utility helpers (ANSI colour codes, verbose printers, YAML and type-coercion helpers) are isolated in `conversation_manager_utilities.py`.

`ConversationManagerNode` extends `rclpy.lifecycle.LifecycleNode`. Heavy initialisation is deferred to `on_configure`: the YAML configuration file is loaded and validated, the ChromaDB collection is initialised from `upanzi_data.json` (or loaded from disk if it already exists), and the action server is registered. The node is ready to handle queries only after it has been transitioned to the ACTIVE state.

4.2 RAG Pipeline

Knowledge Base Initialisation

On `on_configure`, the node calls `setup_collection()`, which uses `chromadb.PersistentClient` in embedded mode (no separate ChromaDB server required). If the named collection already exists on disk it is reused directly; otherwise, `upanzi_data.json` is parsed by `parse_upanzi_format()`, which extracts structured documents for the categories `lab_info`, `goals`, `impact`, `facilities`, `thrust_areas`, and `projects`. Each document is indexed with its title, keywords, and body text as a single searchable string. Embeddings are computed via the `SentenceTransformerEmbeddingFunction` wrapper with the `all-MiniLM-L6-v2` model, and the collection is stored using cosine similarity as the distance metric.

Semantic Search

At query time, `search()` calls `collection.query()` with the visitor utterance as the query text. The raw cosine distances are converted to similarity scores (score $= 1 - d$), and results below `similarity_threshold` are discarded before being returned to the caller.

Streaming Response Generation

`generate_response_stream()` constructs the message list and opens a streaming Chat Completions request. The LLM is expected to return a JSON object of the form {`"intent": "...", "confidence": 0.0--1.0, "answer": "..."`}. The generator locates the `"answer"` key in the partial JSON buffer and decodes its string value character-by-character as tokens arrive, yielding complete sentences (terminated by `.`, `!`, or `?`) to the caller. Thinking-model chain-of-thought sections enclosed in `<think>...</think>` tags are skipped transparently. After streaming ends, the full raw buffer is available to extract `intent` and `confidence` via `extract_intent_from_raw()`.

4.3 Conversation History

The node maintains `conversation_history` as a list of {`query, response`} dictionaries in memory. On each action call, the most recent `context_turns` entries are injected into the LLM message list as alternating `user`/`assistant` messages before the current query. The list is capped at `max_history_turns` entries after each successful turn.

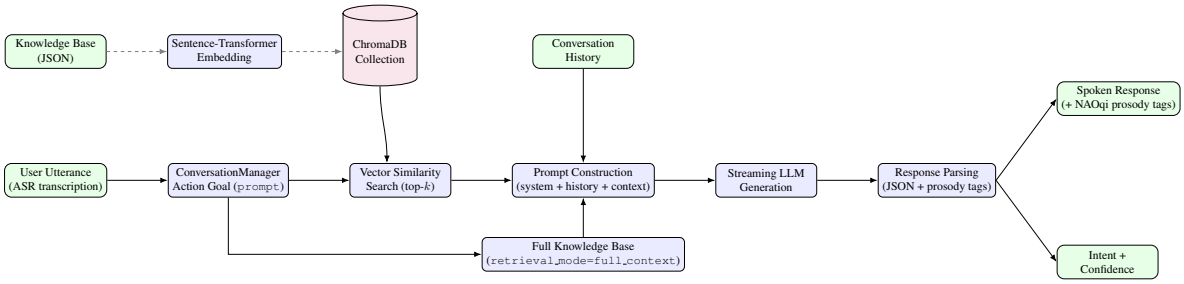

Figure 1: Conversation Manager retrieval-augmented generation (RAG) pipeline. Dashed arrows show the offline knowledge-base ingestion path; solid arrows show the runtime query path. When `retrieval_mode = full_context`, vector search is bypassed and the entire knowledge base is passed to the LLM every turn instead.

5 Implementation

File Organization

```
dec_system
└── conversation_manager
    ├── config
    │   └── converation_manager_configuration.yaml
    ├── data
    │   ├── system_prompt.txt
    │   └── upanzi_data.json
    ├── conversation_manager
    │   ├── __init__.py
    │   ├── conversation_manager_application.py
    │   ├── conversation_manager_implementation.py
    │   └── conversation_manager_utilities.py
    ├── resource
    │   └── conversation_manager
    ├── package.xml
    ├── requirements.txt
    ├── setup.cfg
    ├── setup.py
    └── README.md
```

Figure 2: File structure of the conversation manager package.

Configuration File

The node is controlled by ROS2 parameters declared under the `conversation_manager` node namespace in `config/converation_manager_configuration.yaml` and loaded via the launch file. The `LLM_API_KEY` must be exported as an environment variable and must not appear in this file, since ROS2 parameters are visible through introspection tools such as `ros2paramdump`.

Key	Default / Example	Description
llm_base_url	https://api.deepseek.com/v1	Base URL of the OpenAI-compatible LLM endpoint.
llm_model	deepseek-chat	Model identifier passed to the LLM API.
embedding_model	all-MiniLM-L6-v2	SentenceTransformer model used for ChromaDB embeddings.
retrieval_mode	rag	Retrieval strategy: rag (vector search over ChromaDB, only relevant snippets sent to the LLM) or full_context (skip vector search, send the entire knowledge base each turn).
similarity_threshold	0.15	Minimum cosine similarity score (0.0–1.0) for a retrieved document to be included as context (rag mode only).
top_k	5	Maximum number of documents retrieved per query (rag mode only).
max_history_turns	15	Maximum number of turns stored in the rolling conversation history.
context_turns	10	Number of most recent turns injected into each LLM request.
max_response_sentences	3	Sizes the LLM token budget (sentences $\times$ 50 + 512); does not by itself lengthen the spoken answer, which the system prompt still caps at one sentence.
data_default_path	./data/upanzi_data.json	Path to the JSON knowledge base file.
collection_name	upanzi_knowledge	Name of the ChromaDB collection that stores the indexed knowledge base.
verbose	true	Enables detailed terminal logging of search results, LLM requests, and streamed responses.

Table 1: Configuration file key-value pairs for the conversation manager node.

Topics Subscribed

The conversation_manager node does not subscribe to any ROS2 topics directly. Visitor utterances are received exclusively through the action server interface described below.

Topic Name	Message Type	Description
None — input is received via the action server interface.		

Table 2: Topics subscribed by the conversation manager node.

Topics Published

The conversation_manager node does not publish any ROS2 topics. Streamed sentences are accumulated internally rather than being published incrementally; the complete response text, intent, and confidence are delivered exclusively through the action server's Result, and interim progress through its Feedback field (see Action Server below). Spoken playback is handled downstream by the BehaviorTree SpeechWithFeedback node, which calls the /naoqi_driver/speech_with_feedback action directly — this is outside the conversation_manager package.

Topic Name	Message Type	Description
None — output is delivered via the action server's Feedback and Result fields.		

Table 3: Topics published by the conversation manager node.

Action Server

Field	Type	Description
Action name	—	`/conversation_manager`
Action type	—	`dec_interfaces/action/ConversationManager`
Goal: `prompt`	`string`	Visitor's transcribed utterance to process.
Feedback: `status`	`string`	`"searching"` during vector retrieval; `"generating"` during LLM streaming.
Result: `success`	`bool`	`true` if the query was handled successfully.
Result: `response`	`string`	Full generated answer text.
Result: `intent`	`string`	Classified intent label (see Table 5).
Result: `confidence`	`float`	LLM-reported confidence score in [0.0, 1.0].

Table 4: Action server interface of the conversation manager node.

Intent Classification

The LLM is instructed to classify each visitor utterance and return the intent label in the `intent` field of the JSON response.

Intent	Description
`ASK_EXHIBIT_QUESTION`	Visitor asks a question about DPI, DPGs, cybersecurity, or CMU-Africa research.
`ASK_TOUR_META`	Visitor asks a meta question about the tour itself.
`NAVIGATION_REQUEST`	Visitor requests Pepper to move to a location.
`SOCIAL_SMALL_TALK`	Visitor engages in social conversation (greetings, compliments, etc.).
`OFF_TOPIC`	Visitor asks about something outside the supported topic areas.
`STOP`	Visitor asks Pepper to stop. No `answer` text is returned.
`AFFIRMATIVE`	Visitor expresses agreement or affirmation; `answer` is set to `"yes"`.
`NEGATIVE`	Visitor expresses disagreement or refusal; `answer` is set to `"no"`.

Table 5: Intent classification categories returned by the LLM.

Data Files

File	Description
`data/upanzi_data.json`	JSON knowledge base containing lab information, goals, impact statements, facilities, thrust areas, and project descriptions for the Upanzi Network. Parsed by `parse_upanzi_format()` and loaded into ChromaDB on first run.
`data/system_prompt.txt`	Plain-text system prompt loaded at runtime and prepended to every LLM request. Instructs Pepper to respond only to DPI/DPG/cybersecurity topics, enforce a one-sentence limit, output strict JSON, and classify visitor intent.

Table 6: Data files used by the conversation manager node.

Building and Launching

```
# Export the LLM API key (required before launching)
export LLM_API_KEY=<your-api-key>

# Build the package
cd ~/dec_ws
colcon build --packages-select conversation_manager
source install/setup.bash

# Launch the node (starts UNCONFIGURED - no automatic lifecycle
   transitions)
ros2 launch conversation_manager conversation_manager.launch.py

# Trigger the lifecycle transitions manually
ros2 lifecycle set /conversation_manager configure
ros2 lifecycle set /conversation_manager activate
```

Sending a Query

```
# Send an action goal from the terminal
ros2 action send_goal /conversation_manager \
    dec_interfaces/action/ConversationManager \
    "{prompt: 'What is MOSIP?'}"
```

6 Unit Testing

Test Types

- **Specification verification**: Tests that the node correctly classifies utterances into the eight intent categories and that the confidence score is within $[0,\ 1]$.
- **Requirements validation**: Tests that responses are factually grounded in the Upanzi Network knowledge base and stay within the one-sentence limit.
- **Benchmark evaluation**: Measures end-to-end action latency (from goal submission to the first sentence yielded internally by the streaming generator) and full response latency for a set of DEC-specific prompts.

Example Test Cases

Input Prompt	Expected Intent	Expected Behaviour
"What is MOSIP?"	`ASK_EXHIBIT_QUESTION`	Response grounded in MOSIP content from `upanzi_data.json`; one sentence.
"Please go to the cybersecurity exhibit."	`NAVIGATION_REQUEST`	Polite acknowledgement; one sentence.
"Yes, please continue."	`AFFIRMATIVE`	`answer` field is exactly `"yes"`; confidence ≥ 0.8.
"What is the weather today?"	`OFF_TOPIC`	Polite refusal; no DPI content returned.
"Stop."	`STOP`	No `answer` text; action result `success = true`.

Table 7: Example test cases for the Conversation Manager.

References

Principal Contributors

The main authors of this deliverable are as follows (in alphabetical order).

Yohannes Haile, Carnegie Mellon University Africa.

Document History

Version 1.0

First draft.
Yohannes Haile.
28 February 2026.

Version 1.1

Added System Diagram for the Conversational Manger.
Updates to the topic names.
Yohannes Haile.
08 July 2026.

Version 1.2

Updated the configuration table to the ROS2-parameters format adopted in `pepper4dec`: the node now reads flat parameters declared under the `conversation_manager` node namespace (`llm_base_url`, `similarity_threshold`, `data_default_path`, `verbose`, and so on) loaded via the launch file, rather than a nested plain-YAML file, and the new `collection_name` parameter (the ChromaDB collection name) was added.
Yohannes Haile.
15 August 2026.