Cupertino v1.0.0 "First Light"

The first release I'd actually call stable. Apple's documentation, search that doesn't suck, and an MCP server that an AI agent can read from before it answers.

If you've ever asked an LLM "what does Swift's `Task` do?" and got back something about Mach kernel context switches, you've met the problem cupertino solves. Models hallucinate Apple APIs because their training data is stale, partial, and weighted toward whatever prose happened to mention a symbol — not toward Apple's actual documentation. The fix isn't a bigger model. The fix is making the documentation **available to the model** at inference time, ranked correctly, served over a protocol the model already speaks.

That's cupertino. **v1.0.0 "First Light"** is the first release I'm willing to call stable across the whole pipeline: crawl, index, rank, serve, distribute. Before today every previous release had at least one of those broken. Today none of them are.

The headline: search now returns the right answer

Run this against an installed v1.0:

```
$ cupertino search Task --limit 3
Searched: packages, apple-docs, swift-evolution

[1] Task | Apple Developer Documentation  •  source: apple-docs  •  score:
0.0492
    apple-docs://swift/documentation_swift_task
    A unit of asynchronous work.

[2] task | Apple Developer Documentation  •  source: apple-docs  •  score:
0.0484

apple-docs://kernel/documentation_kernel_kernel_resource_sizes_data_t_1586795-ta
sk

[3] task  •  source: apple-docs  •  score: 0.0476
    apple-docs://foundation/documentation_foundation_urlprotocol_task
```

That's the canonical Swift `Task` struct at #1, where it belongs. The same binary against the same database before today's changes returned a Mach kernel C function as the top result. Cupertino's pre-1.0 ranker — and frankly every documentation search I've seen, including Apple's own — was wrong on common type names. Type "Task", get a kernel essay. Type "View", get a DeviceManagement payload schema. Type "Result", get a Vision associated type.

Multiple compounding bugs caused this. Fixing them was most of what this release is about.

Reciprocal rank fusion that actually understands the query

Cupertino's `cupertino search` (no `--source` flag) fans out across every available corpus in parallel — apple-docs, the Apple Archive, HIG, swift-evolution, swift.org, the Swift Book, the curated packages corpus, the Apple sample-code corpus — and fuses the per-source rankings via **reciprocal rank fusion** (Cormack/Clarke/Büttcher, 2009). The math is `1 / (k + rank)` per source, summed across sources. RRF is the right shape for prose-vague queries where any source might hold the answer ("how do I cancel an async operation"). It is the *wrong* shape for symbol queries where the answer is unambiguously in apple-docs.

The pre-1.0 default was uniform RRF. Every source's rank-1 contributed `1/61 ? 0.0164`. Three sources tied at 0.0164 on a query like `Task`, the alphabetic tiebreak buried apple-docs, and you got "Common Tasks in OS X" from the Apple Archive instead of the Swift `Task` struct.

v1.0 splits the routing decision by query shape. **Symbol-shaped queries** — single token, ASCII identifier, leading uppercase, the kind of thing a Swift developer types on muscle memory — only fan out across `apple-docs + swift-evolution + packages`. The four prose-shaped sources don't even participate in the fusion. Plus an authority weight on RRF: apple-docs gets `3.0`, swift-evolution and packages `1.5`, swift.org and the Swift Book `1.0`, the Apple Archive and HIG `0.5`. apple-docs's rank-1 fuses to `3.0/61 ? 0.0492` and beats peer rank-1's `1.0/61 ? 0.0164` without alphabetic tiebreaks deciding anything.

The `Searched:` line in the output above is the routing decision made visible. It listed only `packages, apple-docs, swift-evolution` because `Task` looks like a symbol. Type `cupertino search "swiftui state management"` and you'll see `Searched: hig, samples, apple-docs, swift-evolution, packages` — five sources, broader fan-out,

because the query is prose.

Inside apple-docs: BM25F, AST symbols, and a ladder of post-rank heuristics

Per-source ranking inside apple-docs is **field-weighted BM25 (BM25F**, Robertson/Zaragoza/Taylor 2004) over an 8-column FTS5 index — `uri`, `source`, `framework`, `language`, `title`, `content`, `summary`, `symbols`. Title gets weight 10×, AST-derived symbols 5×, summary 3×, framework 2×, body and metadata 1×. The `symbols` column is new in v1.0: it's populated by a Swift AST extractor that runs over both code blocks AND declaration lines on every page. So a query like `Observable` ranks the SwiftUI macro page above prose mentions of the word "observable" in unrelated articles.

On top of raw BM25F, there's a ladder of post-rank heuristics that handle pathological cases the indexer alone can't fix:

> **Title-suffix awareness.** Apple writes `<canonical type name> | Apple Developer Documentation` only on the parent landing page of a type. Sub-symbols (properties, methods, nested types) get clean titles. The suffixed page's raw BM25 is *worse* because the suffix dilutes title term-frequency over field length, so a 50× boost is applied to suffixed exact-title matches, while clean-titled siblings keep the existing 20×. Flips canonical-vs-sub-symbol order without touching BM25F.
>
> **Exact-title peer tiebreak.** `Result` matches three apple-docs pages exactly: Swift's `Result` enum, Vision's `VisionRequest.Result` associated type, and Installer JS's runtime `Result`. All three get the same 50×, so BM25F decides — and BM25F has no opinion. v1.0 adds two orthogonal signals: URI simplicity (`documentation_FRAMEWORK_QUERY` is the framework's top-level type page; anything deeper is a sub-symbol) and a small framework-authority map (`swift` 0.5, `swiftui` 0.7, `foundation` 0.7; `installer_js` and `webkitjs` mildly demoted). Fires only inside the exact-title branch — does not crowd out framework-specific symbol queries (`VisionRequest` still resolves to `vision/VisionRequest`).
>
> **Force-include canonical type pages past fetchLimit.** Some canonical pages get buried by raw BM25 length-normalization — Foundation `URL` lands at raw BM25 position 1017, Swift `Identifiable` at 2577, Foundation `Data` past 3000 — well past the 1000-row over-fetch the post-rank multipliers operate on. v1.0 hand-fetches `apple-docs://FRAMEWORK/documentation_FRAMEWORK_QUERY` directly by URI for the top-tier frameworks (swift, swiftui, foundation) and force-prepends them. O(1) per probe via `docs_metadata`'s primary key.

The combined effect: against the v1.0 corpus (405,782 documents across 422 frameworks), **34 of 35 canonical type queries land their canonical apple-docs page at fused #1**. The holdout, `Stack`, doesn't have a canonical Swift / SwiftUI / Foundation type — TVML's Stack is the right answer, and that's what comes back.

The packages corpus gets the same treatment

Same BM25F-buries-the-canonical-repo problem on `package_files_fts`. Type `vapor middleware` and pre-1.0 you got `apple/swift-openapi-generator` because that repo has long prose mentioning Vapor and middleware in the same article; the actual `vapor/vapor` repo's README ranked second. Type `swift testing` and you got `pointfreeco/swift-dependencies` because that package's testing article has more keyword density than the actual `swiftlang/swift-testing` repo.

The fix mirrors the apple-docs canonical force-include. When the query tokens — joined with dashes for multi-word queries, or as single tokens for single-word queries — match an indexed `repo` name exactly, force-fetch that repo's top BM25 file and prepend it. Two priority tiers: dashed forms ("swift-testing") beat single tokens ("swift") so a multi-word query that has a dashed canonical

doesn't *also* pull in the broader-corpus repo via the bare token. `swift testing` resolves to `swiftlang/swift-testing`, not `swiftlang/swift`. `vapor middleware` resolves to `vapor/vapor` because no `vapor-middleware` repo exists; the single-token fallback finds `vapor`.

The probe is bounded — it only fires when there's an indexed canonical match, so queries that genuinely don't have a canonical repo (`alamofire request` against a corpus where Alamofire isn't indexed; `actor isolation` which isn't a repo name at all) fall through to plain BM25 with no overhead.

A SQL gotcha that cost us ~3 seconds per query

While testing the canonical force-include, I shipped it with the wrong SQL shape:

```
SELECT ... FROM docs_fts f
JOIN docs_metadata m ON f.uri = m.uri
WHERE f.uri = ?
LIMIT 1;
```

`docs_fts` is an FTS5 virtual table. Its `uri` column is *not* a queryable index — FTS5 stores columns inside the inverted index, accessible only through `MATCH`. SQLite's planner picked `SCAN VIRTUAL TABLE INDEX 0` and walked the entire FTS5 index for every probe. About **3.2 seconds per probe** on the 3.4 GB v1.0 search.db. Three probes per search call = 10 seconds of pure overhead before the actual query ran.

The fix is a one-line SQL rewrite — query `docs_metadata` (whose `uri` is `TEXT PRIMARY KEY`) directly, pull title and summary out of `json_data` via `json_extract($.title, $.abstract)`, never touch the FTS5 virtual table for this lookup. **5 milliseconds per probe.** Single-process search wall time on the 3.4 GB corpus dropped from ~18 seconds to ~4 seconds.

The same bug was lurking in the older `fetchFrameworkRoot` helper that's been there since #81, untimed and presumably contributing to user-visible latency for months. Both are fixed in v1.0.

Concurrent-search lock contention

Every `Search.Index` initialization used to issue an unconditional `PRAGMA user_version = N` write — even when the version was already correct. SQLite is single-writer. Two parallel `cupertino search` invocations contended on the open-time write lock, and one would fail with `database is locked` because SQLite's default `busy_timeout` is `0`.

Fixed two ways: read-then-write (skip the PRAGMA when the version already matches, so the steady-state path does zero writes), and `sqlite3_busy_timeout(db, 5000)` right after open so any future write contention degrades to a wait-then-succeed instead of an immediate failure. Cupertino's MCP server can now handle concurrent requests against the same database without flaking.

Distribution: one bundle, not two

Pre-1.0 the deployment shape was: `search.db` and `samples.db` shipped on `mihaelamj/cupertino-docs`, while the curated `packages.db` was scoped for a separate `mihaelamj/cupertino-packages` companion repo. The split made sense when the packages corpus was a separate-cadence thing. It turned out it isn't — the same crawl produces all three, on the same schedule, with the same release cycle. Two repos meant two release tags per cut, two sequential downloads on `cupertino setup`, soft-fail logic when one release lagged the other, and two URL constants in `Shared.Constants` to keep in sync.

v1.0 collapses everything to one bundle: `cupertino-databases-v1.0.0.zip` on `mihaelamj/cupertino-docs`, ~833 MB compressed, contains all three databases. `cupertino setup` does **one** download + extract + version stamp. The companion repo is deleted; deleting it was the last action before tagging.

The release tooling caught up too. `cupertino-rel databases` now bundles all three databases into the single zip and uploads to the docs repo. Hard-fails if `packages.db` is missing under `--base-dir` unless `--allow-missing-packages` is passed (lets a release runner publish a partial bundle in genuinely time-sensitive cases without making it the default).

The schema bump (BREAKING)

Schema went from 10 ? 12 across two intermediate steps. v11 added `kind` and `symbols` columns to `docs_metadata`. v12 added the `symbols` column to `docs_fts`. FTS5 doesn't support `ALTER TABLE ADD COLUMN`, so v12 is a hard break — existing v10/v11 databases throw on open with a clear error message:

```
Database schema version 10 requires migration to version 12.
This is a breaking change that adds AST-derived symbols to the FTS index.
Please delete the database and run 'cupertino save' to rebuild:
rm ~/.cupertino/search.db && cupertino save
```

Most users won't see this — they'll just `cupertino setup` once and get the v1.0 bundle directly, with `setup` automatically backing up any pre-existing databases as `.backup-<version>-<iso8601>` siblings (`#249`) before extraction overwrites them. The rollback is one rename.

Users who built their own search.db with `cupertino save` against an older binary need to rebuild, but the docs corpus on disk is unchanged — the same `~/.cupertino/docs/` produces a v12 search.db without re-crawling.

MCP protocol upgrade — 2025-11-25

The Model Context Protocol moved to spec `2025-11-25`. Cupertino now negotiates that as its preferred version and retains backward-compat for `2025-06-18` and `2024-11-05` so older clients keep working through three negotiation hops. The new spec adds `Icon` to `Implementation`, so cupertino's MCP `serve` advertises a 64×64 PNG via a `...` URI (embedded as a Swift literal in the binary, no external bundle).

Crawler hardening

The crawler took a lot of fire across this release.

- **Per-URL JSON-then-WebView fallback.** `cupertino fetch --type docs` does one pass through the queue, trying Apple's JSON API first and falling back to WKWebView when a page has no JSON endpoint. Single-pass, full coverage. (The fallback was already there; the previous "two-pass" orchestration was running the same crawler twice and is now removed.)
- **Auto-resume by default.** If `metadata.json` has an active session matching the start URL, `cupertino fetch` picks it up. The previous `--resume` flag was a log-message switch and is gone.
- **Crash-safe metadata writes.** `JSONCoding.encode(_:to:)` writes with `.atomic` (temp + rename), so a kill mid-save can never leave `metadata.json` corrupt. Mid-save corruption was the one failure mode that could make a multi-day crawl unresumable.
- **Default page cap raised 15,000 ? 1,000,000.** Effectively uncapped for full Apple-corpus crawls (~50–80k pages). The old 15k default would silently truncate at ~15–30% coverage.
- **URL canonicalization (case axis).** `URLUtilities.normalize` now lowercases the URL path. Apple's docs server is case-insensitive, but the crawler used to treat `/documentation/Cinematic/CNAssetInfo` and `/documentation/cinematic/cnassetinfo` as different URLs. Queue inflated ~3× with case duplicates (62% of queue entries on the April 2026 crawl). Fragment + query stripping unchanged.
- **Filename-length cap, queue dedup at enqueue time, autorelease pool around per-page work, full DocC** `references` **walk** — the unglamorous fixes that let a multi-day crawl finish without RSS leaks, dropped pages, or queue explosion.

A reproducible recrawl pipeline (`scripts/recrawl.sh`) wraps the whole thing in 10 phases with named markers so you can tail-follow the log and see "phase 5/10 starting" in real time.

Architecture: the four-package CLI lift

This is more for contributors than users, but worth flagging. Logic that powered four CLI commands — `setup`, `doctor`, `save`, `fetch` — moved out of `Sources/CLI/Commands/*` into four new SPM packages:

- `Distribution` — the download + extract + version-stamp pipeline (`SetupService`, `ArtifactDownloader`, `ArtifactExtractor`, `InstalledVersion`).
- `Diagnostics` — pure-data probes for SQLite + filesystem corpus, used by `doctor`. Zero external deps.
- `Indexer` — write-side counterpart to `Search`. Three indexer services (`DocsService`, `PackagesService`, `SamplesService`) each emit per-stage events. Hosts the preflight pipeline for `cupertino save` + `doctor --save`.
- `Ingest` — package skeleton + `Session` helpers lifted from `FetchCommand`. The seven `<Type>Pipeline` services still live in CLI; lifting them is queued for v1.0.1.

The point of the lift is that MCP tooling, future agent-shell adapters, and tests can now drive the pipelines without depending on `ArgumentParser`. CLI files become thin front-doors that parse flags and render progress.

What "First Light" means

The release name is from astronomy. First light is the first time a new telescope is pointed at the sky and an actual image comes back — not a calibration target, not a test pattern, but real data. Everything before is engineering. First light is when the instrument starts being a telescope.

That's how this release feels. The previous twelve months were engineering — getting the crawler not to leak memory, getting the indexer to handle the field weights, getting the MCP server to negotiate protocols, getting the install to not strand users on stale databases. v1.0.0 is when those parts compose into something that does what it was built to do.

Try it

```
brew install mihaelamj/tap/cupertino
cupertino setup
cupertino search Task
```

`brew install` pulls a notarized universal binary (~13 MB). `cupertino setup` downloads the v1.0.0 database bundle (~833 MB compressed, ~5 GB extracted). `cupertino search Task` should land the canonical Swift `Task` struct as result #1.

If you want it as an MCP server for Claude / Cursor / VS Code Copilot / Zed / Windsurf / opencode / Codex, point your client config at `cupertino serve`. Setup guides for each are in the README.

What's next: v1.0.1

A few items deferred from v1.0.0 that I want to land in a follow-up patch:

- **WKWebView re-crawl on the May 2026 corpus.** The April 2026 corpus shipped with v1.0.0 because rebuilding the schema-12 search.db from on-disk docs doesn't need a re-crawl. The next release will refresh the corpus content itself and pick up Apple-side documentation edits since April plus pages the JSON API doesn't expose.
- **Kind-tier ranking.** Roughly 49% of apple-docs rows have `kind=unknown`. When the crawler-side metadata extraction improves, a `kind ? {enum, struct, class, protocol}` tier slots ahead of the framework-authority tiebreak in HEURISTIC 1. The exact-title peer cluster gets a third orthogonal signal and the few remaining edge cases (`Result`'s lowercase-property competition, `Identifiable`'s conformance-page domination) collapse cleanly.
- `Ingest` **sub-PRs 4b–4f.** The seven `<Type>Pipeline` services still in CLI lift cleanly into the new package once they grow callback-based shapes. Tracked under #247.

That's it. Cupertino v1.0.0 "First Light" is out. Tag, binary, bundle, formula — all live as of today.

Apple's documentation is now something an AI agent can actually read before it answers.