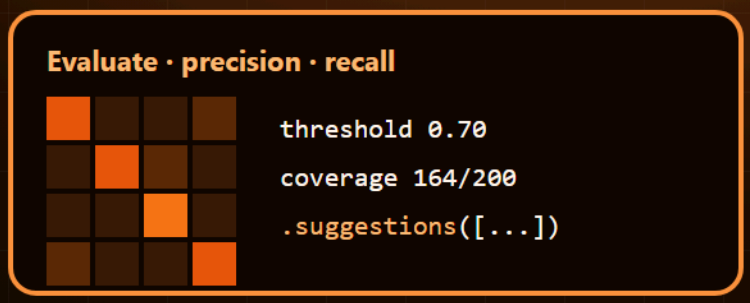



```
let request = VNCoreMLRequest(
   model: birdModel)
try handler.perform([request])
let decision = ConfidencePolicy
   .decide(results)
```





PRIVATE · OFFLINE · ON DEVICE

On-Device Machine Learning for iOS

Core ML, Vision, Natural Language, Speech, Create ML, and Apple's On-Device Foundation Models

Train · Convert · Optimize · Evaluate · Camera · Sound
Guided generation · Tool calling · Privacy · Shipping models

SWIFT 6 · IOS 18 TARGET · IOS 26 FEATURES

THENEXBOOKS

On-Device Machine Learning for iOS

Core ML, Vision, Natural Language, Speech, Create ML, and Apple's On-Device Foundation Models
Private, offline intelligence for your apps — built step by step in one real project

First Edition

THENEXBOOKS

On-Device Machine Learning for iOS
Core ML, Vision, Natural Language, Speech, Create ML, and Apple's On-Device Foundation Models

First Edition, 2026

Copyright © 2026 thenexbooks (thenexbooks.gumroad.com). All rights reserved.

No part of this publication may be reproduced, distributed, or transmitted in any form or by any means without the prior written permission of the publisher, except for brief quotations in reviews and certain other non-commercial uses permitted by copyright law. Code examples may be used freely in your own projects.

The information in this book is provided "as is" without warranty of any kind. While every precaution has been taken in its preparation, the publisher assumes no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein. Apple's frameworks, tools, and device capabilities change over time; APIs, Xcode menus, and feature availability can differ between releases, devices, languages, and regions. Features described as requiring iOS 26 or Apple Intelligence are available only on supported devices and configurations. Always consult Apple's documentation for the versions you use.

Apple, iPhone, iPad, iOS, iPadOS, macOS, Xcode, Swift, SwiftUI, Core ML, Create ML, Apple Intelligence, App Store, and Instruments are trademarks of Apple Inc., registered in the U.S. and other countries and regions. Python is a trademark of the Python Software Foundation. PyTorch is a trademark of The Linux Foundation. Other product names are trademarks of their respective owners. Their use in this book is for identification only and does not imply endorsement. This book is independent and is not affiliated with, sponsored by, or endorsed by Apple Inc. All illustrations and interface sketches in this book and on its cover are original.

Featherfield, the bird-identification app used as the running example in this book, is fictional, as are Larkspur Labs, its team, the Featherfield Backyard Birds dataset, the sightings, the places, the model files, and every evaluation number shown. Evaluation figures are illustrative values for that fictional test set, not measurements of any real model or device. Web addresses use reserved example domains.

Preface

For years, "adding AI" to an app meant sending the user's data to a server and waiting for an answer. That works, but it has costs: the feature stops when the network does, every request costs money, and your users have to trust you with their photos, voices, and notes. Modern iPhones offer another path. They carry dedicated hardware for machine learning and a family of Apple frameworks that let a model run right where the data already lives — on the device, in your app, with nothing uploaded.

This book teaches you to build that kind of intelligence. It is written for iOS developers who already know Swift and SwiftUI and want to add features that see, read, listen, and write: identifying what is in a photo, reading text from the world, transcribing speech, recognizing sounds, tagging and summarizing notes, and — on devices that support it — generating structured text with Apple's on-device foundation model. We explain the machine learning ideas you need as an app developer, without heavy math, and then show exactly how Apple's frameworks put them to work.

We also take the parts that tutorials usually skip seriously: how to tell whether a model is good enough, how to choose a confidence threshold, what to do when the model is unsure, how to keep the feature fast and kind to the battery, how to protect privacy, how to test ML code, and how to ship and update models after release. Machine learning features fail in quiet ways; a large part of building them well is designing for those failures.

Who This Book Is For

- **iOS developers** who are comfortable with Swift and SwiftUI and want to add on-device ML features to real apps.
- **Developers who have tried a Core ML sample** and now need the full picture: data, training, evaluation, optimization, concurrency, testing, and delivery.
- **Teams that care about privacy** and want features that work offline without sending user data to a server.
- **ML practitioners new to Apple platforms** who train models in Python and need to get them into an iOS app correctly.

You do not need a machine learning background. You do need working knowledge of Swift, SwiftUI, and Xcode; this book does not teach the language or basic app building.

What You Will Learn

- The machine learning concepts an app developer needs: training versus inference, datasets, features and labels, confidence, overfitting, and evaluation.
- Core ML from the ground up: model formats, generated classes, inputs and outputs, configuration, compute units, async loading, and Swift 6 concurrency.

- Vision: image classification with your own model, text recognition, barcodes, faces, bodies, animals, feature prints, and a real-time camera pipeline with AVFoundation.
- Create ML: preparing data, training an image classifier, sound and text models, and reading the results honestly.
- Evaluation: confusion matrices, precision and recall, thresholds and coverage, and error analysis that leads to better products.
- Converting PyTorch models with coremltools, and shrinking them with quantization, palettization, and pruning.
- Natural Language, Speech, and Sound Analysis for notes, dictation, and bird songs — and how to combine signals responsibly.
- Apple's Foundation Models framework (iOS 26 and later, on Apple Intelligence-capable devices): availability, sessions, guided generation, tool calling, and responsible design with fallbacks.
- Production practice: privacy by design, testing ML features, performance and battery measurement, and shipping and updating models safely.

How This Book Is Organized

Part I — Foundations of On-Device ML (Chapters 1–5) explains what on-device ML can do, the concepts behind it, the Featherfield project, and Core ML fundamentals. **Part II — Seeing with Vision** (Chapters 6–10) covers Vision requests, photo identification, text and barcodes, detection of faces, bodies, and animals, and live camera inference. **Part III — Building Your Own Models** (Chapters 11–15) covers Create ML, training, evaluation, conversion with coremltools, and model optimization. **Part IV — Language, Speech, and Sound** (Chapters 16–20) covers the Natural Language framework, custom text models, speech recognition, sound classification, and combining signals. **Part V — Apple's On-Device Foundation Models** (Chapters 21–24) covers the iOS 26 Foundation Models framework, structured output, tools, and responsible design. **Part VI — Production ML** (Chapters 25–28) covers privacy, testing, performance, and model delivery, ending with a complete case study. The appendices provide a cheat sheet, templates and reference tables, a production checklist, and a glossary.

Each chapter ends with exercises and a **Key Takeaways** box for quick review.

The Running Example: Featherfield

Throughout the book we build **Featherfield**, a fictional field guide app from a small studio called Larkspur Labs. Featherfield helps people identify backyard and park birds of North America and keep a log of what they see. Version 1.0 knows eight species, from the American Goldfinch to the Northern Cardinal. You point the camera at a feeder or pick a photo, and Featherfield suggests a species — or honestly says it is not sure. It can listen for bird songs, read trail markers, turn spoken field notes into text, tag those notes automatically, blur people out of photos before you share them, and, on devices

that support Apple Intelligence, write a short journal summary of your day from your own sightings. Everything runs on the device: Featherfield's promise is that photos, recordings, and notes never leave the iPhone unless the user shares them.

The sample data is deliberately small — eight species, eight sightings, four places — so you can check every result by hand. The evaluation numbers belong to a fictional 200-image test set and are clearly labeled as illustrative; they are internally consistent, so you can recompute every metric yourself.

Versions and Conventions

The code uses the **Swift 6 language mode** and **Xcode 26-era tooling**, with a deployment target of **iOS 18**. Features that need **iOS 26** — most importantly the Foundation Models framework — are always marked with availability checks and labeled "iOS 26+" in the text; they also require an Apple Intelligence-capable device with Apple Intelligence turned on. Python examples for training, evaluation, and conversion use recent versions of PyTorch, scikit-learn, and coremltools. Xcode menus and some API details change between releases; when something is not exactly where the book says, check Apple's documentation for your version.

> **TIP**
> A recommended practice or a technique that makes your work easier.

> **NOTE**
> Background information, version differences, or a useful aside.

> **WARNING**
> A common mistake or something that can cause wrong results, privacy problems, or crashes.

Code appears in `monospace` with a caption showing its file, such as `Featherfield/ML/BirdClassifier.swift`; long lines are wrapped to fit the page. Output that depends on a device or model is introduced with "you should see something like" because exact values vary. Menu paths are written in bold with arrows, such as **Product > Profile**.

Contents

Preface

PART I — FOUNDATIONS OF ON-DEVICE ML

1 On-Device Intelligence: What Machine Learning Can Do Inside Your App

2 ML Concepts for App Developers: Models, Training, Inference, and Evaluation

3 Meet Featherfield: The Project, the Data Model, and the ML Layer

4 Core ML Basics: Models, MLModel, Inputs, Outputs, and Generated Classes

5 Running Predictions Well: Configuration, Compute Units, Async Loading, and Concurrency

PART II — SEEING WITH VISION

6 Vision Fundamentals: Requests, Handlers, Observations, and Orientation

7 Identifying Birds in Photos with Vision and Core ML

8 Reading the World: Text Recognition and Barcode Scanning

9 Faces, Bodies, and Animals: Detection and Photo Privacy

10 The Real-Time Camera Pipeline: AVFoundation Capture and Live Inference

PART III — BUILDING YOUR OWN MODELS

11 Create ML Essentials: The App, Templates, and Preparing Data

12 Training the Featherfield Bird Classifier

13 Evaluating Models: Metrics, Confusion Matrices, Thresholds, and Error Analysis

14 Converting Models with coremltools: From PyTorch to Core ML

15 Model Optimization: Quantization, Palettization, Pruning, and Trade-offs

PART IV — LANGUAGE, SPEECH, AND SOUND

16 The Natural Language Framework: Language, Tokens, Tags, and Embeddings

17 Custom Text Models: Classifying Field Notes with Create ML and NLModel

18 Speech Recognition: Dictating Field Notes On Device

19 Sound Analysis: Recognizing Bird Songs

20 Combining Signals: Photo, Song, and Context Together

PART V — APPLE'S ON-DEVICE FOUNDATION MODELS

21 The Foundation Models Framework: Availability, Sessions, and Prompts

22 Guided Generation: Structured Output with @Generable

23 Tool Calling: Letting the Model Use Featherfield's Data

24 Designing Responsible Generative Features: Instructions, Safety, Fallbacks, and Testing

PART VI — PRODUCTION ML

25 Privacy by Design for Machine Learning Features

26 Testing and Evaluating ML Features in Your App

27 Performance and Battery: Measuring and Tuning Inference

28 Shipping and Updating Models: Bundling, Downloading, Versioning, and the Featherfield Case Study

APPENDICES

A On-Device ML Cheat Sheet

B Templates and Reference

C Production Checklist

D Glossary

PART I

Foundations of On-Device ML

What machine learning can do inside an iPhone app and where its limits are, the concepts every app developer needs, the Featherfield project and its ML layer, and Core ML from model files to fast, safe predictions under Swift 6 concurrency.

1 · On-Device Intelligence

2 · ML Concepts for App Developers

3 · Meet Featherfield

4 · Core ML Basics

5 · Running Predictions Well

CHAPTER 1 · PART I — FOUNDATIONS OF ON-DEVICE ML

On-Device Intelligence: What Machine Learning Can Do Inside Your App

Your users already carry a capable machine learning computer in their pocket. This chapter explains what it means to run models on the iPhone itself, what Apple's frameworks give you out of the box, where machine learning is the wrong tool, and how the rest of this book builds a complete, private, offline bird-identification app called Featherfield.

Why Put Intelligence on the Device?

For most of the last decade, "adding AI to an app" meant one thing: send the user's data to a server, run a model there, and send an answer back. That architecture still makes sense for some problems. But a growing set of features work better, and are easier to justify to users, when the model runs right on the phone. Recognizing what is in a photo, reading text from a sign, transcribing a short dictated note, classifying a sound, tagging a sentence, and even summarizing a day's worth of notes can all happen locally on recent iPhones.

When we say **on-device machine learning** in this book, we mean something specific: the *inference* step, where a trained model takes an input and produces an output, happens on the user's iPhone. The photo, the recording, or the note never has to leave the device for the feature to work. Training a model is a different step, and it usually happens on a Mac or a larger machine before you ship the app. Chapter 2 explains that distinction carefully; for now, keep in mind that "on-device" describes where predictions are made.

Running inference locally has four practical benefits. None of them requires us to quote a benchmark, and we will not; each one follows from the architecture itself.

Privacy

If the data never leaves the device, there is no server log to leak, no transfer to secure, and no retention policy to write for that data. You still have privacy work to do (Chapter 25 covers it in depth), but the strongest privacy promise you can make is the one where the sensitive data simply is not collected. A bird photo might not sound sensitive, but photos carry location metadata, faces of family members, and the inside of someone's home. A voice recording carries a voice. A field journal records where a person was and when.

Offline operation

Birders, hikers, travelers, and field workers spend a lot of time where the network is weak or absent. A feature that depends on a server is a feature that fails on a trail. An on-device model works the same in a forest as it does on office Wi-Fi.

No per-request server cost

Server inference costs money every time a user taps a button. On-device inference uses hardware the user already owns. That changes the economics of a feature: you can let users identify as many photos as they like without worrying that a popular feature will become an expensive one.

Low latency, in principle

A local prediction avoids the network round trip entirely. Whether the prediction itself is fast depends on the model, the device, and how you call it, and we will measure rather than assume in Chapter 27. But removing the network from the loop is what makes live features possible at all, such as classifying frames from the camera while the user is still pointing it at a branch.

The Limits You Must Design Around

On-device machine learning is not free. Being honest about the trade-offs early saves you from building a feature that cannot ship.

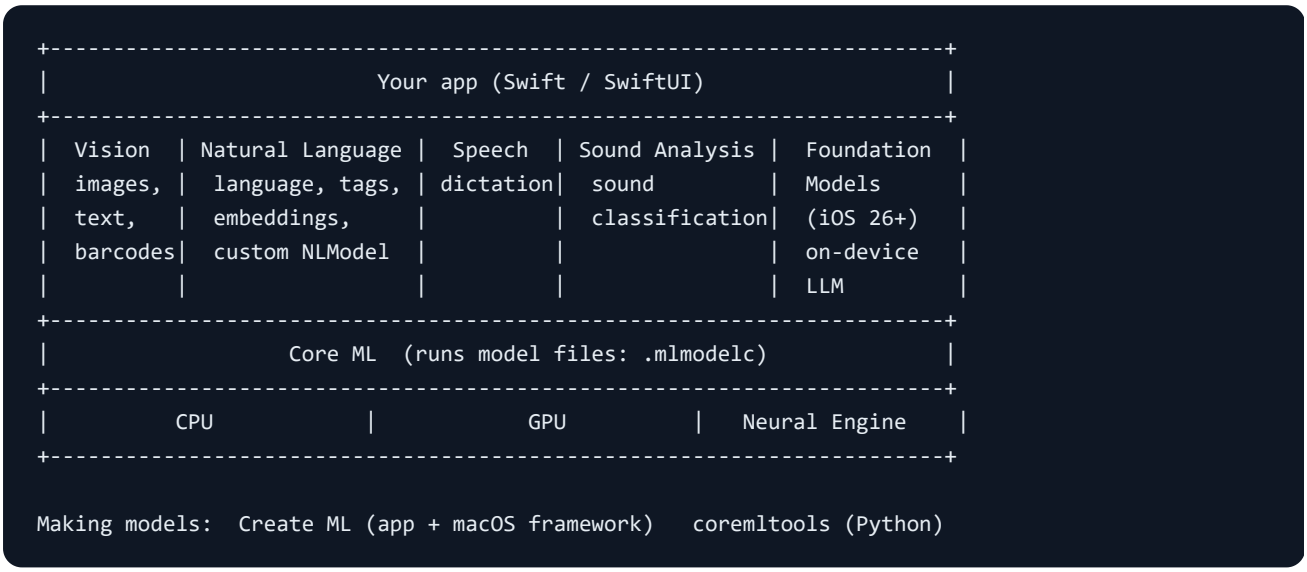
- **Model size.** Every bundled model adds to your app's download size. Very large models are impractical to bundle, and even downloaded models must fit comfortably in storage and memory. Chapter 15 shows how compression techniques shrink weights, and Chapter 28 covers downloading models after install.
- **Device variety.** Your app runs on many iPhone generations. They differ in processors, memory, and available hardware accelerators. A feature that is snappy on a new phone may be sluggish on an older one. Some features, such as Apple's on-device language model, are only available on certain devices at all.
- **Battery and heat.** Running a model continuously, for example on every camera frame, uses energy and generates heat. A responsible app throttles work, pauses when the screen is not visible, and backs off when the device is warm. Chapter 10 and Chapter 27 build those habits into Featherfield.
- **No huge models.** The phone cannot run the largest models that live in data centers. The models you ship are focused: they do one task well. That constraint turns out to be a design strength, because focused models are easier to test, explain, and trust.

> **NOTE**
>
> On-device and server-side inference are not enemies. Many apps mix them. This book deliberately builds an app with *no* server inference so that every technique stays visible and every privacy claim stays simple. Once you understand the on-device side well, you can make an informed choice for your own products.

A Map of Apple's Machine Learning Stack

Apple ships several frameworks that cooperate. It helps to see them as layers, because once you know which layer a task belongs to, you know where to look in the documentation.

Apple ML stack (diagram)



Core ML

Core ML is the engine at the bottom. It loads a compiled model and runs predictions on it, choosing among the CPU, the GPU, and the Neural Engine. You decide which compute units Core ML *may* use; Core ML decides how to split the work. When you drag a model file into Xcode, Xcode compiles it into an `.mlmodelc` folder in your app bundle and generates a Swift class with typed inputs and outputs. Chapter 4 and Chapter 5 cover Core ML directly.

Domain frameworks built on top

Vision handles images: classifying them, finding faces and bodies, recognizing text, reading barcodes, and running your own Core ML image models with correct scaling and orientation. **Natural Language** handles text: detecting a language, splitting text into words and sentences, tagging parts of speech and names, computing embeddings, and running custom text classifiers. **Speech** turns spoken audio into text, and can be told to do so strictly on the device. **Sound Analysis** classifies sounds in audio streams or files, either with a built-in classifier or with your own model.

Foundation Models (iOS 26+)

Starting with iOS 26, the **Foundation Models** framework gives apps access to Apple's on-device language model. It is available on Apple Intelligence-capable devices with Apple Intelligence turned on and the model downloaded, and only in supported languages and regions. That means any feature built on it must be optional and must have a fallback. It is designed for tasks over *your* data, such as summarizing, extracting, and classifying, not as a source of facts about the world. Part V of this book is dedicated to it.

Making models: Create ML and coremltools

When the built-in capabilities are not enough, you train your own model. **Create ML** is Apple's app (and a macOS framework) for training common model types from folders of examples with no ML code. **coremltools** is a Python package that converts models trained in frameworks such as PyTorch into Core ML format and compresses them. Part III uses both.

> **TIP**
>
> Before training anything, check whether a built-in request already solves your problem. Text recognition, barcode reading, face detection, language identification, and general sound classification are all available without shipping a model of your own. A built-in capability costs you no app size and no training data.

What Kinds of Tasks Can a Model Do?

Machine learning vocabulary can make simple ideas sound exotic. Most app features fall into a handful of task types. Learning to name the task is the first step toward picking the right tool.

Task type	Input → output	Featherfield example	Apple tool
Image classification	image → one label and a confidence	"This is a Blue Jay"	Vision + Core ML
Object detection	image → labeled boxes	Where in the frame is the bird?	Vision + Core ML
Text recognition (OCR)	image → strings and positions	Read a trail marker sign	Vision
Text tagging	text → labels per word	Find species names in a note	Natural Language
Text classification	text → one label	Tag a note as "feeding"	Natural Language + Create ML
Transcription	audio → text	Dictate a field note	Speech
Sound classification	audio → label over time	Recognize a song	Sound Analysis

Generation	prompt + data → text or a structure	Summarize a day	Foundation Models (iOS 26+)

Two features of this table deserve attention. First, the outputs are not "answers" in the everyday sense. A classifier outputs a list of labels with confidence scores; a detector outputs boxes with scores; a transcriber outputs a best guess plus alternatives. Your code has to turn those scores into a decision, and that decision logic is where much of the real engineering lives. Second, generation is different in kind from the others. A classifier can only ever pick from the labels it was trained on. A generative model produces new text, which is powerful and also means it can produce text that is wrong. We treat that difference with great care in Part V.

When Machine Learning Is the Wrong Tool

A machine learning model is a way of writing a function from examples when you cannot write it by hand. If you *can* write it by hand, you usually should. Rules are cheaper to build, faster to run, trivial to test, and completely predictable.

Consider some Featherfield features that might tempt you toward a model:

- **Sorting sightings by date.** That is a sort, not a prediction. No model needed.
- **Showing the bird seen most often on a given day.** Summing counts per species and taking the largest is a few lines of Swift. It is also exactly right every time, which a model is not.
- **Validating that a count is a positive number.** A rule.
- **Detecting whether a QR code payload names one of our four places.** Vision reads the code (a solved, built-in problem), but matching the payload against a fixed list is plain string comparison.
- **Recognizing which bird is in a photo.** Now we need a model. Nobody can write rules that turn millions of pixel values into "Northern Cardinal" across every lighting condition, angle, and background.

When a model is the right tool, wrap it in rules anyway. Featherfield uses a model to rank species, then a plain Swift policy decides whether the result is confident enough to show, as Chapter 3 implements.

> **WARNING**
>
> Never use a generative model to compute facts you already have in your data. If the app knows the user saw four Blue Jays, display the number from the data. Asking a language model to "count the birds" invites a plausible but wrong answer. Featherfield's journal assistant summarizes and phrases; Swift code supplies and checks the facts.

Meet Featherfield

Every chapter in this book contributes to one app: **Featherfield**, a bird-identification field guide built by a small fictional studio called **Larkspur Labs**. Using a single running example lets each technique land in a realistic context, with real data types, real edge cases, and real trade-offs, rather than as an isolated snippet.

The Larkspur Labs team has three people you will meet throughout the book. **Mira Okafor** is the product lead; she owns the user promise and asks the hard questions about what users will see. **Tomas Reyes** is the ML engineer; he owns the `Training/` folder where models are trained, converted, and evaluated. **Ana Lindqvist** is the iOS engineer; she owns the app code that loads models and turns predictions into a good experience.

Mira wrote the app's promise to users on the first day, and every technical decision in this book is checked against it:

"Featherfield identifies birds on your iPhone. Your photos, recordings, and notes never leave your device unless you share them."

That promise has concrete consequences. Featherfield has no accounts and no server inference. The only network call in the entire app is an optional download of updated models, which Chapter 28 builds with integrity checks. Everything else, from identifying a photo to summarizing a day, runs locally.

What Featherfield knows

Version 1.0 focuses on backyard and park birds of North America and knows eight species: American Goldfinch, American Robin, Black-capped Chickadee, Blue Jay, Downy Woodpecker, House Sparrow, Mourning Dove, and Northern Cardinal. Eight is a small number on purpose. A focused label set keeps the models small, makes evaluation easy to follow by hand, and mirrors how real products often start: narrow, well tested, and then expanded. In Chapter 28, model version 2.0 adds two more species through a downloaded update.

The features

Featherfield's screens map neatly onto the frameworks in the stack diagram:

- **Identify** — pick a photo and get a species, a short list of suggestions, or an honest "not sure" (Vision and Core ML, Chapters 6 and 7).
- **Live** — point the camera at a bird and see a stable live prediction (AVFoundation and Vision, Chapter 10).
- **Listen** — let the app listen for bird songs and suggest the singer (Sound Analysis, Chapter 19).
- **Log** — a sightings log with dictated field notes, automatic note tags, and species mentions detected in text (Speech and Natural Language, Chapters 16 to 18).

- **Journal** — an on-device assistant that summarizes a day of sightings and answers questions about them, available on iOS 26+ devices that support Apple Intelligence, with a plain template summary everywhere else (Foundation Models, Chapters 21 to 24).
- **Species** — the offline catalog of the eight birds with field marks, which comes from a bundled JSON file, never from a model.

Along the way the app also reads trail marker signs and QR codes to fill in the place name of a sighting, blurs people in photos before sharing, and notices when a cat is lurking near the feeder. Each is a small, concrete feature that teaches a specific Vision capability.

One rule for labels and names

Every model in Featherfield uses the same lowercase snake_case labels, such as `northern_cardinal` and `black_capped_chickadee`. Display names, scientific names, and field marks come only from the bundled catalog, looked up by label. A model never decides what text the user reads about a bird. That single rule prevents a surprising number of bugs, and it is what lets us swap model versions safely later.

A First Taste: The Shape of an Identification

We promised no heavy code in this chapter, but it helps to see where we are heading. The following teaser shows the shape of Featherfield's identification flow. It will not compile on its own yet: `BirdClassifier` and `ConfidencePolicy` are built in Chapters 3, 5, and 7, and Chapter 6 replaces the fixed `.up` orientation with the photo's real one.

Featherfield/Views/IdentifyViewModel.swift (teaser)

```
func identify(imageData: Data) async {
    isWorking = true
    defer { isWorking = false }
    let classifier = self.classifier
    let results = try? await Task.detached {
        try classifier.classify(imageData: imageData, orientation: .up, maxResults: 3)
    }.value
    decision = ConfidencePolicy.decide(results ?? [])
}
```

Read it top to bottom. The classifier runs off the main actor, because a prediction is blocking work. It returns up to three labels with confidences. Then a plain Swift policy turns those scores into one of three outcomes: *confident*, *suggestions*, or *unrecognized*. The model ranks; the policy decides. Chapter 5 replaces the `try?` with proper error handling and explains the concurrency rules in Swift 6 language mode.

Here is what those three outcomes look like for three photos that recur throughout the book. The numbers are illustrative values for Featherfield's examples, not measurements of a real model:

Console output (example)

```
cardinal-feeder.jpg  -> confident: northern_cardinal 0.91
sparrow-hedge.jpg    -> suggestions: house_sparrow 0.48, mourning_dove 0.31,
                        american_goldfinch 0.12
empty-feeder.jpg     -> unrecognized (top: downy_woodpecker 0.22)
```

The third line contains the most important lesson in the book. The photo of an empty feeder contains no bird, yet the classifier still reported a species, because a classifier always picks one of its labels. It has no built-in way to say "none of the above." The policy, with its thresholds, is what lets the app say "I'm not sure" honestly. You will see these three photos, called Photo A, Photo B, and Photo C, again and again.

How Featherfield Is Organized

A clear project layout makes the ML parts easy to find and easy to test. Featherfield separates the plain data model, the ML layer, each framework's code, and the views:

Featherfield project layout (overview)

```
Featherfield/
  App/          FeatherfieldApp.swift, AppModel.swift
  Models/       Species, Sighting, SpeciesCatalog, SightingStore
  ML/           FeatherfieldBirds.mlmodel, FeatherfieldSongs.mlmodel,
                FeatherfieldNoteTags.mlmodel, BirdClassifier, ConfidencePolicy, ...
  Vision/       TrailMarkerScanner, PhotoPrivacyFilter, SceneChecks, ...
  Camera/       CameraPipeline, CameraPreviewView, LivePredictionStabilizer
  Language/     NoteAnalyzer, NoteTagger
  Speech/       FieldNoteDictation
  Sound/        BirdSongListener
  Assistant/    FieldJournalAssistant, DaySummary, SpeciesLookupTool  (iOS 26+)
  Views/        IdentifyView, LiveCameraView, ListenView, JournalView, ...
  Resources/    species.json, sightings-sample.json
FeatherfieldTests/
Training/       (Python and Create ML side, not shipped in the app)
```

Chapter 3 walks through this layout in detail and creates the data model. Notice two boundaries now. First, the `Training/` folder lives next to the app but is never shipped; it is where Tomas trains and evaluates models. Second, the `Assistant/` folder is marked iOS 26+, and every type in it is gated by availability checks so the app still runs on iOS 18.

Featherfield also ships three model files. `FeatherfieldBirds.mlmodel` is an image classifier for the eight species, trained with Create ML. `FeatherfieldSongs.mlmodel` is a sound classifier with the same eight labels plus a `background` label for "no bird song here." `FeatherfieldNoteTags.mlmodel` is a small text classifier that tags field notes as `feeding`, `singing`, `nesting`, `flocking`, or `other`. Part III also builds a converted PyTorch model, `FeatherfieldBirdsLite`, to show the conversion and compression path.

The Plan of This Book

The book has six parts. Each builds on the previous ones, but each part also stands on its own if you want to jump to a particular framework.

1. **Part I — Foundations of On-Device ML (Chapters 1–5).** The vocabulary you need, the Featherfield data model and decision policy, Core ML's model files and generated classes, and how to run predictions correctly with compute units, async loading, and Swift 6 concurrency.
2. **Part II — Seeing with Vision (Chapters 6–10).** Vision's request and observation model, bird identification from photos, text and barcode recognition, faces and bodies for privacy features, and a real-time camera pipeline.
3. **Part III — Building Your Own Models (Chapters 11–15).** Preparing data, training with Create ML, evaluating with confusion matrices and thresholds, converting PyTorch models with coremltools, and compressing models with quantization and palettization.
4. **Part IV — Language, Speech, and Sound (Chapters 16–20).** Natural Language analysis of field notes, a custom text classifier, on-device dictation, bird song recognition, and combining photo and sound evidence.
5. **Part V — Apple's On-Device Foundation Models (Chapters 21–24).** Sessions and prompts, structured output with guided generation, tool calling against the app's own data, and the responsible design of generative features. All of Part V requires iOS 26+.
6. **Part VI — Production ML (Chapters 25–28).** Privacy by design, testing ML features, measuring performance and energy, and shipping and updating models safely, ending with a case study of the whole app.

What we assume you know

This book is for working iOS developers. We assume you are comfortable with Swift, SwiftUI, Xcode, and the basics of async/await. We do not teach Swift syntax, SwiftUI layout, navigation, persistence frameworks, or App Store submission; SwiftUI appears only as thin glue around ML code, usually one view model and one small view. In return, we assume no machine learning background at all. The math never goes beyond percentages and averages, and every new term is defined the first time it appears. Python shows up only in Part III, where training and conversion happen, and every Python line is explained.

What you need

- A Mac with a recent version of Xcode. The book targets Xcode 26-era tooling.
- An iPhone for camera, microphone, and performance work. The Simulator is fine for much of the book, but it cannot stand in for the camera or for real hardware performance.
- For Part V, an Apple Intelligence-capable device running iOS 26 or later, with Apple Intelligence turned on. If you do not have one, you can still read the chapters and build the fallback paths, which are part of the design anyway.

- For Part III, Python 3 on a Mac (or Linux) where you can install packages such as PyTorch and coremltools.
- Your own bird photos and recordings, or properly licensed datasets. This book contains no real bird photos, and the training data described in it, the "Featherfield Backyard Birds v1" dataset, is fictional. When you gather your own data, respect licenses and the consent of anyone who appears in it.

Version Policy: What "Recent" Means in This Book

Apple's ML frameworks evolve every year. To keep the book accurate and useful, we follow a strict version policy.

- **Swift 6 language mode.** All code is written for Swift 6 with its strict concurrency checking. That affects how we pass images, buffers, and results between threads, and we explain each pattern when it appears.
- **Xcode 26-era tooling.** Screens and menu names follow recent Xcode. Xcode labels can differ slightly between versions; when they do, look for the nearest equivalent. Menu paths are shown in bold, such as **File > New > Project**.
- **Deployment target iOS 18.** Featherfield runs on iOS 18 and later. Everything in Parts I through IV and Part VI works there.
- **iOS 26 features are gated.** The Foundation Models framework and the newer `SpeechAnalyzer` API require iOS 26. Every use of them sits behind `if #available(iOS 26, *)` or `@available(iOS 26, *)`, and the prose labels them "iOS 26+".

Where we are not certain which exact version introduced a detail, we say "recent versions of iOS" or "recent Xcode" rather than guessing. Where Apple offers two ways to do something, such as Vision's classic `VN`-prefixed request classes and its newer Swift-only API, we teach the one that is most stable across our deployment range and show the other side by side.

> **NOTE**
>
> We never quote speed, battery, or accuracy figures for real hardware or real models. Performance depends on the device, the model, the OS version, and how you call it. Instead of trusting numbers from a book, you will learn to measure on your own devices in Chapter 27. The accuracy numbers you will see for Featherfield are illustrative numbers for Featherfield's fictional test set, chosen to teach evaluation, and they are labeled that way every time.

How to Read the Code in This Book

Every code listing is captioned with the file it belongs to, for example `Featherfield/ML/BirdClassifier.swift`. When a listing shows only part of a file, the caption says "(excerpt)". Python listings live under `Training/`. Commands you type are captioned `Terminal (Mac)`.

When a listing produces output on a device, we show it after the words "You should see something like:" with a caption of `Console output (example)`. Treat these as examples of the shape of the output, not as exact values. Model outputs vary between OS versions, devices, and model builds; the framework's built-in classifiers in particular can change their labels and scores across releases. When we show the output of a Python script, it is the real output we computed, captioned `Output`.

> **TIP**
>
> Keep a scratch Xcode project next to Featherfield. When a chapter introduces a new framework request, try it in the scratch project first with a single image or a single string. Small experiments make it much easier to tell whether a surprising result comes from the framework or from your app's plumbing.

Thinking Like an ML-Feature Engineer

Before we write real code, it is worth naming the mindset this book tries to build. Traditional app code is deterministic: the same input produces the same output, and a test either passes or fails. ML features are *probabilistic*. In the illustrative numbers for Featherfield's fictional test set, the bird model is right on 177 of 200 photos and wrong on the rest, and both facts matter. That changes how you design, test, and talk about features.

Design for being wrong

Every ML feature will be wrong sometimes. The question is not whether, but how gracefully. Featherfield never silently logs a sighting based on a prediction; the user confirms first. When confidence is middling, the app shows suggestions instead of a single answer. When confidence is low, it says so.

Separate the model from the decision

Models output scores. Products need decisions. Keep those two things in different places in your code. The model layer should return ranked labels with confidences. A small, testable policy should turn those into what the user sees. When you later retrain the model, you can re-tune the policy without touching the UI, and you can unit-test the policy with fixed numbers without loading a model at all.

Evaluate with data, not anecdotes

A model that identifies the three photos on your desk is not necessarily a good model. Chapter 13 shows how to evaluate a model on a held-out test set and how to read a confusion matrix, which tells you not just how often the model is wrong but which species it confuses with which. Those numbers then drive product choices, such as the thresholds that separate confident answers from suggestions.