

Inside the Code Split: A Chunk Census of Claude Desktop 1.19367.0

Release 1.19367.0 split the desktop app’s main process from one 15 MB file into a tiny stub, one 7.7 MB core, and 44 lazy-loaded satellite chunks. This report names every satellite: the agent-session platform behind Claude Code and Cowork, the enterprise sign-in flows, the built-in MCP servers, the Bluetooth bridge behind the Claude Desktop Buddy gadget, and the one satellite the Linux repackaging actually patches.

DOCUMENT	CDL-ANT-0010 · Rev A · FINAL
SUBJECT	Census of the 44 satellite chunks introduced by upstream’s 1.19367.0 main-process code split
FOCUS	What each chunk is scoped to, what loads it at runtime, and what the split means for downstream patching
SCOPE	Official <code>claude-desktop_1.19367.0_amd64.deb</code> (SHA-256 pinned in <code>scripts/setup/official-deb.sh</code>), <code>.vite/build/</code> main-process tree only
TOOLS	<code>ar+tar+@electron/asar</code> extraction, prettier beautification, deterministic require-graph mapping, 44 parallel model-assisted chunk analyses with structured output, selective hand re-verification
HEADLINE	The split is bundler mechanics, not a rewrite: 37 of 44 satellites are first-party feature islands loaded on demand, five carry vendored library code, two are tiny shared helpers, and the boundaries read as a feature inventory, from the agent platform to surfaces still shipped dark.
AUTHOR	Claude Fable 5 · July 12, 2026
REVIEWED BY	Aaddrick Williams

One File Became Forty-Six

Before release 1.19367.0, every line of Claude Desktop’s Electron main process, the privileged Node.js side of the app that owns windows, spawns processes, talks to the OS, and brokers every MCP connection, shipped as a single minified 15 MB file: `.vite/build/index.js`. In 1.19367.0 that file became a 700-byte stub whose only real job is one `require()` call. The code moved into `index.chunk-CNXUb5h4.js`, a 7.7 MB core that still holds the always-running app, plus **44 content-hashed satellite chunks** totaling about 1.95 MB that load lazily, each the first time its feature is touched.

This project repackages Anthropic’s official Linux `.deb` into the formats Anthropic does not serve (RPM, Appliance, Nix, AUR), shipping the official application bytes unmodified except for four small Linux-specific patches. Those patches anchored on the old single-file path, so the split broke our build (fixed in PR #793). Repairing it meant mapping the new layout anyway, and the map turned out to be interesting on its own: **the chunk boundaries are a de-facto feature inventory of Claude Desktop**, drawn by the bundler along the exact seams where the app defers work until a feature is used.

The census answer, in one paragraph: 37 of the 44 satellites are first-party feature code. The biggest cluster is the agent-session platform behind Claude Code Desktop and Cowork (worktree management, terminals, remote SSH targets, session secrets, usage probes). A second cluster is enterprise sign-in (OIDC discovery, Microsoft Entra, GCP Workforce Identity, Anthropic Console key minting). A third is the extension system (MCPB bundles, auto-updates, marketplace migrations). Five chunks carry vendored open-source libraries, two are tiny shared helpers, and the tail holds the app’s most tangible features: the Bluetooth bridge behind Anthropic’s open-source *Claude Desktop Buddy* desk gadget, and an *Imagine* visualization MCP server that still appears to be shipped dark behind feature flags.

Reading guide. Terms that recur: **MCP** is the Model Context Protocol, the plugin standard Claude apps use to talk to external tools. **MCPB** (MCP Bundle) is Anthropic’s packaged-extension format, successor to the older **DXT** desktop extensions. **Cowork** is Claude Desktop’s autonomous-agent workspace. **CCD** appears throughout upstream’s own strings as the tag for Claude Code Desktop, the embedded Claude Code sessions in the app. A **git worktree** is a disposable checkout the app gives an agent session so it can edit code without touching your working copy. Chunk names below are shortened: 8hARSK4E means `index.chunk-8hARSK4E.js`.

How the Census Was Taken

Everything was measured against the official artifact, not our repackage: the SHA-256-pinned `claude-desktop_1.19367.0_amd64.deb` from Anthropic’s APT pool. The `app.asar` archive was extracted with `@electron/asar`, all 44 satellites were run through `prettier` for readability, and the `require` graph between chunks was computed deterministically with `grep` over the minified bytes. Each satellite then got its own analysis pass: 44 parallel model-assisted reviews (Claude Sonnet subagents), each reading one beautified chunk plus its position in the `require` graph, and returning a structured verdict: what the chunk is, what runtime path loads it, and the literal strings that prove it (log prefixes, IPC channel names, endpoint URLs, store filenames).

The evidence rule was strict: a claim counts only if the string is actually in the file. Where a verdict looked thin it was re-verified by hand against the beautified source; the per-chunk descriptions below fold in those corrections. Sizes are minified bytes as shipped. One caveat applies globally: *what triggers loading a chunk* is inferred from the require graph and the chunk’s own strings, not from tracing a live process, so trigger descriptions are confident but not instrumented.

A Stub, a Core, and 44 Satellites

The entry file upstream’s Electron app boots is now almost empty. Minus the Sentry crash-reporting banner that is stamped into every chunk, this is the entire file:

```
// .vite/build/index.js, 1.19367.0 (Sentry banner elided)
require("node:child_process"); require("node:path");
require("node:process");
require("./index.chunk-CNXUb5h4.js"); // <-- the actual main process
require("electron"); require("node:crypto"); require("node:os");
```

index.chunk-CNXUb5h4.js is the old index.js in everything but name: 7.7 MB, always loaded, owning windows, the tray, settings, networking, and the MCP host. The 44 satellites hang off it. 31 of them are required directly by the core; the rest chain through intermediate satellites. Two chunks act as hubs: CVKCxtdY, the Claude Code session backend, pulls in eleven other satellites, and DmcY3fXx, the session API layer, pulls in five. One satellite (BSeLkQD3, the scheduled-tasks handlers) is referenced by nothing statically and is reached through a computed dynamic import.

The 1.19367.0 main-process bundle by the numbers. Sizes are minified bytes inside app.asar; renderer and preload bundles are out of scope.

PIECE	SIZE	ROLE
index.js	0.9 KB	boot stub, one require()
index.chunk-CNXUb5h4.js	7.7 MB	the always-loaded core (old index.js)
44 satellite chunks	1.95 MB	lazy feature islands, this report’s subject
37 of 44		first-party app features
5 of 44		vendored library code (one mixed with feature code)
2 of 44		tiny shared helpers

Why did upstream do this? Almost certainly not by hand: the boundaries match dynamic import() sites, which Vite/Rollup splits into separate files by default. The win is startup cost: the app no longer parses SSH transports, extension packers, and BLE bridges at boot; it parses them the first time you use them. Nothing behavioral moved, which is also why our whole-bundle tripwires kept passing while every patch anchor silently relocated.

Libraries and Dev Tooling

Six chunks are mostly or entirely other people’s code (or Anthropic’s own published packages), pulled out of the

core so their weight is only paid on use.

z2KVH7ws • MCPB extension toolkit, 430 KB

A vendored copy of Anthropic's published `@anthropic-ai/mcpb` package, the toolkit for MCP Bundles: manifest schema validation, packing and unpacking `.mcpb` archives, signing and signature verification, and dependency pruning. It even carries the package's interactive scaffolding wizard (built on `@inquirer/core`), which a desktop app will likely never show. It loads whenever the app needs to validate, verify, or unpack an extension bundle, which makes it the trust root of the extension install path: the signature check that decides whether an extension is allowed to install lives here.

DSARmh3w • electron-devtools-installer, 138 KB

The stock npm package that downloads Chrome extensions (React DevTools and friends) from Google's Web Store endpoint and loads them into Electron. It is reachable only through `B43fo1oX` below, on a development-profiling path; a normal user install should never execute it.

B43fo1oX • React DevTools loader, 2.5 KB

First-party glue in front of the previous chunk: tries `electron-devtools-installer` first, then falls back to scanning a locally installed Chrome profile for the React DevTools extension ID. Every log line is prefixed `[profile]`, marking it as an internal profiling convenience that ships in the production bundle but stays dormant.

Gd80sLOY • Zod re-export barrel, 4 KB

Zod is the schema-validation library used across the app; its implementation lives in the core chunk. This satellite is a pure re-export shim, about eighty `exports.X = core.X` lines, that exists only because Zod's package entry point became its own module boundary during the split. It does nothing, but it is a nice fossil of how mechanical the split was.

DL06Ax9W • shellQuote, 0.7 KB

A single function that POSIX-quotes a string for safe shell interpolation. Three different chunks (the SSH stack, the PTY manager, the session backend) require it, which tells you those are the places the app builds command lines from user-derived paths.

DuRzChhd • Sentry banner stub, 0.6 KB

Nothing but the Sentry release-ID and debug-ID stamp that the build pipeline injects into every chunk for crash symbolication; this one happens to contain nothing else. Included for completeness, and as the baseline noise floor: when a chunk below is described as small, subtract this banner from its size mentally.

05 • AGENT SESSIONS

The Claude Code / Cowork Platform

The largest first-party cluster, eleven chunks, is the machinery behind agentic coding sessions: what happens when you point Claude Desktop at a repository and let it work. The architecture that emerges from the strings is concrete: every session gets an isolated git worktree from a managed, reusable pool; sessions run against local folders, SSH hosts, or WSL; transcripts live in `~/.claude/projects` as JSONL; and a governor evicts idle sessions

under memory pressure.

CVKCxtdY • Claude Code session backend, 374 KB

The heart of the cluster and the third-largest satellite. It owns the full session lifecycle: start-to-outcome telemetry for every message cycle, a session governor with soft-cap eviction, tail-loading of large agent-*.jsonl transcripts, tool-permission auto-approve and deny logic (including a computer-use permission path), GitHub PR-check polling with rate-limit handling, and the wiring that exposes scheduled tasks, transcript search, and session archiving to the model as MCP tools. Eleven other satellites are pulled in through it; if you are using Claude Code inside the desktop app, this chunk is resident.

DmcY3fXx • Session API layer, 46 KB

`createLocalSessionsApi`, the IPC-facing service the UI calls to start sessions and send messages, with one abstraction covering three target types: local folders, remote SSH hosts, and WSL distributions. It handles SSH connection tests and password prompts, WSL path resolution, remote git diff and commit-log retrieval, and workspace-trust checks; strings like *teleport to cloud* mark the handoff hooks that move a local session to a hosted one.

8hARSK4E • Git worktree manager, 64 KB

A `GitWorktreeManager` class that creates, tracks, reaps, and migrates per-session git worktrees, persisted in a `git-worktrees.json` store with corruption recovery (a corrupt store is preserved for forensics and rebuilt empty, per its own log message). It runs git through a PATH- and SSH-agent-aware spawner with prompts disabled (`GIT_TERMINAL_PROMPT=0`), parses diffs and commit logs, and enforces a workspace-trust gate before touching a repo. It also scans diffs for *dangerous feature sources*, a defensive check on what an agent brings into your tree.

CDs06A0Q • Worktree pool, 10 KB

A performance layer over the worktree manager: a pool that leases worktrees to sessions and keeps a configurable number of *warm* pre-created ones so a new session starts instantly. A periodic sweep reaps idle worktrees and resets dirty ones. Feature-flag gated, which suggests upstream was still rolling it out at this release.

BhLwdw68 • Session diagnostics and stats, 21 KB

Three jobs in one chunk. First, a giant error classifier that maps raw crash strings (exit codes, Windows NTSTATUS values, spawn failures, proxy and network errors) into stable category tags for telemetry, which is a readable catalog of every way a Claude Code session has died in the field. Second, the usage-stats aggregator that reads your local transcripts and a stats cache to compute daily activity, per-model token usage, and streaks. Third, the SFTP uploader that copies chat attachments to a remote host's `.claude/uploads` before a remote session references them.

BeyKyjh2 • Claude Code config bridge, 11 KB

The shared reader/writer for the Claude Code CLI's own configuration: it parses `~/ .claude.json` with lock-file-protected atomic saves and backups, tracks per-project trust-dialog acceptance, lists trusted workspaces, and extracts MCP server entries. It also hand-rolls a git-config/INI parser to resolve repo root, branches, and remotes, including worktree indirection. This is the seam where the desktop app and the standalone CLI share state.

BYTSEZgv • Terminal PTY manager, 19 KB

ShellPtyManager, the pseudo-terminal engine behind the integrated terminal panel: spawns interactive shells via node-pty (including a hardened SSH remote-shell path that validates host, port, and identity file), keeps a bounded 256 KB ring buffer per session, strips ANSI escape sequences, and kills every live process through a registered quit handler on app exit.

DuBC1uXg • Agent SDK toolset, 19 KB

The local tool implementations an agent session actually calls: sandboxed bash, read, write, edit, glob, and grep tools (exported in the SDK's beta*Tool naming, bundled as toolset 20260401), plus skill-archive extraction and version resolution. When Claude edits a file on your disk during a session, the code doing it is here.

iS43ruUD • Session secrets materializer, 3 KB

For enterprise (third-party provider) deployments only: writes short-lived credential files under a per-session private directory so a spawned CLI process can read them, with generation tracking against stale writes and a boot-time sweep that clears the whole secrets root on startup. Small chunk, careful code; the sweep-on-boot detail is the kind of hygiene you want to see around credentials on disk.

DIyoLvd8 • Usage probe, 5 KB

Feeds the usage and rate-limit UI by spawning the host-installed Claude Code CLI in a no-tools throwaway session and calling its usage API, whose actual exported name is usage_EXPERIMENTAL_MAY_CHANGE_DO_NOT_RELY_ON_THIS_API_YET. Wrapped in five-minute success caching, backoff, a rate floor, and a shape-drift error class for when that experimental API inevitably changes.

DPyL7Qop • File-open consent dialog, 1.4 KB

A single confirmation dialog, *File access request*, shown before a session opens a generated or referenced file with your OS default application, with de-duplication so repeat requests for the same file do not re-prompt. A one-purpose chunk that exists purely as a human-in-the-loop gate.

06 • AUTONOMY & REACH

PR Automation and Remote Machines

Eight chunks extend sessions in two directions: unattended operation against GitHub, and execution on machines that are not the one running the app. The PR-automation pair is the part most users will not know exists.

DDPvNCKb • AutoFixEngine, 8 KB

A background engine that sweeps active coding sessions that have auto-fix enabled and an attached GitHub pull request. It polls the PR's checks, review comments, and merge state, and when it finds newly failing CI, a merge conflict, or an unaddressed reviewer comment, it wakes the session by injecting a synthesized <ci-monitor-event> prompt telling Claude Code to fix the checks and reply to the review threads. In other words: the desktop app ships a CI babysitter that argues with your reviewers for you, opt-in.

BgxihPuA • AutoArchiveEngine, 4 KB

The companion janitor: every five minutes (verified in source: a 300-second sweep interval), if the

`ccAutoArchiveOnPrClose` preference is on, it archives sessions whose pull requests have all reached a terminal state (merged or closed), skipping running or exempted sessions. Together with `AutoFixEngine` it gives sessions a full unattended lifecycle: work, respond to CI, get archived when the PR lands.

BsymZHkE + Ds_EujvB • PR-state helpers, 0.8 + 1.4 KB

Two micro-chunks of shared vocabulary: predicates for terminal versus open PR states (MERGED/CLOSED vs OPEN/QUEUED), and normalizers that fold GitHub's inconsistent REST and GraphQL mergeable-state answers into one enum. Three different feature chunks share them, which is why the bundler gave them their own files.

BfcG_552 • SSH transport + remote deploy, 427 KB

The second-largest satellite fuses a vendored, patched copy of the `ssh2` npm package (a complete SSH client: key parsing for RSA, Ed25519, OpenSSH and PuTTY formats, packet framing, auth flow) with the first-party code that uses it: detect the remote OS and shell, download or upload the Claude Code CLI binary to the host, write a session token, start a remote daemon, and expose a controller API. This is the chunk that turns *Claude Code on this laptop* into *Claude Code on that server*.

Cc0Cjw5X • Remote controller barrel, 1 KB

A 45-line pass-through that re-exports the remote-server controller getters from the chunk above. It exists so the core can reference the controller without statically dragging in 427 KB of SSH; the price of one indirection buys lazy loading of the whole transport.

Ch2QrYhw • SSH MCP proxy, 7 KB

`SshMcpServerManager`: runs your local stdio MCP servers on the remote host instead, bridging their JSON-RPC over the SSH connection. It filters which environment variables are forwarded to an allowlist, and refuses to proxy servers whose command is a laptop-local absolute path (`/Users/...`, `/opt/homebrew/...`) that the remote could never resolve, a small design detail that prevents a whole class of confusing failures.

r07KBe07 • Remote plugin sync, 5 KB

Before hooks and tools run on a remote host, this chunk walks your local Claude Code plugin directories, content-hashes each, tars and SFTP-uploads them to `.claude/remote/plugins/<hash>` on the host, and extracts them there. It guards against symlinks and path-escaping archive entries, the classic tar-extraction attack shapes.

07 • EXTENSIONS

Install, Update, Migrate

Five chunks manage the desktop extension system (DXT, now MCPB bundles): how extensions update themselves and how upstream is migrating locally stored plugin data to account-level backend storage.

Brn8atpb • Extension update cache, 4 KB

Downloads an extension update from the registry, unzips it to a temp dir, validates the manifest, verifies the package signature (lazily pulling the 430 KB MCPB toolkit only when a signature actually needs checking), enforces the enterprise `signatureRequired` policy and a blocklist, then stages the verified files in a per-extension cache with a metadata sidecar. Notably, policy enforcement happens at cache time, before anything is installed.

Cy4j0ZYe • Extension auto-updater, 5 KB

The loop that uses that cache: every six hours it checks each installed extension for a newer version and stages updates; at startup it applies them, renaming the old extension directory to a timestamped backup and rolling back on failure. Extensions you side-loaded (IDs starting `local.dxt`, `local.mcpb`, `local.unpacked`) are skipped, so dev builds never get clobbered.

PJQ5N4g6 • Org update consent dialog, 2 KB

One localized message box, loaded only when an update arrives for an *organization-provided* extension: install or skip, with the org's name in the prompt. Org-pushed code updates get an explicit human gate that ordinary registry updates do not.

BNkiwKJJ + DNqqgru0 • Marketplace migrations, 4.7 + 5.7 KB

Two one-shot, feature-flagged startup migrations (both gated by the `claudeai_cowork_backend_marketplaces` flag, both leaving run-once sentinels): one zips locally stored uploaded plugins and re-registers them as account-level remote plugins; the other re-registers locally known marketplace entries (sanitized GitHub URLs only) with the backend. Together they mark a clear direction: Cowork plugin state is moving off your disk and into your account.

08 • ENTERPRISE SIGN-IN

Seven Chunks of OAuth

Seven small chunks, all sharing the `custom-3p` log prefix (third-party), implement authentication for enterprise deployments where Claude Desktop talks to a customer's own model gateway (Bedrock, Vertex, Azure) instead of Anthropic's API. Consumer users never load any of them; together they map exactly which identity systems upstream supports.

BBdaXF74 • OIDC discovery, 2.5 KB

The shared foundation: fetches an identity provider's `/.well-known/openid-configuration`, enforces HTTPS on every endpoint (with a deliberate localhost exception for testing), and warns when a provider's token endpoint lives on a different host than its issuer. The two grant-flow chunks below build on it.

5ms3G7cx • External IdP grant, 3.5 KB

The generic flow: a loopback-PKCE browser sign-in against whatever OIDC provider the org configured for its inference gateway, plus silent refresh that distinguishes *fresh*, *no id token*, *invalid grant*, and *transient* outcomes so the UI can tell a re-login from a blip. Google gets special-cased offline-access handling.

CttfaNvQ • GCP Workforce Identity, 4 KB

The Vertex AI variant: after the OIDC sign-in, it exchanges the resulting ID token at Google's STS token-exchange endpoint for a GCP access token (Workforce Identity Federation), so the app can call a customer's Vertex deployment without a Google account in the loop.

DMU53tq3 • Microsoft Entra flows, 7 KB

The Azure variant: device-code grant, browser PKCE grant, and refresh-token exchange against Microsoft En-

tra ID (Azure AD), requesting the Cognitive Services scope used by Azure-hosted model endpoints (Azure AI Foundry). Tokens come back tagged with tenant and client IDs.

oBJ0NwSq • Console key minting, 3 KB

The Anthropic-native variant: *Sign in with Console* runs a PKCE flow against the Anthropic Console's OAuth endpoints, then calls a `create_api_key` endpoint to mint a real API key on your Console org and caches it. The desktop app can bootstrap its own API credentials from a browser login.

Bft8Ec9w • Device-code window, 2 KB

A fixed-size, always-on-top *Verify sign-in code* `BrowserWindow` used by the device-code flows above; it loads the deployment's verification page and exists so the code prompt cannot get lost behind the main window.

K1Q8_qz3 • Auth-expiry toast, 2 KB

The failure UX for all of the above: when a credential expires or an org policy returns a 403, this shows either a *Sign in again* toast with a re-auth action or a non-actionable *access denied by your organization* notice, de-duplicated per credential epoch so a burst of failed requests produces one toast, not twenty.

09 • BUILT-IN SERVERS

MCP Tools, a VM Prefetcher, and a Gadget

The last seven chunks are the most fun: MCP servers the app hosts *inside itself* to give the model extra abilities, a bandwidth-saving VM prefetcher, the desktop end of Anthropic's hardware companion, and one surface that still appears to be unannounced.

BG2Aybw_ • The Imagine server, 252 KB

The fourth-largest satellite is an internal MCP server named `visualize` that gives the model a `show_widget` tool for rendering interactive SVG/HTML visualizations inline in chat, plus a `read_me` tool that serves the model its own instruction manual. Nearly all of the 252 KB is embedded documentation payload: a Claude Design System token vocabulary, an *Imagine, Visual Creation Suite* guide, and form-building guidance, alongside a CDN allowlist (`esm.sh`, `cdnjs`, `jsDelivr`, `unpkg`, Google Fonts) for what a rendered widget may load. It is double-feature-flag gated and restricted to Cowork and Claude Code session types: infrastructure for a richer in-chat canvas, shipped dark.

D3JJQa7m + BSeLkQD3 • Scheduled tasks, 15 + 5 KB

A pair implementing recurring automation: an internal MCP server exposing `list_scheduled_tasks`, `create_scheduled_task`, and `update_scheduled_task` tools (so the model itself can set up cron-style or one-time jobs, each stored as a `SKILL.md` under its task directory), and the main-process dispatch handlers that validate the cron expressions and paths behind those calls. The handler chunk is the one satellite reachable only through a dynamic import, loaded when the scheduled-tasks surface is first used.

Bv2T366m • read_terminal tool, 4 KB

A one-tool MCP server that lets a Claude Code session read the last N lines of your integrated terminal panel, with escape sequences stripped and an optional wait-for-new-output mode. Gated to local (non-SSH) Claude Code sessions and a feature flag. Small, but it closes a real loop: the model can now see the output of the terminal

you are typing in.

Dy70DfSE • Hardware Buddy bridge, 15 KB

The desktop end of *Claude Desktop Buddy*, Anthropic’s open-source BLE desk companion (an ESP32 gadget that shows Claude’s status and lets you approve tool permissions from your desk; the reference firmware and Bluetooth API live at `anthropics/claude-desktop-buddy` on GitHub). This chunk is the app-side bridge: it pairs and scans via Electron’s Bluetooth APIs, opens a dedicated `buddy_window`, streams a live summary of your running agent sessions to the device (running and waiting counts, latest transcript snippet, pending permission approvals, daily token usage), and uploads custom *character* packs of GIFs and text (capped at 1.8 MB) in chunked, acknowledged BLE writes. In-code the device is affectionately a *nibblet*. Opt-in behind a remote feature flag and a `hardwareBuddyEnabled` preference; on Linux this is also a surface worth watching, since Electron’s Bluetooth chooser has its own portal quirks under Wayland.

DPIWtp-Y • Cowork VM warm download, 8 KB

Cowork’s Linux VM image is a multi-gigabyte download, so this chunk prefetches the *next* version in the background while you work: it pulls `zstd`-compressed bundle files from `downloads.claude.ai`, verifies SHA-256 streaming, requires a 5x free-disk margin before starting, and atomically promotes the warm copy when the app update actually lands. For this repository it has a second significance, covered in the closing section: it is the only satellite our patch suite touches.

DVe-Qi89 • TLS fallback fetch, 4 KB

A Node `https` fetch used when a request must bypass Electron’s Chromium networking stack. The clever part: it pins the TLS leaf certificate to the SHA-256 fingerprint Chromium already validated for that host, so stepping outside the browser stack does not mean trusting a different certificate chain. Capped at 4 MB bodies, refuses redirects and event streams; a narrow escape hatch, deliberately kept narrow.

10 • IMPLICATIONS

What the Map Is Good For

For users, the census is a rare unobstructed look at where Claude Desktop is going. The chunk boundaries say what upstream considers optional (everything above loads on demand), the feature flags say what is gated or not finished (the Imagine canvas, worktree pooling, backend marketplaces), and the string literals say how the shipped features actually behave: sessions live in disposable git worktrees from a warm pool, an opt-in engine will respond to your CI failures and reviewer comments autonomously, extensions are signature-checked before they are staged, and enterprise deployments get real OIDC plumbing rather than pasted API keys.

For this repository, the practical findings are narrower. First, the split is mechanical: every one of our patch anchors survived byte-for-byte, they just moved files, which is why the fix in PR #793 is a resolver (`_resolve_main_js` follows the stub’s `require()` to the real core chunk) rather than new patches. Second, exactly one satellite matters to patching today: the Cowork warm-download chunk, whose prefetch our `cowork-bwrap` patch suppresses on setups where the multi-gigabyte pull is unwanted; it is resolved by its stable `[warm]` log literal, not its content hash, because **every chunk hash in this report rotates on every release**. Third, the census sets the watch list: features that live in always-loaded core code stay in the core, but anything lazy (VM downloads, worktrees, SSH)

can migrate between satellites in any future release, so file paths are never load-bearing, only string anchors are.

The durable point: 1.19367.0 looks like a big structural change and is not one; it is the same application with its lazy seams made visible. The visibility is the value. One release's bundler flag turned a 15 MB blob into a labeled map of Claude Desktop's present features and near future, and this report is that map, written down.