

In one line: Core ML runs a *trained model file* on the device and splits its graph across **CPU, GPU and Neural Engine**; **Vision**, **Natural Language** and **Speech** are task frameworks on top (Apple's models, or yours); **Create ML** and **coremltools** make the file. iOS 27 adds **Core AI** for modern neural nets. The data never leaves the phone — the *model* ships in your bundle.



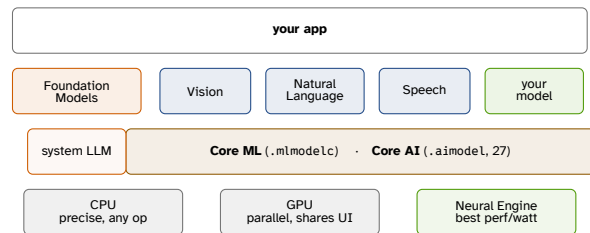
How it works — Core ML

- **Formats:** `.mlmodel` (one file, older *neuralnetwork* type) · `.mlpackage` (folder, **ML Program**, weights apart; `coremltools`' default) · `.mlmodelc` (compiled, what runs). Xcode compiles at build and generates a class.
- `MLModelConfiguration().computeUnits: .all` (default) · `.cpuOnly` · `.cpuAndGPU` · `.cpuAndNeuralEngine`. Core ML *partitions* the graph; ops the ANE can't run fall back. **Background:** no GPU — use CPU/ANE. `MLComputePlan` shows where each op runs.
- **Load:** `await MLModel.load(contentsOf:configuration:)`; the first load on a device specialises (ANE compile) and is cached — later loads are fast.
- **Predict:** `prediction(from:)` (`MLFeatureProvider` in/out); iOS 17 **async** overload: thread-safe, cancellable, runs concurrently — cap in-flight requests (memory). Batch: `predictions(fromBatch:)`.
- **iOS 18:** `MLTensor` (tensor maths), stateful models (`makeState()` → `MLState`, e.g. an LLM KV-cache), multifunction models.
- **Compress** (`ct.optimize.coreml`): `palettize_weights` (k-means LUT, 1–8 bit) · `linear_quantize_weights` (int8/int4) · `prune_weights`. Smaller app, less memory traffic; re-check accuracy.
- **On-device training:** updatable models + `MLUpdateTask` (personalise on user data, never uploaded).
- **Core AI (iOS 27):** `.aimodel`, `AIModel`, `InferenceFunction` `.run(inputs:)`, `NDArray`; `PyTorch` → `coreai_torch`; ahead-of-time compile, `AIModelCache`. Docs: non-neural models (trees, tabular) → Core ML.

The task frameworks

- **Vision:** iOS 18 **Swift-only API** — struct requests (`RecognizeTextRequest`, `ClassifyImageRequest`, `DetectFaceRectanglesRequest`, `DetectBarcodesRequest`, `CoreMLRequest...`), `try await req.perform(on:)` → typed observations. Legacy: `VNImageRequestHandler` + `VNRequests`, cast results. iOS 26: `RecognizeDocumentsRequest` (paragraphs, tables, lists). Coordinates are **normalised 0–1** — convert before drawing.
- **Natural Language:** `NLLanguageRecognizer`, `NLTokenizer`, `NLTagger` (lemma, names, sentiment), `NLEmbedding`, `NLContextualEmbedding`, custom `NLModel`.
- **Speech:** iOS 26 `SpeechAnalyzer` (actor) + `SpeechTranscriber` module; `AssetInventory` downloads the language model; on-device, long-form. Older: `SFSpeechRecognizer`.
- **Create ML:** no-code training (image/text/sound/tabular), transfer learning on Apple's feature extractors → small `.mlmodel`.

Picture — where each framework sits



Vision's `CoreMLRequest` = your model + Vision's crop/scale/orientation

Example — OCR, then your own model

```

var ocr = RecognizeTextRequest() // Vision, iOS 18
ocr.recognitionLevel = .accurate
let lines = try await ocr.perform(on: cgImage)
    .compactMap { $0.topCandidates(1).first?.string }

let cfg = MLModelConfiguration()
cfg.computeUnits = .cpuAndNeuralEngine // keep GPU for UI
let m = try await MLModel.load(contentsOf: url, configuration: cfg)
let req = CoreMLRequest(model:
    try CoreMLModelContainer(model: m, featureProvider: nil))
let result = try await req.perform(on: cgImage) // classifier
  
```

Which one?

Foundation ML	language: summarise, extract, tag; no model to ship
Vision/NL/Speech	common tasks (OCR, faces, barcodes, dictation) — try first
Core ML / AI Server	your domain (defects, own classes), offline, per frame huge models, world knowledge; cost, latency, privacy

Interview traps

- ! Loading per request / on main — load once, early, reuse.
- ! `.all` in a background task — GPU denied; pick CPU/ANE.
- ! “On-device = free”: battery, heat, app size — compress, measure (Core ML instrument, Xcode performance report).
- ! Vision boxes are normalised (VN: bottom-left origin).
- ! The `.mlmodelc` in the IPA is extractable — encrypt it if it is IP.

Remember

“Task framework first, own model second, server last” · load once, compress, pick the unit.

Likely questions

1. `.mlpackage` vs `.mlmodelc`? — source vs compiled.
2. Why the ANE? — perf/watt; unsupported ops fall back.
3. Shrink a model? — `palettise` / `quantise` / `prune` in `coremltools`.
4. Vision vs raw Core ML? — Vision scales/crops; typed results.