TECHNICAL WHITE PAPER · CENSOR

Censor: On-Device Few-Shot Annotation and Dataset Organization for Apple Platforms

On-device few-shot annotation and dataset organization for Apple platforms.

Few-Shot | On-Device | Vision Framework | iPad / macOS | Zero Network Calls

ABSTRACT

Building a supervised computer-vision dataset is a slow, manual bottleneck — the process of drawing and labeling boxes across hundreds or thousands of files is, in aggregate, frequently more expensive than training the model that consumes the result. Censor is a native, universal SwiftUI application for iPadOS and macOS that collapses this from "annotate everything" to "annotate a handful of examples": a user draws and labels a few boxes on a small seed set of files, and an on-device few-shot matching pipeline — built entirely on Apple's system frameworks, with no bundled models and no network calls — proposes matching labels across the rest of a folder for the user to confirm or reject. This paper surveys the academic foundations of the annotation bottleneck, few-shot metric learning, and active human-in-the-loop labeling; describes Censor's seed → propagate → review → export pipeline and its on-device architecture; and analyzes why an on-device few-shot design was chosen over the more conventional alternative of a cloud vision-language model.

01 Introduction

Supervised object detection and document field-extraction models require labeled training data — bounding boxes tied to categories, applied consistently across a dataset large enough to generalize. Producing that data is, empirically, the dominant cost center of the machine-learning lifecycle rather than a preliminary formality: a substantial body of data-management research now treats data collection and labeling as a first-class systems problem in its own right, not an afterthought subordinate to model architecture [6].

Two conventional responses to this bottleneck each carry a structural cost. Manual annotation tools assume the user will do all of the labeling work themselves, and the effort scales linearly (at best) with dataset size. Cloud vision-language models solve the scaling problem by inferring labels directly, but introduce per-image API cost, a network dependency, added latency, and — critically for scanned business documents, medical images, or any personal or confidential material — a requirement that the images leave the device before a label is ever produced.

Censor's founding constraint, inherited from its predecessor application (ModelBuilder), is that everything must run on-device, on Apple hardware, universally across iPad and Mac, with no bundled foundation models — large vision-language or multimodal models do not fit comfortably in iPad memory regardless of chip generation. That constraint rules out the cloud-model answer outright and shapes every subsequent design decision in this paper.

02 Background and Related Work

2.1 The Annotation Bottleneck

Data collection and labeling has been characterized directly in the data-management literature as a major bottleneck in applied machine learning and an active research problem in its own right, spanning data acquisition, data augmentation, and label generation as distinct sub-problems [6]. The cost is structural rather than incidental: unlike model training, which can be accelerated with more compute, manual labeling accuracy and speed are bounded by human attention, and the cost does not fall as the dataset grows — it compounds.

2.2 Few-Shot and Metric-Learning Approaches

Few-shot learning addresses the case where only a small number of labeled examples are available per class, and has matured into an extensively surveyed subfield of machine learning [7]. A dominant family of approaches within it is metric-based: rather than learning a classifier directly, the system learns (or is given) an embedding space in which examples of the same class are close together, and classification reduces to a nearest-neighbor or nearest-prototype lookup in that space. Prototypical Networks [2] formalize this as computing a class "prototype" — typically the mean embedding of that class's known examples — and assigning a new example to the class whose prototype it is closest to. Censor's propagation mechanism follows this same nearest-exemplar structure directly, substituting Apple's built-in perceptual feature-print embedding [8] for a learned metric-embedding network, trading some open-vocabulary generality for a zero-download, zero-training-step on-device implementation.

2.3 Active Learning and Human-in-the-Loop Systems

Active learning is the study of how a learning system should choose which examples to ask a human to label next, in order to reach a target accuracy with the fewest possible labeled examples [3]. The foundational insight — that the value of a human label depends on where in the input space it falls relative to what the system has already seen — motivates Censor's review step directly: a confirmation or rejection is not merely a correction, it changes what the system looks for on its next pass over the same folder, in the same spirit as an active-learning query loop, but driven by the user's own natural review workflow rather than an explicit query interface.

2.4 Object Proposal and Saliency Methods

Before a region can be classified, it must first be proposed as a candidate worth classifying. Objectness measures — scoring image windows by how likely they are to contain a coherent object of any class, independent of what that class is — were formalized as a standalone problem distinct from classification itself [1], and remain the conceptual basis for the class-agnostic region-proposal stage used ahead of any

exemplar-matching step. Censor's candidate-generation stage applies this same class-agnostic principle via Apple's Vision framework, combining a general objectness-based saliency signal with rectangle- and text-region detectors tuned for structured documents.

2.5 Standard Dataset Formats

Two dataset interchange formats dominate applied object detection: the COCO annotation format, introduced alongside the Common Objects in Context dataset [4], and the YOLO format, introduced alongside the single-stage detector of the same name [5]. Both are now treated as de facto standards that downstream training tools are expected to consume directly, which is why Censor treats correct, standards-compliant export — rather than in-app model training — as the boundary of its own responsibility.

---

03 Design Principles

On-device only. Vision, PDFKit, and SwiftData — all system frameworks. Zero external dependencies, zero network calls, zero bundled model weights. A user's documents never leave their device.

Few-shot, not zero-shot. Censor does not attempt to be a general open-vocabulary detector — that effectively requires a foundation vision-language model, which was ruled out for memory reasons on iPad-class hardware. Instead, it leans into the fact that a person is present and willing to label a handful of examples, and turns that into the mechanism itself: propagation works by visual similarity to what the user has already drawn [2], not by interpreting a typed description.

Human-in-the-loop, not human-replaced. Propagation drafts labels; it does not finalize them. The review step is not a rubber stamp — confirming or rejecting a proposed box actively changes what the system looks for next on the same folder [3]. The system's "understanding" of a category is never more than the sum of the exemplars the user has personally confirmed.

Export, don't lock in. Censor's job ends at producing a clean, standards-compliant dataset [4], [5]. It does not train a model itself. Training happens wherever the user wants — including in ModelBuilder, or any other tool that reads COCO, YOLO, or Create ML formats.

---

04 The Pipeline

Censor's interface has four sections, mapping directly onto four pipeline stages: **Import → Annotate → Propagate → Review → Export.**

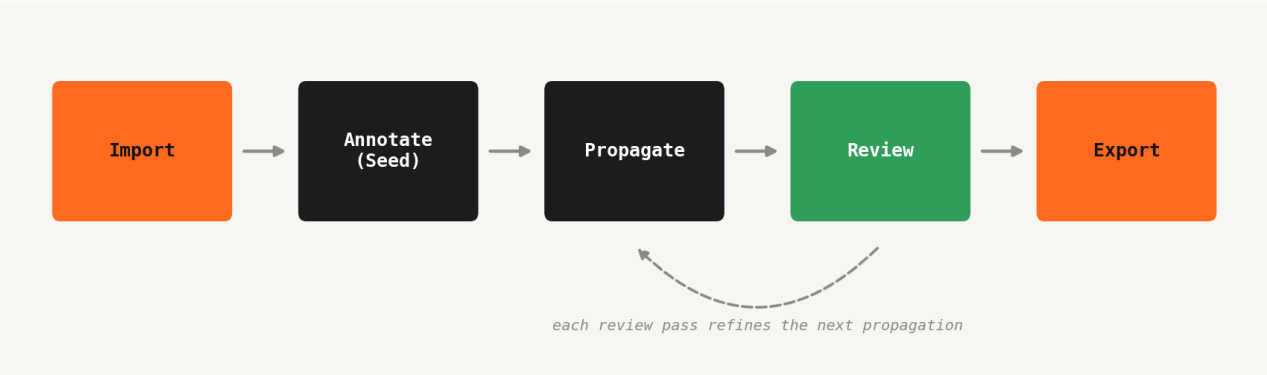

Figure 1. The seed → propagate → review → export loop. Each review pass refines the exemplar set the next propagation pass matches against.

Stage	What Happens	Key Mechanism
Import	User points Censor at a folder. It is scanned for images and PDFs.	Security-scoped folder bookmark; PDFKit renders each PDF page as an independently labelable unit.
Annotate	User draws boxes and assigns labels on a handful of seed files.	Drawing and labeling are a single atomic action — draw, then immediately pick or create a label.
Propagate	The system proposes matching boxes across the rest of the folder.	Vision region proposal + feature-print embedding + nearest-exemplar matching (§4.3).
Review	User confirms, rejects, or adjusts each proposed box.	Confirmations become new exemplars; rejections tighten that category's match threshold.
Export	The confirmed dataset is written out.	Cropped-by-category folders, or a COCO / YOLO / Create ML manifest.

4.1 Import

A workspace is a folder the user grants Censor access to. Under the app sandbox, that access is persisted as a security-scoped bookmark, so the same folder can be reopened across launches without re-prompting. The folder is scanned recursively for images and PDFs; each PDF page becomes its own labelable unit, rendered to a raster image via PDFKit at annotation time.

4.2 Annotate (Seed)

On a handful of files, the user drags to draw a box, and finishing that drag immediately opens a label picker — pick an existing category or type a new one. This collapses what is normally a two-pass workflow (draw every box, then return to tag everything) into a single pass, since the two-pass version is where most annotation fatigue and label inconsistency accumulates.

4.3 Propagate — the Core Mechanism

This is the technical heart of the labeling pipeline, and it is built entirely without a foundation model:

1. **Candidate region proposal.** For each unlabeled file, Censor generates candidate boxes without knowing what it is looking for, using `VNGenerateObjectnessBasedSaliencyImageRequest` [9] (general object-like regions, suited to photos) alongside `VNDetectRectanglesRequest` and `VNRecognizeTextRequest` (rectangular and text-block regions, suited to forms and documents) — a class-agnostic proposal stage in the same spirit as classical objectness scoring [1].
2. **Feature embedding.** Every seed crop the user drew, and every candidate region on every unlabeled file, is embedded using `VNGenerateImageFeaturePrintRequest` [8], Apple's built-in perceptual feature print. There is no model to ship, download, or fine-tune.
3. **Matching.** Each candidate is compared by feature-print distance against every category's accumulated exemplars, following the nearest-prototype structure of metric-based few-shot classification [2]. The nearest category wins, but only if the distance is under that category's own threshold — a candidate with no good match is correctly left unlabeled rather than forced into the closest available bucket.

Because there is no open-vocabulary foundation model on-device, Censor does not support "type a description, find it." Matching is grounded entirely in the visual exemplars the user has drawn — a real limitation on recall, but also a property that keeps every match explainable in a single sentence ("it looked like this example you drew"), which a cloud vision-language model's classification generally is not.

4.4 Review — the Active-Learning Loop

- **Confirming** a proposed box promotes it to a real exemplar. The category's understanding of "what this looks like" widens to include it, informing the next propagation pass.
- **Rejecting** a proposed box does more than delete it — it tightens that category's match threshold, so future propagation for that category requires a closer visual match before guessing again.

Run propagation, review the results, run it again: each pass is measurably better than the last, converging on the user's actual intent over the course of a folder [3], entirely from box-drawing interactions the user was already performing.

4.5 Export

Detection datasets and classification datasets have different shapes, so Censor produces two distinct outputs:

- **Cropped by category.** Every confirmed box is saved as its own image file, sorted into a folder named for its label — immediately usable for training an image classifier.
- **Detection manifest.** Whole labeled images are copied alongside a manifest — COCO JSON [4], YOLO (`.txt` + `classes.txt`) [5], or Create ML JSON — describing every box and its category, for training an object detector. The Create ML format feeds directly into ModelBuilder's existing training pipeline.

Only reviewed annotations — user-drawn or user-confirmed — are ever exported. Boxes still sitting in the review queue are never mistaken for ground truth.

---

05 Architecture

Censor is a single SwiftUI codebase targeting a shared multiplatform destination (`supportedDestinations: [iOS, macOS]`), generated via `xcodegen` from a `project.yml`, with a deployment target of iPadOS 26 and macOS 26. There is no iPhone or watch target — the annotation workflow assumes a larger canvas.

State is persisted with SwiftData across four models: `Workspace` (a folder, held as a security-scoped bookmark), `DocumentItem` (one image or PDF page), `Category` (a label, with its accumulated exemplars and its own match threshold), and `Annotation` (a box, tagged with its source — user-drawn, auto-propagated, or user-confirmed — so the pipeline always knows which boxes are ground truth and which are still drafts).

The propagation engine sits behind a `RegionMatcher` protocol, with `OnDeviceRegionMatcher` as its only current implementation. That seam is deliberate: a different backend — for instance, a cloud vision-language model, for users willing to trade privacy for open-vocabulary recall on a specific project — could be added later without reworking the rest of the application. Nothing beyond the protocol boundary exists yet; it is a design decision, not a shipped feature.

---

06 Why Not Just Use a Cloud Model?

The more conventional design would call a hosted vision-language model with the seed crops and a description, and let it find matches anywhere. It would likely recognize a wider range of objects on the first try. It was ruled out for this project on the following grounds:

Dimension	Fully Manual	Cloud Vision-Language Model	Censor (on-device few-shot)
Privacy	Full — human review only	Images leave the device	Full — never leaves the device
Cost per image	High (labor)	Per-call API cost	None (local compute only)
Works offline	Yes	No	Yes
Open-vocabulary recall	N/A (human judgment)	High	None — grounded in the user's own exemplars

Dimension	Fully Manual	Cloud Vision-Language Model	Censor (on-device few-shot)
Explainability	Full	Low (opaque model)	High — every match traces to a specific exemplar
Setup required	None	API key / account	None

The trade Censor makes is explicit: it gives up open-vocabulary recall in exchange for privacy, zero marginal cost, offline operation, and explainability. For the target use case — a user with a folder of private documents or photos who is willing to label a handful of examples — that trade favors the on-device approach. The `RegionMatcher` seam exists precisely so that trade does not have to be permanent for every user.

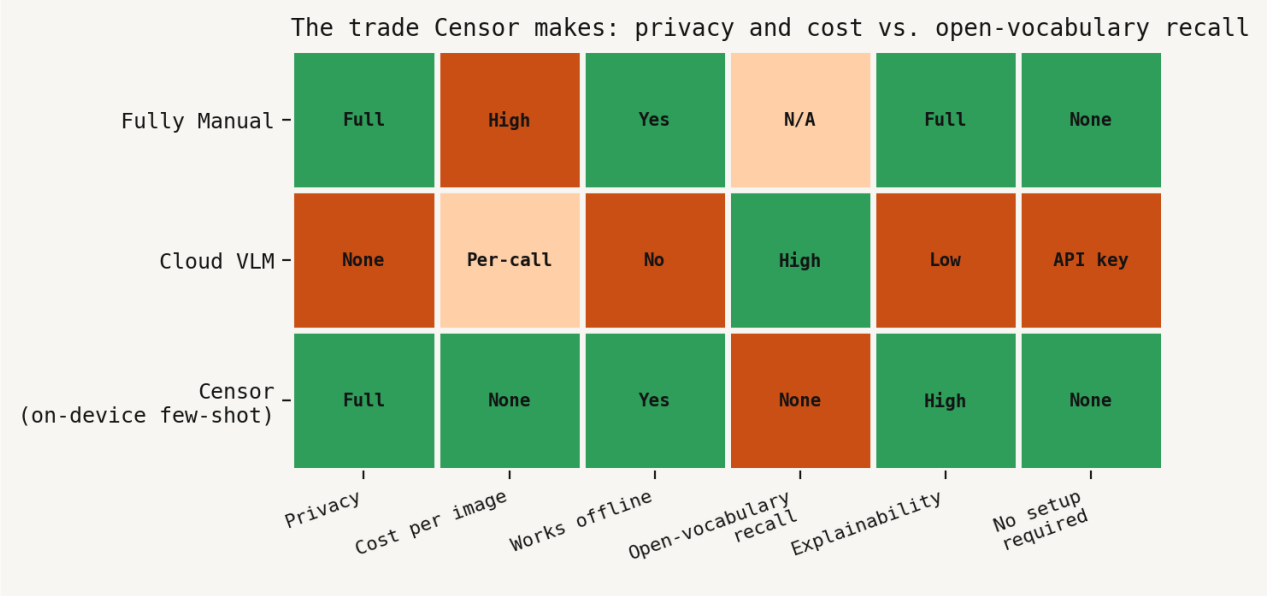

Figure 2. Censor trades open-vocabulary recall for privacy, zero marginal cost, offline operation, and explainability.

07 Economic and Privacy Analysis

7.1 The Marginal Cost of Labeling

Manual annotation cost scales with labor time, not compute, and is conventionally priced per-label or per-image by commercial annotation vendors, with per-image rates compounding quickly across datasets of the size needed for reliable model generalization. A cloud vision-language labeling pipeline replaces labor cost with API cost, which is more scalable per call but does not go to zero and recurs on every dataset revision. Censor's marginal cost per additional labeled image, once a category's exemplar set exists, is local compute

time only — no per-call charge and no incremental labor beyond the review action the user was already performing to confirm quality. This mirrors the on-device economic argument made for ModelBuilder's inference workloads, applied here to the dataset-construction stage that precedes training rather than the trained-model stage that follows it.

7.2 Privacy for Regulated and Confidential Material

For scanned business documents, medical images, or personal photographs, the requirement that images "leave the device" for cloud labeling is not merely an inconvenience — for regulated data (healthcare records under HIPAA, financial documents, attorney-client material) it can be a categorical blocker on using a cloud vision-language labeling service at all without a formal data-processing agreement. Censor's on-device design satisfies data-minimization requirements by construction rather than by contractual promise with a third-party processor, consistent with the architecture-level privacy argument made across this document family for on-device inference under GDPR-style data-protection regimes [10].

7.3 Where the Trade Is Worth Making

The comparison in §6 is not a claim that on-device few-shot matching dominates a cloud vision-language model on every axis — open-vocabulary recall is a real, acknowledged loss. The economic case for Censor specifically targets the population of users for whom that loss is acceptable: practitioners with a folder of visually consistent objects (a fixed set of document layouts, a fixed set of product photos, a fixed set of specimen images) who are already willing to hand-label a handful of examples, and for whom privacy, offline operation, or per-call cost would otherwise rule out labeling assistance entirely.

---

08 Current Scope and Roadmap

In scope for v1: PDFs and images, as spatial bounding-box annotation targets. PDF pages and photos flow through the identical pipeline once rendered to a raster image, so supporting both was a natural extension rather than two separate features.

Deliberately out of scope for v1: tabular or spreadsheet data. Row/column labeling is a fundamentally different annotation paradigm from bounding boxes, and folding it into v1 would have diluted the core loop before it was proven. It remains a plausible future extension, not a rejected idea.

Deliberately excluded, not deferred: in-app model training. Censor's scope ends at producing a clean labeled dataset. Training is left to dedicated tools — including ModelBuilder, whose Create ML pipeline can consume Censor's export directly — so Censor stays a labeling specialist rather than duplicating a role another tool already fills.

---

09 Conclusion

The expensive part of building a computer-vision dataset was never really "training a model" — it was getting a human to look at enough examples and agree on what they mean [6]. Censor does not try to remove the human from that loop; it tries to make each interaction in that loop count for more than one example, by turning a handful of drawn boxes into a matching signal, grounded in established objectness [1] and metric-based few-shot [2] techniques, that propagates across a folder, and turning every review decision into a correction that improves the next pass in the spirit of active learning [3]. Built entirely on Apple's system frameworks, it does this without a single image ever leaving the device.

// References

[1] Alexe, B., Deselaers, T., and Ferrari, V., "Measuring the Objectness of Image Windows," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, no. 11, pp. 2189–2202, 2012.

[2] Snell, J., Swersky, K., and Zemel, R., "Prototypical Networks for Few-shot Learning," Advances in Neural Information Processing Systems 30 (NeurIPS 2017), pp. 4077–4087, 2017.

[3] Settles, B., "Active Learning Literature Survey," Computer Sciences Technical Report 1648, University of Wisconsin-Madison, 2009.

[4] Lin, T.Y., Maire, M., Belongie, S., et al., "Microsoft COCO: Common Objects in Context," Computer Vision — ECCV 2014, pp. 740–755, Springer, 2014.

[5] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A., "You Only Look Once: Unified, Real-Time Object Detection," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779–788, 2016.

[6] Roh, Y., Heo, G., and Whang, S.E., "A Survey on Data Collection for Machine Learning: A Big Data-AI Integration Perspective," IEEE Transactions on Knowledge and Data Engineering, vol. 33, no. 4, pp. 1328–1347, 2021.

[7] Wang, Y., Yao, Q., Kwok, J.T., and Ni, L.M., "Generalizing from a Few Examples: A Survey on Few-Shot Learning," ACM Computing Surveys, vol. 53, no. 3, Article 63, 2020.

[8] Apple Inc., "VNGenerateImageFeaturePrintRequest," Apple Developer Documentation, 2024. https://developer.apple.com/documentation/vision/vngenerateimagefeatureprintrequest

[9] Apple Inc., "VNGenerateObjectnessBasedSaliencyImageRequest," Apple Developer Documentation, 2024. https://developer.apple.com/documentation/vision/vngenerateobjectnessbasedsaliencyimagerequest

[10] European Parliament and Council, "Regulation (EU) 2016/679 (GDPR)," Official Journal of the European Union, L 119, April 2016.

[11] Apple Inc., "PDFKit," Apple Developer Documentation, 2024. https://developer.apple.com/documentation/pdfkit

[12] Apple Inc., "SwiftData," Apple Developer Documentation, 2024. https://developer.apple.com/documentation/swiftdata