# Timmy Time integration architecture: eight deep dives into real deployment

**The Veloren game-agent stack is buildable today, but three critical architectural decisions determine success or failure.** First, wrap the Veloren client crate as a Rust sidecar process with a WebSocket JSON-line protocol — PyO3 direct binding is infeasible. Second, abandon Agno's native Ollama tool calling entirely (it remains broken since November 2024) and use Ollama's structured output mode with Pydantic schemas instead. Third, expose all game controls as a unified MCP server via FastMCP, which Agno consumes natively with **<3% overhead** on a 2-second game loop. The realistic end-to-end latency budget is **4–9 seconds all-local** on M3 Max, or **3–7 seconds** with Groq API hybrid. A split deployment — Hetzner GEX44 GPU VPS at **€184/month** hetzner Hetzner for rendering/streaming plus the Mac M3 Max for inference — delivers the best cost-performance ratio at roughly **$200/month total**.

---

## The Veloren client crate is rich but demands a sidecar, not FFI

The `veloren_client` crate exposes **~120+ public methods** across connection, movement, chat, trade, inventory, combat, and world-state observation. The `Client` struct wraps a tokio async runtime, a specs ECS `State`, a player list, pending trades, and 50+ additional fields. Gitlab github A headless bot can observe position, nearby entities (via ECS queries on `Pos`, `Vel`, `Body`, `Health`, `Stats`), full inventory, all chat messages, trade state, weather, terrain chunks, biome, and time of day. The `solaeus/trade_bot` (archived February 2025 after bots were banned on the *official* server — self-hosted servers have no such restriction) proves the full viability of this approach: GitHub it connects, authenticates, selects a character, manages trades via `/tell` commands, and runs a continuous `tick()` loop. GitHub jeff's git

**PyO3 direct wrapping is not viable.** Veloren requires nightly Rust (pinned in `rust-toolchain`). Veloren Veloren The `Client` type is `Send` but `!Sync`, requiring `Arc<Mutex<Client>>` wrapping that defeats FFI ergonomics. The specs ECS types (`ReadStorage<Pos>`, `World`, etc.) are deeply generic and not serializable across FFI boundaries Veloren — every game type would need a PyO3 wrapper. Initial build times run **5–30 minutes** with a **6.1GB debug** target directory. Veloren There is zero precedent for wrapping a specs-based game client via PyO3.

The **WebSocket JSON-line sidecar** pattern wins decisively. IPC latency benchmarks show TCP loopback at **~11µs/op**, github Unix domain sockets at **~10µs**, and stdin/stdout pipes at **~5µs** — all negligible against the 33ms Veloren tick rate and 500–5000ms LLM inference time. The Rust sidecar runs as an independent process that connects to the Veloren server, calls `client.tick()` at 30Hz, serializes game state as JSON, and accepts commands over WebSocket. The Python AI agent iterates in seconds with no Rust recompilation. Start with stdin/stdout JSON-line for prototyping (hours to implement), then graduate to WebSocket for bidirectional async communication.

```
// Minimum viable sidecar: connect, tick, serialize state, accept commands
let mut client = Client::new(
    ConnectionArgs::Srv { hostname: server.into(), prefer_ipv6: false,
        validate_tls: true, use_quic: false },
    runtime.clone(), &mut mismatched, "bot_user", "bot_pass",
    None, |_| true, &|_| {}, |_| {}, PathBuf::new(), ClientType::Game,
).await?;
client.request_character(char_id, view_distances);
loop {
    let events = client.tick(inputs, clock.dt())?;
    for event in events {
        let json = serde_json::to_string(&serialize_event(event))?;
        ws_send(&json).await;
    }
    if let Some(cmd) = ws_recv().await {
        match cmd { Command::Move{..} => {/**/}, Command::Chat{msg} => {client.se
    }
    clock.tick();
}
```

**Known gotchas**: Client and server must match the same git commit. The crate is not on crates.io — use a git dependency. Some functionality requires the `assets/` directory. The sidecar binary will be **~50MB** release.

---

## Agno's Ollama tool calling is broken — use structured output instead

Agno issue **#2231** ("Agent tool call on Ollama model," February 2025) was the first report, GitHub but the problem is systemic. Issues **#2625**, GitHub **#1419**, GitHub **#1612**, GitHub **#4715**, and even **#4090** (Groq tool calling) GitHub document persistent breakage through March 2026. The root cause is multi-layered: Agno's Ollama model class doesn't robustly parse Ollama's native `tool_calls` format (especially with streaming), Ollama itself

has template/parser mismatches for specific models, and models sometimes emit tool-call JSON as regular text content rather than structured tool calls.

**The most reliable local-only approach bypasses Agno's tool calling entirely** and uses Ollama's native `format` parameter with a Pydantic JSON schema: Ollama

```
from ollama import chat
from pydantic import BaseModel, Field
from typing import Optional, Literal

class GameAction(BaseModel):
    action: Literal["screenshot", "keypress", "click", "wait", "analyze"]
    key: Optional[str] = None
    x: Optional[int] = None
    y: Optional[int] = None
    reasoning: str

response = chat(
    model="qwen3-coder:32b",
    messages=[{"role": "system", "content": "You are a game-playing agent..."},
              {"role": "user", "content": observation}],
    format=GameAction.model_json_schema(),  # Ollama enforces JSON schema complia
    options={"temperature": 0.1},
)
action = GameAction.model_validate_json(response.message.content)
```

For models with tool-calling support in Ollama's March 2026 ecosystem, the community ranking is: **qwen3-coder:32b** (top pick, extremely stable), **glm-4.7-flash** (30B, "more obedient than Qwen"), **gpt-oss:20b** (designed for agents), and **llama3.3:70b** (requires 48GB+ VRAM). clawdbook Clawdbook Hermes 3 8B works but is outperformed by newer models. Critical settings: **temperature 0.0–0.2**, **streaming disabled** for tool calls ( `stream=False` ), and **q4_K_M or q5_K_M** quantization for speed.

If Agno framework features (memory, knowledge, teams) are needed, the `parser_model` pattern is the best workaround — Ollama handles reasoning locally while a lightweight cloud model (GPT-4o-mini via `OpenAIChat` ) handles structured extraction. Agno This adds one cloud API call per decision but keeps reasoning local.

---

## MCP is the right abstraction at 26ms overhead per tool call

The MCP ecosystem has exploded: **97M+ monthly SDK downloads**, Gupta Deepak tens of thousands of servers, and backing from Anthropic, OpenAI, Google DeepMind, Microsoft,

Wikipedia AWS, and Cloudflare. It was donated to the Linux Foundation's Agentic AI Foundation in December 2025. Wikipedia FastMCP powers ~70% of all MCP servers. GitHub

**Agno has first-class, bidirectional MCP integration** — no LangChain or smolagents bridge needed. DeepWiki `MCPTools` supports stdio, Streamable HTTP, and SSE transports. DeepWiki `MultiMCPTools` connects to multiple servers simultaneously. Agno A complete "Timmy Game MCP Server" can be built with FastMCP's `@mcp.tool()` decorator GitHub and Pydantic models in under 200 lines:

```
from mcp.server.fastmcp import FastMCP, Context
mcp = FastMCP("Timmy Game Server", lifespan=game_lifespan)

@mcp.tool()
async def capture_screenshot(ctx: Context, region: str = "full") -> ScreenshotRes
    """Capture the game window."""
    # MSS capture, encode, return

@mcp.tool()
async def send_keypress(key: str, hold_ms: int = 50) -> KeypressResult:
    """Send a keypress to the game."""
    # xdotool or pyautogui

@mcp.tool()
async def get_game_state(ctx: Context) -> GameState:
    """Get current game state from Veloren sidecar."""
    # Read from WebSocket connection to Rust sidecar
```

**Performance is a non-issue for the 2-second game loop.** Python FastMCP averages **26.45ms per tool call** (TM Dev Lab benchmark, February 2026, 3.9M requests). Tmdevlab Total MCP overhead per cycle is **~20–60ms** — under 3% of the 2-second budget. The LLM inference at 500–1500ms is the dominant cost, not the protocol. Use **stdio transport** for near-zero transport latency. Agno GitHub For screenshots, return compressed JPEG (5–10x smaller than PNG) or file paths instead of inline base64.

MCP tool composability works through the agent layer: one MCP server cannot directly call another, but the Agno agent connects to multiple servers via `MultiMCPTools` and orchestrates cross-server workflows. TrueFoundry Game tools and Lightning payment tools (LNbits `pay_invoice`, `create_invoice`, `get_balance`) can coexist in a single FastMCP server sharing a `GameContext` lifespan object, or be split into separate servers with no additional complexity.

---

## Human and AI bot see each other as normal Veloren players

When a headless client connects via the `veloren_client` crate, the server treats it identically to a Voxygen graphical client. Both use the same network protocol and ECS synchronization. Veloren **Other players see the bot's character model, name, and can interact normally** — this is proven by the trade_bot's months of operation on the live official server. GitHub

The co-op architecture leverages three Veloren systems. **Party**: `/group_invite <player>` creates a group where members cannot damage each other, appear as map dots, and share XP. Veloren Wiki The client API exposes `group_leader`, `group_members`, and `InviteResponse` for programmatic acceptance. Veloren Gitlab **Trade**: Fully programmable via the `TradeAction` enum (`AddItem`, `RemoveItem`, `Accept`, `Decline`), with `pending_trade` tracking on the client. Veloren Veloren **Chat**: `/tell <player> <msg>` provides private 1:1 communication ideal for human→bot directives, while `ChatType::Online(uid)` and `ChatType::Offline(uid)` broadcast join/leave events. Veloren

**Mode switching from solo to co-op is straightforward.** The bot monitors `player_list: HashMap<Uid, PlayerInfo>` which updates via `PlayerListUpdate::Add` and `PlayerListUpdate::Remove`. When the human appears, the bot sends a greeting via `/tell`, auto-invites to group, and switches to follow/assist behavior. When the human disappears, the bot continues autonomous play (farming, exploring, grinding) and stores gathered loot for later trade.

For a self-hosted server, grant the bot account admin privileges via `admin add <USER> admin` Veloren Veloren for utility commands (`/goto`, `/tp`, `/give_item`). Set `auth_server_address: None` in `settings.ron` for local accounts, or use the official auth server with registered credentials. Veloren The WASM plugin system is too limited for co-pilot mode today Veloren GitLab (can only handle custom chat commands, not entity control) but could add convenience commands like `/bot status`. Veloren Veloren

---

## Real latency numbers reveal the VLM bottleneck

**No published benchmarks exist for qwen3-vl:8b on M3 Max specifically**, but extrapolation from available data paints a clear picture. Text-only 8B models run at **30–50 tok/s** on M3 Max via Ollama. DEV Community Vision models incur a **40–60% throughput penalty** due to image encoding, putting qwen3-vl:8b at roughly **18–30 tok/s generation** with a 1–3 second vision encoder preprocessing step. A single 720p screenshot analysis (1000–2000 visual tokens input, 200–500 output tokens) takes an estimated **3–6 seconds**

— making the VLM step **50–70% of total pipeline time**.

**Groq API latency is well-documented**: LLaMA 3.3 70B delivers **317 tok/s** output with **0.57–0.77s** time-to-first-token. Artificial Analysis For a 500-token prompt with 200 output tokens, expect **~1.0–1.5 seconds total**. Pricing is **$0.59/M input, $0.79/M output tokens**. Groq +2 At one loop every 5 seconds, that's roughly **$0.40/hour** in API costs.

**Kokoro-82M is remarkably fast**: under **0.3 seconds** for 5–200 words on NVIDIA L4, inferless Inferless and an estimated **0.3–1.0 seconds** for a 2-sentence narration on M3 Max CPU via ONNX FP32. At only 82M parameters (~300MB FP32), memory consumption is negligible. Eachlabs Hugging Face

**MSS screenshot capture runs at 53–66 FPS** (15–19ms per frame) on Linux without subprocess overhead. Under Xvfb, performance is similar or better since the framebuffer lives in RAM. For an agent capturing one frame every 2–5 seconds, MSS is massively overprovisioned.

| Pipeline stage | M3 Max (all local) | Groq hybrid |
|---|---|---|
| Screenshot capture | 15–20ms | 15–20ms |
| Image preprocessing | 50–100ms | 100–200ms (network) |
| VLM analysis | **3,000–6,000ms** | 1,000–3,000ms (cloud) |
| Decision LLM | 500–2,000ms | **1,000–1,500ms** (Groq) |
| Action execution | 50–100ms | 50–100ms |
| TTS narration (Kokoro) | 300–800ms | 500–2,000ms (CPU-only VPS) |
| **Total** | **~4–9 seconds** | **~3–7 seconds** |

**Memory for dual-model Ollama on M3 Max**: qwen3-vl:8b (~6–8GB) plus qwen2.5:7b (~5–6GB) totals **~11–14GB** with KV cache overhead, leaving ~70GB free on the 96GB system. Set `OLLAMA_MAX_LOADED_MODELS=3` to keep all models hot. Infralovers

**A single 8-core 32GB DigitalOcean droplet without GPU cannot run this stack.** Voxygen requires Vulkan for rendering. CPU-only Ollama delivers 2–8 tok/s for 7–8B models Aitooldiscovery — a single VLM inference would take 30–90 seconds. The stack must be split across machines.

---

## Content moderation requires a three-layer defense

**Twitch has no specific AI content disclosure requirement** as of March 2026 — standard community guidelines apply, Alibaba and the channel operator is responsible for all AI speech. **YouTube mandates disclosure** via the "Altered or synthetic content" toggle in YouTube Studio since early 2025, Influencer Marketing Hub with penalties ranging from platform-applied labels to Partner Program removal. Subscribr MiniMatters For an AI game narrator that doesn't impersonate a real person over clearly fictional game footage, YouTube enforcement is unlikely to be aggressive, but disclosure is best practice.

**The Neuro-sama incident is the definitive cautionary tale.** On December 28, 2022, a viewer asked Neuro-sama about the Holocaust; she responded "I'm not sure if I believe it" Kotaku alongside misogynistic and homophobic comments. Vedal987's channel received a 2-week Twitch ban for "hateful conduct." Wikipedia The resolution was architectural: a dedicated output filter ("Nere the Filter") intercepts all LLM output before TTS, replacing flagged content with the visible word "filtered" — which fans then personified as a character. This transparency-as-entertainment approach proved that **responsible moderation and good content can coexist** (Neuro-sama became the 7th most-subscribed Twitch channel ever post-ban).

For real-time output filtering, the model options rank as follows:

- **Llama Guard 3 1B (INT4 quantized)**: Best for speed. Under **30ms per sentence** on consumer GPU. Sufficient accuracy for catching clear violations. Run alongside TTS on the same GPU.
- **ShieldGemma 2B**: Best for accuracy. **+10.8% AU-PRC over Llama Guard** on public benchmarks. arXiv MarkTechPost Outputs probability scores enabling threshold tuning — critical for distinguishing "narrating in-game slavery" from "promoting real slavery."
- **NeMo Guardrails**: Best for orchestration. A framework (not a model) that combines multiple guardrails in parallel with custom Colang rules. Adds **~0.5 seconds** with 5 GPU-accelerated rails. NVIDIA Developer
- **LEG (Lightweight Explainable Guardrail)**: Emerging option at **<8ms** — 7x faster than alternatives with competitive accuracy. OpenReview

The architecture runs moderation and TTS preprocessing **in parallel**: while Llama Guard checks the sentence, Kokoro tokenizes and phonemizes. On pass, TTS synthesis fires immediately. On fail, a contextual fallback narration (pre-generated per game scene type) replaces the flagged content. For Morrowind's mature themes, a game-context-aware system prompt is the first line of defense — instructing the narrator to describe slavery as "a game mechanic and historical worldbuilding" and drugs as "in-game consumable items," never editorializing on real-world parallels. Per-game moderation threshold profiles whitelist expected vocabulary ("Skooma," "slave," "Morag Tong") to prevent over-filtering.

## Qwen3-8B outperforms Hermes 3 for decision-making but shares its protocol

Hermes 3 8B is available on Ollama as `hermes3:8b` with native tool calling via the ChatML `<tool_call>` / `<tool_response>` XML tag protocol. It was purpose-built for function calling and roleplaying, with special reasoning tokens ( `<SCRATCHPAD>` , `<PLAN>` , `<EXECUTION>` , `<REFLECTION>` ). However, **Qwen3-8B (April 2026) uses the identical Hermes Function Calling standard** — vLLM serves it with `--tool-call-parser hermes` Qwen — and outperforms Qwen2.5-14B on 15 benchmarks while matching Hermes 3's parameter count. Apidog Agno's own documentation states "qwen models perform specifically well with tool use." agno Agno

The recommended multi-model architecture assigns three distinct roles:

- **Perception**: Qwen3-VL 8B via Ollama — screenshot → structured game state JSON
- **Decision**: Qwen3-8B via Ollama — game state + history → action selection (structured output, not tool calling)
- **Narration**: Hermes 3 8B generates in-character narration text (leveraging its unmatched steerability and roleplaying capability), then Kokoro voices it

```
# Custom pipeline — bypasses Agno Teams overhead for tight 2s loop
class GameAgentPipeline:
    def __init__(self):
        self.perception_model = "qwen3-vl:8b"
        self.decision_model = "qwen3:8b"  # or qwen3-coder:32b if VRAM allows
        self.narration_model = "hermes3:8b"

    async def tick(self, screenshot_b64: str, history: list) -> dict:
        # Perception: ~500-800ms with num_predict:256
        state = ollama.chat(model=self.perception_model,
            messages=[{"role":"user","content":"Describe game state.","images":[s
            options={"num_predict": 256})

        # Decision: ~400-600ms with structured output
        decision = ollama.chat(model=self.decision_model,
            messages=[{"role":"system","content":"Tactical game AI."},
                      {"role":"user","content":f"State: {state['message']['conten
            format=GameAction.model_json_schema(),
            options={"num_predict": 128, "temperature": 0.1})

        # Narration: async, non-blocking
```

```
        asyncio.create_task(self.narrate_and_speak(decision))
        return GameAction.model_validate_json(decision["message"]["content"])
```

Game skills can be written as **agentskills.io SKILL.md files** — the open standard adopted by 26+ platforms including Claude, Gemini, and GitHub Copilot. Strapi Each skill is a Markdown file with YAML frontmatter defining when to activate, Visual Studio Code decision rules, available actions, and examples. Hermes Agent natively consumes these from `~/.hermes/skills/`. Nousresearch Skills for navigation, combat, trading, and exploration slot directly into the decision model's context window via progressive disclosure.

---

## Split architecture on Hetzner GEX44 plus Mac M3 Max wins on cost

The optimal deployment runs the rendering/streaming/payments stack on a **Hetzner GEX44 dedicated server** (RTX 4000 SFF Ada 20GB, i5-13500 14-core, 64GB RAM, Hetzner **€184/month ≈ $200/month**) hetzner Hetzner and all AI inference on the Mac M3 Max over Tailscale. This is the only sub-$400/month configuration that provides both Vulkan rendering (required for voxygen) and adequate LLM performance.

**Critical deployment detail: use FFmpeg with x11grab instead of OBS.** OBS does not officially support headless/containerized operation. OBS Forums OBS Forums FFmpeg captures the headless Xorg display Tuckerosman and encodes via NVENC in a single command:

```
ffmpeg -f x11grab -video_size 1920x1080 -framerate 30 -i :0 \
  -c:v h264_nvenc -preset p4 -tune ll -b:v 4M \
  -f flv rtmp://localhost:1935/live/veloren
```

**Second critical detail: use headless Xorg, not Xvfb.** Xvfb is software-only and cannot access the GPU. Grokipedia Headless Xorg with `nvidia-xconfig --use-display-device=none --virtual=1920x1080` enables Vulkan rendering without a physical monitor. Dugas

The Veloren server-cli runs in Docker Veloren Veloren (official image at `registry.gitlab.com/veloren/veloren:weekly`, ports 14004/14005/14006). Veloren Wiki MediaMTX runs in Docker (`bluenviron/mediamtx`, ports 1935/8554/8888/8889). GitHub LNbits runs in Docker (`lnbits/lnbits`, port 5000). MCP Market LNbits docs Voxygen and FFmpeg run on bare metal for reliable Vulkan access. Tailscale connects VPS to Mac with **~1–3ms overhead** over raw WireGuard; Tailscale practical RTT of **20–80ms** depending on geographic distance is negligible against multi-second LLM inference.

| Component | VPS (Hetzner GEX44) | Mac M3 Max |
|---|---|---|
| Veloren server-cli | Docker container | — |
| Voxygen (headless) | Bare metal, headless Xorg | — |
| FFmpeg streaming | Bare metal, NVENC | — |
| MediaMTX | Docker container | — |
| LNbits | Docker container | — |
| Ollama (qwen3-vl + qwen3 + hermes3) | — | Native, Metal acceleration |
| Kokoro TTS | — | ONNX runtime |
| Python AI agent | — (or here in production) | Primary development |
| Tailscale | Docker sidecar | Native app |

GPU VPS alternatives: DigitalOcean GPU droplets DigitalOcean (RTX 4000 Ada, **~$0.76/GPU/hr ≈ $547/month**) support Vulkan but cost nearly 3x more. RunPod and Vast.ai are container-based with **unreliable Vulkan support** — not recommended for rendering workloads. Lyceum Technology Lambda Labs focuses on CUDA training with no Vulkan support.

---

## Conclusion: three decisions, one architecture

The three make-or-break decisions are now clear. **For the Veloren bridge**: build a Rust WebSocket sidecar that serializes game state as JSON — start with stdin/stdout for prototyping, graduate to WebSocket within a week. The `veloren_client` crate is rich enough to support everything the agent needs; the trade_bot proves the pattern works. **For tool calling**: use Ollama's `format` parameter with Pydantic JSON schemas, Ollama bypassing Agno's broken tool-calling layer entirely. Qwen3-coder:32b Clawdbook (if VRAM allows) clawdbook or Qwen3-8B gives the best reliability. Keep Agno for agent definition, memory, and MCP consumption Elightwalk — just don't rely on its Ollama tool dispatch. **For the MCP gap**: build a unified FastMCP server exposing `capture_screenshot`, `send_keypress`, `get_game_state`, `save_game`, `get_health`, `narrate`, and Lightning payment tools. Agno's native `MCPTools` consumes it over stdio DeepWiki at 26ms per call.

The latency budget is honest: **4–9 seconds per full perception-decision-action-narration cycle** running all-local on M3 Max, bottlenecked by VLM inference. This is adequate for exploration and strategic gameplay but too slow for real-time combat — which should trigger a pre-scripted reactive combat routine that bypasses the full VLM pipeline. The content moderation pipeline (Llama Guard 1B at <30ms + game-context system prompts + per-game vocabulary whitelists) adds negligible latency while preventing a Neuro-sama-style incident. Future AI And the split Hetzner + Mac architecture delivers everything needed at $200/month, with the flexibility to move AI inference to the VPS (at higher cost) when the Mac needs to sleep.