Technical: Project Cartesian

Technical Architecture

Code formatting / Directions

The github is: https://github.com/Trygon117/ProjectCartesian
Call this **CartesianOS** in the code.
My name is Abraham Furlan
My email is abrahamfurlan@gmail.com

Layer 1: The Foundation (Arch Linux)

- **Base OS:** Arch Linux.
- **Reasoning:** Rolling release provides latest CUDA/ROCm drivers and AI libraries (llama.cpp updates) without dependency hell.
- **Init System:** systemd.
- **Compositor:** Hyprland (Wayland). With optional X11 Fallback.
- **Audio Server:** PipeWire (Required for complex channel routing).

Layer 2: The "Cartesian Core"

- **Framework:** Rust Native (Iced).
- **Role:** The Monolith. Replaces separate daemon scripts.
- **Modules:**
    - **lobotomy.rs:** The Process Monitor & Signal Injector.
    - **main.rs:** Multi-threaded Event Loop.

    - **inference.rs:** Bindings for llama-cpp-rs (text) and moondream (vision).
    - **witness.rs:** Manages the Ring Buffer and Vision Pipeline.
    - **hippocampus.rs:** Manages ChromaDB vector storage and decay logic.
- **Safety Protocol:**
    - **No `rm -rf`:** File deletion MUST use `send2trash` (Freedesktop.org Trash Spec compliance).
    - **Strict Typing:** No `unwrap()` allowed in production code.

Layer 3: The "Face" (Rust / Iced)

- **Role:** The Visuals & Interaction Layer.
- **Stack:** Iced (Rust Native GUI)
- **Communication:** Native Rust Channels (Tokio/MPSC) & Iced Subscriptions.
- **UX Design:**
    - **Holo-Badge**: Transparent, click-through overlay.
    - **Audio-First:** Push-to-Talk (Windows Key, Fn Key, etc) interaction to minimize focus stealing during games.

Layer 4: The "Mind" (AI Model Stack)

- **Manager Model:** Gemma 2 9B (Quantized 4-bit).
    - **Role:** Complex reasoning, macro generation.
    - **Hardware:** Requires GPU VRAM.
- **Sidekick Model:** Gemma 2 2B (Quantized 4-bit).

    - **Role:** Quick Wiki lookups, Chat.
    - **Hardware:** System RAM (CPU).
- **Vision Sidekick:** Moondream2 (1.6B).
    - **Role:** "What is this item?" analysis.
    - **Latency:** ~3-5s on CPU.

Feature Implementation Details

The "Witness" System (Adaptive Pipeline)

Path A: The Replay Buffer (Hard-Coded)

- **Mechanism:** GPU Render -> NVENC/VAAPI (Hardware Encode) -> Ring Buffer (/dev/shm).
- **Constraint:** Uses dedicated Encoder Silicon, protecting Game FPS.
- **Format:** H.264/HEVC (Compressed).

Path B: The AI Eye (Adaptive)

- **Primary (Wayland/Modern):** DMA-BUF Zero-Copy. The AI reads raw textures directly. Ultra-low latency.
- **Fallback (X11/Legacy):** XShm or Screen Copy. The AI decodes the latest frame from the Replay Buffer. Higher latency, but broad compatibility.

The "Lobotomy" Logic (Resource Governor)

Scenario A (God Mode): Manager (9B) on GPU.

Condition: No Heavy Game detected + VRAM available.

State: Manager (9B) loaded on GPU. High intelligence.

Scenario B (Sidekick/Gaming Mode):

Prerequisite: > 6 CPU Cores AND > 16GB System RAM.

Trigger: Gaming Process Detected (Steam, Gamescope, Lutris).

Action:

- Evacuate GPU: Unload Manager Model immediately.
- Load Sidekick: Load Gemma 2B into System RAM.

The Leash:

- Apply nice -n 19 (Lowest Priority).
- **E-Core Pinning:** Force affinity to Efficiency Cores (Intel 12th+ Gen) to protect Game FPS.
- **Panic Switch:** If Frame Pacing variance > 20ms, send SIGSTOP to pause AI.

Scenario C (Potato Mode): Hard disable AI.

Condition: < 6 Cores OR < 16GB RAM.

Action: Hard Disable.

- **Zero Idle Strategy:** AI is completely unloaded. Pressing PTT triggers a "Cold Boot" (load from NVMe -> RAM), incurs ~1.5s latency, performs inference, then auto-unloads.

Gaming Compatibility

Proton Integration: The OS ships with steam and proton-ge-custom.

Wine Prefix Management: A Rust wrapper automatically creates and manages prefixes for non-Steam games, applying dxvk and vkd3d patches

Addendum A: The "Hippocampus" Architecture (Rev 2.0)

A.1 Core Philosophy: Fluid vs. Crystallized Intelligence

The central architectural challenge of CartesianOS is the "Hollow AI" problem: how to make a pre-trained model feel native to the user's specific hardware state without retraining the model from scratch (which freezes knowledge in time).

We resolve this by bifurcating the AI's "Mind" into two distinct layers:

1. **Fluid Intelligence (State / Context):**
   - **Definition:** Transient data that changes second-by-second (e.g., "CPU is at 90%," "User is looking at `main.rs`").
   - **Mechanism:** Retrieval-Augmented Generation (RAG).
   - **Implementation:** This data is never trained. It is injected into the System Prompt at inference time via a "State Header."
2. **Crystallized Intelligence (Behavior / Protocol):**
   - **Definition:** Static behavioral rules (e.g., "Always output JSON," "Be concise," "Do not hallucinate commands").
   - **Mechanism:** Low-Rank Adaptation (LoRA) Fine-Tuning.
   - **Implementation:** We train a lightweight adapter (PEFT) on a dataset of "Perfect OS Interactions" to bake the *persona* of a System Administrator into the weights.

A.2 The Data Funnel (Context Management)

To prevent "Context Stuffing" (overflowing the 8k token limit) and latency spikes, raw system data passes through a four-stage funnel before reaching the LLM.

Layer 1: The Firehose (Raw I/O)

- **Sources:** `lobotomy.rs` (Procfs metrics), `witness.rs` (DMA-BUF Screenshots), `stdin` (User Chat).
- **Volume:** ~50MB/minute.
- **Action:** Discard immediately after processing. Never send raw data to the LLM.

Layer 2: The Transcriber (Normalization)

- **Visuals:** The "Vision Sidekick" (Moondream) samples the screen buffer on a 30s interval or Trigger Event (Window Switch). It converts pixels to text descriptions (e.g., "User is editing a Rust file").
- **Metrics:** `lobotomy.rs` parses raw logs into semantic anomaly reports (e.g., "Process `gcc` exceeded 80% CPU").

Layer 3: The Librarian (Vector Storage)

- **Engine:** ChromaDB (Local).
- **Process:** Text descriptions from Layer 2 are embedded and stored as vectors.
- **Retention:** Decay algorithm applied; older, accessed-less vectors are pruned to keep the DB lean (~500MB cap).

Layer 4: The Context Injection (Inference)

When the user queries the Manager (Gemma 9B), the system constructs the prompt dynamically:

1. **System Header (LoRA):** Activates the "SysAdmin" adapter.
2. **State Slot (Live JSON):** Injects the *current* milliseconds-old state (Active Window, Top CPU Proc).
3. **Memory Slot (RAG):** Injects the top-3 relevant vectors from ChromaDB based on the user's query.
4. **Chat Slot:** The actual conversation history.

A.3 Control Protocol: Grammar-Constrained Generation (GBNF)

To prevent the "Chatty AI" risk (where the model discusses an action instead of performing it), we utilize GBNF (Grammar-Based Normalization Form) via `llama.cpp`.

- **Constraint:** The model is forcibly restricted to output valid JSON matching the `ActionSchema`.
- **Safety:** The generation process terminates immediately if the model attempts to generate conversational text outside the JSON structure.

A.4 Roadmap: Phase 1 vs. Phase 2

- **Phase 1 (The Prototype):** Pure RAG implementation. Rely on massive system prompts to guide behavior. High latency, high token cost.
- **Phase 2 (The Master's Strategy):** Harvest interaction data from Phase 1 to train a QLoRA adapter.
    - **Goal:** Offload "Behavior" instructions from the Prompt (Context) to the Weights (LoRA), reducing latency and token usage by ~40%.