Architecting Local, Air-Gap-Capable Desktop Clients for Agentic AI Pair Programmers: Protocol Compatibility, Process Isolation, and Host Memory Safety

dgxcoder
dgxcoder@dreamference.ai

October 2026

Abstract

Autonomous agentic coding assistants have progressed from rudimentary terminal scripts into complex, interactive graphical desktop environments capable of repository navigation, multi-file code synthesis, terminal multiplexing, and background task management. However, prevailing commercial desktop clients remain tethered to remote vendor clouds, closed model APIs, and persistent background telemetry daemons. Deploying a vendor-free, air-gap-capable agent on modern local workstation hardware—specifically unified-memory CPU–GPU architectures such as the NVIDIA GB10 (128 GB unified LPDDR5X, Blackwell SM121)—presents intricate systems challenges spanning display driver compatibility, multi-process privilege separation, operating system sandboxing, and auditable network isolation.

In this paper, we present the architectural design, formal capability model, and empirical evaluation of Ragon Desktop, an open-source desktop environment that speaks the same app-server protocol as the upstream vendor's desktop client, keeps Codex-compatible defaults, runs inference on the user's own hardware, and confines every sandboxed command to the workstation when the user turns `/airgapped on`. We report five core contributions: (1) **A Documented Reference Point**: A description of the upstream vendor's desktop app drawn only from its public documentation and observable behaviour, separating the open app-server protocol it drives from the window around it, which an open client re-implements with auditable components. (2) **Display Subsystem Architecture**: An empirical investigation demonstrating that standard WebKitGTK desktop wrappers fall back to CPU rasterization on NVIDIA's driver because it refuses kernel Direct Rendering Manager (DRM) dumb buffers (`DRM_IOCTL_MODE_CREATE_DUMB`), causing 40–50 ms per keystroke frame on a 4096×1700 window; an Electron/Chromium pipeline using ANGLE over NVIDIA's own EGL/GL libraries restores hardware compositing. (3) **Formal Capability Gating**: A three-tier multi-process architecture with an allowlist gatekeeper (`policy.json`) over the 170-method JSON-RPC protocol, proving that no method outside the allowlist—among them every vendor account, feedback and remote-control channel—reaches the agent from the renderer, and that the allowlisted unconfined or networked primitives (the user's `!` command via `thread/shellCommand`, plugin installs and marketplace fetches) are refused whenever `/airgapped on`. (4) **Host Sandboxing under User-Namespace Restrictions**: A Linux sandboxing model for Ubuntu's restriction of unprivileged user namespaces (`kernel.apparmor_restrict_unprivileged_userns=1`) via targeted AppArmor profiles, with no SUID root helper (`chrome-sandbox`) in the packaged app, and binary-level Electron fuses. (5) **Workstation Stability & Audited Egress**: A two-layer host-memory safety design (a pre-load host check and a Pressure Stall Information (PSI) watchdog) beside a resident 27B NVFP4 language model server, and an on-demand `strace` egress audit that fails a session whose traced process tree reaches a non-loopback address or sends a DNS query.

1 Introduction

The integration of Large Language Models (LLMs) into developer workflows has shifted decisively from passive inline code completion toward autonomous agentic loops [8, 13–15]. Modern coding agents do not merely suggest text; they autonomously read repository structures, formulate multi-file patches, execute build tools in local environments, parse compiler error diagnostics, and iterate until test suites pass.

To support human-in-the-loop oversight and steering, developer interfaces have similarly evolved. Where agents initially operated within basic Command-Line Interfaces (CLIs), developers increasingly rely on dedicated desktop workspaces that integrate visual diff review panes, interactive terminal tabs, git worktree managers, file tree navigators, and live artifact previewers.

Despite the release of open-source client harnesses, commercial desktop applications remain fundamentally cloud-dependent. Commercial clients transmit repository source files, developer prompts, and tool execution logs to remote vendor endpoints. Concurrently, background telemetry (crash reporters, metrics collectors, and feature-flag services) keeps outbound network connections open. In sectors governed by strict regulatory mandates—such as aerospace (ITAR), healthcare (HIPAA), financial infrastructure, and offensive cybersecurity research—transmitting proprietary intellectual property or vulnerability exploit code to third-party cloud infrastructure is legally prohibited or operationally unacceptable.

Deploying an autonomous coding agent entirely on-premises on a single desk-side workstation introduces substantial systems engineering challenges that do not manifest in cloud-hosted web wrappers:

1. **Display Subsystem and Driver Incompatibilities**: High-performance local AI workstations increasingly adopt unified CPU–GPU memory architectures (e.g., NVIDIA Blackwell GB10, Grace Hopper, Apple Silicon). On Linux, interactions between GPU drivers and generic webview windowing frameworks frequently fail, degrading interactive input performance.

2. **Privilege Separation vs. Developer Capabilities**: A desktop agent client must perform high-privilege actions—running shell processes, inspecting git repositories, and modifying files—while ensuring that the webview renderer cannot be compromised by untrusted repository contents or malicious prompt injections.

3. **Modern Kernel Sandboxing Barriers**: Contemporary Linux distributions (e.g., Ubuntu 24.04 LTS) restrict unprivileged user namespaces by default (via the kernel sysctl knob `apparmor_restrict_unprivileged_userns`), breaking standard Chromium and container sandboxes unless appropriate kernel security profiles are established.

4. **Hardware Stability under Unified Memory**: Because CPU and GPU share a common physical memory pool, unbounded memory growth in desktop sidecars can raise memory pressure until the host stalls or the kernel's Out-Of-Memory (OOM) killer ends a process, possibly the resident language model inference server.

5. **Verifiable Egress Assurance**: Showing that no vendor channel exists, and that a strictly confined session stays on loopback, requires kernel-level network tracing of real sessions.

In this paper, we describe the architecture of Ragon Desktop, an open-source, vendor-free and air-gap-capable desktop environment built for the NVIDIA GB10 workstation. We describe the upstream vendor's desktop app from its public documentation, detail the resolution of GPU driver rendering bottlenecks, formalize the IPC capability gating model, and report the latency, sandboxing, and network egress results we hold records for, marking the remaining measurements as to be made.

Defaults and guarantees. Ragon follows Codex, the open-source upstream project its agent is built from, in its defaults: at `/airgapped off` (the default level) sandboxed commands have the network, plugins can be installed, the user's `!` commands run in the user's own shell, and the user may choose Full Access. Two guarantees hold on top of that. First, at every level there is no vendor account, sign-in, telemetry or analytics channel (Ragon's patches 0005, 0013, 0015 and 0016 to the upstream agent, checked by the egress audit of Section 8). Second, at `/airgapped on` confinement is strict: every sandboxed command runs in an empty network namespace, the host's control sockets are masked, and Full Access and `!` commands are refused.

1.1 Threat Model

We assume a zero-trust threat model with respect to repository contents, visual renderer context, and network egress:

- **Threat T1: Malicious Repository / Prompt Injection.** An agent navigating a public or third-party repository may encounter hostile files (e.g., crafted Markdown, SVG, or HTML) designed to exploit webview vulnerabilities or inject malicious system prompts into the agent loop. The system must prevent renderer compromise from escalating to host execution.
- **Threat T2: Unauthorized Egress & Exfiltration at `/airgapped on`.** When the user turns `/airgapped on`, untrusted third-party dependencies, language server extensions, or rogue model outputs must be prevented from opening outbound network connections, leaking source code, or performing external telemetry tracking. At `/airgapped off` sandboxed commands have the network, as in Codex, and the guarantee is the narrower one that no vendor or telemetry channel exists.
- **Threat T3: Local Privilege Escalation via SUID Helpers.** Historic browser architectures relied on SUID root helpers (`chrome-sandbox`) for process isolation. Attackers can leverage SUID binary vulnerabilities to achieve root privilege escalation on the developer's workstation.
- **Threat T4: Unified Memory Exhaustion.** An unconstrained visual client or renderer leak could exhaust system RAM, destabilizing the unified memory subsystem and crashing resident model inference processes.

2 The Upstream Desktop App, from Public Material

To fix a reference point, we describe the upstream vendor's desktop app only from its public documentation, its release notes and what a user sees. Nothing was downloaded and unpacked, decompiled or read from the app's bundle, and no part of it was copied; every feature Ragon offers in its place is re-implemented from that description. Running the app was outside this study as well, because it asks for a sign-in with the vendor's account before any surface opens.

2.1 What the App Is

Codex launched as a desktop app on macOS in February 2026 and on Windows in March 2026. Since July 2026 the vendor ships it as a view inside its general desktop app, beside the vendor's other views and with its own history. The documented platforms are macOS 14 or later, Windows on x64 and ARM64, and, as a preview since August 2026, Linux on x64 and ARM64, installed as a `.deb` or `.rpm` package from a signed vendor package repository. The app requires the vendor's account, and inference runs on the vendor's hosted models.

2.2 Documented Features

The launch announcement and the vendor's documentation and changelog describe the app as a command centre for agents:

- **Projects and parallel threads**: agents run in separate threads organized by project, and the user switches between them without losing context. Session history and configuration are shared with the Codex CLI and IDE extension.
- **Review in the thread**: the agent's changes are reviewed as diffs (staged and unstaged, per file and per hunk), commented on, and opened in the user's editor.
- **Local and worktree modes**: a thread works in the user's checkout or in a Git worktree, and can be handed between the two; commit, push and pull-request creation are offered from the app.
- **Skills and automations**: skills are created and managed in the app and work in the CLI too; automations run instructions on a schedule and put their results in a review queue.
- **Terminal and sandbox**: an integrated terminal, and the CLI's sandbox with approval for elevated commands.

2.3 The Documented Protocol

The vendor documents app-server as the interface Codex uses to power rich clients, and it is open source in Codex's repository [4]. It is JSON-RPC 2.0: client requests, server requests (approvals), and streamed notifications, served over a local `stdio` connection to a desktop client. At the pinned release (`rust-v0.158.0`) it defines 170 client request methods, 11 server request methods, 85 server notifications and 1 client notification, and `codex app-server generate-ts` writes TypeScript types for them. The protocol has no commit, push, worktree or pull-request method, so a window that offers those does them itself.

2.4 What Follows for an Open Client

Everything that makes the desktop app a coding agent sits behind an open protocol served by an open binary. What is the vendor's own is the window around it: its layout and interaction design, its git and terminal plumbing, and the features tied to the vendor's account and cloud. An open client can therefore drive the same agent core, here `ragon app-server`, and re-implement the window from auditable components, which is what the rest of this paper describes.

3 Hardware and Display Architecture on Unified Memory

On local AI workstations featuring unified CPU–GPU memory architecture, graphics rendering performance is inextricably coupled with kernel driver memory management.

3.1 The WebKitGTK Dumb-Buffer Bottleneck

Initial implementations of lightweight desktop wrappers utilized toolkits such as Tauri, which bind to system-installed WebKitGTK (version 2.52.3 on Ubuntu 24.04). While functional for basic dialogs, typing inside an offscreen WebKitGTK view on the NVIDIA GB10 workstation exhibited severe frame lag on a large window (Table 1).

Table 1: Time per keystroke frame in an offscreen WebKitGTK 2.52.3 view on NVIDIA GB10 (driver 580.173.02; a textarea receiving one keystroke per animation frame over 60 frames, with the app's setting `WEBKIT_DISABLE_DMABUF_RENDERER=1`).

Toolkit	Window	Scale	Time per Frame	Compositing Path
WebKitGTK 2.52	1280 × 860	1.0×	16 ms	CPU Rasterization (Cairo)
	1280 × 860	2.0×	30 ms	CPU Rasterization (Cairo)
	4096 × 1700	1.25×	**40–50 ms**	CPU Rasterization (Cairo)
Electron ANGLE	*To be measured with the same protocol.*			Hardware EGL / GLX

To isolate the root cause, we traced system calls made by the WebKitGTK rendering process during initialization. With its DMABUF renderer enabled, WebKitGTK under X11 attempts to create shared Direct Rendering Manager (DRM) "dumb buffers" to share rendered surfaces with the display server:

```
ioctl(drm_fd, DRM_IOCTL_MODE_CREATE_DUMB, &args);
```

Under NVIDIA's 580.173.02 driver (open kernel modules, proprietary user-space GL/EGL), this ioctl systematically fails:

- When invoked on the primary display card node (`/dev/dri/card1`): returns error `ENOSYS` ("Function not implemented").
- When invoked on the render node (`/dev/dri/renderD128`): returns error `EACCES` ("Permission denied").

The failure stems from a design difference in NVIDIA's driver: it manages framebuffers via its own memory manager (NVKMS) and implements only part of the generic DRM interface, omitting support for the dumb-buffer ioctls on the display node. Because WebKitGTK's one GPU path on X11 depends on DRM dumb buffers, the refusal leaves no window at all when the DMABUF renderer is on. The app therefore ran with `WEBKIT_DISABLE_DMABUF_RENDERER=1`, where WebKitGTK draws with Cairo on the CPU.

3.2 CPU Rasterization Cost Grows with Window Area

Every setting that avoids the GPU path (DMABUF off; compositing off; compositing forced; GLX forced) lands at 40–50 ms per keystroke frame for the 4096 × 1700 window, a near-full-screen window on a 5120 × 2160 monitor at 1.25× scale: letters trail the keys at 20–25 frames per second. The default 1280 × 860 window stays smooth at 16 ms, while the same window at 2× scale already takes 30 ms. The cost therefore grows with the number of pixels the CPU must rasterize per frame. We do not attribute it to a specific bottleneck (memory bandwidth or rasterization compute); that profile is to be measured.

3.3 The Chromium ANGLE Hardware Pipeline

Chromium sidesteps the DRM dumb-buffer limitation through its Almost Native Graphics Layer Engine (ANGLE) abstraction layer. ANGLE translates WebGL and Chromium compositor draw commands into native desktop OpenGL ES, EGL, and GLX API calls. Chromium communicates directly with NVIDIA's user-space driver libraries (`libEGL.so.1`, `libGLESv2.so.2`) and composites

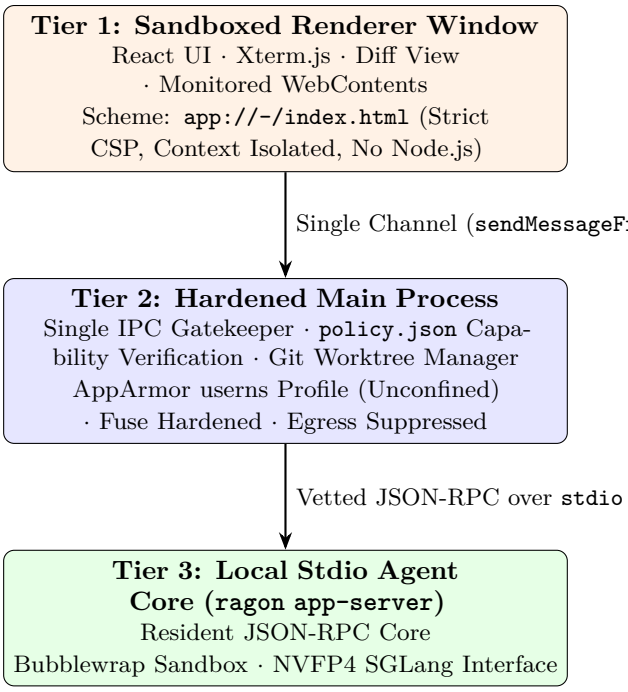


Figure 1: Three-tier architecture of Ragon Desktop: Sandboxed Renderer, Policy-Vetted Main Process, and Stdio Agent Core.

directly into X11 Pixmaps, never asking for a dumb buffer. The same page in a Chromium browser on the same machine did not lag.

By transitioning to an Electron runtime (Electron 42.11.11, pinned exactly), Ragon Desktop restores native hardware GPU compositing on NVIDIA Blackwell hardware. Its time per keystroke frame across window sizes, with the protocol of Table 1, is to be measured.

4 System Architecture and Process Isolation

Ragon Desktop is structured as a three-tier multi-process architecture engineered to isolate untrusted visual components from host operating system privileges (Figure 1).

4.1 Tier 1: Sandboxed Renderer Window

The visual layer is rendered within a frameless Electron window loading exclusively from a privileged custom scheme: `app://-/index.html`. The scheme is registered via `protocol.registerSchemesAsPrivileged` with flags `standard`, `secure`, `stream`, and `supportFetchAPI`.

To prevent cross-site scripting and unauthorized resource loading, the application delivers a strict Content Security Policy (CSP) on all HTML responses:

```
default-src 'none';
script-src 'self' 'wasm-unsafe-eval';
style-src 'self' 'unsafe-inline';
font-src 'self' data:;
img-src 'self' app: blob: data:;
connect-src 'self';
worker-src 'self' blob:;
```

```
base-uri 'none'; form-action 'none';
```

The renderer executes with `sandbox: true`, `contextIsolation: true`, and `nodeIntegration: false`. No native filesystem or operating system APIs are injected into the DOM. The preload script exposes a minimal surface: one messaging function, a helper that resolves a dropped file to its path, and a string naming the window type:

```
window.electronBridge = {
  sendMessageFromView: (msg) =>   // large messages go in acknowledged pieces
    ipcRenderer.invoke("ragon_desktop:message-from-view", msg),
  pathForFile: (file) => webUtils.getPathForFile(file),
};
window.ragonWindowType = "electron";
```

4.2 Tier 2: Hardened Main Process Gatekeeper

The main process acts as an immutable capability gatekeeper. It listens on the `ragon_desktop: message-from-view` IPC channel, with a second channel that only acknowledges the pieces of large messages. Before forwarding any request to the underlying agent core, the main process validates the request against a compiled capability policy (`policy.json`), the same file `ragon web` enforces; both enforcers run the same conformance cases.

The main process also answers the page's own local requests—git and `gh`, worktrees, Night Shift, project tasks, the `/airgapped` toggle and voice—each checked to come from the app window. These are not app-server methods and lie outside the formal model of Section 5.

4.3 Tier 3: The Stdio Agent Daemon

The agent core executes as an independent, unprivileged child process:

```
ragon -c features.code_mode_host=true app-server
```

Communication is conducted exclusively over standard input and standard output pipes. By avoiding loopback network sockets (e.g., `127.0.0.1:3000`), the architecture prevents malicious local web pages or local unprivileged users from querying the agent daemon via standard browser `fetch` calls.

5 Formal Capability Gating and Protocol Compatibility

The pinned upstream agent protocol (`rust-v0.158.0`) specifies 170 client request methods, 11 server request methods, 85 server notifications, and 1 client notification. Rather than implementing an unconstrained proxy, Ragon formalizes access control via a capability classification model.

5.1 Formal Protocol Partitioning

Definition 1 (Protocol Universe)**.** *Let $\mathcal{M}$ denote the universe of 170 JSON-RPC request methods defined in `app-server-protocol`. We define a partition of $\mathcal{M}$ into four mutually disjoint capability tiers:*

$$\mathcal{M} = \mathcal{T}_0 \cup \mathcal{T}_1 \cup \mathcal{T}_2 \cup \mathcal{T}_3 \tag{1}$$

where:

- $\mathcal{T}_0$ *(**Read-Only State**): Queries that inspect configuration, thread history, or models without mutating disk state (e.g., `model/list`, `thread/read`, `config/read`, `project/list`).*

Table 2: Protocol capability partitioning and policy enforcement across the 170 JSON-RPC client methods.

Tier	Count	Example Methods	Gatekeeper Action
$\mathcal{T}_0 \cup \mathcal{T}_1 \cup \mathcal{T}_2$ (Allowed)	73	`model/list`, `thread/read`, `thread/name/set`, `project/update`, `command/exec`	Forwarded; thread openers scrubbed
of which $\mathcal{A}$	4	`thread/shellCommand`, `plugin/install`, `marketplace/add`	Refused at `/airgapped on`
$\mathcal{T}_3$ (Not exposed)	97	`account/login/start`, `feedback/upload`, `remoteControl/enable`	**Rejected** (Returns $\perp$)

- $\mathcal{T}_1$ (***Confined Mutation***)*: Controlled mutations of threads, projects and settings (e.g.,* `thread/start`*,* `thread/compact/start`*,* `project/update`*,* `thread/name/set`*).*
- $\mathcal{T}_2$ (***Execution***)*: Execution primitives. The agent's and the terminal's commands (*`command/exec`*,* `command/exec/terminate`*) run under the permission profile the user picks; a* `!` *command the user types in the composer (*`thread/shellCommand`*) runs in the user's own shell.*
- $\mathcal{T}_3$ (***Not Exposed***)*: Vendor and cloud channels, refused always (e.g.,* `account/login/start`*,* `feedback/upload`*,* `remoteControl/enable`*), and methods the UI does not implement, refused until reviewed.*

We further distinguish $\mathcal{A} \subset \mathcal{T}_1 \cup \mathcal{T}_2$, the networked or unconfined primitives allowed only at `/airgapped off`*:* `thread/shellCommand`*,* `plugin/install`*,* `marketplace/add` *and* `marketplace/upgrade`*.*

About a third of the 97 methods in $\mathcal{T}_3$ are vendor or cloud channels (account, feedback upload, remote control, user verification, plugin sharing, realtime sessions). The rest are local features of the vendor's app that Ragon's UI does not implement; the allowlist refuses them, as it refuses any method a future protocol adds, until they are reviewed.

Theorem 1 (Capability Containment Theorem)**.** *Let $\mathcal{G} : \mathcal{M} \times \mathcal{P} \times \{off, on\} \to (\mathcal{M} \times \mathcal{P}) \cup \{\perp\}$ denote the capability gating function enforced by Tier 2, where $\mathcal{P}$ represents request parameters and the third argument is the* `/airgapped` *level of the thread the request names. For every request $r = (m, p)$ at level ℓ:*

$$\forall m \in \mathcal{T}_3, \quad \mathcal{G}(m, p, \ell) = \perp \qquad \text{and} \qquad \forall m \in \mathcal{A}, \quad \mathcal{G}(m, p, on) = \perp. \tag{2}$$

Furthermore, thread-opening requests (`thread/start`*,* `thread/resume`*,* `thread/fork`*) have instruction, provider, tool and path fields removed before delivery to Tier 3, and Ask threads are confined to a scratch folder. Other allowed requests are forwarded unchanged; what they may do is bounded by the agent's own permission profile, which at* `/airgapped on` *refuses Full Access (patch 0023).*

Proof. The gatekeeper compiles `ragon-rs/web/policy.json` into an immutable hash set $\mathcal{S}_{\text{allowed}}$ of 73 permissible methods and a set $\mathcal{A} \subset \mathcal{S}_{\text{allowed}}$ of 4 methods refused at `/airgapped on`. By construction:

$$\mathcal{S}_{\text{allowed}} \cap \mathcal{T}_3 = \emptyset \tag{3}$$

When a message arrives at `ragon_desktop:message-from-view`, the handler computes:

$$\mathcal{G}(m, p, \ell) = \begin{cases} (m, \text{scrub}(p)) & \text{if } m \in \mathcal{S}_{\text{allowed}} \setminus \mathcal{A}, \text{ or } m \in \mathcal{A} \text{ and } \ell = \text{off} \\ \bot & \text{otherwise} \end{cases} \tag{4}$$

where scrub is the identity except on the thread openers and the configuration import. Because membership is verified before invocation, no method in $\mathcal{T}_3$ can reach Tier 3, and no method in $\mathcal{A}$ can reach it at `/airgapped on`. □

5.2 `thread/shellCommand`: User-Initiated, Refused at `/airgapped on`

`thread/shellCommand` is user-initiated: it runs a `!` command the user typed in the composer, in the user's own shell; the agent cannot issue it. Ragon allows it while `/airgapped` is off, as Codex does, and refuses it at `/airgapped on` in both enforcers (Table 2). The agent's commands run under the sandbox mode the user picks (read-only, workspace-write or Full Access), as in Codex. Only at `/airgapped on` does Ragon add confinement: an empty network namespace, the host's control sockets masked, and Full Access and `!` commands refused. Commands that need the network or host daemons fail only there.

5.3 Prompt Injection Defense via Parameter Scrubbing

When initiating, resuming or forking threads (`thread/start`, `thread/resume`, `thread/fork`), untrusted visual extensions could attempt to inject malicious prompt overrides. The gatekeeper enforces strict parameter scrubbing by stripping `droppedThreadFields`:

```
"droppedThreadFields": [
  "baseInstructions", "developerInstructions", "modelProvider",
  "config", "personality", "dynamicTools", "environments",
  "selectedCapabilityRoots", "history", "path"
]
```

This prevents a renderer compromise from subverting the agent's core system prompt or hijacking tool capability roots. Requests that change settings (`config/value/write`, `config/batchWrite`) and `command/exec` are forwarded as sent; they act with the user's own authority, bounded by the permission profile above.

6 Linux Host Safety and Modern Sandboxing

6.1 User Namespace Restrictions in Ubuntu 24.04

Ubuntu 24.04 LTS introduced an important kernel hardening feature:

```
kernel.apparmor_restrict_unprivileged_userns = 1
```

When enabled, the Linux kernel refuses unprivileged calls to `unshare(CLONE_NEWUSER)` or `clone(..., CLONE_NEWUSER, ...)`, returning error `EPERM` ("Operation not permitted").

Historically, Chromium-based applications bypassed this restriction by installing a setuid root binary (`chrome-sandbox`). However, SUID root binaries present a dangerous attack surface: vulnerabilities in SUID helpers have historically enabled local privilege escalation to root (e.g., CVE-2016-8655, CVE-2022-2588).

6.2 SUID-Free AppArmor Profile Integration

The packaged Ragon Desktop ships no SUID helper. The packaging build script explicitly deletes `chrome-sandbox` post-package. Instead, the `.deb` installs a dedicated AppArmor profile at `/etc/apparmor.d/ragon`:

```
abi <abi/4.0>,
include <tunables/global>

profile ragon "/usr/lib/ragon/Ragon" flags=(unconfined) {
  userns,
  include if exists <local/ragon>
}
```

By granting the `userns` flag specifically to the desktop executable path, the kernel allows Chromium's multi-process sandbox to create unprivileged user namespaces while keeping host system privileges unconfined. Development builds from source checkouts install a development profile (`ragon-desktop-dev`) with `sudo`, so a checkout sandboxes the same way as the Debian package. An unpacked build without `sudo` falls back to Chromium's setuid sandbox, using the helper an installed Chrome, Edge or Brave provides.

6.3 Binary Fuse Hardening

Standard Electron binaries allow runtime inspection and arbitrary script evaluation via environment variables (e.g., `ELECTRON_RUN_AS_NODE=1` or `-inspect=9222`). To permanently disable these debug backdoors, the packaging pipeline applies `@electron/fuses` v2:

- **Flipped Off**: `RunAsNode`, `EnableNodeOptionsEnvironmentVariable`, `EnableNodeCliInspectArguments`, and `GrantFileProtocolExtraPrivileges`.
- **Flipped On**: `EnableCookieEncryption`, `EnableEmbeddedAsarIntegrityValidation`, `OnlyLoadAppFromAs` and `WasmTrapHandlers`.

These fuses mutate binary bits in the compiled ELF executable, so Node's run-as-node, `NODE_OPTIONS` and inspect hooks are disabled in the binary even if an attacker controls environment variables. Chromium's own command-line switches, such as `-remote-debugging-port`, are not fuses and are outside this guarantee.

7 Workstation Memory Stability on Unified Hardware

On unified-memory architectures like the NVIDIA GB10, host CPU RAM and GPU VRAM reside on the same physical 128 GB LPDDR5X bus. When serving a 27B-parameter language model (Qwen3.8-27B in NVFP4) with a block diffusion drafter via SGLang, the weights take about 20 GB; SGLang's static share (memory fraction 0.50) holds them and the KV pool, and about 39 GB of host memory stays available while serving. A timeline of memory residency over a long session (model, KV pool and desktop client) is to be measured.

On unified memory, a model load can freeze the whole host. Ragon guards it with two layers: `check_host_safety()` refuses a load on a host that is not safe, and a watchdog (`MemoryPressureWatchdog`) follows Linux Pressure Stall Information (PSI) during the load and stops the model container, through the fastest kill path the host can still run, before the host stalls. Once the model is resident, a leaking visual client could still raise memory pressure until the kernel's OOM killer ends a process, possibly the model server. Ragon Desktop bounds its own memory via three rules:

Table 3: What the audited session may reach. At `/airgapped off` the web-search container and enabled messenger bridges, outside the traced process tree, are named exceptions.

Destination Port	Binding	Protocol	Authorized Service
Configured model port	`127.0.0.1`	HTTP/JSON	Local SGLang / vLLM model inference server.
Port 8767	`127.0.0.1`	HTTP/JSON	Local read-only Gmail sidecar.
Port 8768	`127.0.0.1`	HTTP/JSON	Local image search service.
Port 8888	`127.0.0.1`	HTTP/JSON	Local SearXNG web search engine.
Disallowed Ports	**Any**	**Any**	**Audit Verdict: FAIL**
Port 53 (DNS)	External	UDP/TCP	Prohibited (no DNS query may leave the traced tree).
All External IPs	Non-Loopback	TCP/UDP	Prohibited for the traced process tree.

1. **Single-Process Daemon**: Every window—the main one, new windows, pop-outs, and detached terminal and diff panes—multiplexes over a single resident `ragon app-server` daemon, eliminating multi-instance process duplication.

2. **Cache Evacuation in Development**: When the app is run from a checkout (`ragon-admin desktop run`), Chromium's HTTP cache directory (`Cache/`) is purged at start; the packaged app and `ragon app` keep Chromium's default cache.

3. **Bounded Async Operations**: Filesystem access to the user's folders times out after 10 seconds, and `git` and `gh` output is capped at 16 MB, so large directory trees and outputs cannot exhaust the heap.

8 On-Demand Network Egress Auditing

To verify that the desktop application opens no vendor channel and that its session stays on loopback, Ragon incorporates an automated kernel-level egress audit (`ragon-admin audit egress -app`).

8.1 Audit Methodology

The egress auditor runs the packaged desktop application with hidden windows, in an isolated scratch environment, for a 40-second session under system call tracing; `ragon-admin codex build` runs the audit too:

```
strace -f -e trace=connect,sendto,sendmsg,sendmmsg,execve,write,writev
```

The audit monitors every socket creation, destination IP address, and payload transmission. Crucially, tracing `sendmmsg` captures asynchronous DNS queries issued by `glibc`. It judges the traced `ragon` or app process tree against loopback. Services outside that tree reach the internet by design at `/airgapped off`: the SearXNG container behind loopback port 8888, which queries search engines for `ragon-search`, and any messenger bridge the user turned on, which the audit names as a declared exception rather than failing it.

8.2 Egress Suppression in Renderer Previews

When rendering live HTML previews, web pages frequently attempt background network calls via DNS prefetching:

```
<link rel="dns-prefetch" href="//telemetry.example.com">
```

Testing revealed that the Chromium Blink renderer resolves DNS prefetch links directly from the renderer process, bypassing standard Electron web request filters. To neutralize this leak, the desktop application initializes with:

- `-blink-settings=dnsPrefetchingEnabled=false`
- Custom response header: `X-DNS-Prefetch-Control: off`
- Command-line switches: `-disable-background-networking`, `-disable-component-update`.

The audit's app session passes only if the traced process tree makes no non-loopback connection and sends no DNS query (Table 3).

9 Cross-Platform Architectures: macOS and Windows on Arm

While Linux on NVIDIA Blackwell represents the primary deployment platform for local model workstations, extending the desktop pair-programming architecture to developer laptops (such as Apple Silicon macOS and Windows 11 on Arm PCs) exposes distinct platform-specific systems constraints.

9.1 macOS Pipeline: Metal Compositing and TCC Filesystem Gates

On macOS (Apple Silicon M-series unified memory architectures), the graphics subsystem differs fundamentally from Linux X11:

1. **Metal Rendering Pipeline**: Rather than translating through OpenGL ES or Vulkan, Chromium ANGLE maps directly to Apple's Metal API framework. This delivers hardware compositing without the dumb-buffer allocation failures observed under Linux X11 drivers; frame timing on macOS is to be measured.
2. **Transparency, Consent, and Control (TCC) Deadlocks**: macOS enforces strict sandbox consent via the TCC subsystem. The first programmatic filesystem access to sensitive developer directories (`~/Documents`, `~/Desktop`, or `~/Downloads`) triggers an interactive system dialog. In preliminary builds, executing synchronous directory walks (`fs.readdirSync`) during startup blocked the main `libuv` event loop while awaiting user consent. This induced an unrecoverable windowless freeze where teardown failed to respond even to `SIGTERM`. Ragon Desktop resolves this by mandating asynchronous non-blocking walks (`fs.promises`) governed by bounded 10-second timeouts.
3. **Keychain Subsystem Mocking**: On ad-hoc signed or development preview builds, Chromium's `os_crypt` framework queries the macOS Keychain ("Ragon Safe Storage") via `SecItemCopyMatching`, hanging the process if an unlocked login keychain is unavailable. Because the app holds no vendor credentials and talks to its agent over local pipes, the application launches with `-use-mock-keychain`, completely eliminating interactive keychain prompts.

9.2 Windows 11 on Arm: Direct3D ANGLE and the Agent's Windows Sandbox (Preview)

The desktop app has no released Windows build. This subsection records the design for Windows 11 on Arm (targeting Snapdragon X Elite and NVIDIA N1X developer workstations) and says which parts exist in code:

1. **Direct3D Compositing via DXGI**: On Windows, Chromium binds ANGLE to Direct3D 11 via the DirectX Graphics Infrastructure (DXGI), so the driver-level dumb-buffer issue does not arise; frame timing is to be measured.
2. **Codex's Windows Sandbox at `/airgapped on`**: Without Linux network namespaces, Ragon relies on the Windows sandbox of the upstream agent, with its offline account and firewall rules. Ragon's patch `0024` forces that sandbox's network to `Restricted` at `/airgapped on`. This is built, but has not yet been exercised on a Windows PC.
3. **Job Objects for Unified Memory Protection (Proposed)**: Windows lacks Linux Pressure Stall Information (PSI). The proposed replacement is Windows Job Objects with hard commit limits (`JOB_OBJECT_LIMIT_JOB_MEMORY`) and monitoring of kernel memory resource notifications (`CreateMemoryResourceNotification`). This is not built.
4. **Event Tracing for Windows (ETW) Egress Auditing**: The Linux `strace` audit has an ETW counterpart for `ragon exec`, monitoring the `Microsoft-Windows-Kernel-Network` and `Microsoft-Windows-DNS-Client` providers. An ETW audit of the desktop app is not built.

10 Experimental Evaluation

We evaluated Ragon Desktop on an NVIDIA GB10 workstation (128 GB LPDDR5X, kernel 6.17.0-1029-nvidia, Ubuntu 24.04.4 LTS). The keystroke measurements of Table 1 are the results we hold records for. The following measurements are to be made, each to be reported with its log:

- **IPC Throughput and Latency**: JSON-RPC turnaround across standard input/output pipes (`stdio`), Unix Domain Sockets and TCP loopback, for turn-sized, diff-sized and tree-sized payloads (mean, P95, throughput). *Table to be measured.*
- **Display**: Electron/ANGLE time per keystroke frame at the window sizes of Table 1, and tail latencies for both toolkits.
- **Workstation Stability under Sustained Workload**: model serving throughput, desktop client RSS over time, and PSI memory pressure over a long pairing session beside the resident model server.

11 Differences from the Vendor's Desktop App and Remaining Limitations

Ragon Desktop speaks the same app-server protocol as the vendor's app, with the vendor channels removed, and keeps Codex-compatible defaults. What stays different by design is short: there is no vendor sign-in, one model is served on the user's own hardware, and local image generation is off. This section compares resource use, the plugin ecosystem, command execution, distribution and maintenance with the upstream vendor's desktop application (Table 4).

Table 4: Comparison: the upstream vendor's desktop app vs. Ragon Desktop.

Dimension	Vendor's Desktop App	Ragon Desktop (Ours)
Model Capacity	The vendor's hosted frontier models.	Local 27B-parameter model (Qwen3.8-27B in NVFP4), 262K-token context.
Host Memory Footprint	Thin client; inference in the vendor's cloud.	On a node: model co-located (about 20 GB of weights; half of memory reserved by SGLang). On a client machine: the app and `ragon` only.
Ecosystem & Plugins	Plugin marketplace, the vendor's catalogue and plugin sharing, cloud integrations.	Codex plugins from git or local marketplaces at `/airgapped off`; the vendor's catalogue and sharing closed; plugins off at `/airgapped on`.
Command Sandbox	Read-only, workspace-write or Full Access; `!` commands in the user's shell.	The same at `/airgapped off`; at `/airgapped on`, empty network namespace, host control sockets masked, Full Access and `!` refused.
Code Signing / Trust	Signed and notarized, auto-updating.	Preview: ad-hoc signed on macOS, unsigned on Windows; the `.deb` installs its AppArmor profile; `ragon update` verifies Ed25519-signed releases.
Cross-Device Sync	Sync through the vendor's cloud accounts.	Thread history synced through the user's own node, off until enabled per device; phone access through `ragon web` and messenger bridges.

11.1 Model Scale and Reasoning Asymmetry

The most significant performance asymmetry between the two systems stems from the physical constraints of local hardware versus hyperscale cloud clusters:

1. **Reasoning Depth on Deeply Layered Codebases**: The vendor's application routes prompts to its hosted frontier reasoning models running across clusters of high-bandwidth memory (HBM3e) accelerators. In contrast, Ragon is bound to the physical memory of a single workstation (128 GB LPDDR5X on NVIDIA GB10). Its context is not the constraint: the local model serves a 262,144-token window. The local 27B-parameter model (Qwen3.8-27B in NVFP4) resolved 68 of 100 sampled SWE-bench Verified tasks on this machine (about 69% estimated), a figure that compares configurations rather than a leaderboard score; it can be expected to show lower reasoning capacity than frontier models when synthesizing large-scale multi-module architectural refactorings or resolving subtle type-theoretic invariants.

2. **KV Cache Memory Pressure**: In cloud-hosted architectures, context caching and KV pool allocations are dynamically scaled across multi-tenant GPU nodes. In Ragon, the KV pool is shared by every session on the node, so the launcher sets the automatic compaction limit once per launch to 60% of the server's KV pool: sessions compact at about 60% of the shared pool (about 94K tokens of about 157K) rather than at the 262K window.

11.2 Workstation Resource Competition on Unified Memory

The vendor's application functions as a classic thin client: because all token generation, vector indexing, and embeddings occur in the vendor's cloud, the local desktop application uses little memory and virtually no GPU compute. A developer can run resource-intensive local builds (`cargo build -release`, C++ compilation, or Android emulator execution) without impacting the speed of agent code generation.

Ragon separates a node from a client. A client machine (a laptop or a Mac) runs only the app and `ragon`, as a thin client of a GB10 node it finds on the local network. On the node itself, the whole stack is co-located:

- The model server (SGLang) reserves half of the unified memory for the weights (about 20 GB) and its KV pool; about 39 GB stays available for the host.
- Heavy local compilations compete directly for unified memory bandwidth and CPU cores with the speculative drafting engine (DFlash2).
- Any memory leak or uncontrolled heap expansion in the Electron desktop shell directly threatens workstation stability: an unchecked client process could raise Linux PSI memory pressure until the kernel's OOM killer ends a process, possibly the resident model server.

11.3 Plugins and Multi-Device State

1. **Plugins**: Ragon allows Codex plugins (skills, MCP servers, hooks and apps from git or local marketplaces) while `/airgapped` is off; the upstream vendor's catalogue and plugin sharing are closed at every level. At `/airgapped on` a session starts with plugins disabled, and both enforcers of the bridge policy refuse plugin installs and marketplace fetches, drop plugin items from imports and open every thread without plugins. A plugin's MCP servers and hooks run as the user outside the command sandbox, so Ragon treats them like MCP servers the user configured.
2. **Multi-Device State**: Ragon synchronises threads between the user's desktop, browser and phone through a sync hub on the user's own node. Thread history, titles and archive state reach every paired device; nothing passes a vendor cloud, and the session databases stay local because each device rebuilds them from the synced transcripts. It is off until the user enables it per device. Following live turns and answering them from any paired device is the next phase and is not built yet.

11.4 Command Execution: Codex's Modes, Confinement at `/airgapped on`

`thread/shellCommand` is user-initiated: it runs a `!` command the user typed in the composer, in the user's own shell; the agent cannot issue it. Ragon allows it while `/airgapped` is off, as Codex does, and refuses it at `/airgapped on` in both enforcers. The agent's commands run under the sandbox mode the user picks (read-only, workspace-write or Full Access), as in Codex. Only at `/airgapped on` does Ragon add confinement: an empty network namespace, the host's control sockets masked, and Full Access and `!` refused. Commands that need the network or host daemons (e.g., local Docker sockets) fail only there.

11.5 Code Signing, Packaging, and Enterprise Fleet Deployment

Commercial applications benefit from established enterprise software supply chain infrastructure:

1. **Code Signing and Notarization**: The vendor's desktop client is signed on Windows and notarized on macOS. This enables frictionless double-click installation across corporate device fleets without operating system security warnings.
2. **Friction of Ad-Hoc Signing**: As observed on our macOS deployment (Section 9), Ragon's desktop preview bundles are ad-hoc signed (`flags=0x2(adhoc)`), and Windows builds are

unsigned. On macOS Monterey and later, this triggers Apple Gatekeeper quarantines, requiring developers to manually execute terminal overrides (`xattr -cr`) or navigate system preferences to authorize execution. The desktop app has no in-app updater yet; the command-line binaries update through `ragon update`, which verifies Ed25519-signed releases.

3. **Linux AppArmor Privileges**: Ubuntu 23.10+ refuses a usable user namespace to programs without an AppArmor profile. The `.deb` installs the desktop app's profile itself; an unpacked build without `sudo` falls back to Chromium's setuid sandbox, using the helper an installed Chrome, Edge or Brave provides. The agent's bubblewrap sandbox has no root-free route on Ubuntu 24.04 (the pinned Codex's Landlock fallback refuses restricted filesystems), so it requires a one-time administrative profile per machine, which `ragon` detects and prints; Ubuntu 25.04+ ships one.

11.6 Maintenance Fragility of the Fork-Minimal Architecture

Finally, Ragon's long-term sustainability faces architectural tension against upstream development velocity. The upstream Codex codebase comprises over 120 Rust crates and nearly 2 million lines of code, evolving with dozens of commits daily. While Ragon's 29-patch series (about 57 KB in total) minimizes merge surface area, major architectural refactorings upstream—such as modifying internal message processor state machines or restructuring `app-server-protocol` schemas—demand continuous rebasing, schema reconciliation, and regression testing to prevent protocol desynchronization.

12 Compatibility Status for Users of the Vendor's App

While systems researchers prioritize formal capability proofs and hardware latency invariants, software developers evaluate developer tools primarily through immediate usability, workflow ergonomics, and behavioral predictability. For six behaviours a user of the vendor's desktop app expects, Table 5 gives what Ragon Desktop does and why.

12.1 No Vendor Authentication

A developer coming from the vendor's app finds no vendor account sign-in:

1. **No Vendor Sign-In**: The upstream application includes dedicated login flows (`account/login/start`) supporting web-based OAuth, session tokens, and Enterprise Single Sign-On (SSO). Ragon's patch 0005 removes the sign-in screens, and its patches 0013, 0015 and 0016 close the analytics, vendor network and online diagnostic channels, so no vendor channel remains. `ragon` uses the local or node model server instead, and prompts go to the local model.

2. **Model Provider**: The launcher points `ragon` at the local model server by default. A provider and model the user names explicitly (`-c model_provider=...`, `-model`) are honoured.

12.2 One Served Model, Reasoning Effort, and Images

In commercial agent environments, model routing is treated as a fluid, per-turn preference:

- **The Model Picker**: Upstream users switch between the vendor's lightweight and reasoning models from a picker. Ragon Desktop has a picker too (Ctrl+Shift+M in the composer):

it lists the one served model and lets the user choose its reasoning effort. Because model weights are resident in workstation unified memory (LPDDR5X), swapping models is not an instantaneous API header change; it requires shutting down the active SGLang server, freeing its share of memory, and loading an alternate checkpoint (`ragon-admin server start -model ...`).

- **Images**: Screenshots, mockups and diagrams can be pasted or attached to the prompt, and the local model reads them. Ragon also provides a local image search sidecar (`ragon-admin images`). Local image generation is switched off (`DIFFUSION_ENABLED = False`) to keep unified memory for the model.

12.3 Several Windows on One App-Server

Modern power developers frequently configure dual- or triple-monitor workstations, positioning code editors, terminal multiplexers, and agent chat interfaces across separate displays:

- **Several Windows**: Ragon Desktop opens several windows (*File* → *New Window*, Ctrl/Cmd+Shift+N), each with its own thread or project. Terminal and diff panes detach into their own windows and dock back, and the window layout is restored at start (from 1.7.2). Each window shows *Ask* (conversational assistance) or *Work* (active repository inspection and diff review), and a global shortcut can open a floating Quick Entry box (off by default).
- **One App-Server**: Every window shares the single `ragon app-server` behind a single-instance lock; the main process routes the one vetted IPC channel per window and gives each thread one composer, so a thread open in two windows is mirrored read-only, never double-sent.

12.4 In-Place Edits by Default, Worktrees on Request

Interactive sessions in Ragon (the terminal agent, the desktop app's Work mode and `ragon web`) edit the user's checkout in place by default, as Codex's CLI and app do, so the editor shows every change as it happens. Isolation is opt-in: `/worktree on` or the app's Worktree toggle starts a session in a fresh Git worktree on its own branch. The footer names the folder and branch being edited, and `/worktree apply` brings the changes back as uncommitted changes, refusing on conflict. Night Shift is the deliberate exception: each unattended task and each of its best-of-N attempts runs in its own temporary worktree, and only the chosen result is kept on branch `night/<id>`. Nothing is merged automatically; the morning report names the branch, and `ragon night apply <id>` applies it to the current checkout.

12.5 Network Access in Sandboxed Commands

In cloud-backed agents, shell commands run in environments that permit dynamic package resolution:

- **At `/airgapped off`**: As in Codex, every sandboxed command has the network, Read Only included, so `pip install`, `npm install` and `curl` work inside agent commands.
- **At `/airgapped on`**: Commands execute in a Bubblewrap sandbox with an empty network namespace (`CLONE_NEWNET`). Package managers fail with `Network is unreachable` (`ENETUNREACH`) or DNS resolution errors, and `curl` or `wget` cannot connect. The user chooses this level for work that must not reach the network.

- **Reading the Web**: Ragon's audited `ragon-search` and `ragon-fetch` command-line utilities give the agent web search and page reading.

12.6 Mobile Access and Multi-Device Sync

Finally, modern developers have grown accustomed to ubiquitous cross-device continuity:

- **The Full UI on a Phone**: Upstream vendors provide polished iOS and Android companion apps that synchronize conversation history, active tasks, and code diffs across devices via cloud servers. Ragon has no native mobile app yet. A phone pairs with `ragon web` by QR code and uses its UI in the browser (voice and the clipboard API need a secure context, which plain HTTP on the LAN does not give); an installable web app (PWA) comes first, native apps later. Thread history reaches the phone through the user's own node when sync is enabled.
- **Messenger Bridges**: Self-hosted messenger bridges, off until a command turns one on, carry text, images, files and voice: `ragon signal setup` (restricted to Signal Note-to-Self) and `ragon chat` (Telegram, and Matrix with a homeserver started by `ragon-admin matrix start`). Telegram adds rich blocks and a Mini App that opens the full `ragon web` UI.

13 Related Work

Autonomous Coding Agents. ReAct [15], SWE-agent [14], and OpenHands [13] established agentic scaffolds for code reasoning and tool calling. While SWE-bench [8] standardizes benchmark evaluation, prior literature evaluates headless CLI harnesses. Our work addresses the architectural, display driver, and security challenges of interactive desktop environments.

Multi-Process Browser Security. Barth et al. [1] and Reis et al. [12] formalized process isolation in modern web browsers. Electron security guidance [7] outlines guidelines for CSP and context isolation, while Bubblewrap [11] established unprivileged Linux containerization. We extend these models to air-gap-capable developer workstations.

Hardware-Accelerated Model Inference. Efficient memory serving in vLLM [9] and FlashAttention [5, 6] enabled high-throughput local execution. Quantization via AutoRound [3] and speculative drafting [2, 10] enable interactive 27B-parameter inference on single workstations. Our research integrates these inference runtimes safely alongside hardware-accelerated desktop shells.

14 Discussion and Threats to Validity

Internal Validity. Keystroke latency measurements were collected with WebKitGTK only, on a single NVIDIA GB10 system under driver 580.173.02; the Electron measurements are still to be made. Driver updates could introduce dumb-buffer support on the display node in future releases; however, NVIDIA's preference for its own buffer manager suggests this constraint will persist.

External Validity. Our implementation was targeted at Linux AArch64 workstations. While the Electron/ANGLE architecture generalizes to macOS (via Metal) and Windows (via Direct3D), Linux-specific sandboxing primitives (AppArmor and Bubblewrap) require platform-specific translation on other operating systems.

15 Conclusion

Architecting interactive desktop environments for local, air-gap-capable coding agents requires overcoming critical systems bottlenecks across graphics drivers, multi-process privilege separation, and operating system sandboxing. We showed that a WebKitGTK wrapper reaches 40–50 ms per keystroke frame on a large window under NVIDIA's driver, which refuses DRM dumb buffers, whereas an Electron/ANGLE pipeline restores hardware compositing. By combining a three-tier multi-process architecture with an allowlist capability gatekeeper (`policy.json`), AppArmor user namespace profiles, and an on-demand `strace` network egress audit, Ragon Desktop speaks the same app-server protocol as the vendor's app with the vendor channels removed, keeps Codex-compatible defaults, and confines every sandboxed command to an empty network namespace when the user turns `/airgapped on`.

Data Availability

The source code of Ragon, including the desktop app, the gatekeeper policy and the egress audit described here, is available in the public repository `https://github.com/dreamference/ragon` (AGPL-3.0).

References

[1] Adam Barth, Collin Jackson, Charles Reis, and The Chromium Team. The security architecture of the chromium browser. In *Technical Report, Stanford University and Google*, 2008.

[2] Jian Chen, Yesheng Liang, and Zhijian Liu. DFlash: Block diffusion for flash speculative decoding. *arXiv preprint arXiv:2602.06036*, 2026.

[3] Wenhua Cheng, Weiwei Zhang, Haihao Shen, Yiyang Cai, Xin He, Kaokao Lv, and Yi Liu. Optimize weight rounding via signed gradient descent for the quantization of LLMs. *arXiv preprint arXiv:2309.05516*, 2023.

[4] Codex contributors. Codex CLI: a lightweight coding agent that runs in your terminal. `https://github.com/openai/codex`, 2026. Release `rust-v0.158.0`, commit `064c6b8`.

[5] Tri Dao. FlashAttention-2: Faster attention with better parallelism and work partitioning. In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2307.08691.

[6] Tri Dao, Daniel Y. Fu, Stefano Ermon, Atri Rudra, and Christopher Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2022. arXiv:2205.14135.

[7] Electron Authors. Electron security tutorial and checklist. `https://www.electronjs.org/docs/latest/tutorial/security`, 2026.

[8] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations (ICLR)*, 2024. arXiv:2310.06770.

[9] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large

language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles (SOSP)*, 2023. arXiv:2309.06180.

[10] Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning (ICML)*, 2023. arXiv:2211.17192.

[11] Project Atomic. Bubblewrap: Unprivileged sandboxing tool for linux. `https://github.com/containers/bubblewrap`, 2024.

[12] Charles Reis, Alexander Moshchuk, and Nasko Oskov. Site isolation: Process separation for web sites within the browser. In *Proceedings of the 28th USENIX Security Symposium*, pages 1661–1678, 2019.

[13] Xingyao Wang, Boxuan Li, Yufan Song, Frank F. Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, Hoang H. Tran, Fuqiang Li, Ren Ma, Mingzhang Zheng, Bill Qian, Yanjun Shao, Niklas Muennighoff, Yizhe Zhang, Binyuan Hui, Junyang Lin, Robert Brennan, Hao Peng, Heng Ji, and Graham Neubig. OpenHands: An open platform for AI software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024.

[14] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. SWE-agent: Agent-computer interfaces enable automated software engineering. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2024. arXiv:2405.15793.

[15] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*, 2023. arXiv:2210.03629.

Table 5: Compatibility status: the upstream vendor's app vs. Ragon Desktop.

Category	Vendor's App	Ragon Desktop Behavior	Status / Mechanism
Vendor Authentication	Sign-in with a vendor account.	No vendor account: `ragon` uses the local or node model server, and prompts go to the local model.	By design: the sign-in screens are removed (patch 0005), and no vendor channel remains (0013/0015/0016).
Model Selection	Picker across the vendor's hosted models.	Picker lists the one served model and its reasoning efforts (Ctrl+Shift+M).	By design: one resident model; another model needs `ragon-admin server start -model`.
Window Management	Multi-window support; detachable terminal and diff panes.	Several windows (*File* → *New Window*, Ctrl/Cmd+Shift+N), each with its own thread or project; terminal and diff panes detach into their own windows and dock back; window layout restored at start (from 1.7.2).	No longer a gap: every window shares the single `ragon app-server` behind a single-instance lock; the main process routes the one vetted IPC channel per window and gives each thread one composer, so a thread open in two windows is mirrored read-only, never double-sent.
Filesystem Mutation	Direct in-place editing of files in the developer's checkout.	In place by default (terminal agent, Work mode, `ragon web`); a fresh Git worktree is opt-in.	Same as Codex. Night Shift is the exception: each unattended task runs in its own worktree, and the chosen result is kept on branch `night/<id>`.
Networked Tooling in Sandbox	Inline `pip install`, `npm install`, or `curl` inside agent commands.	Network fails with `ENETUNREACH` only at `/airgapped on`; below it every sandboxed command has the network, Read Only included.	Air-gapped network namespace (`CLONE_NEWNET`) at `/airgapped on`; reading the web via `ragon-search`/`ragon-fetch`.
Mobile Experience	Official iOS/Android apps with cloud conversation synchronization.	`ragon web` on a paired phone (QR pairing); messenger bridges (Signal, Telegram with a Mini App, Matrix) carrying text, images, files and voice.	An installable web app (PWA) first, native apps later; threads synced through the user's own node.