Pennyroyal AI Platform

An Implemented, Human-Governed Multi-Agent Architecture

Data sovereignty, event-driven agent execution, human-in-the-loop skill creation, and long-context inference

Architecture and integration: John Pezzulli
AI-assisted implementation and review: OpenAI Codex and locally hosted models
Architecture snapshot: 21 September 2026
Version: v0.3

Draft architecture document · v0.3 · 21 September 2026
This document describes an implemented personal system and will continue to evolve. It is not a formally validated reference architecture.

Contents

Executive summary

This deployed multi-user AI architecture is built around Pennyroyal, a customized SGLang inference runtime. **Data sovereignty is a core property of work executed entirely through Penny:** model prompts and completions, conversations, tool execution, generated artifacts, image and speech processing, and durable file originals remain on operator-controlled hosts. Connected search retrieves external information without moving model inference or conversation storage to a hosted AI service.

The Personal Open WebUI Fork, pennyroyal-owui, provides the application and agent surface. Native per-user terminals execute tools; a durable executor coordinates long-running assignments and event-driven return to Codex; Versity S3-compatible storage retains file originals; and separate services provide research, browser interaction, image processing, speech, and document creation.

The system supports long-context, tool-using work by several people with different permissions. Its principal design choices are local control of the AI data plane, separation of inference from agent control, durable task state from notifications, durable originals from working files, and shared capability installations from per-user execution state. Those choices allow the same model service to support interactive work, supervised engineering, file-intensive analysis, and multimodal production while preserving distinct ownership for each concern.

This document records the implemented components, interfaces, request and data flows, design rationale, and measured results. Engineering teams and individual builders can adapt the patterns to their own workloads. Deployment-specific addresses, credentials, and customer source material are omitted.

Concern	Implemented component	Responsibility
Inference	Pennyroyal / customized SGLang	Model execution, long context, speculation, and reusable prefix state
Application	Personal Open WebUI Fork	Authentication, chats, context assembly, tool presentation, and file interaction
Agent control	Penny executor and Codex supervisor	Durable task lifecycle, wakeup, review, and steering
Execution	Identity router and native Open Terminal	Tool work under each authorized Linux identity
Durable data	Versity Agent Library	Per-user and shared S3-compatible file originals
Specialized processing	AI Tools, ComfyUI, Dubline, research tools	Images, speech, and connected work through bounded interfaces

Data sovereignty in the Penny execution path

In this architecture, data sovereignty means that the operator controls the machines and services that assemble model requests, execute inference, retain conversations, run tools, process media, and store outputs. Open WebUI and its task ledger run on room101; Pennyroyal and its GPU, HiCache, and NIXL tiers run on thegrid. Native terminal work runs under mapped Linux identities. Versity Agent Library stores durable originals on thegrid. AI Tools, ComfyUI, and Dubline process images and speech on thegrid and Neomatrix. The resulting files return through authenticated Open WebUI or terminal routes.

Data or execution	Operator-controlled location
Model prompts, token generation, and long-context state	Open WebUI request assembly and Pennyroyal inference; GPU, host RAM, and local SSD cache
Chats, settings, task states, and work traces	Open WebUI application data and executor ledger on room101
Tool processes, browser profiles, and working files	Native per-user terminal accounts and homes on room101

Data or execution	Operator-controlled location
Uploaded originals and deliberate published outputs	Versity S3-compatible Agent Library on thegrid
Generated images and edits	AI Tools and ComfyUI on thegrid and Neomatrix, then authenticated Open WebUI display and archive
Speech recognition and synthesis	Dubline on Neomatrix, with Open WebUI handling the authenticated user session
Editable documents and presentations	Native user workspace, shared local tools, and authenticated file delivery

Research remains connected. A search provider receives the query sent to it, and a visited website receives normal browser requests; both return information for Penny to evaluate. They do not host Pennyroyal's model prompts, generation, chat history, or artifact store. The full model interaction stays within the local architecture even when external information is consulted. The OpenAI-backed Codex supervisor is a separate workflow; the sovereignty claim here concerns execution through Penny without that supervisor.

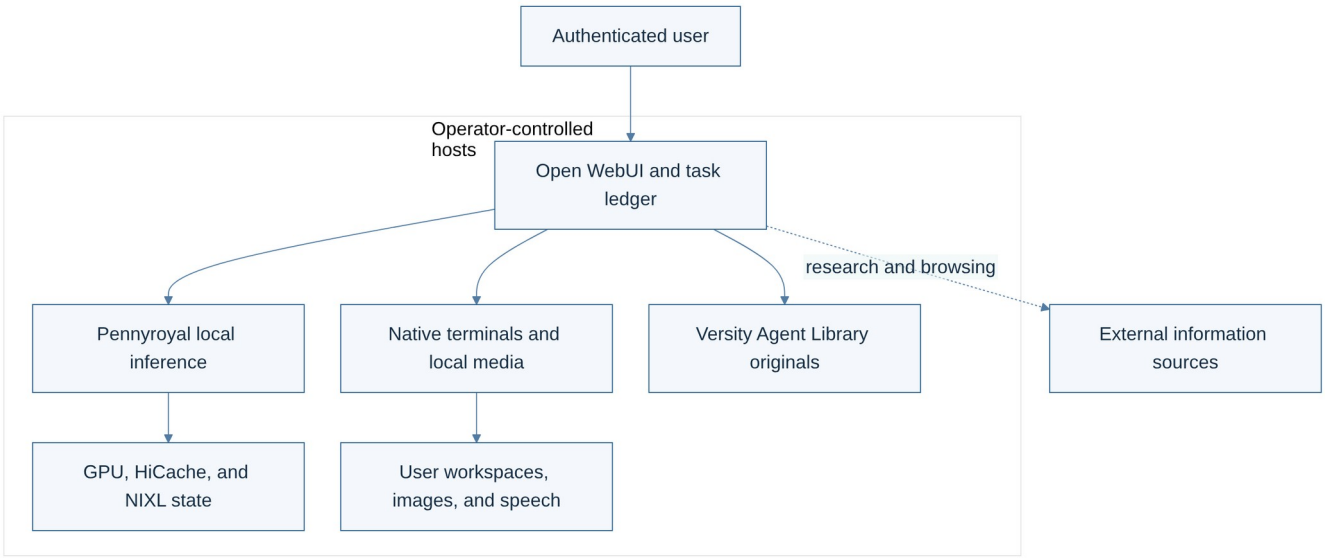

Figure S1. Penny data sovereignty boundary

This arrangement gives the operator control over both computation and persistence. The S3 interface is local Versity storage, not an Amazon S3 bucket. User grants and native identities decide which stored objects and tools an account can reach; authenticated delivery governs what leaves the system as a finished artifact. A workflow can use outside facts while keeping its model execution and working record local.

Implemented agentic capabilities

This is a deployed agentic system in the operational sense: agents can pursue a bounded objective through multiple model and tool turns, retain state across pauses, delegate selected work, and return evidence for human review. It is also a **multi-agent architecture** at two levels. Codex acts as a supervisor and Penny as a separately executing worker; within OWUI, Penny can use bounded native subagents for separable work. These roles do not share one undifferentiated prompt or execution identity.

Term used in agent architecture discussions	Implemented mechanism	Governing boundary
Multi-agent orchestration	Codex delegates a durable assignment to Penny; Penny can delegate bounded subtasks inside OWUI	Codex reviews Penny's result; the parent Penny turn remains responsible for subagent results

Term used in agent architecture discussions	Implemented mechanism	Governing boundary
Bounded multi-agent review	The supervisor reviews substantive work and can assign fresh low-context reviewers	Review stops after at most three cycles; unresolved findings trigger a human design decision
Event-driven agents	Actionable executor state changes queue a notification to the originating Codex task	Codex reads the ledger and evidence; the event itself carries no decision authority
Human in the loop	The human owner sets scope, answers blocked questions, reviews artifacts, and authorizes consequential changes	Human checkpoints occur at decisions and persistence boundaries
Durable agent execution	Ledger, saved chat, retained workspace, and resume path	Interrupted or blocked work continues from retained state rather than being silently replayed
Human-approved agent learning	Tool-using chats can produce personal skill proposals	The account owner reviews, revises, rejects, or approves the exact persistent change
Context engineering	Layered guidance, selected skills, latest-request usage display, and compaction	The application controls what enters a model request; the cache controls reuse of its computation

The human-approved skill path is an especially useful example. It turns observed agent work into a *candidate reusable procedure* while retaining an explicit owner decision before that procedure becomes persistent and discoverable in later chats. Chapter 5 traces this loop in detail.

1. System requirements and architecture boundary

The workload includes multi-turn engineering assignments, long-context analysis, browser and repository interaction, image and document production, and later retrieval of user files. The system must retain task state when an agent waits for input, assemble context within the model window, execute tools under the correct user's authority, and return finished files through an authenticated path.

Pennyroyal serves inference. Open WebUI owns conversation and tool orchestration. The executor owns assignment lifecycle and compact results; saved OWUI chats retain the complete work trace. Native terminals own command and file execution. Agent Library owns durable originals. Specialized services own image and speech processing. Each responsibility has an independent source and operating boundary.

The deployment serves a small trusted group. Its user boundary relies on OWUI grants, mapped Linux accounts, application ownership checks, and selected filesystem namespaces. This is the stated workload and security model for the architecture.

Requirements translated into architecture

Requirement	Implemented response	Design consequence
Long, repeated context	A 524,288-token served window, application compaction, and tiered prefix-state reuse	Conversation continuity and compute reuse are managed by different layers.
Work that outlives a chat turn	Executor ledger, retained OWUI chat, stable task workspace, and actionable notifications	A supervisor can return after Penny pauses or finishes without relying on an open connection.
Different users and capabilities	Additive OWUI groups, identity routing, native Linux users only for workspace access	A user can receive research or image capability without receiving another person's terminal or files.
Reusable source files	Agent Library objects separate from OWUI display files and native working copies	A generated or uploaded original can survive a chat or workspace cleanup.
Multiple specialized models and tools	Inference, image, speech, and research services behind explicit contracts	Each processor can be changed or measured without moving the whole application.

Requirement	Implemented response	Design consequence
Reviewable outcomes	Saved execution trace, compact task result, source revisions, and representative acceptance checks	A completion notification initiates inspection; it does not itself establish correctness.

The system has two important axes. The **data plane** carries prompts, model state, tool inputs and outputs, images, audio, and files. The **control plane** carries authentication, grants, task states, notifications, approval decisions, and source selection. Several implementation choices follow directly from keeping those axes distinct. For example, the notification channel carries a small status event, while the executor ledger retains the assignment's current state. Similarly, an internal signed storage URL can move image bytes between services, while an authenticated OWUI route delivers a result to a person.

The architecture also separates *shared capability* from *shared identity*. A single managed installation of document tools or a browser binary avoids duplicate maintenance. The state produced by those tools—profiles, downloads, temporary files, and working documents—belongs to the Linux account under which they run. This distinction makes the multi-user system feasible without building a custom container orchestrator for every account.

The remainder of the document follows an actual request and its related data through these boundaries. The physical deployment illustrates one implementation; the interface and ownership decisions are the reusable parts.

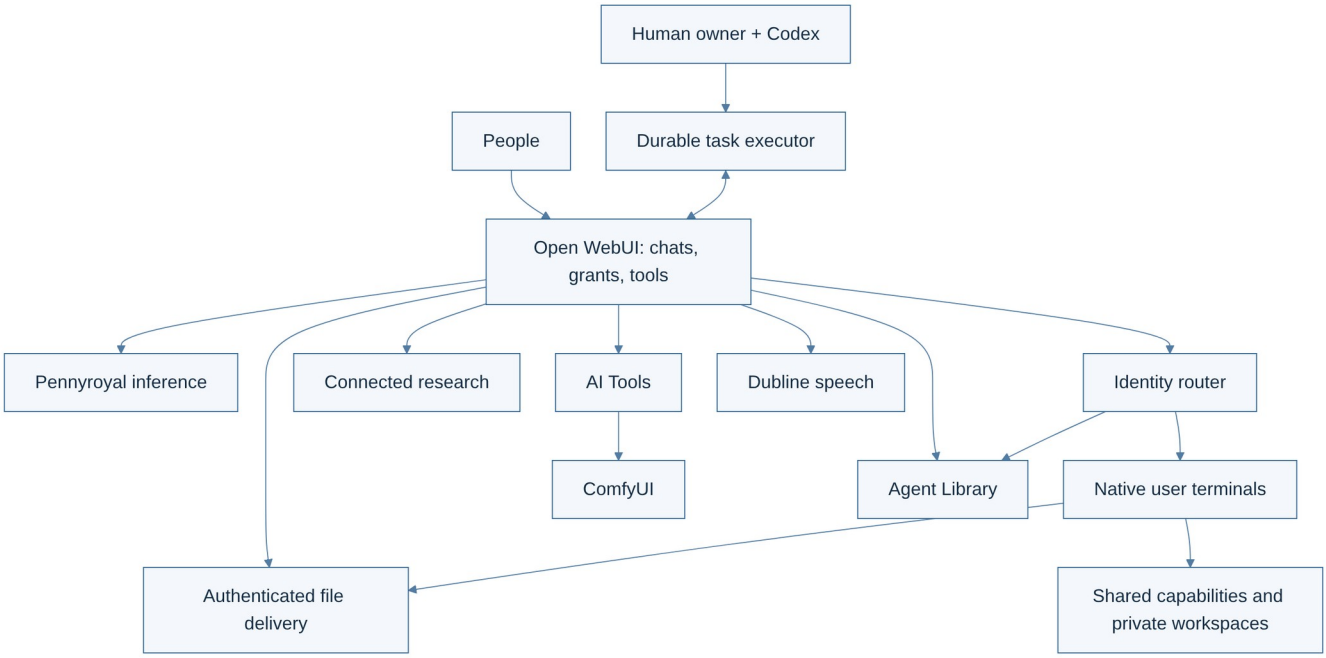

Figure 1. Logical architecture

Figure 1 identifies logical service responsibilities. File archival enters through an OWUI upload hook or the terminal identity router; Chapter 8 traces both paths.

2. Physical deployment and service placement

Thegrid is the physical server for Pennyroyal inference, public ingress, the image-tool interface, and Agent Library storage. Room101 is a VM on thegrid, hosting the Personal Open WebUI Fork, its task executor, identity router, and native terminals. Neomatrix is a separate workstation hosting ComfyUI, Dubline speech, and the normal Codex Desktop work surface. The model GPU is on thegrid; image and speech generation use Neomatrix's GPU. Room101 does not need a local inference GPU.

The placement allows the application, inference runtime, and specialized processors to change independently. It also exposes network and resource crossings that require end-to-end verification: model health alone does not establish image processing or authenticated file delivery.

Placement and resource boundaries

Thegrid is the stable physical center. Its RTX PRO 6000 Blackwell GPU serves Pennyroyal; the machine also holds the model files, the NIXL cache filesystem, Agent Library objects, public ingress, and the room101 VM. The VM isolates the browser application and user execution services from the native inference process while keeping them on the same host's local network. The model's weights and cache therefore remain with the GPU; the chat database, user sessions, and terminal processes remain with the application VM.

Neomatrix contributes a second GPU and a different class of workloads. ComfyUI uses it for image generation and editing; Dubline uses it for local speech recognition and synthesis. Codex Desktop also runs there, but Codex is a separate supervisory work surface rather than another process inside the OWUI container. The division creates two resource domains: heavy language-model inference on thegrid and image/speech generation on Neomatrix. No general GPU scheduler coordinates every service across both hosts; the document therefore describes measured paths and resource contention rather than promising uniform latency.

Room101 runs Open WebUI as a rootless Podman container. The terminal router and native Open Terminal instances are host services with separate identities. That combination preserves an application container boundary while giving authorized users normal filesystem and tool behavior under their Linux accounts. The application reaches the router through the container's host gateway; the router then selects the native terminal associated with the authenticated OWUI identity.

External users enter through Caddy on thegrid and authenticate in OWUI. Caddy provides HTTPS ingress, not identity substitution or file authorization. OWUI and its terminal proxy enforce the authenticated download paths described later. The separate public artifact service used elsewhere in the household is outside this flow.

There are several physical dependencies worth showing without turning this document into an operations manual. The room101 VM depends on thegrid. ComfyUI and Dubline depend on Neomatrix being available. Agent Library originals depend on their storage and service state, including metadata. NIXL cache state is disposable performance data and has a different recovery priority from those originals. A failure in any one path can produce a partially working application: model text may still work while images, speech, or file retrieval do not.

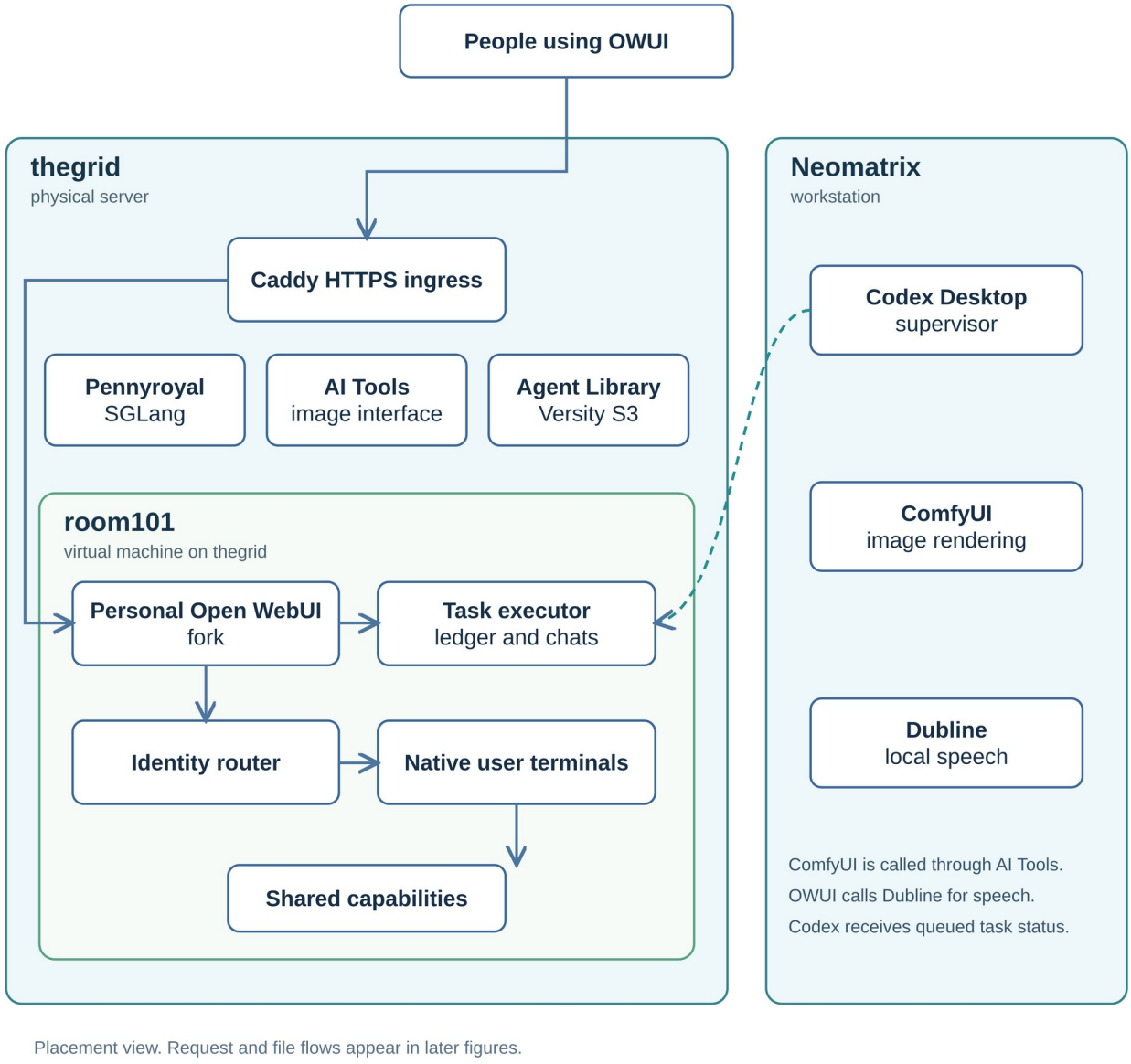

Figure 2. Deployment architecture

Figure 2 maps the logical components onto the deployed hosts. The dashed arrow is a status message queued to the originating Codex task through existing SSH and the Codex CLI. A small notifier script restricts and formats that call; there is no separate notification service or result payload on the arrow. Exact private addresses and public hostnames are omitted.

3. Pennyroyal inference runtime

Pennyroyal is the OpenAI-compatible inference endpoint. Its SGLang-derived source qualifies two Qwen3.8 profiles on one RTX PRO 6000 Blackwell GPU: Flash-Next NVFP4 with native speculative decoding, and a 27B FP8 target with DFlash2. The deployed launcher selects the Flash-Next recipe with an OrcaRouter checkpoint derivative. The performance campaign below used the named RadixArk checkpoint and remains the published benchmark for the corresponding runtime configuration.

The endpoint receives assembled requests from Open WebUI. It does not own OWUI accounts, chat history, skills, permission grants, or files. That division matters when comparing clients: two applications can call the

same model while producing different agent behavior because they assemble different context, expose different tools, and maintain different state.

The two qualified configurations solve different model-serving problems within the same source line:

Profile	Precision and speculation	Architectural point
Flash-Next	NVFP4 target, native NEXTN multi-token prediction, with an FR-Spec draft-vocabulary option	Keeps the target model's full verification policy while reducing draft work; its hybrid recurrent, sparse-attention, and PLE state needs coordinated recovery. This is the profile selected by the live launcher.
27B	FP8 target and DFlash2 draft	Uses a separate draft model and its own target/draft cache state. The public 27B measurements identify the checkpoint derivative used, which matters when interpreting behavioral results.

Runtime modifications and validation

The selected accelerator and model family did not reduce to a stock launch command. The source line integrates model-specific execution paths and fixes for the SM120 GPU architecture. Flash-Next combines ordinary attention KV with recurrent GDN state, sparse attention, and a large PLE table. Native NEXTN multi-token prediction generates draft candidates without a separate draft checkpoint; FR-Spec reduces the vocabulary considered by that draft path. The target model still verifies candidates and controls acceptance. By contrast, the 27B profile uses DFlash2 as a separate draft model with its own state.

Correctness and performance are coupled in this work. A speculative path must preserve accepted state when a draft branch is rejected, when a request resumes from a cached prefix, and when a captured CUDA graph is replayed. Kernel routing must select implementations that actually support SM120 and the data representation in use. The public source identifies narrow overrides for GDN phases, QSA sparse attention, output and recovery paths, and multimodal rotary computation. It retains fallbacks where an apparently related backend cannot execute correctly on this GPU. These are concrete runtime integrations, not a collection of unchecked acceleration flags.

The qualified recipes fix served context, precision, speculative configuration, memory sizing, and cache layout together. A checkpoint or speculation-map change can alter accuracy and cache representation. The repository therefore records qualified profiles separately from experimental variants.

Model-serving capacities

A 524,288-token context window is the maximum served request context in the qualified recipes. The GPU KV pool is a different allocation: it holds state for active and reusable requests, and its capacity can exceed one request's context. HiCache host memory and NIXL storage extend the persistence of compatible prefixes beyond that GPU pool. The architecture must preserve this distinction when explaining both concurrency and long conversations. More shared KV capacity can help several requests coexist, while a larger per-request window would change the model-serving contract.

The runtime does more than select a checkpoint. The owning source documents SM120-specific routing for attention and recurrent kernels, CUDA graphs for target, draft, verification and recovery, speculative decoding, multimodal rotary handling, and cache persistence. Some optimizations are narrow precisely because a kernel that works on a related GPU architecture may not have an SM120-compatible binary or the required state semantics. The configuration and validation records make those choices inspectable instead of treating the GPU as a generic accelerator.

SM120 execution and accepted-state recovery

The selected NVIDIA RTX PRO 6000 is an SM120 Blackwell GPU. Pennyroyal's SGLang line resolves a collection of model phases to kernels that are both fast and valid for that architecture. The Flash-Next profile uses FlashInfer for selected GDN decode and prefill phases, a sparse QSA prefill path, and a FlashInfer QSA wrapper that resolves to an SM120-capable decode implementation. Other phases retain Triton or SGLang CUDA implementations. The source documents cases where a backend available on SM100 is not executable on SM120, and keeps the guard rather than forcing an incompatible binary. That restraint is part of the engineering work: a dispatch change must be qualified for the phase and representation it actually serves.

Flash-Next's speculative path uses native NEXTN multi-token prediction. It proposes several token steps before the target model verifies and accepts them. FR-Spec reduces the draft vocabulary to a mapped subset, making draft scoring cheaper while leaving target verification over the full vocabulary. The accepted state after verification must align with the chosen tokens. The runtime records target, draft, verify, prefill, and recovery graph behavior in its qualification material because a speculative success rate or decode speed is useless if the next turn resumes from the wrong recurrent state.

The 27B/DFlash2 profile is a distinct qualified path. It has a separate draft model, target and draft KV pools, and DFlash2-specific verification and state materialization. It shares the overall SGLang-derived source line and HiCache/NIXL approach, but its measured timings and persistence layout must not be copied into the Flash-Next description. The source also documents broader upstream-compatible model use without the qualified HiCache/NIXL persistence; those cases are outside the two complete recipes described here.

These implementation details have direct application consequences. Long conversations and tool continuations repeatedly exercise prefill, speculative decode, cancellation, grammar-constrained output, and resume. The release work includes tool-markup handling and cache/prefill reliability fixes because an agent relies on correct structured output over many turns, not only fast free-form text.

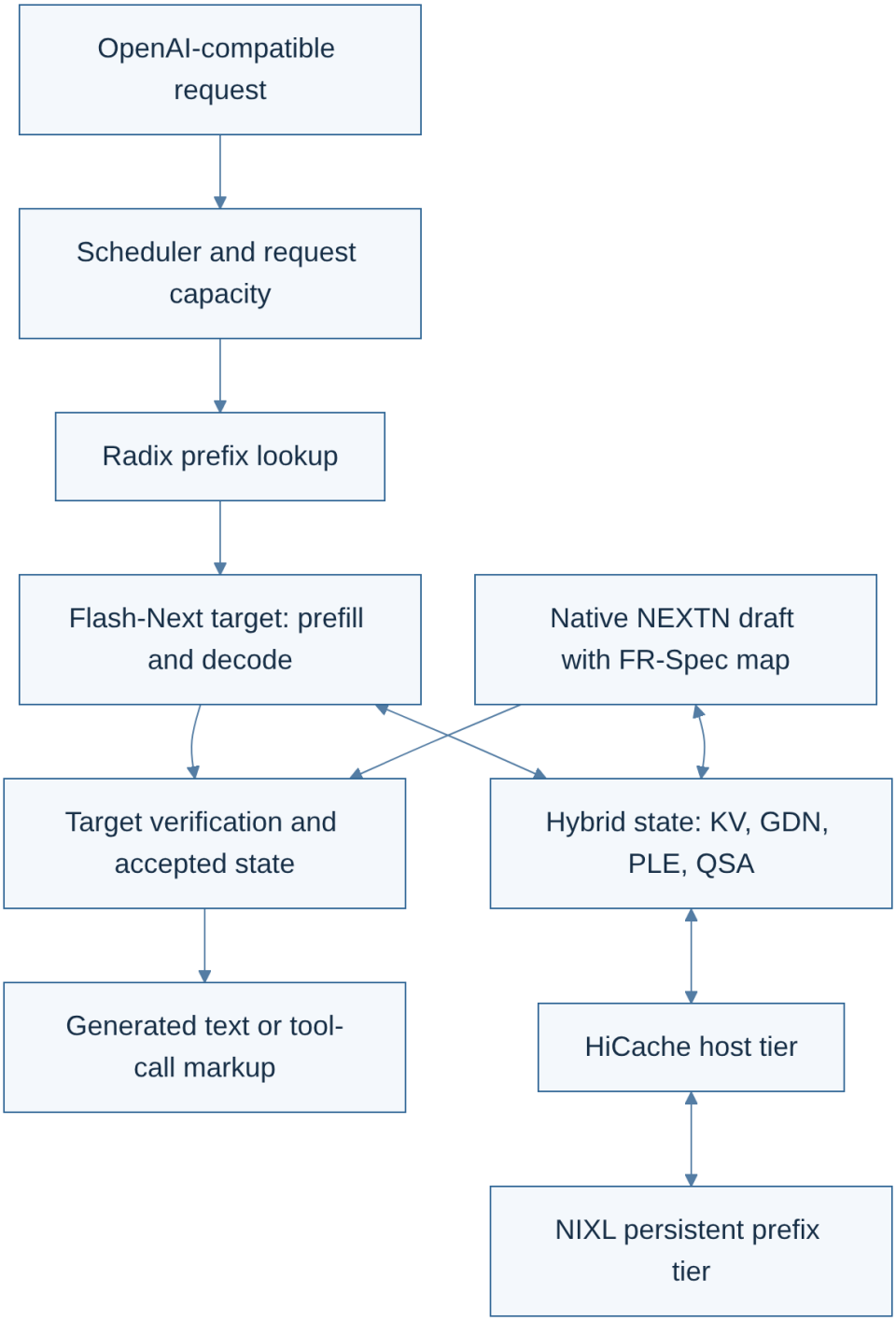

Figure 3a. Inside the selected inference profile

The draft proposes likely next tokens; target verification decides what is accepted. FR-Spec narrows the draft vocabulary while the target retains its full vocabulary and acceptance policy. The source also has explicit SM120 kernel-routing and recovery work because correct speculative continuation needs the accepted model state, not just plausible text. Source: v2.5.1 runtime overview.

A request moves from tokenization through prefix lookup and prefill, then generation and possibly a tool-call continuation. The 524,288-token served context is a per-request limit; the larger shared KV pool controls how much state can be resident across requests. They are related capacities, not the same number. A single throughput figure cannot describe cold prefill, warm reuse, single-request decode, four-request aggregate work, and multimodal/tool-heavy sessions equally.

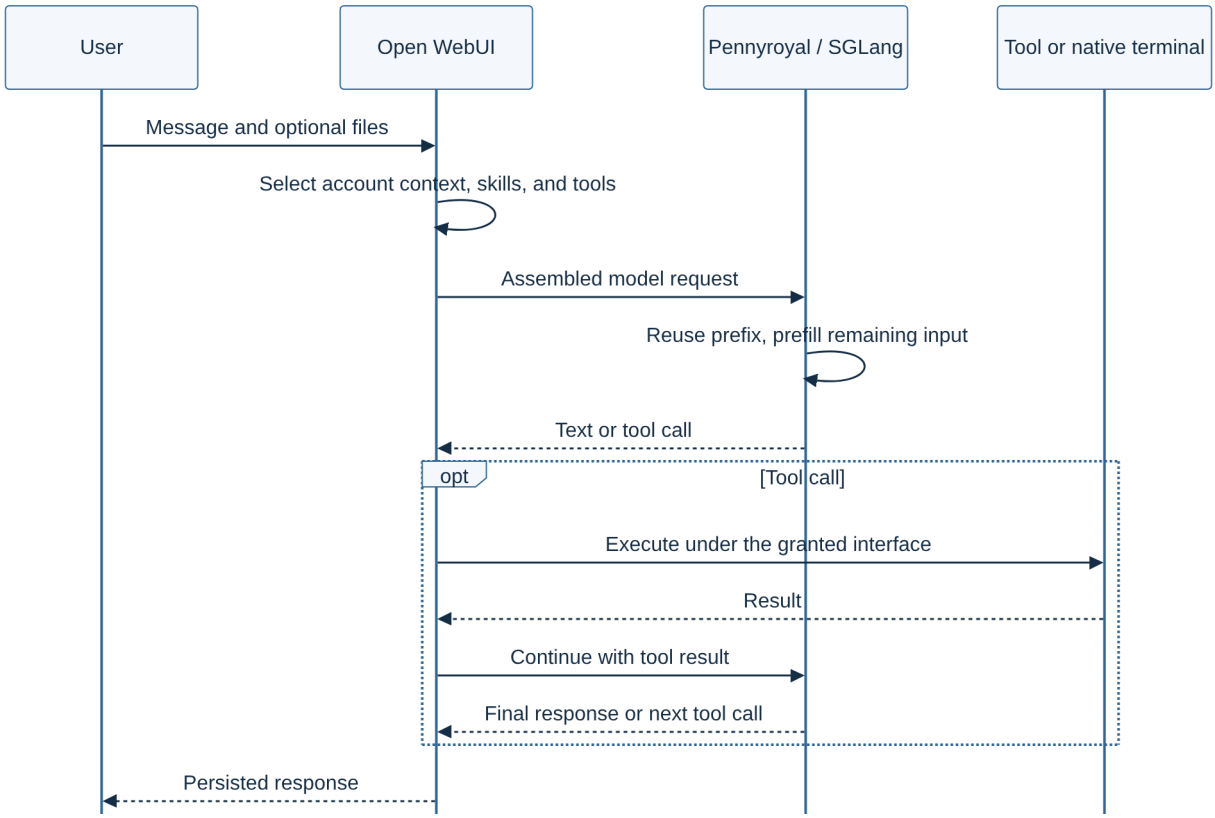

Figure 3b. An inference request

The tool result changes the next request. That continuation is one reason cache reuse, context measurement, and mid-turn compaction matter to agent work.

4. HiCache, NIXL, and long-context reuse

During prefill, the model computes attention state for the input prefix. Repeating a long unchanged prefix from scratch costs time and compute. SGLang's radix cache retains reusable state close to the GPU. HiCache extends the useful lifetime of that state into host memory; the selected NIXL POSIX storage backend extends it to a local SSD. The SSD is a performance tier, not the authoritative chat record.

For this model family, "KV cache" is a convenient shorthand but not the full persistence contract. Correct reuse can depend on model-specific sibling state as well as target KV: the SGLang repository documents the additional Flash-Next state that must be restored consistently. A prefix that appears textually similar is not enough if the model, tokenizer, layout, or speculative method is incompatible. Cache misses and short recomputed tails remain normal.

Cache identity incorporates the representation and relevant model/speculative configuration, so an incompatible prior state is not silently loaded as a hit. The documented Flash-Next host tier is configured for 32 GiB; that is a limit in the qualified recipe, not a measured total of every host allocation.

A separate optional SSD feature streams Flash-Next's large PLE table from a prepared immutable snapshot instead of keeping that table pinned in RAM. This NVMe PLE option changes where model data is read; it is **not** the NIXL prefix-cache tier. The default documented recipe retains RAM-backed PLE. Source: v2.5.1 memory and persistence notes.

Cache identity and restoration

A useful cache hit requires more than matching the first several words of a prompt. Tokenization, model weights, precision, speculative method, and the structure of model-specific state all affect whether an old representation is valid. The NIXL namespace records the representation identity needed to keep incompatible cache material separate. Flash-Next persistence includes packed target/native-MTP KV and associated GDN, PLE, and QSA state; the 27B path has its own target, draft, and recurrent-state requirements. The source treats this as a correctness boundary. Restoring only attention KV while omitting a dependent state component could make the next generation wrong even though the text prefix matched.

The GPU radix tree is the hot path. HiCache provides a page-oriented host-memory tier with write-through behavior in the qualified configuration. NIXL uses its POSIX FILE backend for the SSD tier with io_uring and direct I/O. The first request for a new prefix pays prefill cost and can populate reusable state; a later request may restore pages from a lower tier and compute only a tail. Storage cleanup and namespace changes can remove a potential hit without damaging the authoritative chat. An absent cache therefore affects performance and time to first token, not the integrity of the OWUI conversation.

The SSD has two conceptually separate potential jobs. NIXL stores reusable *computed prefix state*. Optional NVMe PLE stores an immutable *model table* and streams it through bounded buffers. The PLE option trades fixed host-RAM residency for storage reads; it does not turn the model's PLE table into a chat cache. The distinction matters when interpreting memory savings, SSD traffic, and restart behavior.

Representation persistence and failure scope

The persistent prefix is a representation of model computation, not a serialized transcript. The Flash-Next state includes packed target and native-MTP KV, complete recurrent GDN state, PLE-related sibling state, and compressed QSA index keys in the qualified layout. Restoring these components together allows subsequent target verification and long continuation to operate on the same accepted state as an uninterrupted request. The source includes ordering and copy-on-write work to keep a load from racing with writes or exposing a partially restored representation.

The cache is namespaced by representation identity. A change in checkpoint, tokenizer, precision, draft map, PLE placement, or other relevant configuration can select a different namespace. That behavior sacrifices reuse when necessary to protect correctness. A restart can reuse compatible on-disk pages; a new or incompatible namespace begins cold. Similarly, storage cleanup can discard cached state without deleting the OWUI chat or Agent Library object. The hierarchy therefore has a clear failure hierarchy: cache loss increases computation; loss of application or file state has a different consequence.

The published qualification did more than show that files appeared on an SSD. It tested exact long-context needle retrieval after restart, concurrent restoration, and continuation after restoring a long prefix. For the Flash-Next example cited below, almost all of the 489,879-token input was restored and the remainder was recomputed. The short tail matters: page boundaries and new user material can leave a small amount of real prefill even when restoration succeeds.

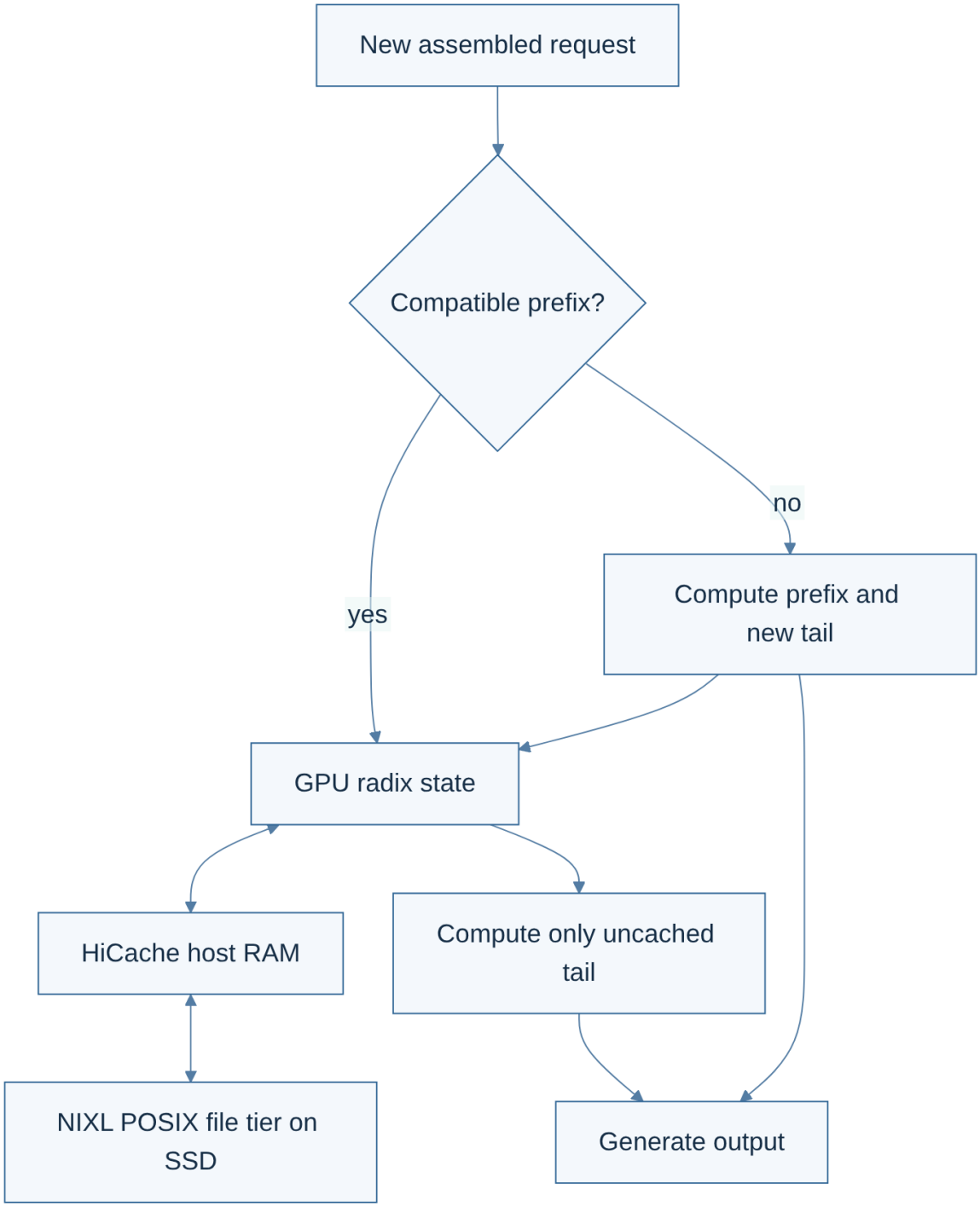

Figure 4. Cache hierarchy and request reuse

GPU-resident state can satisfy a request without SSD I/O. Colder compatible state can be restored from host memory or the NIXL storage tier. In the Flash-Next restart qualification, 489,856 of 489,879 input tokens were restored, 23 were recomputed, and all three long-context needles remained exact. Source: v2.5.1 HiCache/NIXL persistence.

Measured performance campaigns

The v2.5.0 online-FP8 campaign measured post-first-token decode on one RTX PRO 6000 with the same Flash-Next serving shape and RadixArk checkpoint on both sides of the comparison. Decode increased **15.8–28.3%**

across the short, 128K, and 490K request shapes. Graph capture also left about 3.86 GiB more GPU memory available. Cold time to first token did not improve in the long samples. Source: v2.5.0 campaign carried in the v2.5.1 README.

The v2.5.1 release retained that decode campaign and changed cache retention, prefill reliability, tool/parser behavior, and optional capacity settings. In its nine-conversation host-cache revisit test, substantial restores improved from 0/9 to 9/9, and median revisit time fell from 9.04 to 1.05 seconds with correct answers. Source: v2.5.1 changes.

The v2.5.0 speed campaign exercised the full suite and published the changed results. The unchanged v2.4.0 cold-prefill and four-request throughput values carry forward in the documented performance baseline; v2.5.0 adds the online-FP8 decode comparison, and v2.5.1 adds the cache-retention result.

Campaign	Workload	Result
Carried forward from v2.4.0	63,864-token cold prefill; server rate and client time to first token	14,842 input tokens/s; 5.242 s to first token
Carried forward from v2.4.0	489,879-token cold prefill	8,773 input tokens/s; 60.613 s to first token
Carried forward from v2.4.0	Four concurrent 1,024-token outputs; median of three groups	446.49 output tokens/s aggregate
v2.5.0 online-FP8 decode	Short request; 1,024 output tokens; three-run medians	161.47 → 207.12 tokens/s, **+28.3%**
v2.5.0 online-FP8 decode	128K input; 1,024 output tokens; one observation per arm	154.70 → 195.63 tokens/s, **+26.5%**
v2.5.0 online-FP8 decode	490K input; 1,024 output tokens; one observation per arm	149.14 → 172.64 tokens/s, **+15.8%**
v2.5.1 cache-retention correction	Nine-conversation Flash-Next host-cache revisit	Substantial restores 0/9 → 9/9; median revisit 9.04 → 1.05 s
Flash-Next restart restoration	489,879-token input with exact needle checks	489,856 restored; 23 recomputed

These campaigns measure different mechanisms: cold prefill, concurrent throughput, FP8 decode, and cache retention. The release label identifies where a result was measured; unchanged results carry forward into later releases.

Performance implication

Agent sessions repeatedly submit substantial shared context with new tool output and decisions appended. Reusing a compatible prefix reduces repeated prefill work; persistence extends that benefit across GPU eviction and qualified service restarts. Context-window management remains an application responsibility, handled separately through measurement and compaction.

5. Context engineering and human-approved skill learning

Open WebUI turns the inference endpoint into a usable multi-user work surface. It stores chats, authenticates people, grants models and capabilities, handles files, presents native tools, and assembles the actual request sent to Pennyroyal. The personal fork also carries the durable executor and several agent-continuation and context features. The deployment repository owns the desired configuration, prompts, skills, router, and capability installation; the private application fork owns downstream Open WebUI source. “Personal” identifies ownership of that fork; the deployed application itself is multi-user. Upstream changes are reviewed before integration as an update policy, separate from the fork’s name.

The prompt architecture is deliberately described here at the level of responsibility. A shared layer supplies common behavior. Account-specific guidance calibrates the interaction for each person. Project context describes the local system. Skills are task-specific instructions exposed by grants and selected when relevant. Tool schemas and file content enter at runtime. Publishing full prompt text would obscure that architecture and disclose material unnecessary to understanding it.

Context assembly and model-facing state

The model does not receive a raw transcript alone. OWUI assembles the current request from conversation messages, shared and account guidance, selected skill content, tool descriptions, relevant file references, prior tool results, and any compaction summary. These inputs have different origins and different lifetimes. A shared contract is stable across accounts; an account overlay changes the expected interaction with one person; skills are loaded for particular work; a browser page or a retrieved document is untrusted task data. Their roles can be documented without reproducing private prompt text.

The model-facing context is also distinct from the full saved chat. OWUI can retain the complete conversation and execution trace while sending a smaller assembled request after compaction. Compaction preserves a checkpoint summary and enough recent working material for continuation. The selected deployment's threshold is below the model's maximum input allowance because tool schemas, skills, memory, and other material can be added after an early estimate. A prior acceptance exercise exposed a request that exceeded the model window by a small margin; the corrected threshold reserves additional headroom rather than assuming the estimate and final provider tokenization are identical.

The context circle makes the current scale visible to a user. Its numerator is the latest provider request's prompt-token count when available, and its denominator is the configured model window. Cumulative billing tokens across many turns would answer a different question and would make the circle misleading. A new conversation can legitimately show unknown; an estimated value is marked separately from a provider count. This interface does not perform compaction itself. It makes context pressure observable while the backend's compaction path manages the actual request.

Capability discovery and personal learning

A skill is a discoverable capability with instructions for a bounded class of tasks. OWUI can expose shared skills through model registration and grants; private skills remain limited to their owner. Mentioned skills may contribute full content, while other selected skills can be loaded through the skill-view tool. The system's document, diagram, browser, and artifact skills route Penny toward installed tools and delivery paths; they do not grant operating-system access by their wording. Grants and native identity mapping remain the enforcement points.

The personal fork also implements a proposal-driven personal skill-learning loop. After enough qualifying tool-using iterations in an owned saved chat, a post-turn reviewer may propose creating or revising a personal skill. The review runs without blocking the user's response. Its output is staged in an owner-facing proposal conversation, where the owner can approve, reject, or request revision. An ambiguous response applies nothing. Approval claims the exact staged mutation with concurrency and staleness checks, and the resulting personal skill becomes discoverable only to its owner. This design uses agent experience to identify reusable procedures while preserving a human checkpoint before persistent behavior changes.

The deployment disables automatic File Context injection while retaining OWUI's built-in file tools. Penny can deliberately inspect an attachment instead of having every file converted into background RAG context. That choice improves traceability for file-intensive agent work: the tool call, file selected, and result become part of the visible work path.

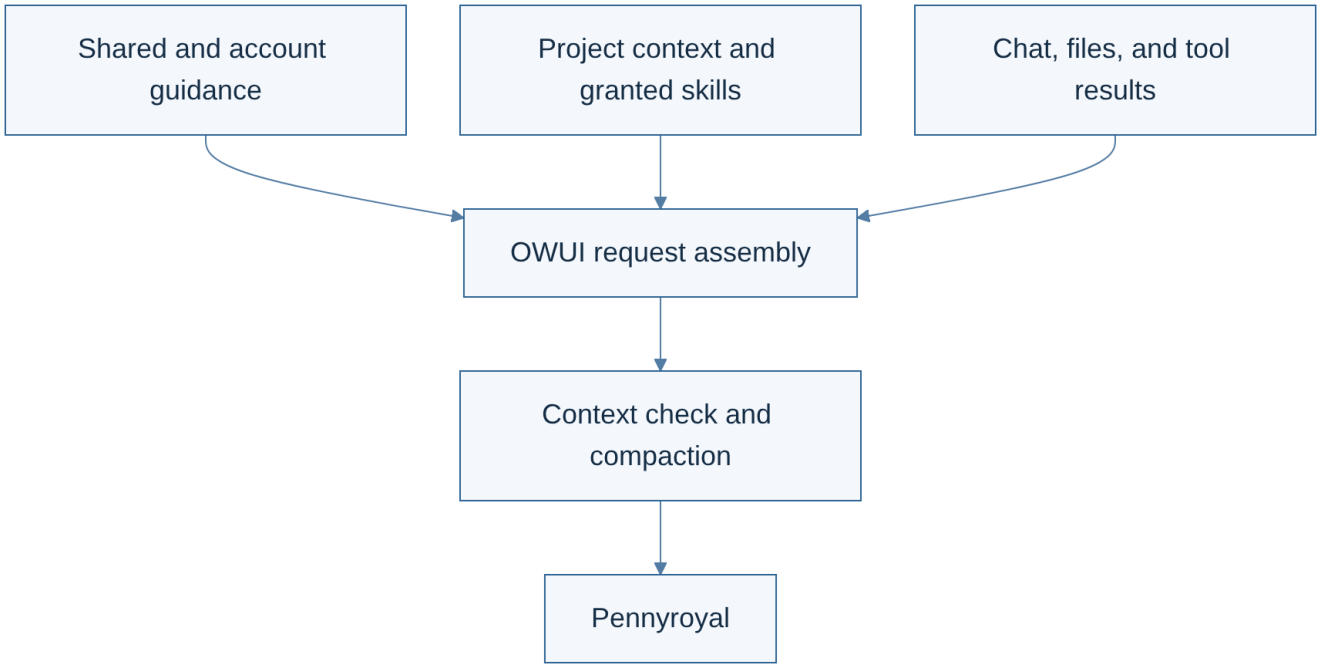

Figure 5. Request context is assembled from layers

The visible context circle reports the latest provider request's prompt usage against the model window. It is an ambient measure of the request that just ran, not a sum of all billed tokens in the conversation. Before there is a provider count, the interface can show an estimate or unknown state. Compaction is a separate threshold and safety mechanism: it condenses older conversation state when a forthcoming request would become too large, including tool continuations that occur within a turn.

The fork also contains a personal skill-learning path. After qualifying tool use in a saved user chat, a background review can propose a new or revised personal skill. It creates an owner-facing proposal; the owner can approve, reject, or request revision. Approval turns the staged proposal into a skill discoverable in that owner's later chats. This retains useful work patterns while keeping persistent behavior under human control.

Context budget example

The current Pennyroyal model window is 524,288 tokens. The deployment reserves a default 65,536-token output allowance and 16,384 tokens of additional headroom for estimate error and context assembled after an early preflight. Its compaction threshold and cap are 442,368 input tokens. The arithmetic is intentional: 442,368 + 65,536 + 16,384 = 524,288. The threshold is a safety decision, not the circle's denominator. The circle still represents the full configured model window and displays the latest request's input consumption against it.

An admission failure motivated the reserve. An early estimate allowed a 458,897-token input, but the requested 65,536 output tokens made the total 524,433—145 over the model window. The revised threshold leaves space for later-assembled material. Context display and admission control therefore use related measurements for different purposes.

Quantity	Current declared value	Meaning
Model window	524,288 tokens	Maximum combined request budget presented by the provider configuration
Default output allowance	65,536 tokens	Space reserved for a model response
Post-assembly headroom	16,384 tokens	Margin for later additions and estimation differences
Compaction threshold/cap	442,368 input tokens	Point at which application summarization protects the next request

Quantity	Current declared value	Meaning
Context circle numerator	Latest provider prompt-token count when available	What the last assembled request actually consumed

The model's KV cache may retain reusable computation for part of the assembled request, but the provider still counts the tokens in the request. Cache reuse changes cost and latency; it does not make those tokens disappear from context accounting.

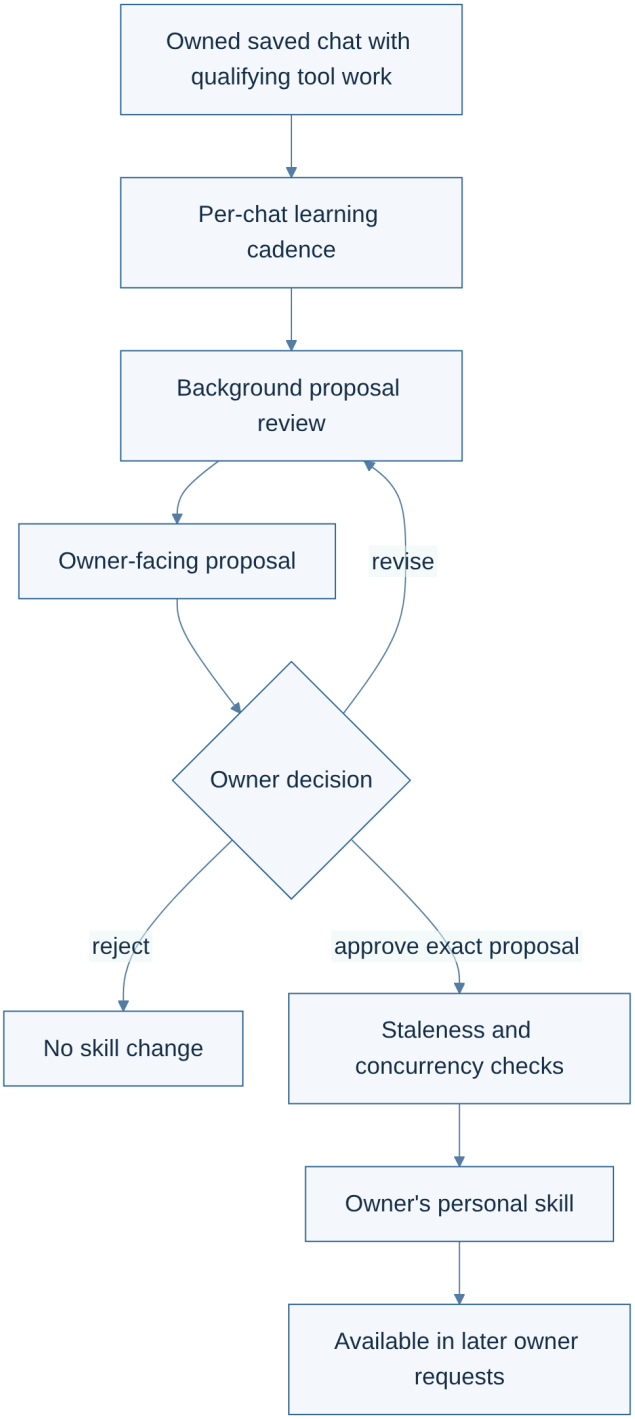

Figure 5a. Human-in-the-loop skill creation

The diagram traces authority and state. The agent drafts a candidate procedure from completed work; the account owner approves any persistent personal skill. Shared skills use a separate central deployment and grant path.

Human-in-the-loop learning as a system capability

The skill proposal loop is more consequential than another memory feature. A normal chat may contain a useful procedure, but that procedure disappears into conversation history unless a person extracts and maintains it. Automatically promoting every apparent success would create a different problem: a model could write lasting

instructions from incomplete evidence, accidental behavior, or untrusted task content. This implementation makes the boundary explicit. The agent may identify and draft a candidate; the account owner controls whether that candidate becomes persistent behavior.

The path begins after an eligible, user-owned saved chat has completed tool-using work. Internal, proposal, and Penny-executor worker chats are excluded from this reviewer path. A per-chat cadence tracks qualifying tool-call iterations, with an explicit user request to learn also able to trigger review. The background reviewer receives bounded conversation context and the owner's existing personal skills so it can propose a new skill or a revision rather than duplicate an existing one. It produces staged proposal data, then an ordinary saved proposal chat in that owner's Skill Creation project. The source chat remains the evidence for the proposed procedure; the proposal chat is the decision surface.

The owner can approve the exact staged text, reject it, or ask for revision. An ambiguous message does not apply a skill change. Approval checks ownership, claims the proposal against concurrent decisions, and checks whether the target skill has changed since the proposal was prepared. An approved personal skill is then available for discovery in that owner's later requests. It does not become a shared capability and does not widen the user's OWUI grants or Linux access. Shared skill changes still follow the centrally managed deployment path.

For example, an artifact workflow can gather source facts, generate an editable diagram, render it, inspect layout, and deliver through an authenticated file route. The agent can propose that sequence as a personal skill. The owner can narrow, revise, reject, or approve it; later chats can discover an approved version under that account.

This is a strong human-in-the-loop pattern because the human checkpoint occurs at a **future-behavior boundary**. A transient answer affects one conversation. A persistent skill can influence many later conversations. The architecture therefore requires explicit owner action for the latter while allowing routine tool use to proceed without constant approval prompts. The result is adaptive capability with accountable persistence.

Human checkpoint	Agent autonomy before the checkpoint	Human authority at the checkpoint
Assignment scope	Codex and Penny can investigate within an agreed objective	The human owner decides goals, material tradeoffs, and accepted boundaries
Blocked execution	Penny can complete independent work and preserve state	The human owner or supervisor answers a decision the worker cannot own
Result review	Penny can implement and run checks	the supervisor agent and human owner inspect evidence and accept or return defects
Skill creation	Background review can stage a candidate procedure	The account owner authorizes the exact persistent skill
External publication or deployment	Agents can prepare reviewable source and artifacts	The human owner authorizes the consequential release step

The table illustrates a broader principle of this build: human involvement is concentrated where authority changes. It enables multi-step agent work without converting the assistant into an unsupervised publisher, administrator, or author of permanent policy.

6. Event-driven multi-agent execution and human review

A normal tool call is contained within an interactive chat turn. Engineering and other substantial tasks need a longer lifecycle: they may run across many turns, wait for input, survive a disconnected supervisor, and return evidence for review. The executor adds that lifecycle to the normal OWUI/Penny environment. It does not create a second model or duplicate the complete conversation.

the supervisor agent and human owner first agree on the objective and boundaries. Codex submits an assignment with a return destination. The executor records a task ID and compact durable state, and Penny works in a saved OWUI chat using the tools granted to its dedicated identity. A task receives its own workspace;

code assignments can use an isolated Git worktree, while non-code assignments do not require a repository. The current configuration permits two active tasks; a full executor returns busy rather than silently queuing more work.

Penny can continue, block on a question, finish ready for review, or stop in another actionable state. A blocked task retains its chat, workspace, question, and completed actions without occupying an active execution slot. When the state warrants attention, a restricted notifier queues a message for the originating Codex task. That notification is an event, not the result payload. Codex reads the ledger and selected evidence, answers or steers as needed, then resumes the same task. Acceptance, publication, and deployment remain explicit human-supervised decisions.

Two levels of agent collaboration

The first level crosses work surfaces. Codex and Penny are distinct agents with different model providers, tools, state, and responsibilities. Codex is the supervisor agent: it formulates a bounded assignment and later examines results with the human owner. Penny executes the assignment through the Open WebUI harness and local Pennyroyal inference. The executor provides stable task identity, retained state, result access, and return notification between them.

The second level occurs within Penny's work. Open WebUI supports native, bounded subagent delegation for separable parts of a task. Penny can ask a helper to perform an independent investigation or analysis, continue its own useful work, and incorporate the returned result. The parent Penny task still owns the final answer and must account for the helper's work. A subagent result does not bypass the executor's review state or grant a new publication authority. The runtime also supports asynchronous helpers, but a helper that outlives one turn must have its result collected before the parent declares ready for review.

The multi-agent architecture includes supervised delegation across Codex and Penny, bounded delegation inside Penny's OWUI execution, and independent low-context agents in the source-review workflow described in Chapter 11. Each delegation retains a parent agent and a human decision owner.

Task record, execution record, and supervisor event

The executor keeps a compact task record: assignment, task ID, state, progress, result, and references to evidence. The saved OWUI chat carries the full sequence of model turns and tool output. These records are complementary. A supervisor can find a task's current state quickly without treating the ledger as a replacement transcript, and can inspect the detailed execution when a result needs review.

The executor creates a stable workspace for every assignment. Repository and base revision are optional. A code assignment can prepare a Git worktree when the repository mapping and revision are available; a research or document assignment can start in a workspace without a checkout. That generality matters because durable execution is a coordination mechanism, not a special case of Git automation. Exact unpublished source held in another worktree is not transferred by naming a repository; it must be made available through an explicit source handoff.

The current configuration allows two active tasks. A third submission receives a busy response when both slots are occupied. Blocked tasks, completed tasks awaiting review, and interrupted tasks retain state without reserving an active slot. Each task has independent runner state, chat, workspace, and supervisor target. This is bounded concurrency with visible backpressure, not an unbounded hidden queue.

State or checkpoint	Meaning for the supervisor	Execution capacity
Continue	Penny has more independent work; the same assignment remains active	Occupies an active slot while running
Blocked	A precise user/supervisor input is needed; chat and work are retained	Does not occupy an active slot while waiting
Ready for review	Penny reports implementation or analysis complete with evidence references	Does not occupy an active slot

State or checkpoint	Meaning for the supervisor	Execution capacity
Interrupted	Work stopped with retained state requiring reconciliation before resumption	Does not occupy an active slot
Failed or cancelled	Closed outcome requiring diagnosis or a new decision	No active slot

The notification path is deliberately narrow. On an actionable transition, room101's sender validates the task ID, status, and originating task target. Neomatrix and thegrid use a restricted SSH forced command to queue the fixed notification to that Codex task; room101-originated tasks use a local queue call. The notification can wait while the originating Codex task is busy; it does not inject a result into a running analysis or grant shell access. Codex reads the ledger and, when needed, detailed evidence through the executor interface. Uncertain notification delivery is recorded separately and is not grounds for blindly repeating side effects.

A blocked native question is resumed in the same saved chat and workspace. The answer is submitted as steering, then the task resumes through OWUI's existing continuation path. This preserves tool-call identity and completed effects; the agent does not start over. Review feedback follows the same principle: defects return to the existing Penny task rather than silently creating a replacement task on a different checkout.

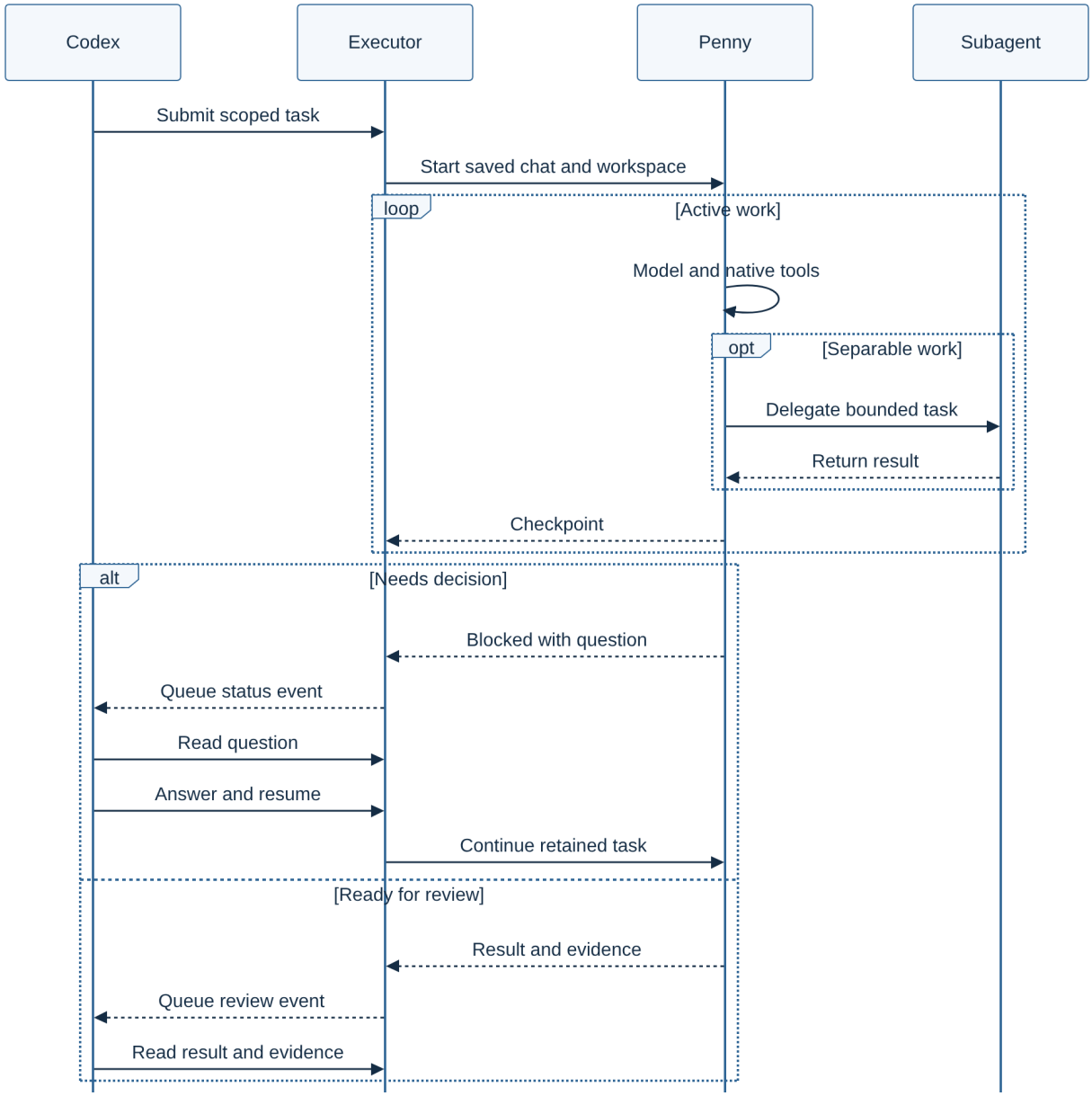

Figure 6. Event-driven multi-agent assignment and return

This design separates the event from the source of truth. The notification wakes the originating supervisor task; the ledger provides current status and result references; the saved chat retains the execution trace. Dated live acceptance exercised pause, answer, resume, and return to review. Broader failure recovery is assessed separately.

7. Identity, authorization, and native execution

OWUI capability groups grant access to models, research, image work, document tools, and agent workspaces independently. Only people who need native workspace execution receive Linux accounts and native Open Terminal services. A chat- or image-only account need not have a shell. The identity router maps an authenticated, allowed OWUI identity to that person's terminal and rejects missing or unknown identities.

Shared binaries and skills live in a centrally managed capability tree. User-written files, browser profiles, caches, and experiments live under that user's account. the platform owner's private presentation assets are in a

separate private capability tree. This makes it possible to share a document renderer or browser installation while preserving separate writable state and private material.

Two authorization layers

OWUI grants answer *which capabilities may this account invoke?* Linux identities answer *whose processes and files are used when the capability runs?* The system needs both. An image-only user can invoke an allowed image tool without receiving a native terminal. A workspace-enabled user has an account-specific native terminal and a mapped home directory. The identity router checks the authenticated OWUI identity and an allowlist before proxying the Open Terminal interface; an absent or unknown identity fails closed.

Capabilities are additive. Global access stays narrow, and groups supply model, research, image, workspace, document, or private resources. This avoids a separate model copy for every person's combination of permissions. It also keeps personal interface preferences under each user's control: deployment reconciliation manages shared invariants and selected prompts/grants, rather than treating every user-owned setting as drift.

Boundary	Mechanism	Architectural purpose
OWUI account to capability	Authenticated session plus group/resource grants	Limits visible models, skills, tools, and administrative functions
OWUI account to terminal	Identity router with explicit user mapping	Prevents a missing identity from reaching a default or another person's terminal
Terminal to filesystem	Native Linux UID and per-user home	Keeps writable work and browser state separate
Terminal to shared tools	Root-managed capability tree with read/execute group access	Shares runtimes without granting users control of installations
Private material	OWUI grants plus filesystem and mount-namespace controls	Keeps private documentation and brand assets out of friend workspaces
Browser file delivery	OWUI session and terminal/file ownership checks	Avoids public artifact URLs for personal results

The shared browser illustrates the difference between installation and state. The agent-browser binary and matching skill are installed once. Browser processes, profiles, sockets, downloads, and screenshots live under the invoking user's home. The default persistent browser session lets one user's task continue across tool calls, while concurrent browser jobs under that same account require distinct sessions. A web page's contents and metadata are treated as input to evaluate, not instructions with authority over the agent.

Thegrid's shared NFS artifact mount maps client identities to one server-side identity. Native terminal services therefore use filesystem and mount-namespace boundaries to separate user-private paths. This is a specific correction to a real infrastructure constraint, not a generic claim that NFS permissions enforce per-user privacy in this deployment.

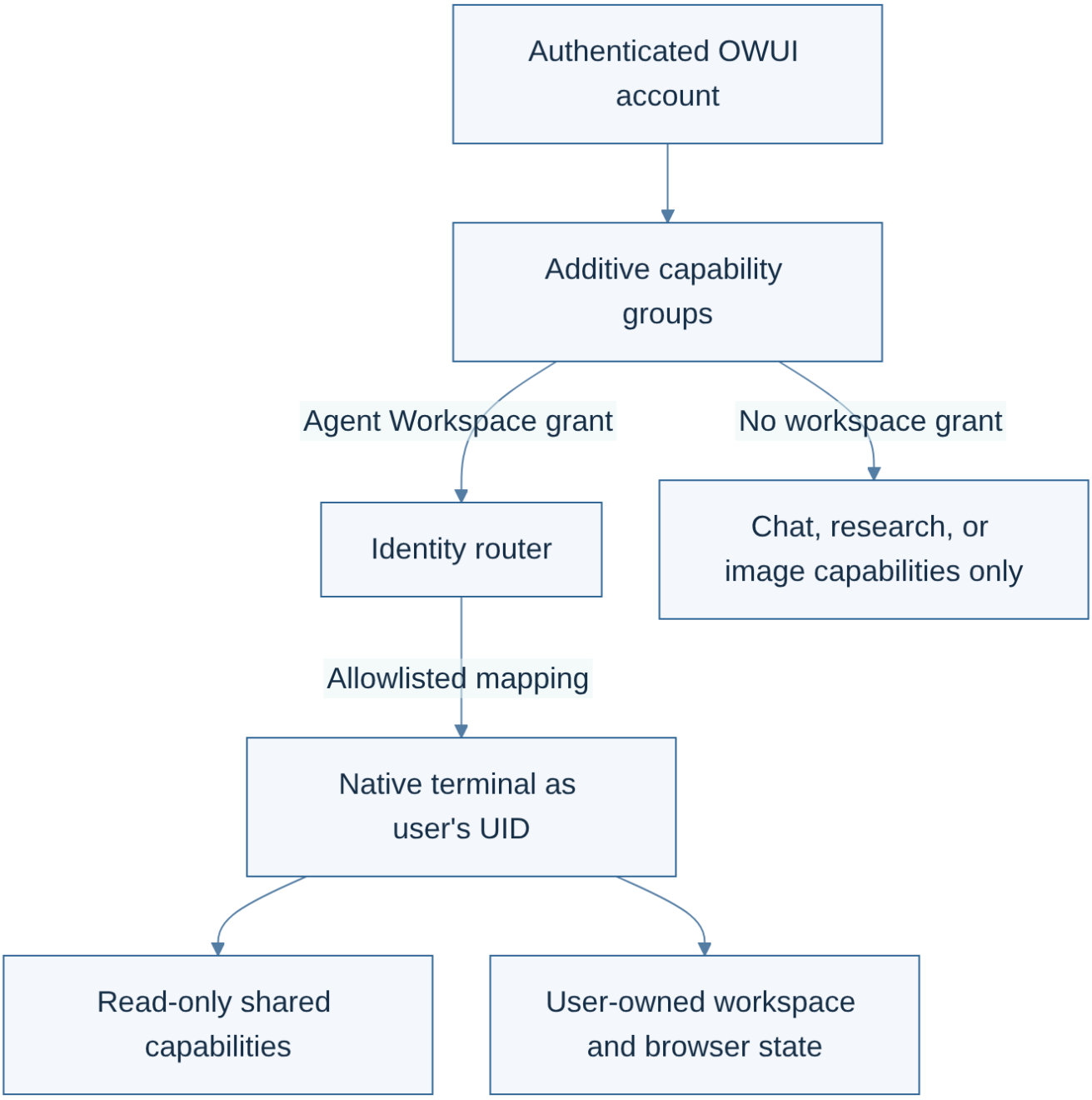

Figure 7. From grant to operating-system identity

This small trusted-group design uses OWUI ownership checks and Unix ownership. Shared NFS artifact storage uses squashed identity, so native terminals hide protected paths through service mount namespaces; server-side mode bits alone cannot separate those paths.

8. Agent Library and authenticated file delivery

A file may have three lives in this system. Open WebUI keeps a display copy associated with a chat. Agent Library keeps a durable original in S3-compatible Versity storage under the user's private or shared identity. A native workspace holds a working copy for tools to inspect or modify. These copies differ in ownership and lifetime, so the document treats them separately.

Copy	Purpose	Lifetime and access
OWUI file	Display and chat attachment	Tied to OWUI's authenticated file handling and chat experience.
Agent Library object	Durable original or deliberate published result	Stored under the mapped user's private or shared S3 permissions; independent of a working chat.
Native workspace file	Tool input, intermediate, or finished working copy	Owned by the Linux user; can be removed or regenerated from the library when appropriate.

The identity router uses the mapped user's library identity for private and shared objects; a missing mapping does not fall back to another account. Renaming changes display metadata while retaining the same object identity and bytes. Neither behavior is a substitute for application authentication at download time.

An OWUI upload hook archives files handled by its upload path, including native MCP image responses. When a chat attachment goes through the terminal router, the router archives it before forwarding a workspace copy. Library tools let Penny list, fetch, publish, and rename objects within the mapped user's permissions. A new published result receives a new identity rather than overwriting the earlier object. The library is not a replacement frontend; the user normally remains in OWUI.

Object identity, permissions, and copies

Versity provides the S3-compatible API and object storage. The integration maps OWUI users to private buckets and a shared bucket. The router uses credentials for the mapped regular users and does not use the storage administrator identity for ordinary agent work. Bucket ownership and policy implement the library access model; the application's authenticated identity and router mapping determine which storage identity a given request receives.

Library keys are UUID-prefixed so two uploads called image.png do not collide. The original filename and a display name can remain metadata. Renaming the display name updates metadata on the same key and preserves the object identity and bytes. A new output publish assigns a new UUID rather than replacing an older original. These mechanics are mundane but important: agent work often produces iterations, and overwriting an input or losing the distinction between a filename and object identity would make later retrieval unreliable.

Two ingress paths serve different user actions. OWUI's upload hook archives content handled by its file upload route and preserves OWUI's own file ID for display. A terminal attachment passes through the identity router, which archives it before placing the UUID-prefixed working copy in the user's workspace. Both paths avoid forcing a user to manage a separate S3 console for ordinary work. The standalone storage Explorer remains an administrative or direct-storage interface, not the normal OWUI file-delivery path.

A library fetch returns a working copy that tools can read or edit. A publish call records a deliberate output in the library. Fetch and publish can return an OWUI-relative download route, which Penny can present without inventing a URL. OWUI and its terminal proxy still authenticate the browser request. This is why the location of the working copy matters: removing a copy that backs a current terminal download link breaks that link, even though the library original remains available for another fetch.

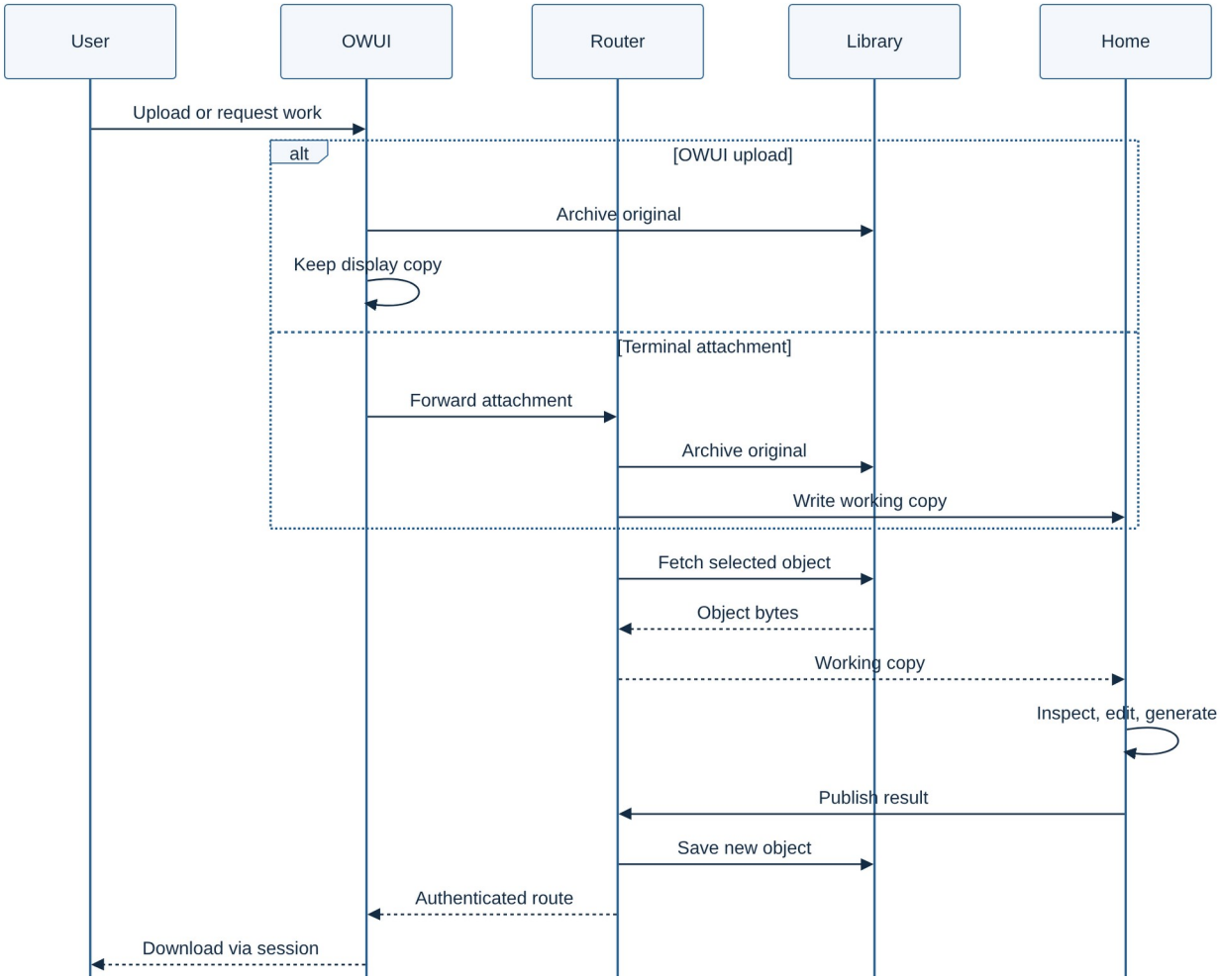

Figure 8. File lifecycle and delivery

The browser does not need direct S3 access. Library-only results can be fetched into the user's workspace and delivered through the authenticated terminal file route. Files already owned by OWUI use its authenticated file route. A live download link depends on the working copy remaining present; the durable library object can be fetched again if needed.

Image editing has a different internal input path: a selected library object can receive a short-lived signed URL for ComfyUI to read into memory. That URL is for service-to-service processing and is not the user-facing download link. The source integration records a one-hour validity period. The distinction can be explained without publishing a signed example URL.

This architecture makes a useful separation: the original survives beyond an individual chat, tools get disposable working copies, and people receive files through their authenticated application. Backup and restore of all library state are a separate operational question; the existence of S3-compatible storage alone does not prove a tested restore.

Failure behavior and retention boundary

Archival is part of the declared upload contract. A storage failure fails the upload visibly rather than returning success while silently losing its durable copy. The router writes transfer data through private temporary files and cleans them up after use. If archival succeeds but forwarding to a terminal fails, it attempts to remove the new object. That compensation is best effort. There is no distributed transaction across OWUI, the router, the native terminal, and S3; an interrupted request can leave an object that an operator must identify. This is a narrower, more honest guarantee than claiming every cross-service step is atomic.

The integration's configured upload limit is 512 MiB. The router spools its transfer to private temporary storage, but upstream OWUI proxy and Open Terminal paths still buffer some uploads in memory. This affects capacity planning for very large files and prevents the library chapter from implying a fully streaming, arbitrary-size pipeline. It does not change the normal object identity or authenticated download model.

Durable recovery requires Versity object files, metadata, IAM state, and application mappings together. This storage domain is separate from OWUI's chat database. Archival, retrieval, cross-user denial, and byte-exact downloads were exercised; a full library restore is a separate recovery procedure.

Authenticated download as an architecture boundary

A browser download has a different trust relationship from a service-to-service signed input. The former should be checked against the logged-in OWUI user and the selected file route. The latter is a temporary capability used by a processor such as ComfyUI to read a specific object. The generated artifact can be archived and then offered through OWUI. Keeping these routes separate means a workflow can use S3-compatible storage internally without making object storage public or teaching an agent to expose signed URLs as final results.

Dated multi-user acceptance exercised these boundaries directly: duplicate filenames remained independent, OWUI and fetched-library downloads matched original bytes, one user's private object was denied to the other, and unauthenticated download attempts failed. Those tests validate the selected paths at the time they ran; they do not replace review when the application fork or storage integration changes.

9. Connected processing services

The main agent path can select connected research tools, an interactive browser, image services, speech, and document tools. Their contracts are intentionally different.

Research services return information for Penny to evaluate and cite. The interactive browser is a stateful Chromium session running through a user's native terminal, with that user's profile and downloads. It can continue across tool calls; page content remains untrusted input. The distinction matters because finding a page and filling a form are different activities with different state and consequences.

For images, AI Tools owns the named MCP contract and invokes tracked ComfyUI workflows on Neomatrix. ComfyUI performs generation or editing; image bytes return to the harness for archival and authenticated display. Agent Library supplies selected edit inputs through temporary signed URLs rather than a permanent shared input directory.

For speech, OWUI owns microphone upload, transcription display, read-aloud interaction, and personal voice selection. Dubline on Neomatrix owns local decoding, speech inference, voice references, and GPU serialization. The current integration uses MOSS transcription and IndexTTS output. Speech and ComfyUI share workstation GPU resources; this build does not claim a general scheduler for them.

Document tools run in the native user's workspace using shared installations and, when granted, private assets. That places generated files within the same identity and delivery scheme as other agent work.

Connected research and interactive browsing

Research is available in the normal Pennyroyal experience through centrally managed MCP connections. The Research grant controls access to those connections. Native OWUI web search is disabled in this deployment, avoiding two competing search paths with different tool selection and citation behavior. A research result is input for Penny to evaluate; neither the search provider nor a retrieved page owns the conversation's instructions.

Browser Research and Interactive Browser solve different tasks. Browser Research selects, retrieves, and extracts information from the web. Interactive Browser operates a persistent Chromium session under the user's native Linux identity. It can click, enter text, navigate tabs, wait for page state, and handle downloads across terminal calls. The agent reads the installed browser skill and uses the browser's accessibility snapshot and

element references rather than assuming page coordinates or inventing selectors. Its profile and socket are isolated by the user's home, and page content remains untrusted even when it appears inside a tool result.

This distinction is relevant to enterprise workflow design. Research can inform a decision; browser interaction can change external state. The latter therefore needs an explicit user or supervisor checkpoint for credentials, CAPTCHA, or consequential external actions. The platform's architecture makes that difference visible through capability grants and tool contracts.

Image generation, image editing, and result presentation

AI Tools is the named image-service interface presented to the agent. It owns MCP tool contracts and the tracked workflows sent to ComfyUI on Neomatrix. ComfyUI owns model execution and rendering. Image bytes return through the service path and are handled by OWUI as native image content, which gives the application a chance to archive the output and serve an authenticated display URL.

An edit adds a source-selection problem. Penny can identify one to three Agent Library objects under the user's library access, request short-lived source URLs, and pass them to AI Tools in the requested order. ComfyUI's URL input node loads the images into memory for the edit graph. The signed URL is a transport credential with a short lifetime; it does not become a permanent artifact or a browser download link. The final PNG returns to OWUI, where the ordinary archive and display path applies.

The application fork's native MCP image-reference handling matters at this boundary. A tool can return image content without first publishing a public URL. OWUI persists the display copy and supplies an authenticated reference. Model-facing guidance lets Penny embed that reference in a response while the server enforces access control. Generated-image and reference checks exercised this path.

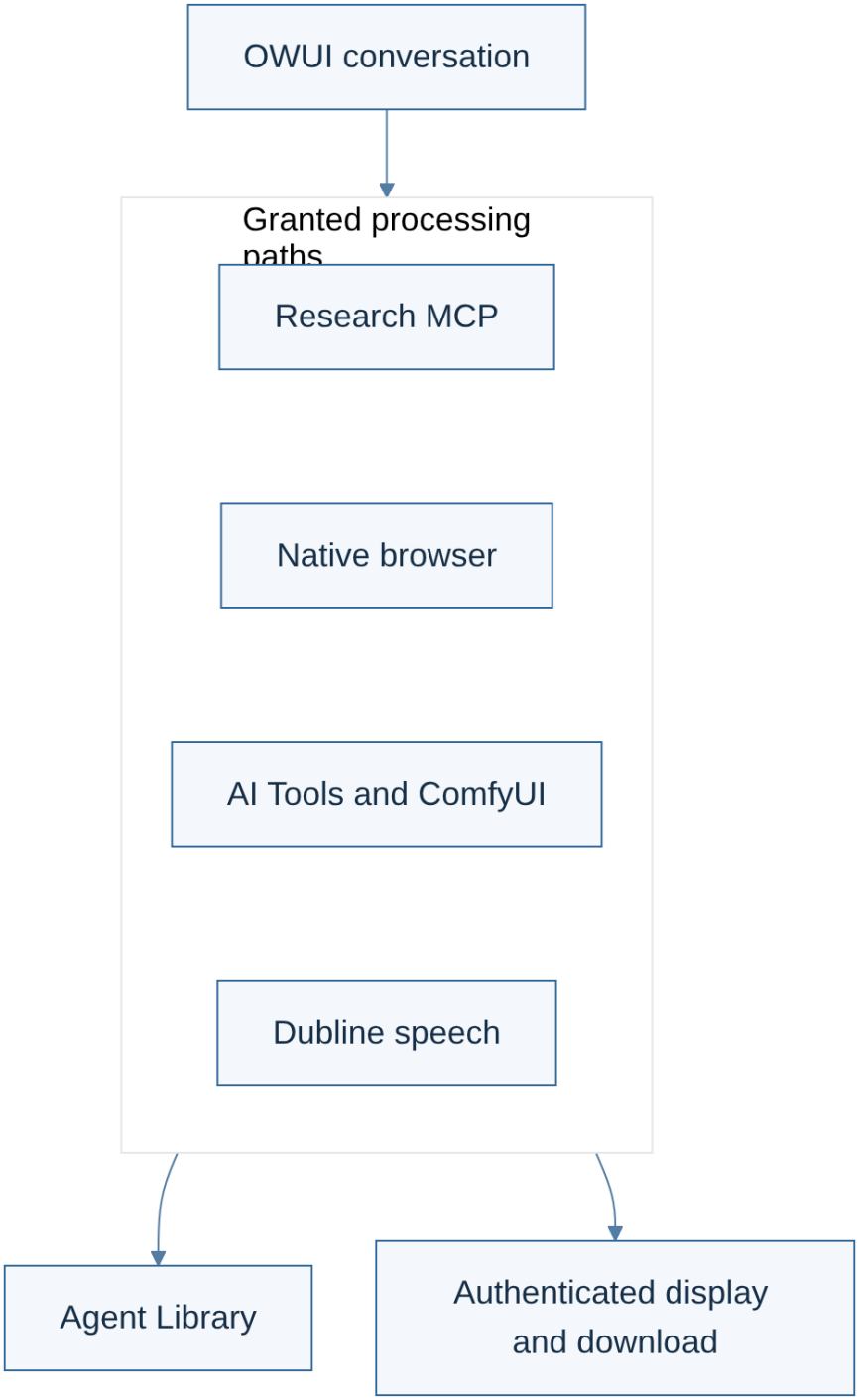

Figure 9. Specialized processing rejoins the authenticated user path

Image editing across the storage and processing boundary

Image editing demonstrates why the three-copy file model is useful. The source image can be retained as an Agent Library object even after its original chat is closed. Penny selects the specific object under the user's library identity and requests a short-lived internal URL. AI Tools submits that URL to a tracked ComfyUI workflow. ComfyUI reads the source bytes into memory, renders the edit, and returns the resulting PNG through AI Tools as native MCP image content. OWUI creates an authenticated display file and the archival hook records the result. If the user decides the revision is worth retaining as a named library result, a publish operation records another object without overwriting the source.

The separation keeps each actor's responsibility narrow. Agent Library controls retained objects and per-user S3 access. AI Tools owns the image-tool contract and workflow selection. ComfyUI performs GPU rendering. OWUI owns the logged-in user's display and download. The signed input URL is not displayed as a permanent artifact link, and ComfyUI does not become a general-purpose object browser.

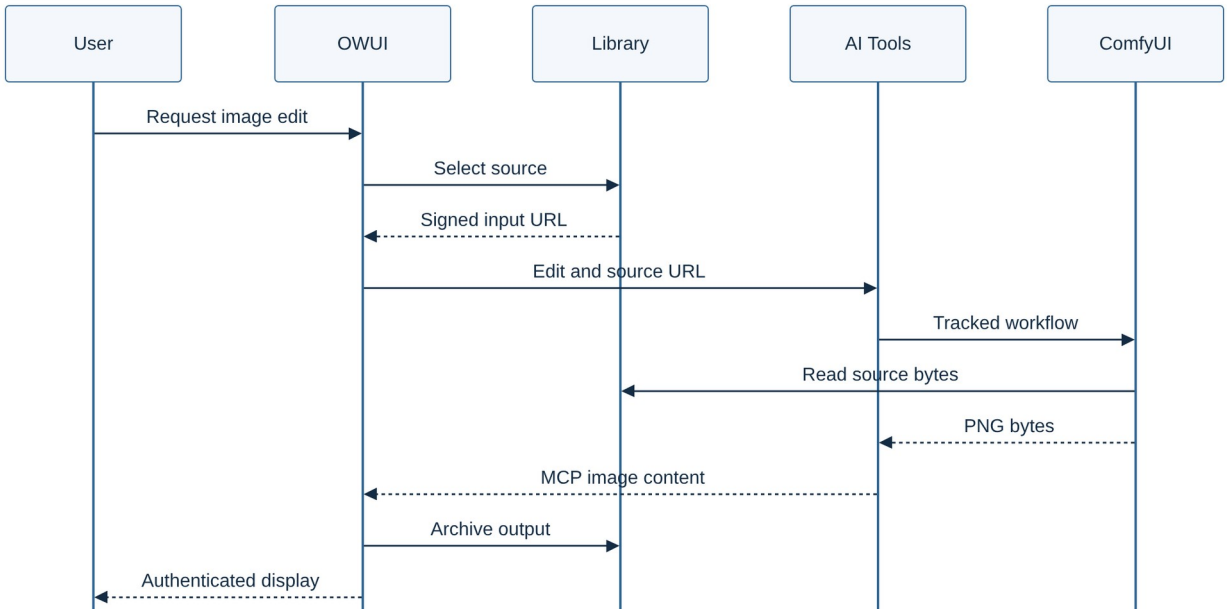

Figure 9a. Image-edit sequence

Speech input and playback

OWUI is the browser-facing speech client. It receives microphone recordings, shows transcript text, initiates read-aloud, and retains each person's voice selection. Dubline on Neomatrix provides the local speech API. Its service owns decoding, MOSS transcription, IndexTTS synthesis, voice references, model lifecycle, GPU serialization, and request cleanup. The browser does not use a separate Dubline UI.

The input contract matters. OWUI's audio bypass passes the original uploaded recording through once, allowing the backend to maintain its decoding and speaker context. Without that bypass, OWUI's normal preprocessing can convert or split a large upload before it reaches the speech model. The same bypass affects output processing, so the TTS provider is configured to request MP3 and the backend returns actual MPEG audio. Input and output behavior were therefore qualified together.

Read-aloud splits text by paragraph to stay within Dubline's per-request limit. A single paragraph longer than that limit can fail explicitly; the system does not claim unlimited synthesis by virtue of splitting. Voice choice is user-owned in OWUI. A model-level voice override would take precedence and would undermine that personalization. Browser Voice Mode was accepted as a practical user flow, while individual voice quality remains a subjective choice.

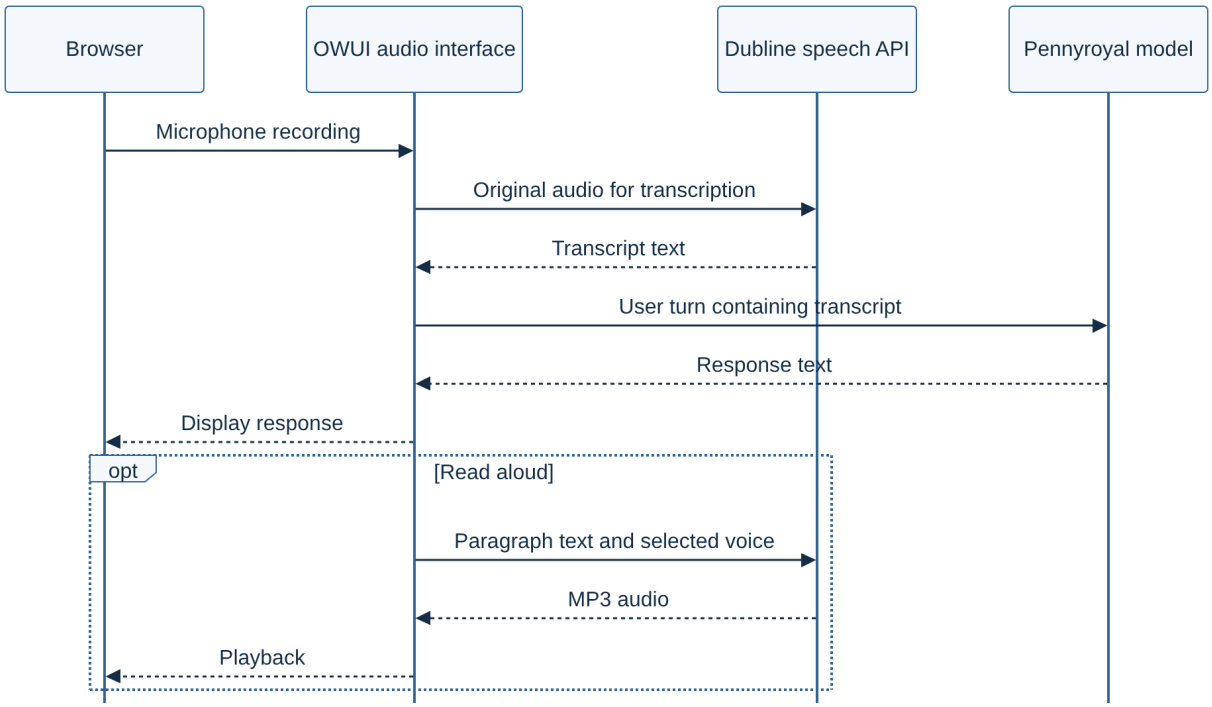

Figure 9b. Speech ownership and return path

A successful speech backend response verifies only part of the user path. Transcription must reach the chat, Penny must answer, and playback must reach the browser for the full interaction to work. The same principle applies to image output: a successful ComfyUI render does not by itself verify OWUI persistence or authenticated display.

Documents and editable diagrams

Document creation runs through the native terminal and shared capability installation. The shared tree contains document, spreadsheet, PDF, OCR, presentation, and diagram tools. A granted user can write and modify work in their own home while the shared installation remains read-only to them. This provides a consistent route for input inspection, generation, rendering, validation, and authenticated delivery.

The architecture values editable outputs. A presentation remains a PPTX with source material and rendered pages for review. A hardware or network diagram remains a draw.io file with reusable shapes and connections rather than only a flat image. Catalog-backed assets improve the physical specificity of a diagram, but they do not validate an invented topology. Technical source facts and visual layout still require review.

Private brand capabilities overlay the shared document stack. The deck owner's HPE assets, registrations, and licensed fonts remain in a separate private capability tree with corresponding OWUI grants. Application authorization and filesystem placement both protect them.

10. Representative workflows

The following walkthroughs trace four different outcomes through the same architecture: supervised engineering, file-backed image editing, editable presentation production, and long-context continuation. Each identifies the initiating authority, the services and state involved, the returned artifact, and the acceptance boundary.

Workflow A — Human-in-the-loop engineering assignment

Assignment contract. The human owner and supervisor agent establish the problem, source state, boundaries, and acceptance evidence. Codex submits the bounded assignment to the executor with an originating task

destination. The executor records the task and prepares a retained workspace. If a configured repository and resolvable base revision are present, it can prepare an isolated worktree. General assignments can start without a Git checkout. An exact description of unpublished source held elsewhere does not transfer that source; the supervisor must arrange a real handoff.

Execution path. Penny Engineering runs in a normal saved OWUI chat with its own non-admin identity and native terminal. It can inspect source, run commands, use granted tools, and ask focused questions. The model/tool turns remain OWUI behavior. The executor owns lifecycle, capacity, progress checkpoints, and compact result recording. A command that continues beyond one terminal response must have its result collected or its running state made explicit before Penny declares completion. An asynchronous helper can proceed while Penny does independent work, but the helper's result must be incorporated into the final checkpoint.

Pause and return. When a human decision is needed, a native question moves the task to blocked. The saved chat, pending tool-call identity, workspace, and completed actions remain intact. The notifier queues an event to the originating Codex task. Codex reads the exact question through the executor interface, supplies an answer as steering, and resumes the same assignment. Waiting does not occupy one of the two active slots. A dated live acceptance exercised question, answer, and continuation in the same chat and invocation, including a file effect performed once rather than duplicated after resume.

Evidence and authority. At ready for review, Penny returns a compact result with evidence references. Codex inspects the actual artifact or diff, test outcomes, and relevant logs or chat excerpts before recommending acceptance. Defects return as steering on the same task and checkout. A ready status does not grant publication or deployment authority. The source repository, deployment repository, and live application remain separately controlled.

This workflow supplies a concrete event-driven-agent pattern: a durable assignment, bounded execution, an actionable transition event, a supervisor that reads current state, and a human-governed outcome. The notification is intentionally small. It wakes the reviewer at the right time; it does not pretend to be the audit record.

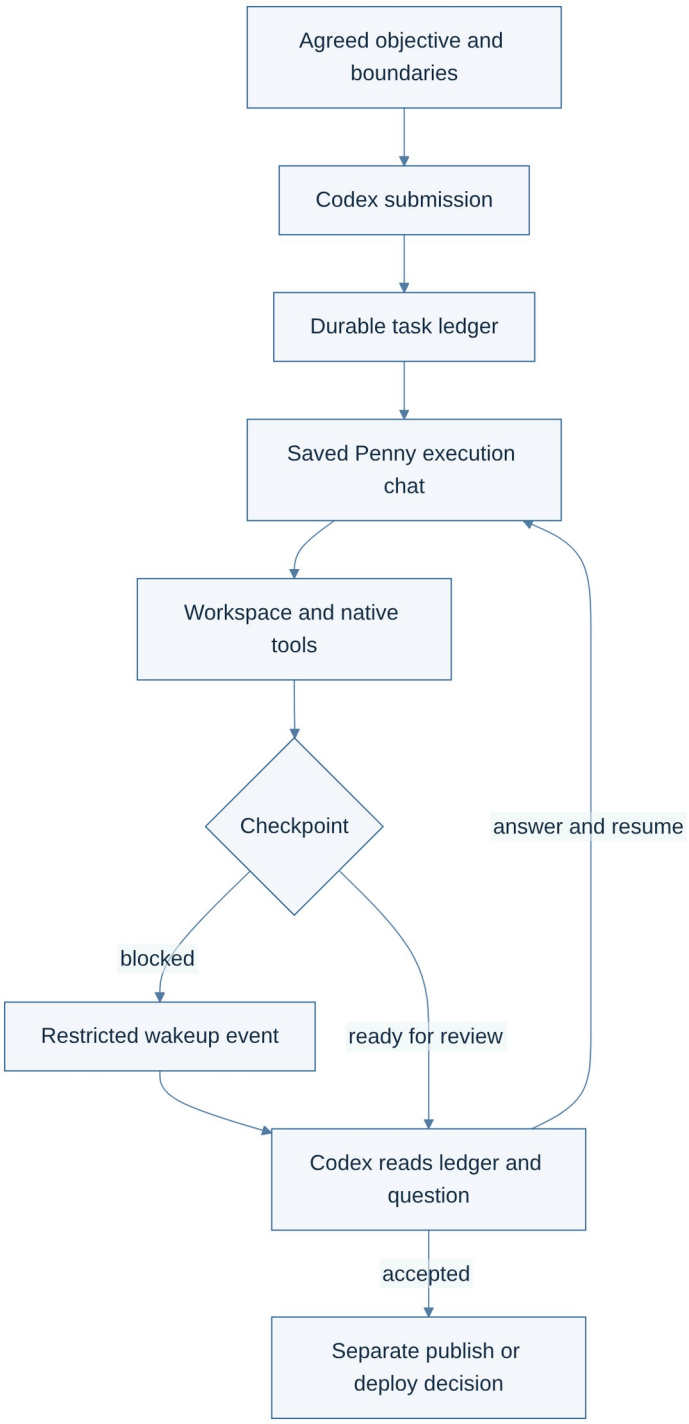

Figure 10. Engineering assignment control and evidence

Workflow B — A retained image used in a later edit

A user first supplies an image to OWUI. The application maintains the authenticated chat display copy, while the integration archives the original bytes in Agent Library with a unique key and metadata. This prevents a later file called image.png from overwriting the earlier image. The durable object remains useful after the source conversation is no longer the active workspace.

In a later chat, Penny lists or searches the user's allowed library space and fetches the exact source. The terminal receives a working copy under the user's Linux identity. A display-name change can make the object easier to locate without changing its stable ID or bytes. If the task is an image edit, Penny requests a short-lived

source URL and supplies it to AI Tools in an explicit input order. ComfyUI reads the source into the edit graph and returns PNG bytes through AI Tools. OWUI archives and displays the new image. A deliberate publish operation can retain the final result as another library object, leaving the input unchanged.

The download path is independent of the S3 processing URL. OWUI can serve its own file object through its authenticated file API. A library-only object can be fetched into the native workspace and presented through the authenticated terminal file route. If that workspace copy is removed, its existing terminal link may stop working; the library object can still be fetched again. That behavior is visible in the architecture and preferable to treating a temporary signed storage URL as a permanent deliverable.

The same pattern applies to non-image files: preserve an original, work on a copy, publish deliberate results with distinct identity, and let the application mediate user delivery.

Workflow C — An editable HPE presentation

The HPE deck path demonstrates a private, user-specific capability built on a shared toolchain. It is included as a compact business case, not as the subject of the platform.

1. **Brief and authority.** The deck owner supplies the audience, argument, approved facts, and source material. The HPE presentation capability is granted only to the authorized OWUI identity. Its brand overlay and licensed assets live in a private filesystem tree; generic presentation tools are shared.
2. **Creation.** The agent uses the private capability within the owner's native workspace. The toolchain creates editable slide content and a PPTX, rather than a slide-shaped raster image. Source material and intermediate files remain available for revision.
3. **Review.** The deck is rendered to pages and inspected for hierarchy, font use, legibility, overflow, visual consistency, and fidelity to the brief. The deck owner reviews technical and customer-specific claims. Large prose changes are made in the editable source and regenerated, preserving layout quality.
4. **Revision and delivery.** The owner requests changes through the normal conversation. The agent revises, rerenders, and returns the deck through authenticated file delivery. A deliberate library publish can preserve a result for later work.

A representative HPE deck passed practical quality review during OWUI acceptance. It demonstrates a useful business workflow: private brand grants, shared presentation tools, native workspace ownership, editable PPTX output, render-and-review, and authenticated delivery. Customer facts remain subject to the deck owner's approval.

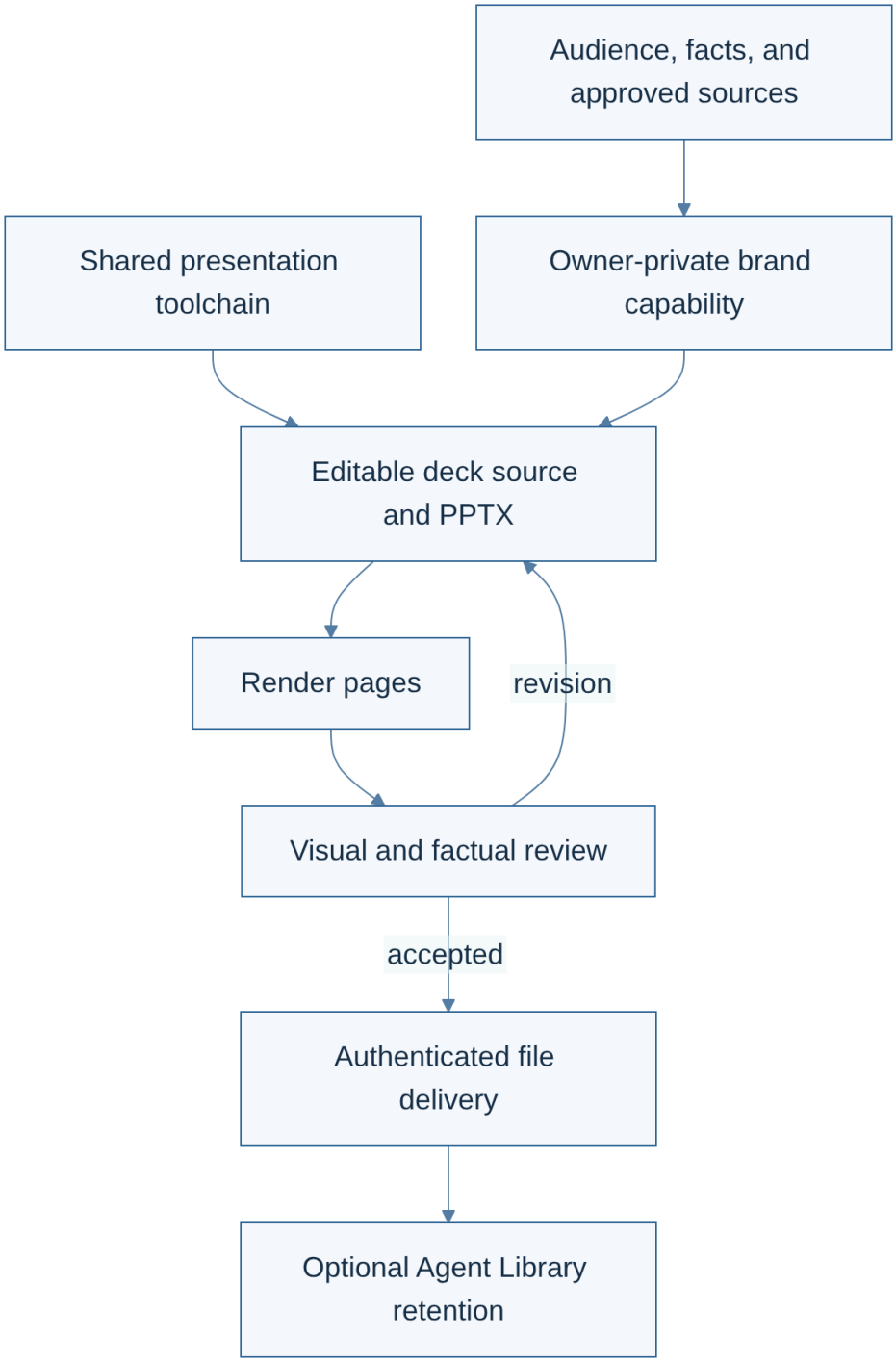

Figure 11. Presentation artifact pipeline

Workflow D — Long-context continuation

A long agent session repeatedly presents shared background, recent messages, and new tool results to the model. OWUI measures the assembled request against the configured model window. The context circle displays the latest provider prompt count when available. Backend compaction can replace older model-facing turns with a checkpoint summary when a forthcoming request approaches its safety threshold, while the complete saved chat remains available for inspection.

Pennyroyal addresses a different cost. If the assembled prefix is compatible with cached state, the runtime can reuse GPU pages or restore older state from HiCache/NIXL and prefill only the new tail. Application compaction

decides what information enters the next prompt. Prefix caching decides how much computation required for that prompt can be reused. One mechanism does not replace the other.

The Flash-Next restart check restored 489,856 tokens from a 489,879-token input. For a long-running engineering agent, compatible prefix reuse avoids recomputing most shared context as new tool results and decisions accumulate.

11. Design decisions and constraints

The components are independently useful, but the design decisions at their boundaries are what turn them into a working system. Several choices favored explicit control and ordinary operating-system behavior over a larger platform abstraction. They should be evaluated against the actual workload: a few trusted users, long-context local inference, substantial agent work, and files that must remain available across sessions.

Decisions and consequences

Decision	Reason in this workload	Consequence and limit
Operator-controlled AI data plane	Model prompts, generated outputs, conversations, media, and files should stay on owned infrastructure	Penny-only work uses local inference and storage while connected research supplies outside facts
One shared local inference service	Several clients and agent paths need the same model without duplicating the GPU runtime	Clients share model capacity but retain separate chats, prompts, permissions, and application state
Personal Open WebUI fork	Agent continuation, executor integration, context indication, and file behavior extend upstream application behavior	Source updates require explicit review, integration, and deployment
Native per-user terminals	Users need normal tools and writable homes under their own identities	Operating-system accounts are required only for workspace users; shared capability installation remains protected
Explicit identity router	OWUI's authenticated identity must reach the correct terminal	Missing or unknown identities fail closed; router becomes a narrow trusted mapping service
Separate task ledger and saved chat	Supervisors need quick state readback without losing full execution evidence	Ledger holds compact state; OWUI chat remains the detailed work record
Restricted status notification	Supervisors should wake on actionable transitions without polling continuously	Notification carries only validated status and task identity; result inspection follows through the executor
Bounded active execution	A local system cannot safely treat agent demand as infinite	Two active slots are configured; a full executor returns busy instead of building an implicit queue
S3-compatible Agent Library	Inputs and useful outputs must outlive an individual chat or working directory	Originals require storage identity, metadata, and a recovery strategy separate from transient copies
Authenticated artifact delivery	Multi-user results need owner checks at the browser boundary	User links go through OWUI or its terminal proxy rather than a public file server
Deliberate file inspection	Automatic File Context injection obscured what Penny actually read	Built-in file tools remain available while automatic RAG ingestion stays off
Human-approved skill learning	Useful procedures can emerge from repeated tool work	Candidate skills remain proposals until the account owner approves the exact change
Dedicated image and speech processors	Different models and GPUs serve different media tasks	Cross-host calls and shared workstation GPU use create visible dependencies

Together, these decisions make the workload tractable for a small trusted group: shared compute and tools, separate user state, durable work, and explicit human authority at consequential transitions.

Source ownership and evolution

The build has more than one source authority. The public SGLang-derived repository owns the Pennyroyal inference runtime, qualified recipes, and its benchmark evidence. The private Open WebUI fork owns downstream application source, including the executor and context/skill mechanisms. The deployment repository owns desired configuration, account/resource grants, identity routing, service definitions, shared capabilities, and the acceptance path. AI Tools and ComfyUI own their image contract and rendering internals; Dubline owns speech inference and voice catalog behavior.

This separation keeps durable source authority outside the live container and checkout. A reviewed application update is selected as a specific source revision, built, checked, and deployed explicitly. Open Terminal follows its separate latest-release policy as a shared native tool. OWUI deployment faults are repaired in place without retaining a parallel old installation.

Bounded multi-agent review of substantive changes

The engineering review workflow applies to substantive coding and serious behavior changes across the system, not only the personal Open WebUI fork. It is another multi-agent pattern with a different purpose from runtime delegation. Runtime subagents help complete a user's assignment. Review agents try to find defects and challenge assumptions before a change is accepted.

When Penny implements, Codex performs the first independent supervisory review of the actual diff, relevant source, and evidence. A fresh low-context reviewer is added when risk, uncertainty, or findings justify another perspective. The reviewer receives the intended behavior, constraints, focused source and tests, and a request for concrete defects. It does not inherit the implementation conversation or earlier review conclusions. That limited context reduces anchoring while preserving the information needed for a meaningful challenge.

Findings return to the same implementation task and checkout. The changed behavior is checked again after a material correction. The process is capped at **three review cycles in total**; three is a ceiling, not a quota. A clean review ends the cycle. If the third review still identifies substantive issues, the supervisor stops and brings the design back to the human owner. Repeating reviews indefinitely can polish local defects while missing that the proposed architecture is wrong for the change.

The review effort follows change risk. Executable behavior, permissions, migrations, and recovery paths receive independent scrutiny; documentation edits and routine registration use proportionate checks.

The review process does not authorize publication or deployment. Once a substantive change is accepted, its owning source must be committed to an appropriate named ref. The personal Open WebUI fork, Pennyroyal inference source, and deployment configuration have distinct source authorities. A clean, explicitly selected application checkout is built and deployed only at the separately agreed checkpoint.

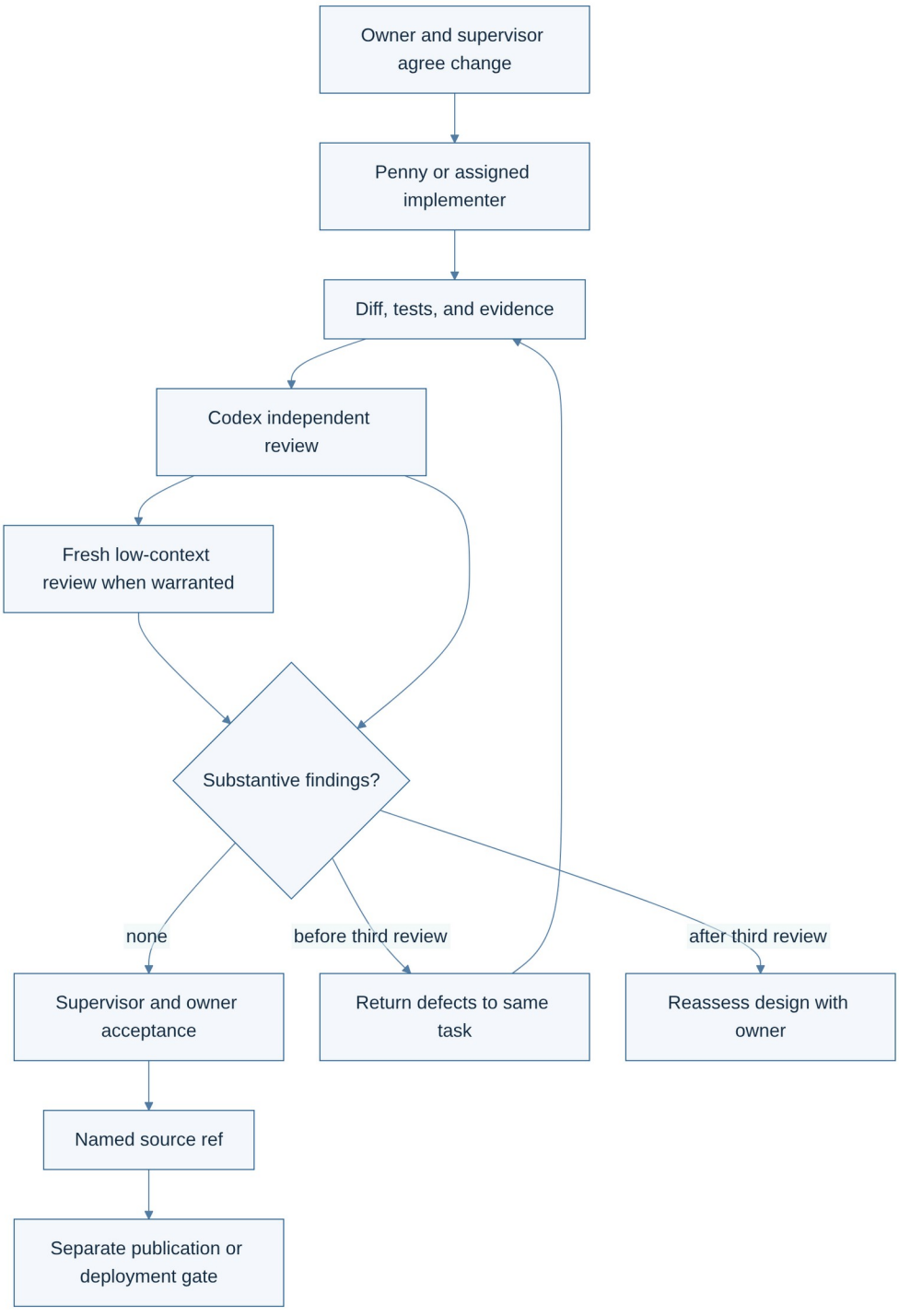

Figure 12. Bounded multi-agent review and architecture stop

The three-cycle ceiling is a control on reasoning, not a test-count target. It forces a design conversation when independent reviews keep finding problems.

Failure boundaries visible to the user

A service diagram becomes more useful when it predicts partial failure. The system can remain responsive on one path while another path is broken.

Failed or unavailable boundary	Likely user-visible effect	Authority for diagnosis or repair
Pennyroyal inference	OWUI may load, but model turns fail or stall	SGLang runtime and selected launcher
NIXL cache filesystem	Compatible prefixes may lose SSD restoration; new computation may still proceed if configuration permits	Inference cache configuration and filesystem state
Room101 OWUI application	Authentication, chats, and tool routing are unavailable despite a healthy model	Reviewed OWUI service and application state
Terminal router or mapped terminal	Chat can work while native file/command tools fail for affected users	Router mapping and native terminal services
Versity Agent Library	Archive, fetch, or publish operations fail; an existing OWUI display copy may still render	Library service, IAM mapping, objects and metadata
Neomatrix image service	Text/research can work while image generation or editing fails	AI Tools contract and ComfyUI execution
Neomatrix speech service	Typed chat can work while transcription or playback fails	OWUI audio configuration and Dubline
Codex task notification route	Penny can finish while Codex is not automatically awakened	Executor ledger and notification delivery record

The table describes ownership and visible behavior, not an automatic failover policy. It also guides acceptance: checking one healthy API cannot establish the whole user workflow. For a file task, verify storage, working copy, and authenticated download. For a supervised task, verify the saved execution, task state, notification, and review readback. For speech, verify recording to transcript to Penny response to playback.

Why the human-governed learning loop matters

Many agent deployments can call tools. Fewer make *capability evolution* a visible part of the architecture. Here, a completed workflow can provide evidence for a candidate skill. The candidate is staged as a durable owner-facing proposal, can be revised in normal conversation, and becomes discoverable in later work only after the owner approves the exact change. The resulting skill is personal; it does not silently alter every user's behavior or grant access to new resources.

That loop gives the human owner control over reusable instructions while the agent performs extraction and drafting. The owner evaluates whether a pattern is correct, general enough, and appropriate to persist. The proposal and source chat make the decision reviewable. Approval checks make concurrent or stale application less likely to corrupt an existing personal skill.

This is the clearest example of the system's broader agentic design. Multi-agent execution produces work; event-driven coordination returns attention when needed; retained state makes the work inspectable; human review decides which outcomes become durable artifacts, code, or future behavior. The same control principle spans several different kinds of persistence.

Transferable architecture patterns

Several patterns can be reused independently of these host names or exact tools.

Keep the AI data plane under operator control. Place model inference, conversation state, tool execution, media processing, and durable objects on infrastructure the operator manages. Connected research can still supply external information without becoming the model or artifact store.

Separate inference from agent governance. The model server should remain replaceable without absorbing user identity, chat persistence, task approval, or file ownership. This allows experimentation with model kernels and cache representation without rebuilding the entire work surface.

Treat long-running agent work as a state machine. An assignment has an identity, a workspace, checkpoints, capacity limits, and explicit blocked and review states. A supervisor event carries a pointer back to current state. This pattern applies to event-driven analysis, engineering automation, and operational workflows where work can pause for human input.

Make files first-class durable objects. Keep originals under stable per-user identity, give processors temporary inputs, and deliver results through an authenticated application. A storage URL, a workspace path, and a browser download URL serve different purposes and should not be exchanged casually.

Let shared tools coexist with per-user work. Shared installations reduce duplicated runtimes and maintenance. Native accounts and grants define whose data and browser state each invocation may touch. The precise isolation technology can change, but the distinction between shared code and private state should remain.

Publish measurements with the configuration that produced them. Long-context prefill, post-first-token decode, aggregate concurrency, and cache restoration answer different performance questions. Model checkpoint, precision, request shape, cache state, and test date are part of the result. The same discipline applies to subjective artifact acceptance: a deck that met a particular brief demonstrates a workflow, not a universal quality score.

The architecture's contribution is the integration of these patterns into one operating system that handles real work. Its reusable content lies in the mechanisms and their boundaries; the exact machine inventory is one implementation.

Appendix A. Interface and state reference

The preceding chapters use diagrams to show major paths. The following inventory records the interface contract at each crossing. It is intentionally limited to architectural responsibility rather than ports, credentials, or installation commands.

Service interfaces

Caller	Callee	Interface and payload	Identity and returned state
Browser or phone	Caddy and OWUI	HTTPS application session; messages, uploads, and downloads	OWUI authenticates the person and owns chat/file access decisions
OWUI	Pennyroyal	OpenAI-compatible model request containing assembled context and tool definitions	Inference returns text, tool calls, usage, and model output; OWUI owns conversation state
OWUI	Identity router	Authenticated terminal-work requests, including HTTP, streaming files, and WebSocket traffic	Router maps allowed OWUI identity to one native terminal; missing identity is rejected
Identity router	Native Open Terminal	Proxied tool execution under the mapped service	Linux UID owns process, home, browser state, and working files
OWUI	Connected research MCP	Tool invocation and query payload	Grant-controlled result becomes untrusted source material in the chat
OWUI	AI Tools MCP	Named image generation/editing call; ordered inputs where needed	AI Tools returns image content; OWUI persists and displays it
AI Tools	ComfyUI	Tracked image workflow and temporary source inputs	ComfyUI renders and returns PNG bytes, without becoming the durable library

Caller	Callee	Interface and payload	Identity and returned state
OWUI	Dubline	Speech transcription or synthesis request	OWUI owns browser interaction; Dubline owns media inference and returned text/audio
OWUI upload hook	Versity Agent Library	S3-compatible archive of an OWUI file	Library object gets stable identity; OWUI keeps its display copy
Router library tools	Versity Agent Library	User-scoped list, fetch, publish, rename, and source URL operations	Mapped private/shared S3 identity governs object access
Codex	Penny executor in OWUI	Submit, status, result, evidence, steer, and resume	Executor owns task ledger and saved worker-chat link
Executor notifier	Originating Codex task	Validated status event through a restricted receiver	Codex wakes and reads current executor state; event contains no full result

This inventory also identifies where a substitute implementation could fit. A different model server must preserve the model API and usable context/usage reporting. A different object store must preserve user-scoped object identity and signed processing inputs. A different browser harness would need equivalent authenticated file routes, saved execution state, and tool permissions before the agent workflow would retain the same properties.

State ownership and lifetime

State	Primary owner	Lifetime and recoverability
Model checkpoints and runtime source	Pennyroyal/SGLang installation and owning repository	Selected deliberately; reproducible only from the exact source, weights, and launch configuration
GPU, host, and NIXL prefix state	SGLang runtime and cache filesystem	Reusable computation; disposable when incompatible or lost
OWUI accounts, chats, settings, and uploaded display files	OWUI application state	Persistent application data with account ownership; separate from model cache
Penny task ledger	OWUI executor state	Compact assignment status, results, and references; retained across supervisor attention changes
Penny saved worker chat	OWUI chat state	Full model and tool work trace for the assignment
Task workspaces and Git worktrees	Dedicated native worker identity	Retained through blocked/review states; unpublished source must be handled explicitly
Human user homes and browser profiles	Each user's Linux identity	User-writable, private working state; may be damaged by that user without changing shared tools
Shared capability binaries and skills	Deployment-managed capability tree	Centrally installed, readable/executable by granted users, not writable by ordinary users
Private brand assets and final documentation	Private owner and application/filesystem grants	Deliberately restricted material; excluded from generic shared capability tree
Agent Library originals and metadata	Versity S3-compatible storage	Durable file objects independent of chat and workspace copies; backup/restore needs objects and metadata
Short-lived signed image input URLs	Library signing operation	Temporary service input; not a durable file or user-facing download
OWUI/terminal download links	Authenticated application file routes	Browser access checked by the user session; terminal links depend on the working copy's presence

A common source of architecture errors is to conflate these lifetimes. A NIXL namespace can be cleared while the conversation remains intact. A working file can be deleted while its library original persists. An OWUI chat can contain full worker output while the executor ledger carries only a compact review result. A source checkout can be present on a machine without being the revision selected by the running service. The design assigns each fact to the layer that can prove it.

Control-plane checkpoints

Several transitions intentionally stop automation from acquiring more authority than its input grants.

1. **Account grant:** an administrator assigns OWUI capability groups. The model's instruction text does not create access.
2. **Terminal mapping:** the router resolves the authenticated OWUI user to an explicitly allowed native identity. Unknown users do not inherit a default terminal.
3. **Task submission:** the supervisor supplies objective, boundaries, and a return target. The executor's capacity limit can reject the submission.
4. **Blocked task:** a native question pauses execution while retaining state. The supervisor supplies an answer before the same task resumes.
5. **Personal skill proposal:** the owner accepts, rejects, or revises a staged candidate before it becomes persistent behavior.
6. **Artifact review:** a rendered deck, diagram, code change, or image result is inspected before it is accepted as finished.
7. **Publication or deployment:** reviewed source or customer-facing material moves outward only through a separately authorized decision.

These checkpoints are not a universal human-in-the-loop rule for every tool action. They are placed where this deployment crosses ownership, persistence, or consequence boundaries.

Appendix B. Evidence map for architectural claims

Architectural claim	Source or exercised path
Penny-only model prompts, generation, and persistent state remain local	Deployment placement and owning sources for Open WebUI, Pennyroyal, native terminals, Versity, AI Tools, ComfyUI, and Dubline
Two Qwen3.8 profiles and HiCache/NIXL	Public Pennyroyal source, recipes, and qualification records
Near-490K restart restoration	Flash-Next restart test with exact restored and recomputed token counts and needle checks
Personal Open WebUI Fork selected	Application source revision and active service selection
Multi-user native terminals	Deployment configuration and authenticated checks with two distinct user accounts
Retained agent pause, notification, and resume	Executor source and live same-chat acceptance
Per-user Agent Library and authenticated delivery	Byte-exact download, cross-user denial, and signed-input checks
Native MCP image display	Application integration and authenticated image checks
Editable HPE deck workflow	Rendered PPTX and owner acceptance
Browser speech path	Voice Mode acceptance from recording through response and playback

The public runtime references below identify the performance and persistence campaigns. Application, deployment, and capability sources are maintained in their owning repositories.

Source and verification notes

This architecture snapshot was checked against owning source and selected services on 21 September 2026. The performance tables reproduce the public Pennyroyal qualification campaigns.

Primary source map

- Public Pennyroyal SGLang source: v2.5.1 README, RUN, RESULTS, CHANGES, and memory/persistence notes.
- OWUI deployment source: owuimulti-deploy architecture, current operations, executor, Agent Library, speech, browser, and Phase 8 acceptance notes.
- Personal Open WebUI fork source: pennyroyal-owui, especially the executor, context usage/compaction, and owner-approved personal skill-learning paths.
- AI Build system map and software orientation provide placement and ownership context; observed source and runtime selection take precedence where they differ.

Appendix C. Architectural decisions and build trajectory

Hands-on development began on 4 June 2026 with local inference on an ASRock Radeon AI PRO R9700. The 21 September architecture snapshot follows **109 elapsed calendar days** of iterative design and integration. The system grew through workload-driven decisions rather than a fixed sequence of planned phases.

The dates below identify when a decision or resulting state was recorded. Where the source is a documentation generation, that date may be later than the implementation itself.

Recorded date	Decision of consequence	Architectural result
4–24 June	Move from initial R9700 exploration to one 96 GB RTX PRO 6000 for larger local models	A single-GPU inference center, with hardware limits and power delivery treated as part of model qualification
8 July	Move beyond direct model chat toward sustained tool-using agent work	An explicit agent-control layer; later harness experiments informed the durable workflow now in Open WebUI
31 July–2 August	Make model selection reproducible through frozen reasoning, tool, and long-context exercises	Candidate runtimes were judged on correctness and workload fit, including near-million-token retrieval
18–27 August	Own the SGLang-derived runtime and qualify two profiles in one source line	SM120-specific kernels, speculation, long-context execution, and HiCache/NIXL restoration became one documented inference architecture
21 August–19 September	Make Open WebUI a multi-user work surface with native per-user execution and a reviewed personal fork	Shared tools remained centrally managed while identity, chats, workspace state, and application behavior gained explicit owners
17 September	Retain originals outside chats and workspaces	Versity Agent Library added stable S3 object identity, per-user access, temporary processing inputs, and authenticated delivery
19–21 September	Add retained agent assignments and human-approved capability learning	The executor gained ledger-backed pause, notification, review, and resume; skill proposals gained an owner approval boundary

The detailed research ledger behind this selection is maintained separately.