Project Morrowind: technical feasibility and implementation guide

Timmy Time is architecturally viable but requires navigating one dormant dependency, one high legal risk, and several integration gaps that don't yet have off-the-shelf solutions. The core concept — an autonomous AI agent playing Morrowind over TES3MP, streamed live, with Lightning-paid audience interaction and eventual community agent expansion — maps onto proven patterns from Voyager, Cradle, Twitch Plays Pokémon, and the emerging Nostr DVM ecosystem. The critical bottleneck is TES3MP itself: effectively dormant since May 2022, stuck on OpenMW 0.47 (three major versions behind mainline), and missing the fine-grained movement/combat APIs that modern OpenMW Lua scripting now provides. Every other component — streaming, payments, TTS, agent orchestration, the web platform — has mature or near-mature tooling available today.

This guide is organized into three phases: what exists today, what must be built, and what could kill the project.

Phase 0: the TES3MP reality check

TES3MP 0.8.1 shipped **May 1, 2022** github and has received roughly 20 commits since. The GitHub repository shows ~80 open issues with 2 open PRs. github Core developers David Cernat and Stanislav Zhukov appear to have moved on. Community activity continues — Steam group comments through late 2025, active Lua script projects — but no new releases are planned. The sole active fork is **EncoreMP** (Tower-Numid/EncoreMP), a mechanical overhaul targeting Tamriel Rebuilt compatibility.

Apple Silicon is a mixed bag. Pre-built universal binaries exist (samkaufman/tes3mp-mac-pkg) and the client runs well on Apple Silicon. The server, however, has known issues on ARM GitHub — likely related to CrabNet (the RakNet fork) under Rosetta 2. Running the server in a Linux Docker container (TES3MP/server-container) on the same M3 Max is the pragmatic fix.

The **OpenMW version gap** is the deepest technical debt. TES3MP is built on OpenMW 0.47; github current OpenMW is **0.50.0**. OpenMW OpenMW 0.49+ introduced a comprehensive client-side Lua scripting API OpenMW with `Controls.overrideMovementControls()` and `Controls.overrideCombatControls()` — exactly the APIs needed for an autonomous agent. Readthedocs TES3MP's Lua scripting is server-side only and predates this entirely.

Rebasing TES3MP onto modern OpenMW would be a significant engineering effort that nobody is currently undertaking.

The server-side Lua API is nonetheless powerful for state manipulation. It exposes **21 API categories** — Actor, Book, Cell, CharClass, Chat, Dialogue, Faction, GUI, Item, Mechanics, Object, Position, Quest, RecordsDynamic, Server, Setting, Shapeshift, Spell, Stats Tes3mp — with full read/write access to player position (`tes3mp.GetPosX/Y/Z` , `tes3mp.SetPos`), Tes3mp inventory, quest journal state, dialogue topics, faction standing, and character creation. Console commands can be executed on players via `logicHandler.RunConsoleCommandOnPlayer()` . Steam Community Event hooks cover connect/disconnect, death, cell change, equipment changes, journal updates, object activation, and more. GitHub

What server-side Lua cannot do is the problem. There is no walk/turn/strafe API — only teleportation. NPC activation (talking, trading) requires client-side input. Combat actions (swing, cast, block) require client-side input. The agent cannot navigate the world step-by-step through server Lua alone.

Capability	Server Lua	Needs client input
Teleport to coordinates	✅	—
Add/remove inventory	✅	—
Advance quest journal	✅	—
Set stats, class, race	✅	—
Execute console commands	✅	—
Walk/run/swim	❌	✅
Talk to NPC	❌	✅
Combat (attack, cast, block)	❌	✅
Open doors/containers	❌	✅
Read screen state	❌	✅

Pathfinding data exists but is not exposed via Lua. OpenMW uses Recastnavigation for dynamic navmesh generation, Rpgcodex cached in SQLite via the standalone `openmw-navmeshtool` binary. GitLab This cache can be read externally by a Python process, and recast mesh can be dumped to .obj files via `enable write recast mesh to file = true` .

Original Morrowind path grids are accessible through ESM/ESP file parsing. Neither is API-accessible from TES3MP scripts.

Practical player limit is approximately 20–30 concurrent connections before packet-related instability. For the single-agent-plus-spectators model, this is not a concern.

Phase 1: the agent architecture

The strongest prior art comes from four projects that collectively define the design space.

Voyager (Minecraft, 6.7K GitHub stars) proved that LLM-generated code calling a structured game API outperforms reinforcement learning by 15× on key milestones. Its architecture — automatic curriculum, reusable skill library, iterative prompt refinement — works because Mineflayer provides a rich structured API. **GITM** (Ghost in the Minecraft, 634 stars) added hierarchical goal decomposition: LLM Decomposer → LLM Planner → structured actions → operations, achieving 67.5% success on ObtainDiamond. **Cradle** (BAAI, accepted ICML 2025) demonstrated the first AAA game agent using only screenshots and keyboard/mouse simulation on Red Dead Redemption 2. **The Immersive Morrowind LLM AI project** (drzdo/immersive_morrowind_llm_ai, 78 stars, archived March 2026) proved the MWSE Lua → TCP → Python → LLM pipeline is viable specifically for Morrowind, though it was limited to NPC dialogue and used vanilla MWSE rather than OpenMW.

The recommended architecture for Timmy is a **hybrid control model**:

Perception should be primarily API-driven, not vision-driven. TES3MP server Lua provides structured game state (position, inventory, journal, nearby actors, quest progress). This data is converted to text descriptions for the LLM. Vision (via screen capture + a VLM) serves as a supplementary channel for spatial awareness, verifying game state the API can't access, and reading in-game text. The research consistently shows that structured API perception produces dramatically better agent performance than screen-only approaches. VideoGameBench found a persistent "knowing-doing gap" where VLMs articulate correct goals but fail to execute correct action sequences. Substack

Reasoning should use hierarchical decomposition with knowledge augmentation. The main quest line becomes: "Complete the main quest" → "Become Hortator of House Hlaalu" → "Talk to Crassius Curio" → "Travel to Vivec, Hlaalu Canton" → specific navigation and dialogue actions. UESP wiki data, pre-processed into a RAG-queryable knowledge base, provides quest walkthroughs, NPC locations, item databases, and map data. A DAG-based reasoning structure (from SPRING) with ReAct-style interleaved thought/action traces handles step-level decisions.

Memory should follow the three-tier pattern validated by Smallville's generative agents and refined by CHIM (the Skyrim mod that runs at ~$5/month on a 70B model via OpenRouter):

- **Working memory**: Current context window — active quest, nearby NPCs, inventory, last 5–10 actions, health/stats
- **Episodic memory**: Chronological log of key events with embeddings for retrieval, scored by recency + importance + relevance, periodically summarized Substack
- **Semantic memory**: Static knowledge from UESP wiki, game data files, quest walkthroughs — retrieved via RAG

Action requires the hybrid approach. Define a **constrained action vocabulary** of ~20 structured actions: `move_to(location)`, `talk_to(npc, topic)`, `pick_up(item)`, `equip(item)`, `cast_spell(spell, target)`, `attack(target)`, `rest(hours)`, `use_item(item)`, `open(object)`, `buy(item, merchant)`, `sell(item, merchant)`, `read(book)`, `journal_check()`, `wait(seconds)`. Actions that can be executed via server Lua (teleport, inventory, quest advancement, console commands) are handled directly. Actions requiring client interaction (walking, NPC activation, combat) are executed via **client-side input injection** using macOS CGEvent API or pyautogui.

Model tiering keeps costs manageable and latency acceptable:

- **Routine actions** (navigation, basic interactions): Local 7–8B model via Ollama (~0.3–1s latency, free)
- **Complex reasoning** (quest planning, dialogue strategy, combat tactics): Hermes 3/4 70B via Ollama on M3 Max 128GB (~8–12 tok/s at Q4_K_M, fits entirely in unified memory)
- **Difficult decisions or recovery from failures**: Claude or GPT-4o API (1–3s, ~$5–15/hr of heavy play)
- **Combat**: Pause game during inference (Morrowind's real-time combat is slow enough; CombatVLA validated this approach), or use pre-computed behavior trees (OpenMW's MercyCAO mod demonstrates Lua behavior trees for combat AI) GitHub

The REST API bridge connects everything. TES3MP's Lua runtime can load `luasocket`, enabling TCP communication with an external Python process. The architecture is: **FastAPI (Python) ↔ TCP socket ↔ TES3MP Lua server script ↔ Game state**. Community projects have built Discord bots, IRC bridges, and a Rust-based REST API using this pattern. A minimal Rust REST API for TES3MP already exists as prior art. GitHub

---

Phase 1.5: the input bridge gap

The most significant "must be built" component is the **client-side input bridge**. No existing project automates OpenMW player movement from an external process.

On macOS, the options are:

1. **CGEvent API** (recommended): Programmatic keyboard/mouse injection at the OS level. Python bindings via `Quartz` framework. Can target the OpenMW window specifically. Latency is sub-millisecond.
2. **pyautogui**: Cross-platform, simpler API, but requires the window to be focused and visible. Less precise timing.
3. **Modified OpenMW client**: Fork OpenMW to add a local socket that accepts movement/action commands, bypassing SDL2 input entirely. Most robust but highest development cost.
4. **AppleScript/osascript**: Basic keystroke injection. Unreliable for real-time control.

The recommended approach is CGEvent for movement and basic interaction, with server-side Lua handling everything it can (teleportation for long-distance travel, inventory management, quest advancement, stat changes). This hybrid minimizes the fragile surface area of input simulation.

Phase 2: live streaming pipeline

Frame capture on macOS uses ScreenCaptureKit (WWDC 2022, macOS 12.3+), which delivers **60fps at native resolution** with GPU-backed capture buffers, 50% less CPU and 15% less RAM than the legacy CGWindowListCreateImage approach. OBS Studio on macOS already uses ScreenCaptureKit via the "macOS Screen Capture" source with per-window targeting. OBS

Encoding leverages Apple Silicon's dedicated VideoToolbox hardware encoders. The M3 Max supports H.264 and HEVC encoding at 4K 120fps with ~20% CPU usage versus 100% for software encoding. Martin Riedl FFmpeg command for low-latency capture:

```
ffmpeg -f avfoundation -framerate 30 -capture_cursor 1 -i "<screen_index>:none" \
  -c:v h264_videotoolbox -realtime 1 -b:v 5000k \
  -profile:v high -pix_fmt yuv420p -g 30 \
  -f flv rtmp://localhost:1935/gamestream
```

Distribution uses a dual-protocol architecture. **MediaMTX** (v1.11.2, single Go binary, zero

dependencies) auto-converts between RTMP, WebRTC, HLS, RTSP, and SRT simultaneously from a single ingest. GitHub Push RTMP from OBS, get WebRTC (<300ms glass-to-glass) and LL-HLS (2–5s for CDN scale) automatically. For higher scale, **LiveKit** (Apache 2.0, Go/Pion-based WebRTC SFU) GitHub supports 100,000 simultaneous users per session, Software Suggest with Ingress (RTMP from OBS), Egress (HLS to S3/CDN, RTMP to Twitch), and a Python Agents SDK.

Latency budget for the full pipeline:

Stage	Latency
ScreenCaptureKit capture	~8–16ms
VideoToolbox encoding	~16–33ms
Local RTMP transport	<5ms
MediaMTX conversion	<10ms
WebRTC delivery	**50–200ms**
LL-HLS delivery	**2,000–5,000ms**

Overlays (chat, Lightning events, agent reasoning) are best implemented as OBS Browser Sources — embedded Chromium instances rendering HTML/CSS/JS, composited at the OBS level before encoding. A small React/Svelte web app connects to the agent's state via WebSocket and displays current action, reasoning chain, and event feed. Neuro-sama (the most-subscribed Twitch AI streamer, 162K+ active subs as of January 2026, ~$400K/month) uses exactly this pattern. SuperAGI

Phase 3: the Nostr DVM layer

NIP-90 (Data Vending Machines) is **merged, specified, and functional** GitHub with a small but active ecosystem. The specification defines kind range 5000–7000: job requests (5000–5999), job results (request kind + 1000), and job feedback/status (kind 7000). GitHub Nostr Payment flows through Lightning invoices embedded in event tags. Nostr GitHub The kind registry at data-vending-machines.org includes text generation (5050), image generation (5100), transcription (5250), content discovery (5300), and translation (5005).

NIP-89 (Application Handlers) provides the **discovery layer**. DVMs publish kind:31990 events advertising supported job kinds; users publish kind:31989 recommendations. Nostr

Clients query their social graph for DVM recommendations, Nostr creating a web-of-trust discovery mechanism.

Python tooling for DVM development:

- **nostr-sdk** (v0.44.2, PyPI): Rust-nostr Python bindings PyPI via UniFFI. The primary library used by both major DVM frameworks. Supports arbitrary event kinds, relay management, async patterns. Marked ALPHA with potential breaking changes. PyPI
- **ezdvm** (PyPI): Minimal DVM framework — pick a kind, implement `do_work()`, run. PyPI Under 20 lines for a working DVM.
- **nostrdvm** (v1.1.3, PyPI): Full-featured framework with NIP-89 auto-registration, LNbits wallet integration, bot interface. GitHub OpenSats-funded. OpenSats
- **DVMDash** (dvmdash.live): Monitoring/directory tool GitHub GitHub and playground for testing.
- **DVMCP/ContextVM**: Bridges MCP servers to the Nostr DVM ecosystem, enabling AI tools to discover and use MCP servers via Nostr. GitHub GitHub

Kind 2020 review/rating system remains an **unmerged proposal** (GitHub issue #1515 on the NIPs repo). No implementations exist. The platform would need to implement its own reputation layer or wait for community standardization.

Honest ecosystem assessment: NIP-90 is the right architectural foundation but is **early-stage/alpha**. Libraries are unstable, GitHub few mainstream Nostr clients deeply integrate DVMs, relay behavior varies, and the developer community numbers in dozens, not hundreds. Being an early builder confers influence over emerging standards but requires tolerance for rough edges.

Phase 4: Lightning payments infrastructure

The **L402 protocol** (formalized as bLIP-0026) combines HTTP 402 status codes with Lightning invoices and macaroon-based authentication tokens. GitHub The flow is: client requests → server responds 402 with invoice → client pays → client resubmits with `Authorization: L402 <macaroon>:<preimage>` → server verifies GitHub via single SHA-256 hash check. Lightning Labs Blockonomi **Aperture** (Go, Lightning Labs) is the production L402 reverse proxy GitHub used by Lightning Loop and Pool. GitHub lightning

Lightning Labs released **lightning-agent-tools** on **February 11, 2026** — seven composable skills: LND node management, remote key isolation, scoped macaroons, lnget (L402-aware

HTTP client), Aperture hosting, MCP server (18 read-only tools for node state), and a commerce meta-skill for end-to-end buyer/seller workflows. Lightning Labs KuCoin The tools are CLI/shell-oriented and work with any agent framework that can execute shell commands. Lightning Labs lightning **No dedicated Python SDK exists** — Python integration uses LND's gRPC API via generated protobuf stubs or the REST API via `requests`.

Nostr Wallet Connect (NIP-47) enables programmatic wallet access through E2E-encrypted messages over Nostr relays. GitHub Nostr Supported commands include `pay_invoice`, `make_invoice`, `get_balance`, and `list_transactions`. GitHub Nextra Alby Hub is the primary NWC implementation. GitHub Bringin **No mature standalone Python NWC client library exists** — implementation requires `nostr-sdk` plus custom NIP-47 message handling.

@getalby/lightning-tools (npm ~v5.0.0) provides `fetchWithL402()` for consuming L402-protected resources from JavaScript. GitHub Alby also ships **@getalby/agent-toolkit** (v0.1.0) for LangChain/Vercel AI SDK integration, npm and an **MCP server** (`@getalby/lightning-tools-mcp-server` v1.0.0) for AI agent integration. npm

For the Lightning node: **LND on macOS Apple Silicon** compiles natively (Go has excellent ARM64 support since 1.16) and is the recommended choice. The entire Lightning Labs ecosystem — Aperture, lnget, agent-tools — is built around LND. Use **Neutrino mode** for development (no full Bitcoin node required). For production, **Voltage** ($27–38/month) Voltage or **Alby Cloud** (21,000 sats/month) provides managed hosting with full API access.

Phase 5: voice pipeline

Kokoro-82M is the recommended primary TTS engine. Despite only **82 million parameters**, BentoML it ranks #1 in TTS Spaces Arena Elo rankings, beating models 5–15× its size. bentoml On Apple Silicon via **mlx-audio** (`pip install mlx-audio`), it achieves **178ms for 4 words** and **604ms for 36 words** on M4 Max Runanywhere — comfortably real-time for game narration. Apache 2.0 license. Hugging Face Output is 24kHz high-quality audio. No built-in voice cloning. Unreal Speech

Piper TTS v1.4.1 (February 2026, now under OHF-Voice/piper1-gpl) PyPI GitHub serves as a fallback for ultra-low-latency short utterances. Native ARM64 wheel ships on PyPI. PyPI Real-time factor ~0.2 on CPU (5× faster than real-time). arXiv Sub-50ms for short sentences on M3 Max. Supports custom voice training from ~30 minutes of clean audio via fine-tuning from checkpoints, ssamjh though training requires an NVIDIA GPU. GitHub +2 License changed to **GPLv3** as of v1.3.0. GitHub

Chatterbox-Turbo (Resemble AI, 350M params, MIT license) is the best option if expressive commentary is a priority. BentoML Sub-200ms inference, voice cloning from short clips, and emotion control with paralinguistic tags (`[laugh]` , `[cough]`) BentoML bentoml — ideal for reactive game commentary.

The **real-time pipeline** uses **RealtimeTTS** (`pip install realtimetts[all]`), which supports Piper, Kokoro, and Coqui engines with built-in streaming, sentence segmentation, and callbacks. PyPI GitHub Audio mixing with the game stream uses `sounddevice` , ducking game audio by 6–12dB during narration.

Phase 6: the /tower web platform

No integrated, production-grade platform currently combines Lightning payments with an AI agent marketplace. The closest competitors are CASCDR (workflow builder + Lightning APIs, small team, early stage), Stacker News Routstr (decentralized LLM routing via Nostr + Cashu), Substack and Lightning Labs' agent-tools (infrastructure, not a consumer platform). The opportunity for a polished, consumer-facing "App Store for sovereign AI agents" is wide open.

The **web stack** is mature. Next.js 15 with App Router and Server Components is battle-tested; React 19.2 (October 2025) adds `<Activity>` for pre-rendering and `useEffectEvent` for stable callbacks; React Tailwind CSS 4 (January 2025, Rust-based Oxide engine) delivers **5× faster full builds and 100× faster incremental builds** with CSS-first configuration; DesignRevision shadcn/ui provides 40+ components fully compatible with React 19 and Tailwind 4. Use **pnpm** to avoid npm peer dependency conflicts with React 19. DEV Community

For 3D elements, **Three.js WebGPURenderer** (default since r171, September 2025) provides automatic WebGL 2 fallback. Utsubo WebGPU's async architecture prevents context loss issues entirely. CopyProgramming Browser support: Chrome 113+, Firefox 141+ (Windows) / 145+ (macOS ARM), Safari 26+. Always implement context loss handlers on the WebGL fallback path Medium and dispose GPU resources on component unmount.

Serverless GPU infrastructure for AI inference: **RunPod Serverless** is recommended for production Koyeb (best cold-start performance at 48% under 200ms, zero egress fees, transparent per-second billing). Runpod RTX 4090 at **$0.32/hr** on Community Cloud Neurocanvas or **$3.96/hr** Flex for serverless. **Modal** ($30/month free credits, Python-native decorators, sub-second container spin-up) is ideal for development. Modal Vast.ai offers 20–50% cheaper raw hourly rates but with variable host quality. Runpod

---

Phase 7: agent orchestration and local inference

Agno (formerly Phidata, `pip install agno`) is the recommended orchestration framework. BestAIAgents Benchmarked at **529× faster instantiation** than LangGraph and **24× less memory** Medium (October 2025, Apple M4). GitHub Native MCP integration via `MCPTools` class. AgentOS provides a production-ready FastAPI backend serving agents as APIs. GitHub Model-agnostic Agno — works with OpenAI, Anthropic, Groq, and any OpenAI-compatible endpoint including Ollama. MGX

Model Context Protocol (MCP) has achieved massive adoption since its November 2024 launch: **97 million monthly SDK downloads**, 5,800+ servers, 300+ clients. Gupta Deepak Donated to the Linux Foundation's Agentic AI Foundation in December 2025, co-founded by Anthropic, OpenAI, and Block. Wikipedia **FastMCP** (v3.1.1, ~1M daily downloads, powers ~70% of MCP servers) can generate an MCP server from an existing FastAPI app in one line: `mcp = FastMCP.from_fastapi(app=app)`. PyPI

Hermes 3 70B on the M3 Max with 128GB unified memory is the local reasoning backbone. At Q4_K_M quantization, the full model fits in memory with room for KV cache, delivering **~8–12 tokens/second** via Ollama with Metal acceleration. Hermes 4 (August 2025) adds hybrid reasoning with togglable `<think>...</think>` chains. Nous Chat +2 The 70B Q4 variant is the sweet spot for 128GB unified memory. Nousresearch Available on Ollama as `hermes3:70b`; Hermes 4 70B is available as GGUF from HuggingFace Hugging Face but not yet in the Ollama library.

Skip AirLLM entirely. Real-world testing shows ~10 minutes per response for 70B models on Apple Silicon due to sequential layer-by-layer loading. Medium Ollama/llama.cpp with Metal acceleration is **100–1000× faster** for the same models on the same hardware. Studyhub

For development tooling, run **both Continue.dev and Tabby**: Continue.dev as a VS Code extension with agent mode, chat, and tab autocomplete booststash (all via Ollama); Ollama Tabby as a self-hosted Rust server with Metal acceleration for team-wide code completion. github Both share the same Ollama backend. Nisrulz

---

Legal risk map

Gambling classification is the highest-risk area. U.S. gambling law requires three elements: consideration (something wagered), prize (something won), and chance. Users paying satoshis clearly constitutes consideration. If there is **any mechanism where users can win money, prizes, or items of value**, the platform enters gambling territory. The New

York AG's 2026 lawsuit against Valve for loot boxes demonstrates aggressive enforcement. **The entire economic design must ensure users are exclusively paying for entertainment** — injecting events for the fun of watching the AI react — with no prizes, no payouts, and no "winning."

Federal money transmission risk is low. FinCEN guidance (FIN-2013-G001, FIN-2019-G001) classifies businesses accepting cryptocurrency as payment for goods or services as "users," not money transmitters. The platform is not transmitting funds between parties, exchanging crypto on behalf of others, or holding custodial balances. Stripe **State-level risk is moderate** — 49 states plus DC have money transmitter licensing, and New York's BitLicense is particularly strict. The merchant exception and agent-of-payee exemption likely apply, but a state-by-state analysis is warranted.

Bethesda IP is a moderate risk. Their fan video policy explicitly encourages Let's Play videos and fan tributes, but it's a unilateral grant that may not contemplate a commercial, crypto-powered, AI-operated streaming platform. Seeking explicit written permission from Bethesda/Microsoft is recommended, or consider using an open-source game to eliminate this risk entirely.

Tax compliance requires recording the fair market value in USD of each Lightning micropayment at the time of receipt (IRS Notice 2014-21). The IRS does not distinguish between on-chain and Lightning transactions. A de minimis exemption has been proposed by AICPA but not adopted. Stripe

---

What exists today versus what must be built

Component	Status	Primary tools
TES3MP server + Lua scripting	✅ Exists (dormant)	TES3MP 0.8.1, Docker server container
Client-side input bridge	🔨 Must build	CGEvent API (macOS), pyautogui
Server↔Agent REST bridge	🔨 Must build (pattern exists)	FastAPI + luasocket TCP
Navmesh/pathfinding integration	🔨 Must build (data exists)	SQLite navmesh cache + external A*
AI agent reasoning core	🔨 Must build (patterns proven)	Agno + Ollama + Hermes 70B

Game knowledge base (RAG)	🔨 Must build	UESP wiki → embeddings → vector store
Memory system	🔨 Must build (architecture proven)	Three-tier: working/episodic/semantic
Screen capture + encoding	✅ Exists	OBS + ScreenCaptureKit + VideoToolbox
Low-latency streaming	✅ Exists	MediaMTX or LiveKit
Overlay system	✅ Exists (pattern proven)	OBS Browser Source + WebSocket
Nostr DVM integration	✅ Exists (alpha)	nostr-sdk + ezdvm/nostrdvm
NIP-89 service discovery	✅ Exists (alpha)	nostr-sdk + kind:31990 events
L402 API gating	✅ Exists	Aperture + LND
Lightning payments (receive)	✅ Exists	LND + Voltage/Alby Cloud
NWC for user wallets	⚠️ Partial (no Python client)	@getalby/sdk (JS) + custom Python
LNURL-auth	✅ Exists	lnurl (Python v0.8.3) + coincurve
TTS voice pipeline	✅ Exists	Kokoro-82M via mlx-audio + RealtimeTTS
Custom voice training	✅ Exists (needs NVIDIA GPU)	Piper training pipeline
Web platform (Next.js stack)	✅ Mature	Next.js 15 + React 19 + Tailwind 4 + shadcn/ui
3D/WebGL frontend	✅ Mature	Three.js WebGPURenderer
Serverless GPU inference	✅ Exists	RunPod Serverless, Modal
Agent orchestration	✅ Exists	Agno + FastMCP
Local LLM inference	✅ Excellent	Ollama + Hermes 3/4 70B on M3 Max

Review/reputation system	❌ Not standardized	Kind 2020 proposal unmerged

---

Known risks ranked by severity

1. 🔴 **Gambling misclassification**: If any feature allows users to "win" something of value, the platform faces gambling regulation in every U.S. jurisdiction. Design the economic model as entertainment-only from day one.
2. 🔴 **TES3MP abandonment**: The project is effectively dormant with a widening version gap against OpenMW mainline. If a blocking bug is discovered, there may be no upstream fix. Mitigation: budget for maintaining a fork, or evaluate migrating to pure OpenMW with a custom multiplayer/automation layer.
3. 🟡 **Client-side input fragility**: CGEvent-based input injection is inherently brittle — window focus changes, resolution changes, or macOS security updates can break the pipeline. This is the weakest link in the agent control chain.
4. 🟡 **Bethesda IP revocation**: The fan video policy is not a license. Bethesda/Microsoft could issue a takedown at any time, especially for a commercial crypto-powered use case.
5. 🟡 **Nostr ecosystem immaturity**: nostr-sdk is marked ALPHA. GitHub DVMs have limited client integration. Relay behavior is inconsistent. Plan to work around rough edges and potentially run dedicated relay infrastructure.
6. 🟡 **Lightning micropayment tax complexity**: Tracking FMV of potentially thousands of daily micropayments creates significant accounting burden with unclear IRS guidance on Lightning-specific questions.
7. 🟢 **Performance on M3 Max**: All components — Ollama 70B inference, OpenMW rendering, OBS encoding, TTS synthesis, server processes — should run comfortably on M3 Max with 128GB unified memory, but the aggregate load under peak conditions needs benchmarking.

Recommended implementation sequence

Month 1–2: Stand up TES3MP server (Docker/Linux) + client (macOS). Build the FastAPI ↔ TCP ↔ Lua bridge. Implement basic agent loop: read game state via server Lua → format as text → send to local Hermes 8B via Ollama → parse structured action → execute via server

Lua (teleportation, inventory) or CGEvent (movement, NPC interaction). Stream via OBS + MediaMTX. No payments, no Nostr, no fancy TTS — just prove the agent can navigate Seyda Neen and talk to Sellus Gravius.

Month 3–4: Add knowledge base (UESP wiki → vector store), three-tier memory system, hierarchical quest planning, and upgrade reasoning to Hermes 70B for complex decisions. Build the navmesh integration for pathfinding. Add Kokoro-82M narration. Implement the OBS overlay showing agent reasoning and game events.

Month 5–6: Integrate Lightning payments via LND + Aperture. Implement the spectator interaction system (pay sats → inject game event via server Lua). Add LNURL-auth for user identity. Build the NIP-90 DVM wrapper so the agent's services are discoverable on Nostr.

Month 7+: Launch the /tower web platform. Expose the agent framework as a reusable system for community agents. Implement NIP-89 service discovery. Build the sovereign economy layer with NWC for automated payments between agents.

This sequence front-loads the highest-uncertainty components (TES3MP automation, agent reasoning) and defers the better-understood infrastructure (payments, streaming, web platform) until the core loop is proven.