NerdCarX - Architecture Overview

> **Last updated:** 2026-01-13
> **Current Phase:** 1 (Desktop Complete)

Table of Contents

---

1. Executive Summary

NerdCarX is an AI-powered robot car that can hold conversations in Dutch, display emotions, and perceive its surroundings through a camera. It's built on a PiCar-X platform with a Raspberry Pi 5, but all heavy AI processing happens on a separate desktop server with GPUs.

Why local-first? Everything runs locally - no cloud APIs, no subscription costs, no privacy concerns, no rate limits. The modular architecture means any component (speech recognition, language model, text-to-speech) can be swapped without rewriting the system.

Current capability: The desktop demo is fully functional. You can have hands-free conversations in Dutch, the system recognizes emotional context and responds appropriately, and it can describe what it "sees" through a camera. Next: connecting the actual robot hardware.

---

2. Design Rationale

Why Local-First?

Benefit	Explanation
No recurring costs	Cloud APIs charge per request. Local = one-time hardware investment
No rate limits	Experiment freely without API throttling
Privacy	Conversations never leave your network
Low latency	No round-trip to cloud servers
Learning opportunity	Understanding the full stack, not just API calls
Full control	Tune parameters, switch models, modify behavior

Cloud fallback is planned (Phase 4+) for flexibility, but local-first remains the default.

Why Desktop + Pi Split?

The AI models are too demanding for a Raspberry Pi 5, even with 16GB RAM:

Model	Issue on Pi 5	Verdict
Voxtral 3B (STT)	Requires GPU for acceptable speed; CPU inference far too slow	Incompatible
Ministral 14B (LLM)	Requires GPU; ~20GB VRAM needed; CPU would take minutes per response	Incompatible
Fish Audio (TTS)	Requires GPU for real-time synthesis	Incompatible

The problem isn't just RAM - the Pi 5's CPU simply cannot run these models at usable speeds. Even if models fit in memory, inference would be orders of magnitude slower than on a GPU. These models are designed for GPU compute.

> *Note: Voxtral currently uses ~15GB VRAM due to vLLM's memory management, not model size. Could be optimized, but wasn't a priority with sufficient VRAM available.*

Solution: Desktop does the "thinking" (GPU-heavy AI), Pi does the "sensing and acting" (mic, speaker, motors, display). They communicate over LAN via WebSocket.

Why These Technologies?

Choice	Why	Alternatives Considered
Ollama (LLM serving)	Simple setup, one command to run models, built-in API, great for local development	llama.cpp (lower level), vLLM (overkill for single model)
vLLM (STT serving)	Required for Voxtral - it's a Mistral model needing proper vLLM support, handles batching efficiently	Ollama (doesn't support Voxtral), TGI (less mature)
FastAPI (Orchestrator)	Lightweight, async, full control, no magic. LangChain considered but adds complexity without clear benefit for this use case	LangChain (too abstract), Flask (not async)
Docker	Reproducible environments, GPU isolation, easy to deploy and version	Conda only (works but less portable)
WebSocket (Pi ↔ Desktop)	Bidirectional, low latency, persistent connection for real-time interaction	REST (higher latency, polling needed), MQTT (overkill for point-to-point)

Why These Models?

Model	Why This One	Key Deciding Factors
Voxtral Mini 3B	Dutch is one of only 8 officially supported languages. Better noise robustness than Whisper. Can do audio Q&A, not just transcription.	Tested Faster-Whisper - less robust in noisy environments

Model	Why This One	Key Deciding Factors
Ministral 14B	Native function calling support, vision capability, reasonable size for 24GB GPU. Official Mistral parameters (temp=0.15) reduce hallucinations.	Tested 8B variant - 14B Q8 noticeably better quality
Fish Audio S1-mini	#1 on TTS-Arena2 benchmark, ~1.2s latency (vs 5-20s for Chatterbox). Dutch via voice cloning with reference audio.	Tested Chatterbox (too slow), Piper (less expressive), VibeVoice (Belgian accent and very unreliable low quality results)
YOLO Nano/Small	Runs on Pi 5's GPU, real-time object detection, well-documented, many pre-trained variants	Full YOLO too heavy for Pi
Porcupine (Wake word)	Accurate, low CPU, custom wake words, works offline, has hobby license	Snowboy (discontinued), Mycroft Precise (less accurate)
Silero VAD	Local, no network, reliable voice activity detection, works with Python	WebRTC VAD (less accurate), cloud VAD (defeats local-first)

> For complete decision history with dates and alternatives: DECISIONS.md

---

3. What Can It Do?

Use Case	Description	Status
Voice Conversation	Ask questions in Dutch, get spoken answers	Working
Emotional Responses	Robot maintains persistent emotional state (not per-message) shown on OLED display	Working
Visual Awareness	"What do you see?" - robot describes its surroundings	Working
Movement Commands	"Drive forward", "turn left" - physical robot control	Phase 3

Use Case	Description	Status
Proactive Interaction	Robot initiates conversation when idle or detects a person	Phase 4

Example interaction:

```
User: "Hallo, hoe heet je?"
Robot: [shows happy emotion] "Hoi! Ik ben NerdCarX, een robotauto gemaakt door Ralph."

User: "Wat zie je?"
Robot: [takes photo, analyzes] "Ik zie een bureau met een laptop en een koffiemok."

User: "Je bent stom!"
Robot: [shows sad emotion] "Dat doet pijn om te horen..."
```

> For detailed use cases (UC1-UC9), see archive/0.concept/picar-x-ai-companion-concept.md

---

4. Hardware Platform: PiCar-X

NerdCarX is built on the **SunFounder PiCar-X** platform - an AI-ready robot car kit that comes with significant capabilities out of the box. Our AI integration enhances these, but the hardware itself is quite capable.

Hardware Components

Component	Description
Robot HAT	Expansion board with motor driver, I2S audio, mono speaker, PWM/ADC, LED, button
2-Axis Camera Mount	Pan/tilt servos for camera positioning
Camera Module	OV5647, 5MP, 1080p/30fps, 720p/60fps, 65° FOV
Ultrasonic Sensor	HC-SR04, 2-400cm range, 3mm accuracy

Component	Description
Grayscale Module	3-channel line/cliff detection
DC Motors	Differential drive for movement
Steering Servo	Front wheel steering
Battery Pack	2x 18650, 2000mAh, 7-12V
OLED Display *(added)*	0.96" I2C (128x64) - Shows robot's emotional state (see Emotion State Machine)

Built-in Capabilities (Standard PiCar-X)

These capabilities come standard with PiCar-X using its included scripts, OpenCV, and optional cloud APIs (OpenAI, etc.). Our stack replaces/enhances these with local, high-quality alternatives:

Capability	How It Works	Our AI Enhancement
Basic Movement	Motor control: forward, backward, turn, steer	Voice commands: "drive forward"
Obstacle Avoidance	Ultrasonic detects obstacles, auto-stops/turns	LLM decides how to respond, explains what it sees
Cliff Detection	Grayscale sensors detect edges/drops	Contextual awareness in conversations
Line Tracking	Follow lines on ground using grayscale	Could be voice-activated: "follow the line"
Face Tracking	OpenCV detects faces, servos follow	Combined with emotion recognition
Color Detection	OpenCV detects specific colors	"Drive towards the red object"
Video Streaming	Stream camera to web/app	Real-time monitoring + AI analysis
App Control	SunFounder Controller app	Voice control replaces manual control

Capability	How It Works	Our AI Enhancement
Sound/Music	Play audio via I2S speaker	TTS for voice responses
Keyboard Control	Control via keyboard input	Natural language replaces key presses

What Our Local AI Stack Adds

Standard PiCar-X	+ NerdCarX Local AI
Pre-programmed responses or cloud API calls	Natural conversation in Dutch, fully local
Fixed behavior patterns	Context-aware, adaptive responses
Manual or simple auto control	Voice-commanded actions
Basic TTS (espeak) or cloud TTS	High-quality Dutch TTS (Fish Audio, local)
Command-based interaction	Conversational interaction with personality
Reactive only	Proactive (initiates conversation)
No memory	Remembers context, learns preferences
Single capability at a time	Integrated: talk while moving, emote while responding
Cloud API dependency (optional)	Privacy-first, no recurring costs

Reference

For detailed PiCar-X documentation, examples, and tutorials:

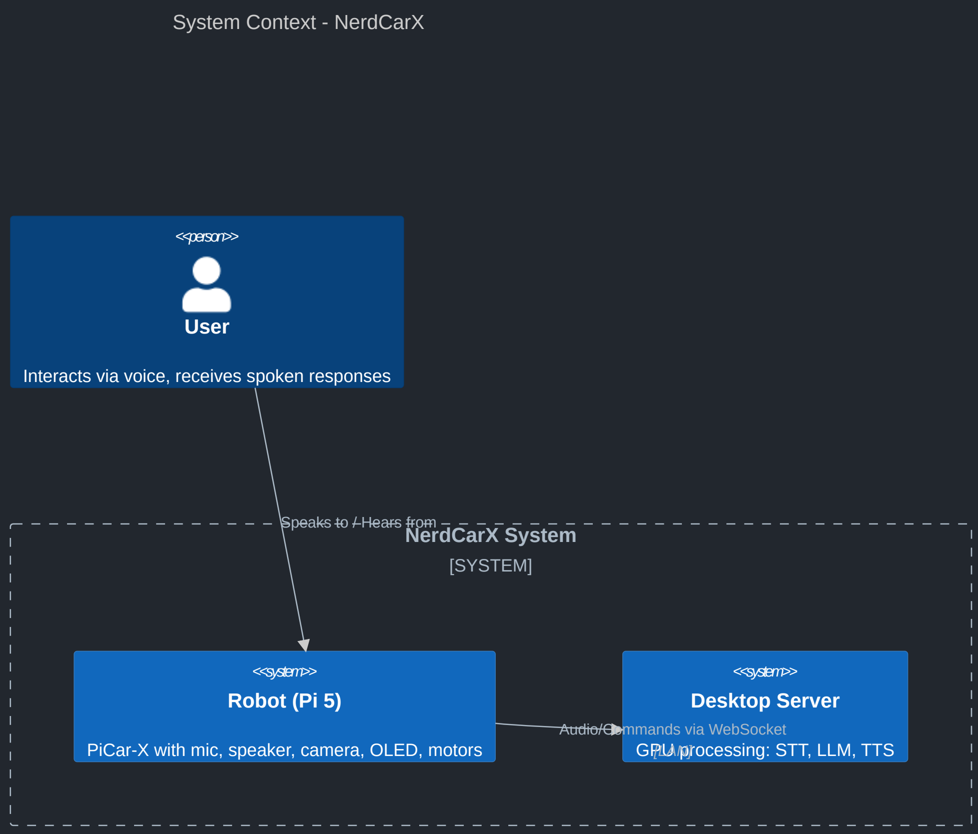
- original_Picar-X-REFERENCE/Documentation/

---

5. System Context (C1)

This diagram shows NerdCarX in its environment - who uses it and what systems it connects to.

Target Architecture (Phase 3+)



> **Current state (Phase 1):** No hardware yet. User interacts via desktop microphone/speaker. All components run on desktop for development and testing.

Why split between Pi and Desktop?

Component	On Pi 5	On Desktop	Reason
Microphone/Speaker	Yes	-	Physical location
Wake word detection	Yes	-	Must always listen, low power
VAD (Voice Activity Detection)	Yes	-	Low latency, local processing
Object Detection (YOLO)	Yes	-	Real-time awareness, runs on Pi GPU

Component	On Pi 5	On Desktop	Reason
OLED/Motors	Yes	-	Hardware control
Speech-to-Text	-	Yes	Needs GPU (15GB VRAM)
Language Model	-	Yes	Needs GPU (20GB VRAM)
Text-to-Speech	-	Yes	Needs GPU, quality matters

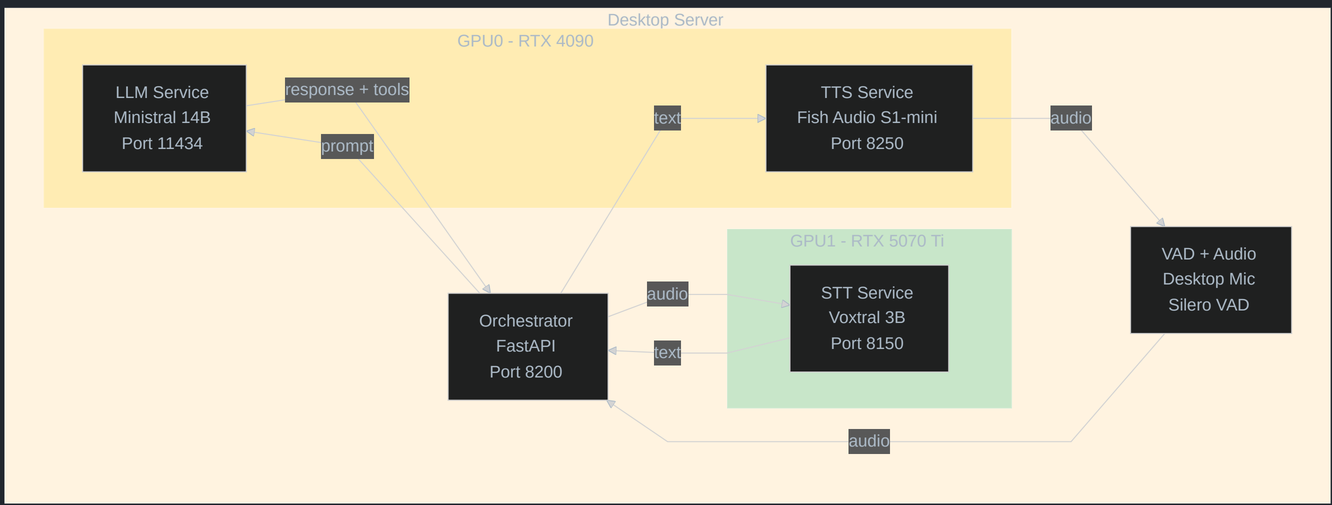
---

6. Container View (C2)

The architecture evolves across phases. Below are diagrams showing each stage.

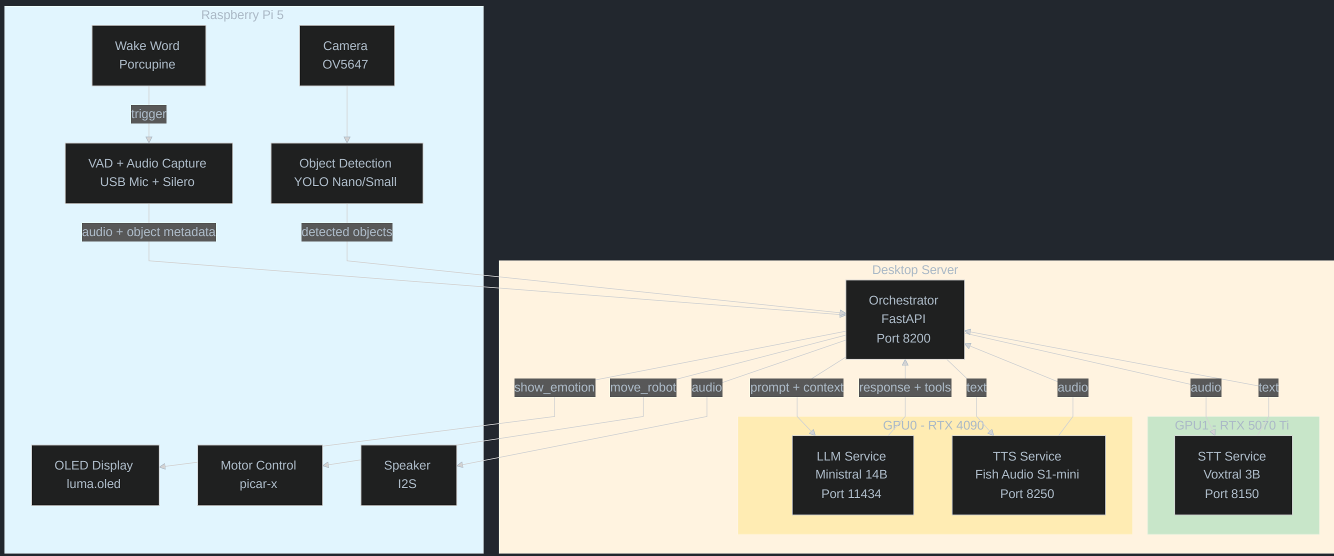
Phase 1: Desktop Complete (Current)

All components on desktop for development and testing without hardware.



Phase 3: Pi Integration

Hardware arrives. VAD, wake word, and object detection move to Pi. Heavy AI stays on desktop.



Dual Vision Approach

Capability	Location	Speed	Detail	Use Case
Object Detection (YOLO)	Pi	Real-time	Objects only	Quick context: "person detected", "obstacle ahead"
LLM Vision (take_photo)	Desktop	5-10s	Full scene	Detailed analysis: "What do you see?"

How they work together:

- YOLO runs continuously on Pi, sends metadata with each request (e.g., `"detected": ["person", "cup"]`)
- LLM receives this context automatically, enabling awareness without explicit photo requests
- Enables use cases like: "drive towards the user" (YOLO detects person, motors respond)
- `take_photo` tool still available for detailed scene description when needed

Service Details

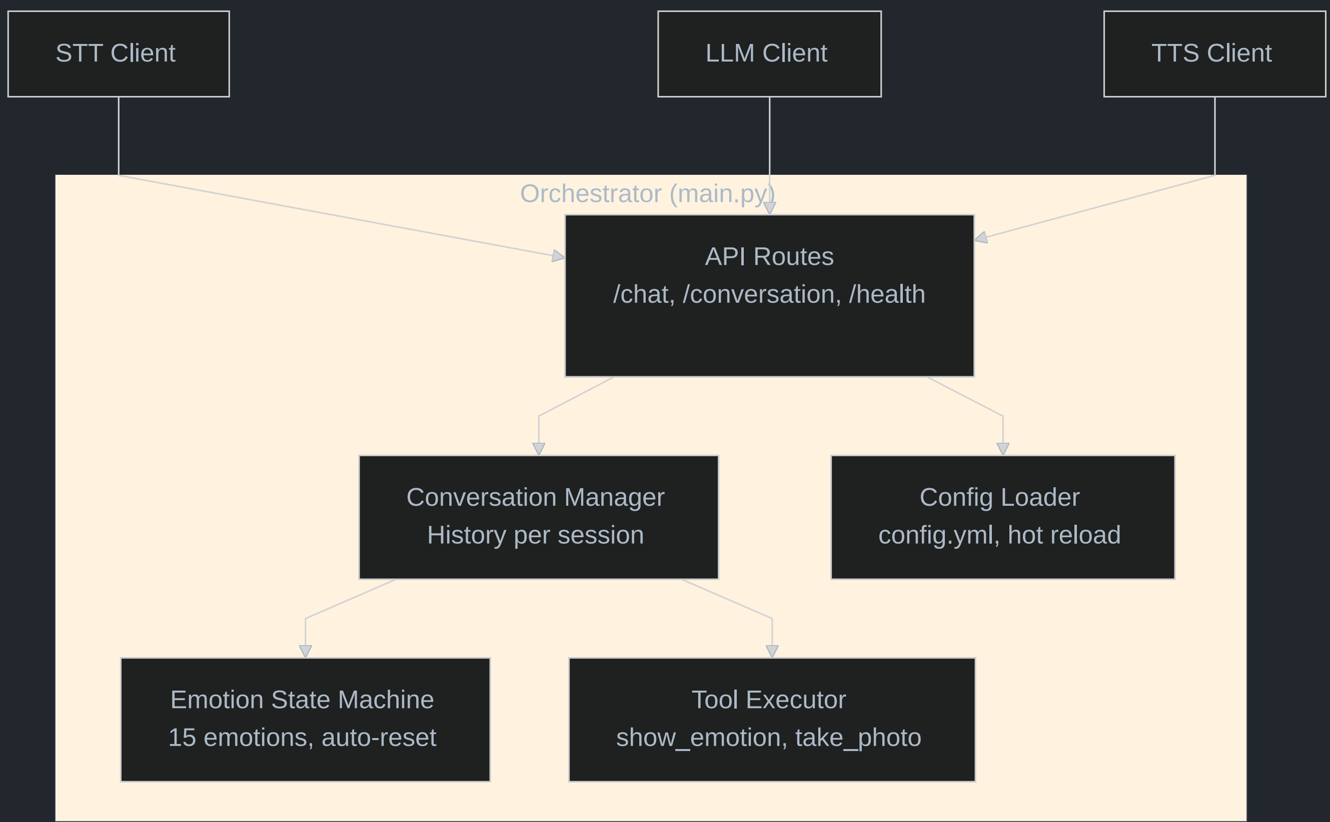
Service	Technology	Latency	Purpose
STT	Voxtral Mini 3B + vLLM	150-750ms	Dutch speech recognition, noise robust

Service	Technology	Latency	Purpose
LLM	Ministral 14B Q8 + Ollama	700-1300ms	Conversation, reasoning, function calling, vision
TTS	Fish Audio S1-mini	~1.2s/sentence	Dutch voice synthesis via reference audio
Orchestrator	FastAPI	-	Routes requests, manages state, executes tools
Object Detection	YOLO Nano/Small	Real-time	Quick object awareness on Pi

---

7. Component View (C3)

Inside the Orchestrator - the brain that coordinates everything.



Key Features

Emotion State Machine:

The robot maintains a **persistent emotional state** - not just showing a smiley per response, but simulating an actual emotional state that evolves over the conversation:

- **15 emotions**: happy, sad, angry, surprised, neutral, curious, confused, excited, thinking, shy, love, tired, bored, proud, worried
- **Stateful**: Emotion persists until explicitly changed (not reset per message)
- **LLM-driven**: The LLM decides when to call `show_emotion(emotion)` based on conversation context
- **Context-aware**: Current emotion is passed to LLM in system prompt, influencing responses
- **Auto-reset**: Returns to neutral after 5 minutes of inactivity
- **OLED display** (Phase 3): Shows current emotion as visual expression on the robot

Example flow:

```
User: "Je bent dom!" → LLM calls show_emotion("sad") → state changes to SAD
User: "Sorry, dat meende ik niet" → LLM calls show_emotion("neutral") → state changes to NEUTRAL
User: "Je bent geweldig!" → LLM calls show_emotion("happy") → state changes to HAPPY
(5 min silence) → auto-reset to NEUTRAL
```

Function Calling:

- `show_emotion(emotion)` - Update robot's emotional display
- `take_photo(question)` - Capture and analyze image
- Supports both native JSON format and text-based fallback parsing

Configuration:

- Single `config.yml` for all settings
- Hot-reload via `/reload-config` endpoint
- No restart needed for parameter changes

> Implementation details: fase1-desktop/orchestrator/main.py

---

8. Technology Stack

Component	Technology	Why This Choice
STT	Voxtral Mini 3B + vLLM	Dutch support (1 of 8 languages), noise robust, audio Q&A
LLM	Ministral 14B Q8 + Ollama	Function calling, vision, low temperature for consistency
TTS	Fish Audio S1-mini	#1 TTS benchmark, 4x faster than alternatives, voice cloning
Orchestrator	FastAPI	Simple, fast, full control, no framework overhead
Wake Word	Picovoice Porcupine	Accurate, low CPU, custom wake words
VAD	Silero VAD	Local, no network, reliable
Containerization	Docker	Reproducible, isolated, easy deployment

> Full decision rationale with alternatives considered: DECISIONS.md

---

9. Current Status

What's Working (Phase 1)

Component	Status	Notes
Speech-to-Text	Working	Dutch, noise-robust, 150-750ms
Language Model	Working	Function calling, vision, 700-1300ms
Text-to-Speech	Working	Dutch voice via reference audio, ~1.2s/sentence
Emotion State Machine	Working	15 emotions, persistent state

Component	Status	Notes
Vision Tool	Working	Photo analysis on demand
VAD Conversation	Working	Hands-free interaction
Central Config	Working	Hot-reload supported

Performance

Metric	Value
STT latency	150-750ms
LLM latency	700-1300ms
TTS latency	~1.2s per sentence
Vision latency	5-10s (dual LLM call)
Total round-trip	**~3 seconds**

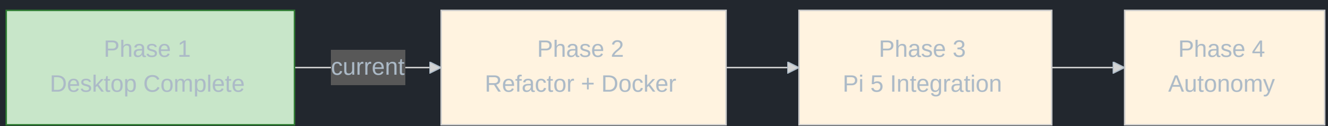
Currently Refining

The following items from fase1-desktop/TODO.md are being addressed:

1. **TTS text normalization** - Acronyms and numbers sometimes sound English (e.g., "API" → "aa-pee-ie")
2. **TTS parameter tuning** - Finding optimal temperature/top_p balance
3. **Optional improvements** - Longer reference audio, pseudo-streaming per sentence

---

10. Roadmap



Phase 1: Desktop Complete (Current)

All AI components working end-to-end on desktop. Hands-free Dutch conversations with emotion awareness and vision capability.

Details: fase1-desktop/README.md

Phase 2: Refactor + Docker

Clean up code (SOLID, KISS, DRY), full Docker Compose stack, API documentation, testing. Goal: `docker compose up` starts everything.

Details: fase2-refactor/PLAN.md

Phase 3: Pi 5 Integration

Connect real hardware - PiCar-X robot, motors, wake word detection, and OLED display for showing the robot's emotional state (driven by the Emotion State Machine). Desktop remains the AI brain, Pi handles physical interaction.

Details: fase3-pi/PLAN.md

Phase 4: Autonomy & Personality

Add "life" to the robot - idle behaviors (blinking, looking around), proactive conversations, long-term memory, obstacle avoidance, consistent personality.

Details: fase4-autonomie/PLAN.md

---

11. Future Possibilities

The modular architecture creates a **foundation** that can grow without rewrites. Here's what the system can support:

Hardware Extensions

Extension	What It Enables
Additional sensors (temp, light, distance)	Environmental awareness, context-aware responses
Multiple cameras	Wider field of view, depth perception

Extension	What It Enables
Different display (LCD, LED matrix)	Richer visual feedback
Additional actuators (arm, gripper)	Physical interaction capabilities
Lidar/depth sensor	Better navigation, 3D awareness

AI Model Flexibility (Local ↔ Cloud)

The architecture is designed (Phase 2 refactor) to easily swap between local and cloud providers. Implementation planned for Phase 4+.

Component	Local (Default)	Cloud Options
STT	Voxtral Mini 3B	Whisper via DeepInfra, AssemblyAI, cloud providers
LLM	Ministral 14B (Ollama)	OpenRouter, OpenAI, Claude, KiloCode, Ollama Cloud
TTS	Fish Audio S1-mini	ElevenLabs (likely primary), other cloud TTS
Vision	LLM vision + YOLO	Cloud vision APIs if needed

Why cloud fallback?

- Higher quality models when needed (e.g., Claude for complex reasoning)
- Reduced local GPU requirements for lighter deployments
- Fallback when local resources are insufficient
- A/B testing different providers

Implementation approach:

- Phase 2: Design service interfaces with SOLID principles (easy to swap implementations)
- Phase 4+: Implement cloud adapters, add configuration switches

Interface Extensions

- **Web dashboard** - Remote monitoring, configuration, conversation history
- **Mobile app** - Control robot from phone, receive notifications
- **REST/WebSocket API** - Third-party integrations, custom clients
- **Home automation** - Home Assistant, possibly MQTT if needed for smart home integration
- **Voice assistant bridge** - Connect to Alexa/Google Home ecosystem

Behavior Extensions

- **Custom tools** - Check weather, control smart home, play music, set timers
- **Learning/memory** - Remember preferences, past conversations, user profiles
- **Multi-robot coordination** - Multiple robots sharing one brain
- **Scheduled tasks** - Routines, reminders, time-based behaviors
- **Emotional learning** - Adapt personality based on interactions

Deployment Flexibility

- **Fully portable** - Battery powered with 4G/WiFi connectivity
- **Multi-robot setup** - Multiple Pi robots sharing one desktop server
- **Edge deployment** - Run smaller models directly on Pi for offline use
- **Hybrid processing** - Split workloads between Pi, desktop, and cloud based on requirements

---

Quick Reference

Resource	Location
Main README	README.md
All decisions with rationale	DECISIONS.md
Phase 1 details	fase1-desktop/README.md
Open improvements	fase1-desktop/TODO.md
Original concept (detailed)	archive/0.concept/

Resource	Location
Central configuration	fase1-desktop/config.yml

Hardware Requirements

Desktop Server (AI Processing):

Component	Minimum	Recommended
GPU	RTX 3080 (10GB)	RTX 4090 (24GB) + RTX 5070 Ti
RAM	32GB	64GB
OS	Linux (Ubuntu 22.04+)	Linux

Robot (Pi 5 Client):

Component	Specification
Raspberry Pi 5	8GB RAM (16GB recommended)
PiCar-X Kit	v2.0
OLED Display	0.96" I2C (128x64)
USB Microphone	Omnidirectional
Camera	OV5647 via CSI

NerdCarX - A learning project for AI-driven robotics