# NAAMIKA

# VOICE ASSISTANT SYSTEM

**Complete Technical Documentation**

---

**Document Version:** v2.1.7 (v2_demo_22Jan)

**Classification:** Internal Technical Reference

**Generated:** January 27, 2026

**Repository:** github.com/bhaveshbakshi633/indu_v2_1_2

**Branch:** v2_demo_22Jan

---

**Prepared by:** Engineering Team

**Organization:** SS Innovations International

---

# TABLE OF CONTENTS

# CHAPTER 1: EXECUTIVE SUMMARY

## 1.1 Document Purpose

This document provides comprehensive technical documentation for the NAAMIKA Voice Assistant System, a voice-controlled interface for the Unitree G1 humanoid robot. It serves as the authoritative reference for system architecture, code implementation, deployment procedures, and operational guidelines.

## 1.2 System Overview

NAAMIKA is a distributed voice-controlled humanoid robot assistant designed for the Unitree G1 platform. The system integrates multiple AI components to enable natural voice interaction and physical robot control.

**Core Capabilities:**

| Capability | Technology | Status |
|---|---|---|
| Voice Recognition | Whisper STT (GPU) | Operational |
| Natural Language Understanding | Ollama LLM + RAG | Operational |
| Text-to-Speech | Chatterbox TTS | Operational |
| Intent Classification | Pattern Matching + LLM | Operational |
| Emergency Stop | Fast-path Detection | Operational |
| Gesture Control | HTTP Action Bridge | Operational |
| Locomotion | Time-based Motion | Operational |
| SLAM Navigation | Unitree SDK | Operational |
| Face Recognition | Shakal | Operational |

## 1.3 Architecture Summary

The system operates across four networked machines:

| Machine | Role | IP Address |
|---|---|---|
| This PC (ssi) | Brain Server, TTS | 172.16.6.19 |
| G1 Robot | Motion Control, Audio | 172.16.2.242 |
| Isaac PC | LLM Inference | 172.16.4.226 |
| New PC (b) | STT Inference | 172.16.4.250 |

## 1.4 Critical Design Principles

**Safety-First Architecture:**

- All physical actions require FSM state validation
- MEDIUM/HIGH risk actions require explicit user confirmation
- Emergency STOP bypasses all processing pipelines instantly
- 22-action whitelist prevents unauthorized command execution

**Distributed Processing:**

- Compute-intensive tasks (LLM, STT) offloaded to GPU servers
- HTTP-based communication for reliability over ROS2
- Graceful degradation with fallback mechanisms

# CHAPTER 2: SYSTEM ARCHITECTURE

## 2.1 High-Level System Diagram

The following diagram illustrates the complete system architecture including all four machines and their interconnections.

```
+=============================================================================+
|                     NAAMIKA VOICE ASSISTANT SYSTEM                          |
|                        Distributed Architecture                             |
+=============================================================================+

+----------------------------------+     +----------------------------------+
|      THIS PC (172.16.6.19)       |     |     ISAAC PC (172.16.4.226)     |
|                                  |     |                                  |
|  +----------------------------+  |     |  +----------------------------+  |
|  |    BRAIN SERVER            |  |     |  |    OLLAMA LLM SERVER       |  |
|  |    Port: 8080 (HTTPS)      |  |     |  |    Port: 11434             |  |
|  |                            |  |     |  |    Model: naamika:v1       |  |
|  |  +----------------------+  |  |     |  +----------------------------+  |
|  |  | VAD Pipeline         |  |  |     |               ^                  |
|  |  | - Speech Detection   |  |  |     |               | HTTP             |
|  |  | - Interrupt Monitor  |  |  |     +---------------|------------------+
|  |  +----------------------+  |  |                     |
|  |           |                |  |     +---------------|------------------+
|  |           v                |  |     |     NEW PC (172.16.4.250)       |
|  |  +----------------------+  |  |     |                                  |
|  |  | STOP Fast-Path       |  |  |     |  +----------------------------+  |
|  |  | Emergency Detection  |  |  |     |  |   WHISPER STT SERVER       |  |
|  |  +----------------------+  |  |     |  |   Port: 8001               |  |
|  |           |                |  |     |  |   Model: medium            |  |
|  |           v                |  |     |  +----------------------------+  |
|  |  +----------------------+  |  |     |               ^                  |
|  |  | Intent Reasoner      |------|----|---------------|                  |
|  |  | - Pattern Matching   |  |  |     +---------------|-----------------+
|  |  | - LLM Classification |  |  |                     | HTTP
|  |  +----------------------+  |  |                     |
|  |           |                |  |                     |
|  |     +------+------+        |  |                     |
|  |     |             |        |  |                     |
|  |     v             v        |  |                     |
|  | [ACTION]    [CONVERSATION] |  |                     |
|  |     |             |        |  |                     |
|  |     v             v        |  |                     |
|  | +--------+  +----------+   |  |                     |
|  | | Action |  | RAG+LLM  |---|--+---------------------+
|  | |Registry|  | Agent    |   |  |
|  | +--------+  +----------+   |  |
|  +----------------------------+  |
|               |                  |
|  +----------------------------+  |
|  |   CHATTERBOX TTS SERVER    |  |
|  |   Port: 8000               |  |
|  +----------------------------+  |
|               |                  |
+---------------|------------------+
                | HTTP (Audio PCM)
                v
+==============================================================================+
|                        G1 ROBOT (172.16.2.242)                               |
|                                                                              |
|  +------------------------------------------------------------------------+  |
|  |                        DOCKER CONTAINERS                                |  |
|  |                                                                         |  |
|  |  +----------------+  +----------------+  +---------------------------+  |  |
|  |  | audio_receiver |  | tts_audio_     |  | startup_health_check     |  |  |
|  |  | Port: 5050     |  | player         |  | Service Monitoring       |  |  |
|  |  +----------------+  +----------------+  +---------------------------+  |  |
|  |                                                                         |  |
|  |  +----------------+  +----------------+  +---------------------------+  |  |
|  |  | g1_orchestrator|  | arm_controller |  | http_action_bridge       |  |  |
```

```
|   |   | Main FSM        |   | API 7106/7108   |   | Port: 5051                  |   |   |
|   |   +----------------+   +----------------+   +---------------------------+   |   |
|   |                                                                               |   |
|   |   +----------------+   +----------------+   +---------------------------+   |   |
|   |   | talking_        |   | status_         |   | shakal                      |   |   |
|   |   | gestures        |   | announcer       |   | Face Recognition            |   |   |
|   |   +----------------+   +----------------+   +---------------------------+   |   |
|   +-------------------------------------------------------------------------+   |
|                                                                                   |
|   +-------------------------------------------------------------------------+   |
|   |              SLAM HTTP SERVER (Native - Port 5052)                        |   |
|   |              systemd: slam_server.service                                 |   |
|   +-------------------------------------------------------------------------+   |
+===============================================================================+
```

## 2.2 Network Configuration

| Service | Machine | IP Address | Port | Protocol |
|---|---|---|---|---|
| Voice Assistant (Brain) | This PC | 172.16.6.19 | 8080 | HTTPS |
| Chatterbox TTS | This PC | 172.16.6.19 | 8000 | HTTP |
| Whisper STT | New PC (b) | 172.16.4.250 | 8001 | HTTP |
| Ollama LLM | Isaac PC | 172.16.4.226 | 11434 | HTTP |
| G1 Audio Receiver | G1 Robot | 172.16.2.242 | 5050 | HTTP |
| G1 HTTP Action Bridge | G1 Robot | 172.16.2.242 | 5051 | HTTP |
| G1 SLAM Server | G1 Robot | 172.16.2.242 | 5052 | HTTP |

## 2.3 Data Flow Pipeline

The voice processing pipeline consists of six stages from user speech to robot action or response.

```
User Speech Input
       |
       v
+-------------------------------------------------------+
| STAGE 1: VOICE ACTIVITY DETECTION (VAD)               |
| File: vad_pipeline/vad_processor.py                   |
|                                                       |
| - Silero VAD model (16kHz, 512 chunk size)           |
| - Multi-frame confirmation (3 frames @ 0.5)          |
| - Interrupt detection (1.0 threshold)                |
+-------------------------------------------------------+
       |
       v
+-------------------------------------------------------+
| STAGE 2: EMERGENCY STOP CHECK (Fast Path)             |
| File: stop_fast_path.py                               |
|                                                       |
| Keywords: "stop", "halt", "freeze", "ruk", "bas"     |
| Action: Immediate DAMP mode, bypass all processing   |
+-------------------------------------------------------+
       |
       v (if not STOP)
+-------------------------------------------------------+
| STAGE 3: SPEECH-TO-TEXT (STT)                         |
| Server: Whisper @ 172.16.4.250:8001                   |
|                                                       |
| - Model: faster_whisper medium                       |
| - GPU accelerated (CUDA, float16)                    |
| - Multi-language: English, Hindi                     |
+-------------------------------------------------------+
       |
       v
+-------------------------------------------------------+
| STAGE 4: INTENT DETECTION                             |
| File: intent_reasoner.py                              |
|                                                       |
| Priority 1: Pattern matching (ACTION_PATTERNS)       |
| Priority 2: LLM classification (if pattern fails)    |
| Output: IntentResult(type, action, confidence)       |
+-------------------------------------------------------+
       |
       +--------------------+--------------------+
       |                                         |
       v                                         v
+--------------------+                 +-------------------------+
| STAGE 5A: ACTION   |                 | STAGE 5B: RAG + LLM     |
|                    |                 | File: naamika_rag.py    |
| - Risk check       |                 |                         |
| - Confirmation     |                 | - FAISS retrieval       |
| - HTTP dispatch    |                 | - Ollama generation     |
| - SLAM client      |                 | - Memory management     |
+--------------------+                 +-------------------------+
       |                                         |
       v                                         v
+--------------------+                 +-------------------------+
| G1 ROBOT           |                 | STAGE 6: TTS            |
| EXECUTION          |                 | Server: Chatterbox      |
|                    |                 |                         |
| - API 7106/7108    |                 | - Audio generation      |
| - API 7105         |                 | - PCM streaming to G1   |
| - SLAM navigation |                 | - Speaker playback      |
+--------------------+                 +-------------------------+
```

# CHAPTER 3: VOICE ACTIVITY DETECTION PIPELINE

## 3.1 Overview

The VAD pipeline detects when a user starts and stops speaking using the Silero VAD neural network model. This enables hands-free voice interaction without push-to-talk buttons.

**File Location:** `vad_pipeline/`

## 3.2 VAD Processor

**File:** `vad_pipeline/vad_processor.py`

### 3.2.1 Configuration Parameters

```
class VADProcessor:
    def __init__(
        self,
        threshold: float = 0.5,           # Speech confidence threshold
        frame_count: int = 3,              # Frames for confirmation
        sample_rate: int = 16000,          # Audio sample rate (Hz)
        chunk_size: int = 512,             # Samples per chunk
        interrupt_threshold: float = 1.0,  # Strict interrupt threshold
        interrupt_frames: int = 5          # Frames for interrupt
    ):
```

### 3.2.2 Core Methods

**Single Frame Detection:**

```
def is_speech(self, audio_chunk: np.ndarray) -> Tuple[bool, float]:
    """
    Process single audio chunk through Silero VAD.
    Returns: (is_speech: bool, probability: float)
    """
    audio_tensor = torch.from_numpy(audio_chunk).float()
    speech_prob = self.model(audio_tensor, self.sample_rate).item()
    return speech_prob >= self.threshold, speech_prob
```

**Multi-Frame Speech Start Detection:**

```
def detect_speech_start(self, audio_chunk) -> bool:
    """
    Requires multiple consecutive frames above threshold.
    Prevents false positives from transient noise.
    """
    is_speech, prob = self.is_speech(audio_chunk)
    self.frame_buffer.append(prob)

    if len(self.frame_buffer) >= self.frame_count:
        recent = self.frame_buffer[-self.frame_count:]
        return all(p >= self.threshold for p in recent)
    return False
```

**Interrupt Detection During Playback:**

```
def detect_interrupt(self, audio_chunk) -> bool:
    """
    Used during TTS playback to detect user interruption.
    Uses stricter threshold across multiple frames.
    """
    is_speech, prob = self.is_speech(audio_chunk)
    self.interrupt_buffer.append(prob)

    if len(self.interrupt_buffer) >= self.interrupt_frames:
        high_conf = sum(1 for p in self.interrupt_buffer
                       if p >= self.interrupt_threshold)
        return high_conf >= (0.67 * self.interrupt_frames)
    return False
```

## 3.3 Pipeline State Machine

The pipeline manager operates as a three-state machine:

```
+---------------+      Speech       +---------------+
|   LISTENING   |----Detected----->|   RECORDING   |
+---------------+                   +---------------+
        ^                                  |
        |                              Silence
        |                              Timeout
        |                                  |
        |                                  v
        |                          +---------------+
        +-------Playback-----------|   PLAYING     |
                Complete           +---------------+
```

## 3.4 Audio Recorder

**File:** `vad_pipeline/audio_recorder.py`

```
class AudioRecorder:
    def __init__(
        self,
        vad_processor: VADProcessor,
        sample_rate: int = 16000,
        silence_timeout: float = 1.5,   # Stop after 1.5s silence
        max_duration: float = 30.0       # Maximum recording length
    ):

    def record_from_monitor(self, audio_monitor) -> bytes:
        """Record audio until silence detected."""
        buffer = []
        silence_start = None

        while True:
            chunk = audio_monitor.get_chunk()
            is_speech, prob = self.vad.is_speech(chunk)
            buffer.append(chunk)

            if is_speech:
                silence_start = None
            else:
                if silence_start is None:
                    silence_start = time.time()
                elif time.time() - silence_start > self.silence_timeout:
                    break

            if self.get_buffer_duration() > self.max_duration:
                break

        return np.concatenate(buffer).tobytes()
```

# CHAPTER 4: SPEECH-TO-TEXT PROCESSING

## 4.1 Whisper Server Architecture

**File:** `whisper_server.py` **Deployment:** 172.16.4.250:8001

### 4.1.1 Server Configuration

```
MODELS_AVAILABLE = {
    "tiny": "tiny",       # 39M params, 1GB VRAM
    "base": "base",       # 74M params, 2GB VRAM
    "small": "small",     # 244M params, 3GB VRAM
    "medium": "medium",   # 769M params, 5GB VRAM (DEFAULT)
    "large": "large-v3"   # 1.5B params, 10GB VRAM
}

DEFAULT_MODEL = "medium"
DEVICE = "cuda"
COMPUTE_TYPE = "float16"
PORT = 8001
```

### 4.1.2 Transcription Endpoint

```
@app.post("/transcribe")
async def transcribe(
    file: UploadFile,
    model: str = "medium",
    language: Optional[str] = None
):
    """
    Process audio file and return transcript.

    Request:
        - file: WAV audio file
        - model: tiny/base/small/medium/large
        - language: en, hi, es, etc. (auto-detect if None)

    Response:
        {
            "text": "transcribed text",
            "language": "en",
            "model": "medium",
            "duration_seconds": 5.2,
            "processing_time_seconds": 0.8
        }
    """
    whisper_model = load_model(model)

    segments, info = whisper_model.transcribe(
        audio_path,
        language=language,
        task="transcribe",
        beam_size=5
    )

    text = " ".join([s.text for s in segments]).strip()

    return TranscribeResponse(
        text=text,
        language=info.language,
        model=model,
        duration_seconds=info.duration,
        processing_time_seconds=time.time() - start
    )
```

## 4.2 Brain Server STT Integration

**File:** `server.py`

```python
async def transcribe_audio(audio_bytes: bytes) -> str:
    """Send audio to Whisper server and get transcript."""
    config = load_config()
    stt_config = config["stt"]["whisper_server"]

    url = f"http://{stt_config['host']}:{stt_config['port']}/transcribe"

    async with httpx.AsyncClient(timeout=30.0) as client:
        files = {"file": ("audio.wav", audio_bytes, "audio/wav")}
        params = {
            "model": stt_config.get("model", "medium"),
            "language": stt_config.get("language", "en")
        }

        response = await client.post(url, files=files, params=params)
        response.raise_for_status()
        return response.json()["text"]
```

# CHAPTER 5: INTENT DETECTION & REASONING

## 5.1 Overview

The Intent Reasoner determines whether user speech is an action command or conversational input. It uses a two-tier approach: fast pattern matching first, LLM classification as fallback.

**File:** `intent_reasoner.py`

## 5.2 Intent Types and Results

```
class IntentType(Enum):
    ACTION = "action"           # Physical robot action
    CONVERSATION = "conversation"  # LLM response needed
    QUERY = "query"             # Information request

@dataclass
class IntentResult:
    intent_type: IntentType
    action_name: Optional[str]    # e.g., "WAVE", "FORWARD"
    confidence: float             # 0.0 to 1.0
    requires_confirmation: bool   # MEDIUM/HIGH risk
    reason: str                   # Human-readable explanation
    original_transcript: str      # Original STT output
```

## 5.3 Action Pattern Dictionary

```
ACTION_PATTERNS = {
    # System actions (HIGH priority)
    "initialize": ("INIT", 0.95),
    "boot up": ("INIT", 0.95),
    "chalu karo": ("INIT", 0.95),     # Hindi

    "ready": ("READY", 0.90),
    "get ready": ("READY", 0.95),
    "taiyaar": ("READY", 0.90),       # Hindi

    "damp": ("DAMP", 0.95),
    "relax": ("DAMP", 0.85),

    # Posture actions
    "stand up": ("STANDUP", 0.95),
    "utho": ("STANDUP", 0.95),        # Hindi
    "sit down": ("SIT", 0.95),

    # Motion actions
    "walk forward": ("FORWARD", 0.95),
    "aage": ("FORWARD", 0.90),        # Hindi
    "walk back": ("BACKWARD", 0.95),
    "turn left": ("LEFT", 0.95),
    "turn right": ("RIGHT", 0.95),

    # Gestures
    "wave": ("WAVE", 0.95),
    "shake hand": ("SHAKE_HAND", 0.95),
    "hug": ("HUG", 0.95),
    "namaste": ("namaste1", 0.95),

    # SLAM
    "start mapping": ("START_MAPPING", 0.95),
    "stop mapping": ("STOP_MAPPING", 0.95),
    "start navigation": ("START_NAVIGATION", 0.95),
    "list waypoints": ("LIST_WAYPOINTS", 0.95),
}
```

## 5.4 Local Pattern Matching

```
def parse_intent_local(transcript: str) -> IntentResult:
    """
    Fast local intent parsing without LLM.

    Priority:
    1. Check if question (always conversation)
    2. Check SLAM parameterized commands
    3. Check direct action patterns
    4. Default to conversation
    """
    lower = transcript.lower().strip()

    # Questions are always conversation
    if is_question(transcript):
        return IntentResult(
            intent_type=IntentType.CONVERSATION,
            action_name=None,
            confidence=0.95,
            requires_confirmation=False,
            reason="Question detected"
        )

    # Check SLAM parameterized commands
    waypoint = is_save_waypoint_command(transcript)
    if waypoint:
        return IntentResult(
            intent_type=IntentType.ACTION,
            action_name=f"SAVE_WAYPOINT:{waypoint}",
            confidence=0.90,
            requires_confirmation=True,
            reason=f"Save waypoint: '{waypoint}'"
        )

    # Check direct pattern match
    match = match_action_pattern(transcript)
    if match:
        action_name, confidence = match
        needs_confirm = action_name in [
            "INIT", "DAMP", "ZERO_TORQUE",
            "STANDUP", "SIT", "FORWARD", "BACKWARD"
        ]
        return IntentResult(
            intent_type=IntentType.ACTION,
            action_name=action_name,
            confidence=confidence,
            requires_confirmation=needs_confirm,
            reason=f"Matched pattern '{action_name}'"
        )

    # Default to conversation
    return IntentResult(
        intent_type=IntentType.CONVERSATION,
        action_name=None,
        confidence=0.8,
        reason="No action pattern matched"
    )
```

## 5.5 Confirmation Flow

### Confirmation Patterns

```
POSITIVE_PATTERNS = [
    "yes", "yeah", "yep", "ok", "okay", "sure", "confirm",
    "go ahead", "absolutely",
    "haan", "ha", "theek", "bilkul", "zaroor"  # Hindi
]

NEGATIVE_PATTERNS = [
    "no", "nope", "cancel", "stop", "don't", "abort",
    "nahi", "mat", "ruk", "rehne do"  # Hindi
]
```

### IntentReasoner State Machine

```
class IntentReasoner:
    def __init__(self):
        self.pending_action = None
        self.awaiting_confirmation = False

    def process(self, transcript: str) -> IntentResult:
        # Check if awaiting confirmation first
        if self.awaiting_confirmation and self.pending_action:
            if is_confirmation(transcript):
                action = self.pending_action
                self.clear_pending()
                return IntentResult(
                    intent_type=IntentType.ACTION,
                    action_name=action,
                    confidence=1.0,
                    requires_confirmation=False,
                    reason="User confirmed"
                )
            elif is_rejection(transcript):
                self.clear_pending()
                return IntentResult(
                    intent_type=IntentType.CONVERSATION,
                    reason="User cancelled"
                )

        # Normal intent parsing
        result = parse_intent_local(transcript)

        # Store for confirmation if needed
        if result.requires_confirmation:
            self.pending_action = result.action_name
            self.awaiting_confirmation = True

        return result
```

# CHAPTER 6: ACTION REGISTRY & SAFETY SYSTEM

## 6.1 Overview

The Action Registry defines all valid robot actions with their risk levels, FSM requirements, and API mappings. This serves as the single source of truth for action validation.

**File:** `action_registry.py`

## 6.2 Risk Classification

```
class RiskLevel(Enum):
    LOW = "low"       # Gestures - immediate execution
    MEDIUM = "medium" # Locomotion - needs confirmation
    HIGH = "high"     # System commands - extra validation

class ActionType(Enum):
    GESTURE = "gesture"
    MOTION = "motion"
    POSTURE = "posture"
    SYSTEM = "system"
    MODE = "mode"
    QUERY = "query"
```

## 6.3 FSM State Constants

```
# Unitree G1 FSM States
FSM_ZERO_TORQUE = 0   # Motors off - robot may fall
FSM_DAMP = 1          # Damping mode - gentle stop
FSM_SQUAT = 2         # Squat position
FSM_SIT = 3           # Sitting position
FSM_STANDUP = 4       # Standing from sit/lie
FSM_START = 500       # Sport/active mode
FSM_READY = 801       # Ready for arm control

# Valid FSM sets
FSM_ANY = {0, 1, 2, 3, 4, 500, 801}
FSM_STANDING = {500, 801}
FSM_READY_ONLY = {801}
```

## 6.4 Action Definitions (Selected)

| Action | Type | Risk | Required FSM | API |
|---|---|---|---|---|
| WAVE | Gesture | LOW | 801 | 7106:0 |
| SHAKE_HAND | Gesture | LOW | 801 | 7106:2 |
| HUG | Gesture | LOW | 801 | 7108:hug |
| NAMASTE | Gesture | LOW | 801 | 7108:namaste1 |
| FORWARD | Motion | MEDIUM | 500, 801 | 7105 |
| BACKWARD | Motion | MEDIUM | 500, 801 | 7105 |
| LEFT | Motion | MEDIUM | 500, 801 | 7105 |
| RIGHT | Motion | MEDIUM | 500, 801 | 7105 |
| STOP | Motion | LOW | Any | 7105:stop |
| STANDUP | Posture | MEDIUM | 1, 4 | 7101:4 |
| SIT | Posture | MEDIUM | 801 | 7101:3 |
| INIT | System | HIGH | Any | orchestrator:init |
| DAMP | System | HIGH | Any | 7101:1 |

## 6.5 Emergency Stop Fast Path

**File:** `stop_fast_path.py`

### Stop Keywords

```
class StopType(Enum):
    EMERGENCY = "emergency"   # "emergency", "emergency stop"
    IMMEDIATE = "immediate"   # "stop", "halt", "freeze"
    CASUAL = "casual"         # "ruk", "bas" (Hindi)

EMERGENCY_KEYWORDS = ["emergency", "emergency stop"]
IMMEDIATE_KEYWORDS = ["stop", "halt", "freeze", "stop now"]
CASUAL_KEYWORDS = ["ruk", "ruk jao", "bas", "band karo"]

# Exclusions (valid commands, not emergencies)
STOP_EXCLUSIONS = ["stop mapping", "stop navigation", "stop talking"]
```

### Detection Logic

```
def detect_stop(transcript: str) -> StopDetectionResult:
    """
    Check for STOP keywords BEFORE any other processing.
    Runs immediately after STT, bypasses LLM entirely.
    """
    lower = transcript.lower().strip()

    # First check exclusions
    for exclusion in STOP_EXCLUSIONS:
        if exclusion in lower:
            return StopDetectionResult(is_stop=False)

    # Priority: emergency > immediate > casual
    for stop_type, keywords in ALL_STOP_KEYWORDS.items():
        for keyword in keywords:
            if keyword in lower:
                return StopDetectionResult(
                    is_stop=True,
                    stop_type=stop_type,
                    matched_keyword=keyword
                )

    return StopDetectionResult(is_stop=False)
```

# CHAPTER 7: RAG-ENHANCED LLM PROCESSING

## 7.1 Overview

The NAAMIKA RAG Agent combines conversation memory with knowledge base retrieval for accurate, context-aware responses about SSi medical robotics.

**File:** `naamika_rag.py`

## 7.2 Architecture

```
User Query
     |
     v
+-------------------------------------------+
| STEP 1: Knowledge Retrieval (RAG)         |
|                                           |
| - Embed query with MiniLM-L6-v2         |
| - Search FAISS vector store             |
| - Retrieve top 5 relevant chunks        |
+-------------------------------------------+
     |
     v
+-------------------------------------------+
| STEP 2: Prompt Construction               |
|                                           |
| - CRITICAL_FACTS (anti-hallucination)   |
| - Conversation history                  |
| - RAG context                           |
| - User query                            |
+-------------------------------------------+
     |
     v
+-------------------------------------------+
| STEP 3: LLM Generation                    |
|                                           |
| - Ollama with naamika:v1 model          |
| - Temperature: 0.2                      |
| - Context: 4096 tokens                  |
+-------------------------------------------+
     |
     v
+-------------------------------------------+
| STEP 4: Response Processing               |
|                                           |
| - Save to conversation history          |
| - Return for TTS                        |
+-------------------------------------------+
```

## 7.3 Anti-Hallucination Facts

```
CRITICAL_FACTS = """
CRITICAL RULES - FOLLOW EXACTLY:
1. If you don't find information, say "I don't have that" - NEVER fabricate.
2. For competitors (da Vinci, Medtronic), redirect to SSi advantages.
3. You are NAAMIKA (NOT INDU), a humanoid robot. NOT the CEO.
4. NO MARKDOWN: Natural sentences only.

DR. SRIVASTAVA FACTS (USE EXACTLY):
- Education: J.L.N. Medical College, Ajmer
- Returned to India: 2011
- Awards: Golden Robot Surgical Award 2025
- World Record: 1,300+ beating heart TECAB surgeries

SSi STATS (December 2025):
- Installations: 168 worldwide
- Surgeries: 7,800+ performed
- Headquarters: Gurugram, India

SSi MANTRA SPECS:
- Components: FOUR (Surgeon Console, Arms, Vision, MUDRA)
- Degrees of Freedom: 7 DOF
- Scaling: 2:1, 3:1, 4:1
- Cost: Less than one-third of competitors
"""
```

## 7.4 NaamikaAgent Class

```
class NaamikaAgent:
    def __init__(
        self,
        knowledge_base_path: str = "naamika_knowledge_base.txt",
        system_prompt_path: str = "naamika_system_prompt.txt",
        model_name: str = "naamika:v1",
        vector_store_path: str = "naamika_vectorstore",
        ollama_host: str = "127.0.0.1",
        ollama_port: int = 11434,
        temperature: float = 0.2,
        max_history: int = 10
    ):
        self.conversation_history = []

        # Initialize embeddings
        self.embeddings = HuggingFaceEmbeddings(
            model_name="sentence-transformers/all-MiniLM-L6-v2"
        )

        # Load/create FAISS vector store
        self.vectorstore = FAISS.load_local(vector_store_path, ...)

        # Initialize retriever
        self.retriever = self.vectorstore.as_retriever(
            search_kwargs={"k": 5, "fetch_k": 20}
        )

        # Initialize LLM
        self.llm = OllamaLLM(
            model=model_name,
            base_url=f"http://{ollama_host}:{ollama_port}",
            temperature=temperature
        )
```

## 7.5 Chat Method

```
def chat(self, user_query: str) -> str:
    """Send query and get response with memory + RAG."""

    # 1. RAG retrieval
    rag_context = self._retrieve_context(user_query)

    # 2. Format conversation history
    chat_history = self._format_chat_history()

    # 3. Build prompt with CRITICAL_FACTS
    prompt = f"""{CRITICAL_FACTS}

CONVERSATION HISTORY:
{chat_history}

RELEVANT KNOWLEDGE:
{rag_context}

USER: {user_query}

RESPONSE:"""

    # 4. Call LLM
    response = self.llm.invoke(prompt).strip()

    # 5. Save to history
    self.conversation_history.append((user_query, response))

    return response
```

# CHAPTER 8: TEXT-TO-SPEECH PIPELINE

## 8.1 Overview

The TTS pipeline converts LLM responses to audio and streams to G1's speaker for natural voice output.

**Files:**

- `chatterbox_tts_client.py` - Client library
- External: Chatterbox server at 172.16.6.19:8000

## 8.2 Chatterbox TTS Client

```
class ChatterboxTTS:
    def __init__(self, base_url: str = "http://172.16.6.19:8000"):
        self.base_url = base_url

    async def generate(
        self,
        text: str,
        voice: str = None,
        output_file: str = "output.wav"
    ) -> bytes:
        """Generate speech audio from text."""
        params = {"text": text}
        if voice:
            params["voice"] = voice

        async with httpx.AsyncClient(timeout=120.0) as client:
            response = await client.get(
                f"{self.base_url}/tts",
                params=params
            )
            response.raise_for_status()
            return response.content
```

## 8.3 Audio Streaming to G1

```
async def send_audio_to_g1(audio_bytes: bytes):
    """Stream PCM audio to G1's audio_receiver."""
    url = f"http://172.16.2.242:5050/audio"

    async with httpx.AsyncClient(timeout=30.0) as client:
        response = await client.post(
            url,
            content=audio_bytes,
            headers={"Content-Type": "audio/wav"}
        )
        response.raise_for_status()
```

## 8.4 Response Chunking

```
async def process_llm_response(response_text: str):
    """Chunk text for low-latency TTS playback."""

    # Split into speakable chunks
    chunks = chunk_text(
        response_text,
        first_chunk_size=50,   # Small first chunk
        rest_chunk_size=150    # Larger rest
    )

    for chunk in chunks:
        audio = await chatterbox_client.generate(chunk)
        await send_audio_to_g1(audio)
```

# CHAPTER 9: SLAM NAVIGATION SYSTEM

## 9.1 Overview

The SLAM system enables autonomous mapping and navigation using the Unitree SDK.

**Files:**

- `slam_client.py` - HTTP client for brain
- `slam_http_server.py` - G1 HTTP server
- `slam_control.cpp` - Unitree SDK binary

## 9.2 SLAM Architecture

```
+-----------------------------------------------------------------+
|                    SLAM NAVIGATION SYSTEM                        |
+-----------------------------------------------------------------+

Brain (172.16.6.19)             G1 Robot (172.16.2.242)

+-------------------+           +----------------------------+
|  slam_client.py  |    HTTP    |  slam_http_server.py       |
|                  |---------->|  Port 5052                  |
|  - start_mapping |            |                             |
|  - stop_mapping  |            |  +------------------------+ |
|  - save_waypoint |            |  | slam_control (C++)     | |
|  - goto_waypoint |            |  | Unitree SDK            | |
|  - list_waypoints|            |  +------------------------+ |
+-------------------+           |              |               |
                                |              v               |
                                |  +------------------------+ |
                                |  | Unitree SLAM Service   | |
                                |  +------------------------+ |
                                |                             |
                                |  +------------------------+ |
                                |  | waypoints.json         | |
                                |  | map_registry.json      | |
                                |  | *.pcd map files        | |
                                |  +------------------------+ |
                                +----------------------------+
```

## 9.3 SLAM API IDs (Unitree SDK)

| API ID | Function |
| --- | --- |
| 1801 | START_MAPPING |
| 1802 | END_MAPPING |
| 1804 | START_RELOCATION |
| 1102 | POSE_NAV |
| 1201 | PAUSE_NAV |
| 1202 | RESUME_NAV |

## 9.4 SLAM Client Methods

```
class SLAMClient:
    def __init__(self, host="172.16.2.242", port=5052):
        self.base_url = f"http://{host}:{port}"

    # Mapping Operations
    def start_mapping(self) -> SLAMResponse
    def stop_mapping(self, map_name: str = None) -> SLAMResponse

    # Navigation Operations
    def start_navigation(self, map_name: str = None) -> SLAMResponse
    def goto_waypoint(self, name: str) -> SLAMResponse
    def pause_navigation(self) -> SLAMResponse
    def resume_navigation(self) -> SLAMResponse

    # Waypoint Management
    def save_waypoint(self, name: str) -> SLAMResponse
    def list_waypoints(self) -> SLAMResponse
    def delete_waypoint(self, name: str) -> SLAMResponse

    # Map Management
    def list_maps(self) -> SLAMResponse
    def delete_map(self, name: str) -> SLAMResponse
```

## 9.5 Voice Commands for SLAM

| Command | Action | Description |
|---|---|---|
| "start mapping" | START_MAPPING | Begin SLAM mapping |
| "stop mapping as office" | STOP_MAPPING:office | Save map as "office" |
| "start navigation office" | START_NAV:office | Load "office" map |
| "save this as kitchen" | SAVE_WAYPOINT:kitchen | Save current position |
| "go to kitchen" | GOTO_WAYPOINT:kitchen | Navigate to waypoint |
| "list waypoints" | LIST_WAYPOINTS | Show saved waypoints |
| "pause" | PAUSE_NAV | Pause navigation |
| "resume" | RESUME_NAV | Resume navigation |

# CHAPTER 10: G1 ROBOT INTEGRATION

## 10.1 HTTP Action Bridge

**Endpoint:** `POST http://172.16.2.242:5051/action`

```
@app.post("/action")
async def execute_action(request: ActionRequest):
    action_name = request.action.upper()

    # Validate action
    if action_name not in VALID_ACTIONS:
        raise HTTPException(400, f"Unknown: {action_name}")

    # Check FSM state
    if not is_fsm_compatible(action_name, get_current_fsm()):
        return {"success": False, "error": "Invalid FSM"}

    # Execute
    if action_name in GESTURE_ACTIONS:
        execute_gesture(action_name)
    elif action_name in MOTION_ACTIONS:
        execute_motion(action_name)

    return {"success": True, "action": action_name}
```

## 10.2 Unitree API Reference

| API ID | Purpose | Example |
|---|---|---|
| 7101 | Set FSM State | SetFsmId(801) |
| 7104 | Height Control | HighStand(), LowStand() |
| 7105 | Move Control | Move(vx, vy, vyaw) |
| 7106 | Built-in Arm Actions | WaveHand(), ShakeHand() |
| 7108 | Custom Arm Actions | ExecuteCustomAction("hug") |

### API 7106 - Built-in Tasks

| ID | Action |
|---|---|
| 0 | WaveHand (no turn) |
| 2 | ShakeHand (Stage 0) |
| 18 | HIGH_FIVE |
| 19 | HUG |
| 26 | WAVE |

### API 7108 - Custom Actions

```
namaste1, namaste2, explain1-5, self, self_hug,
battery, Handshake, release_arm
```

## 10.3 Motion Parameters

```
MOTION_PARAMS = {
    "FORWARD":  {"vx": 0.3,  "vy": 0, "vyaw": 0,    "duration": 2.0},
    "BACKWARD": {"vx": -0.3, "vy": 0, "vyaw": 0,    "duration": 2.0},
    "LEFT":     {"vx": 0,    "vy": 0, "vyaw": 0.3,  "duration": 1.5},
    "RIGHT":    {"vx": 0,    "vy": 0, "vyaw": -0.3, "duration": 1.5},
}
```

## 10.4 FSM State Transitions

```
ZERO_TORQUE (0)
      |
      | DAMP
      v
   DAMP (1)
      |
      | STANDUP
      v
  STANDUP (4)
      |
      | READY
      v
   READY (801)  <-- Arms Enabled
```

**Boot Sequence:** 0 → 1 → 4 → 801

## 10.5 Docker Containers on G1

| Container | Purpose | Port |
|---|---|---|
| audio_receiver | HTTP audio endpoint | 5050 |
| tts_audio_player | Speaker playback | - |
| g1_orchestrator | Main FSM controller | - |
| arm_controller | Real-time arm (500Hz) | - |
| http_action_bridge | HTTP to ROS2 | 5051 |
| talking_gestures | Auto gestures | - |
| status_announcer | Voice feedback | - |
| shakal | Face recognition | - |

# CHAPTER 11: CONFIGURATION SYSTEM

## 11.1 Main Configuration (config.json)

```json
{
  "g1_audio": {
    "enabled": true,
    "host": "172.16.2.242",
    "port": 5050,
    "gain": 0.5
  },
  "stt_backend": "whisper_server",
  "tts_backend": "chatterbox",
  "ollama_model": "naamika:v1",
  "enable_rag": true,
  "vad_threshold": 0.5,
  "silence_duration": 0.5,

  "stt": {
    "whisper_server": {
      "host": "172.16.4.250",
      "port": 8001,
      "model": "medium"
    }
  },
  "llm": {
    "deployment_mode": "remote",
    "remote": {
      "host": "172.16.4.226",
      "port": 11434
    }
  },
  "slam": {
    "host": "172.16.2.242",
    "port": 5052
  }
}
```

# CHAPTER 12: API REFERENCE

## 12.1 Brain Server (172.16.6.19:8080)

| Endpoint | Method | Description |
|---|---|---|
| `/` | GET | Web interface |
| `/stream` | WebSocket | Audio streaming |
| `/api/health` | GET | Health check |

## 12.2 G1 Action Bridge (172.16.2.242:5051)

| Endpoint | Method | Body |
|---|---|---|
| `/action` | POST | `{"action": "WAVE"}` |
| `/health` | GET | - |

## 12.3 G1 SLAM Server (172.16.2.242:5052)

| Endpoint | Method | Body |
|---|---|---|
| `/slam/start_mapping` | POST | - |
| `/slam/stop_mapping` | POST | `{"map_name": "office"}` |
| `/slam/relocate` | POST | `{"map_name": "office"}` |
| `/slam/goto_waypoint` | POST | `{"name": "kitchen"}` |
| `/waypoint/save` | POST | `{"name": "kitchen"}` |
| `/waypoint/list` | GET | - |
| `/map/list` | GET | - |

## 12.4 External Services

| Service | Endpoint |
| --- | --- |
| Chatterbox TTS | `GET /tts?text=Hello` |
| Whisper STT | `POST /transcribe` (multipart) |
| Ollama LLM | `POST /api/chat` |

# CHAPTER 13: DEPLOYMENT GUIDE

## 13.1 System Requirements

| Machine | OS | GPU | Purpose |
|---|---|---|---|
| This PC | Ubuntu 22.04+ | Optional | Brain, TTS |
| Isaac PC | Ubuntu 22.04+ | 8GB+ VRAM | LLM |
| New PC | Ubuntu 22.04+ | 5GB+ VRAM | STT |
| G1 Robot | ROS2 Humble | - | Control |

## 13.2 Startup Sequence

**Step 1: External Services**

```
~/scripts/g1_services.sh start chatterbox
~/scripts/g1_services.sh start ollama
~/scripts/g1_services.sh start whisper
```

**Step 2: G1 Robot**

```
ssh unitree@172.16.2.242
sudo systemctl start slam_server
cd ~/deployed/v2_1_7 && docker-compose up -d
```

**Step 3: Brain**

```
cd ~/Downloads/naamika_brain_v2_1_7
python3 server.py
```

**Step 4: Access** Open: https://172.16.6.19:8080

## 13.3 SSH Reference

| Machine | Command | Password |
|---|---|---|
| G1 | `ssh unitree@172.16.2.242` | 123 |
| Isaac | `ssh isaac@172.16.4.226` | 7410 |
| New PC | `ssh b@172.16.4.250` | 1997 |

# CHAPTER 14: TROUBLESHOOTING

## 14.1 Brain Server

**Port in use:**

```
lsof -i :8080 && kill -9 <PID>
```

## 14.2 STT Issues

**Server not responding:**

```
ssh b@172.16.4.250
curl http://localhost:8001/health
```

## 14.3 LLM Issues

**Ollama not responding:**

```
ssh isaac@172.16.4.226
OLLAMA_HOST=0.0.0.0:11434 ollama serve
```

## 14.4 G1 Robot Issues

**Docker containers:**

```
docker-compose logs -f
docker-compose down && docker-compose up -d
```

**FSM stuck:**

```
curl -X POST http://172.16.2.242:5051/action \
  -d '{"action": "DAMP"}'
```

# APPENDIX A: FILE REFERENCE

## A.1 Brain Directory

```
naamika_brain_v2_1_7/
├── server.py              # Main server
├── config.json            # Configuration
├── intent_reasoner.py     # Intent detection
├── action_registry.py     # Actions
├── stop_fast_path.py      # Emergency STOP
├── slam_client.py         # SLAM client
├── naamika_rag.py         # RAG agent
├── chatterbox_tts_client.py
├── vad_pipeline/
│   ├── vad_processor.py
│   ├── audio_recorder.py
│   └── pipeline_manager.py
└── naamika_vectorstore/   # FAISS index
```

## A.2 G1 Directory

```
/home/unitree/deployed/v2_1_7/
├── docker-compose.yml
├── config/
└── src/
    ├── g1_orchestrator/
    ├── arm_controller/
    ├── http_action_bridge/
    └── audio_player/

/home/unitree/slam_control/
├── slam_control.cpp
└── slam_http_server.py
```

# APPENDIX B: CONFIGURATION VALUES

| Parameter | Value |
|---|---|
| Brain Port | 8080 |
| Audio Receiver | 5050 |
| Action Bridge | 5051 |
| SLAM Server | 5052 |
| Chatterbox | 8000 |
| Whisper | 8001 |
| Ollama | 11434 |
| VAD Threshold | 0.5 |
| Motion Duration | 2.0s |
| Turn Duration | 1.5s |

**End of Document**

*NAAMIKA Voice Assistant System - Technical Documentation v2.1.7*

*SS Innovations International*

*Generated: January 27, 2026*