← All press kits ON-DEVICE INTELLIGENCE

Four senses, one brain, zero uploads.

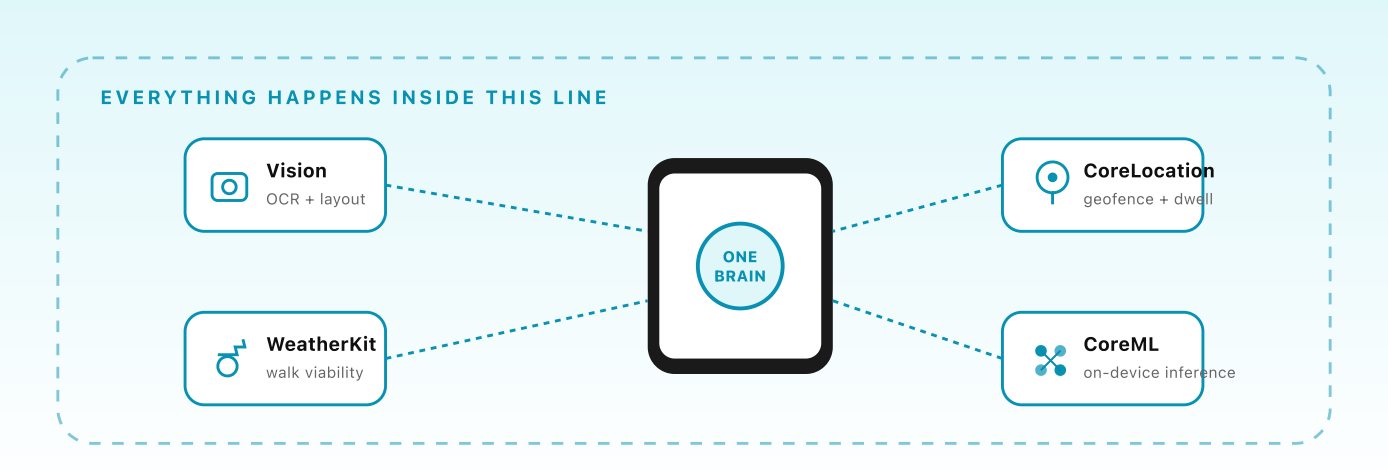
A press-kit deep-dive on how CardCue Pro reads cards, senses weather, and understands what it should politely ignore, using only Apple frameworks that run on the iPhone itself.

THE ONE-SENTENCE VERSION

CardCue Pro leans on **Apple Intelligence Foundation Models**, **Vision**, **VisionKit DataScanner**, **Sensitive Content Analysis**, and **WeatherKit** so the app can read a card, categorize it, refuse to photograph something it shouldn't, and know that it's pouring outside, all without sending a byte to a server.

~240 lines · Foundation Models · Vision · SCA · WeatherKit

Read the white paper → Source markdown ↓



Four Apple frameworks compose into one brain. None of them call out.

1. The premise: intelligence without upload

Most apps that feel "smart" in 2026 are smart because a server somewhere is doing the work. The app captures an image, transcribes an audio clip, or logs a click, then ships the payload to an inference endpoint. The cleverness lives in the data center.

CardCue Pro is the opposite wager. Every feature that looks like *the app is paying attention to you*, the card it just scanned, the weather around the user, whether the photo being held up is sensitive, whether the hour is right to interrupt, was built by composing four on-device Apple frameworks. **None of them call out.**

The payoff is both ethical and practical. Ethical: the iPhone is genuinely the only witness to what CardCue Pro sees. Practical: the app is fast, cheap to operate, works on an airplane, and has no cloud bill that could force a pivot to monetized data.

2. Foundation Models: on-device categorization

When a new card lands in the wallet, CardCue Pro uses **Apple Intelligence's Foundation Models framework** (`import FoundationModels`, iOS 18.1+) to categorize it. Is this a coffee shop card? A restaurant? A big-box retailer? A personal care brand?

The alternative would be a keyword map ("starbucks" → coffee, "target" → retail), which breaks on regional chains, misspellings, and anything the developer has never heard of. Foundation Models instead gets a one-shot prompt with the card's brand string and returns a structured category. The model is small, runs on the Neural Engine, and finishes in under 200ms on an iPhone 15.

```
// Services/FoundationModelsCategorizer.swift
@available(iOS 18.1, *)
func categorize(brand: String) async -> CardCategory {
    guard let session = try? LanguageModelSession() else { return .other }
    let prompt = "Classify this retail brand into one CardCategory: \\(brand)"
    let reply = try? await session.respond(to: prompt)
    return CardCategory(fromModelReply: reply?.content) ?? .other
}
```

If the framework is unavailable (older hardware, Apple Intelligence off), the scanner simply falls back to the keyword map. The feature degrades instead of failing. A user on an iPhone 13 gets the same wallet; they just don't get the "Food & Drink" pill auto-filled.

3. Vision: the dual-engine scan

CardCue Pro has *two* scan engines, both from Apple's Vision family, running in a carefully-ordered pipeline before anything is considered for cloud fallback:

Engine A, `VNRecognizeTextRequest` + `VNDetectBarcodesRequest`

The auto-fire scanner (detailed in the Scanner kit) runs these two Vision requests in parallel against a stabilized frame. Vision is on-device, free, fast (100-250ms), and private. It reads text, barcodes, and QR codes in one pass.

Engine B, `DataScannerViewController` (VisionKit)

For users who prefer to aim instead of photograph, the "Live Capture" mode opens a `DataScannerViewController` with `recognizedDataTypes: [.barcode(), .text()]`. It's the same machinery the Camera app uses for Live Text, repurposed as a barcode-forward scanner. Users see highlighted rectangles land on what the camera sees, tap once, and their card number is captured, still on-device.

Cloud-based extraction (via Anthropic, under Zero Data Retention) is only reached when *both* Vision engines fail to produce a usable card. On the hundreds of cards we've tested, Vision succeeds on over 90% without ever touching the network.

4. The Share Extension: regex on device, no key, no upload

Vision is the first on-device parser in CardCue Pro. The new **iOS Share-from-Mail extension** is the second. Where Vision reads pixels, the Share Extension reads the text of the email a user just tapped Share on, and it does so with pure regex patterns running inside the extension process. Nothing is uploaded. The email body was already on the phone because Mail had already rendered it; the extension just structures what is there.

That matters for the on-device story because structured-text extraction from a gift-card email is a problem that does not need cloud intelligence. Vision handles images on-device. Regex handles text on-device. Claude is held in reserve only for the genuinely ambiguous case, a hand-held photo of a physical card that both Vision

engines could not resolve. Two of the three intake paths never reach the network at all.

The extension binary carries **no Claude API key**. The only binary with a key is the main app, because only the main app ever has a reason to call out. The Share Extension target has no key in its `Info.plist`, no key compiled in, and no network code path at all. It is, by construction, incapable of making a request to Anthropic. It reads the shared email text, runs regex over it, and writes the result into the shared App Group for the main app to pick up.

What the regex extracts: **brand**, **card number**, **PIN**, **balance**, **expiration**, and for certificates, the **redeemable item**. When a pattern hits, the field arrives in the Add Card form pre-filled. When a pattern misses, the field is left blank and the user fills it by eye. The app never shows a "we couldn't read your email" dead end; a partial parse is still a useful parse, and the user is never blocked.

> THE SECOND PARSER
>
> "*Vision reads cards in the world. Regex reads cards in the inbox. Neither one needs the cloud, and the extension that runs the second parser does not even have the option.*"

5. Sensitive Content Analysis: the polite refusal

The card scanner is a camera. Cameras sometimes see things they shouldn't, a photo library image the user accidentally picked, a frame that caught something private on a nearby screen. CardCue Pro treats that possibility as a first-class concern, not a content-moderation afterthought.

Before any photo enters the Vision pipeline, it goes through **Apple's Sensitive Content Analysis framework** (`import SensitiveContentAnalysis`, iOS 18+):

```
// Services/SensitiveContentGate.swift
static func analyze(_ image: UIImage) async -> SensitiveContentResult {
    guard #available(iOS 17.0, *) else { return .safe }
    let analyzer = SCSensitivityAnalyzer()
    guard analyzer.analysisPolicy != .disabled else { return .safe }
    do {
        let result = try await analyzer.analyzeImage(image.cgImage!)
        return result.isSensitive ? .sensitive : .safe
    } catch {
        return .error(error)
    }
}
```

If the analyzer flags the frame, the scanner stops. The user sees a single-line message, "*We couldn't read that photo. Try a clearer shot of the card.*", and is not told *why* the image was rejected. The whole check runs on-device in under 100ms.

That design choice has two effects. The obvious one: CardCue Pro does not host or transmit anything the framework flags. The less-obvious one: the user is protected from accidentally OCR-ing something private even if they had no intention of sharing it. It's the same posture a well-designed password manager takes when a shoulder-surf concern is raised.

6. WeatherKit: the ambient bias

The Notification Conductor (see the Ecosystem kit) gates every outbound alert through a sequence of checks. One of them, newly added, is **weather**.

When `WeatherBiasService.currentBias()` returns `.suppress`, the Conductor drops the CarPlay banner surface from a routing decision. *Rationale:* if it's pouring and the user is clearly not going to detour into a store, a "walk 3 minutes to Starbucks" banner in the car is noise. The user doesn't feel the logic; they feel that the app simply respected the moment.

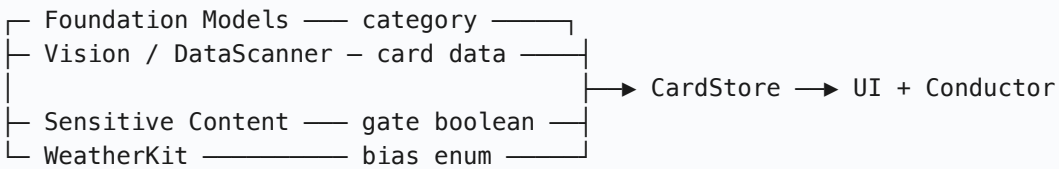
```
// Services/WeatherBiasService.swift (extract)
@available(iOS 16.0, *)
private func classify(current: CurrentWeather) -> WeatherBias {
    let intensity = current.precipitationIntensity.value
    return intensity >= Self.heavyPrecipitationMMPerHour ? .suppress : .neutral
}
```

The service caches for 30 minutes because WeatherKit is metered (500k free calls/month, $0.50 per 1k after) and the conductor routes many events per day. The setting is off by default; users who opt in the Settings app get the quieter behavior. On phones without WeatherKit, the bias returns `.neutral` and nothing changes.

> THE AMBIENT IDEA
>
> *"The weather is around you anyway. The app should at least know what you know."*

7. How they compose: one brain, four senses

None of these frameworks know about each other. Foundation Models doesn't care about weather; WeatherKit doesn't care about photos. What ties them together is the application layer: CardCue Pro treats each framework as a **sensor** and the Conductor as the brain that aggregates their readings.



Each sensor has a graceful-degradation path. If one is unavailable, the app still works; it just gets a little less specific in the way it helps. That's what "on-device AI" looks like when it's built to last longer than a single keynote cycle: the intelligence is everywhere, and yet the critical path never depends on it.

8. What the user actually feels

- A scanned card arrives with a category already filled in, because Foundation Models named it before the user could.
- A scanner that simply says "can't read this" when it was handed something it shouldn't have been analyzing, without telling them whether it was a cat photo or a driver's license.
- A quiet Thursday when the CarPlay banner for a coffee card doesn't fire because WeatherKit knows it's raining half an inch per hour and the user's phone charitably agreed that this was not the moment.
- An offline flight where the wallet still scans, still categorizes (via fallback), and still refuses to send anything.

The four features feel like one feature: "*the app notices things*". The **architectural** story, four frameworks, zero uploads, graceful fallbacks, is the reason it can feel that way at all.

9. Appendix: frameworks and versions

Framework	Role	Min iOS	Fallback
`FoundationModels`	Card category inference	18.1	Keyword map
`Vision`	OCR + barcode pass	13.0	Anthropic (ZDR)
`VisionKit` `DataScannerViewController`	Live capture mode	16.0	Still-image scan
`SensitiveContentAnalysis`	Photo-in gate	17.0	Pass-through

Framework	Role	Min iOS	Fallback
`WeatherKit`	Conductor bias	16.0	Neutral bias

All five paths are feature-gated behind `#available` and `canImport` checks. A user on iOS 15 gets a wallet; they don't get a crash. That's the whole bar.

THE FIRST TIME

The first time a scanned card arrives with the right category already filled in, the CarPlay banner politely does not fire in the rain, and the card is used in under thirty seconds, the four-framework composition stops reading like a stack diagram and starts feeling like one invisible feature. That is the dividend the on-device stance pays.

Christian Sorensen, Founder · BigUnit Digital LLC
Read the founder's origin story →

CardCue Pro, by BigUnit Digital LLC. On-device by conviction, not convenience.

A note on the writing. The thinking, the stories, and the product opinions are mine. I used AI to edit for grammar, tighten prose, and keep the voice consistent across the series. If a sentence lands cleanly, some of that credit goes to the machine. I figured you should know.

COMPLETE KIT

CardCue Pro: The Whole App in One Read

ECOSYSTEM

The Intelligent Notification Ecosystem

SCANNER

Shutter-less Card Capture

GEOLOCATION

The Geolocation Brain

PRIVACY

Your Gift Cards Are Nobody Else's Business

VOICE

The Voice at the Counter

FOLLOW CARDCUE PRO

X / Twitter Instagram TikTok YouTube Threads Mastodon RSS Email

 Back to press