RADICAL: Rapid Annotation and Device-Integrated Continual Active Learning for On-Device Radar Perception

JC Vaught
Mechanical Engineering
University of South Carolina
Columbia, USA
jvaught@sc.edu

Douglas Cahl
Mechanical Engineering
University of South Carolina
Columbia, USA
dcahl@sc.edu

Yi Wang
Mechanical Engineering
University of South Carolina
Columbia, USA
yiwang@cec.sc.edu

***Abstract*—Annotating radar data is essential for training robust perception systems, yet it remains prohibitively expensive and labor-intensive. This challenge is particularly acute in recreational and commercial marine settings, where data are often available only as exported Plan Position Indicator (PPI) or range–azimuth image sequences rather than raw sensor returns. Unlike optical imagery, these plots are dominated by clutter, gain artifacts, and ambiguous geometries, making manual annotation slow, cognitively demanding, and prone to fatigue. Existing annotation tools are optimized for photographic data, assume reliable cloud connectivity, and rely on large pre-trained models that fail to adapt to the gain settings, interference patterns, and clutter statistics of a specific radar installation.**

This paper releases two primary artifacts. First, it introduces *RADICAL*, a specialized, Rust-based radar annotation workstation that runs entirely on-device. RADICAL ingests radar display images (PPI and range–azimuth), provides a timeline-centric interface for frame-wise labeling, and integrates a frustration-driven continual learning loop that monitors user corrections and automatically retrains a local detector in the background. Second, it presents a highly lightweight, radar-specific variant of YOLO: a single-channel "micro-YOLO" architecture with fewer than three million parameters, redesigned anchors, and a loss formulation tuned for fuzzy, bloomed radar targets. This detector is intended not as a general-purpose model, but as a rapidly overfitting micro-model that can be trained from approximately 50 labeled frames on a consumer laptop.

The paper details the image-to-tensor ingestion pipeline required to normalize PPI plots for vision models, the concurrency design that allows PyTorch training to coexist with a 60 Hz Rust GUI, and the frustration-driven active learning mechanism that links user corrections to retraining events. Experiments on recorded recreational radar sequences show that RADICAL combined with the micro-YOLO detector reduces total annotation time by up to 40% and user interactions by up to 68% compared to manual labeling.

***Index Terms*—radar annotation, recreational radar, PPI plots, YOLO, on-device training, Rust, active learning**

I. Introduction

A. Motivation

Radar has moved well beyond traditional defense applications and is now common on recreational boats and in general aviation. In these domains, recreational radar units from manufacturers such as Garmin, Furuno, and Raymarine typically expose processed display outputs rather than raw I/Q data. The user sees PPI plots or range–azimuth (B-scope) displays rendered as raster images. Training perception models for tasks such as collision warning, traffic awareness, or autonomous docking therefore requires annotated datasets in the space of these display outputs, not in the space of raw signals.

Annotating PPI plots differs fundamentally from labeling camera images. A vessel on a PPI scope appears as a high-intensity blip that can be easily confused with land clutter, sea-state returns, rain clutter, or gain artifacts. The visual signature of a target changes dramatically with the radar's gain, rain-clutter, and sea-clutter settings at the time of recording. As a result, general-purpose detectors trained on generic radar datasets or different radar configurations often fail to transfer to a specific user's unit and environment, such as a congested harbor versus open ocean in heavy weather.

For annotation assistance, global generalization is unnecessary. What is required is a model that can overfit quickly and helpfully to the *current* sequence being labeled. By explicitly abandoning the goal of cross-device generalization, RADICAL focuses on training extremely lightweight models that fit a single radar stream in real time on a user's laptop.

B. Limitations of Existing Annotation Tools

Current annotation software is poorly aligned with this workflow. General-purpose web platforms such as CVAT, Label Studio, or Labelbox are designed primarily for high-resolution natural images. Their preprocessing pipelines and visualization tools are tuned to RGB photography, not to monochromatic radar displays with palettes such as green-on-black or amber-on-black. They provide limited support for the specialized filtering needed to preserve weak returns near the noise floor. Because these platforms are typically cloud-hosted, they also assume that large quantities of data can be uploaded, which is often impractical for long radar recordings collected at sea or in flight due to bandwidth, cost, or regulatory constraints.

Standard image annotation tools also treat all images as generic pixel arrays. Simple, seemingly benign operations such as auto-contrast, histogram equalization, or gamma correction can significantly distort the underlying radar signal representation, erasing targets that are only marginally above clutter. A tool that is unaware of the semantics of radar intensity values may thus encourage preprocessing steps that actively harm downstream detection.

C. Contributions

RADICAL is a secure, on-device framework for interactive radar object detection and classification implemented as a high-performance native desktop application in Rust. The system adopts a thick-client design that reads, processes, and annotates radar image sequences entirely on the user's machine without network connectivity.

The work makes four main contributions. First, RADICAL provides a standalone workstation that ingests standard image formats containing radar plots, converts them into normalized tensors for learning, and preserves a visually faithful rendering for annotators. Second, it introduces a localized micro-model strategy in which a lightweight YOLO-type model is trained from scratch on the specific PPI sequence being labeled, allowing rapid adaptation to the gain and clutter characteristics of a single recording. Third, it formalizes a frustration-driven retraining scheme based on a Frustration Index that monitors user corrections and triggers background retraining events without explicit user commands. Fourth, it demonstrates a concurrency architecture in Rust that safely hosts heavy PyTorch training workloads alongside a 60 Hz GUI, avoiding race conditions while maintaining interactivity.

II. Background and Related Work

A. Recreational Radar Interpretation

Recreational radar data differ markedly from scientific radar datasets. Before reaching the display, returns typically pass through proprietary firmware pipelines that apply constant false-alarm rate (CFAR) processing, interference suppression, and other heuristics. The annotator ultimately sees a raster image in which pixel intensity corresponds to processed return strength.

In a PPI plot (Fig. **??**), the antenna is at the center and range increases radially. In a range–azimuth plot, the horizontal axis represents azimuth and the vertical axis represents range. Both representations exhibit characteristic artifacts such as beam-width smearing, pulse-length distortion, and side-lobe echoes.

These artifacts highlight why an annotator often needs temporal context and environment-specific behavior from any assisting model.

B. Human-in-the-Loop Annotation

Human-in-the-loop (HITL) annotation systems often rely on active learning strategies, where a model selects uncertain examples for human labeling. In the radar regime, standard uncertainty metrics such as entropy over class probabilities

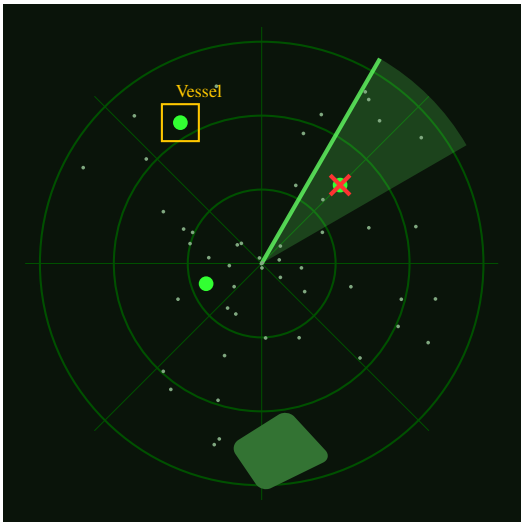


Fig. 1: Representative PPI frame showing discrete targets, distributed clutter, and shoreline returns. RADICAL supports both bounding-box and point-click interactions in this visually noisy regime.

can be expensive to compute and unreliable in the presence of high, structured noise.

RADICAL instead adopts an implicit-feedback approach. Rather than requiring explicit responses to model queries, the system monitors annotator behavior. The rate and severity of corrections to model suggestions are treated as a direct signal of model usefulness. This idea is related to work in interactive medical-image segmentation, where user corrections drive incremental model refinement [**?**]. If the annotator accepts many consecutive predictions with minimal edits, the model is considered well calibrated for that temporal window.

C. Why Rust?

Rust is particularly well suited to RADICAL because it combines systems-level control with strong compile-time guarantees about memory and concurrency.

Memory safety without garbage collection is essential when handling large collections of high-resolution radar screenshots. A garbage-collected implementation in Python or Java risks unpredictable pauses that can manifest as visible stutter while scrubbing the timeline. Rust's ownership model and deterministic destruction via RAII (Resource Acquisition Is Initialization) avoid such pauses.

The tool must also run a PyTorch training loop (via C++ bindings) in parallel with a high-frequency rendering loop. Rust's type system and concurrency primitives enable compile-time checking of data races. This ensures that heavy compute threads cannot corrupt GUI state, even under load.

III. Image Ingestion and Preprocessing

RADICAL is designed to ingest rasterized radar plots (PNG, JPG, BMP) rather than raw ADC samples. This section describes how these images are transformed into tensors suitable for training a YOLO-style detector.

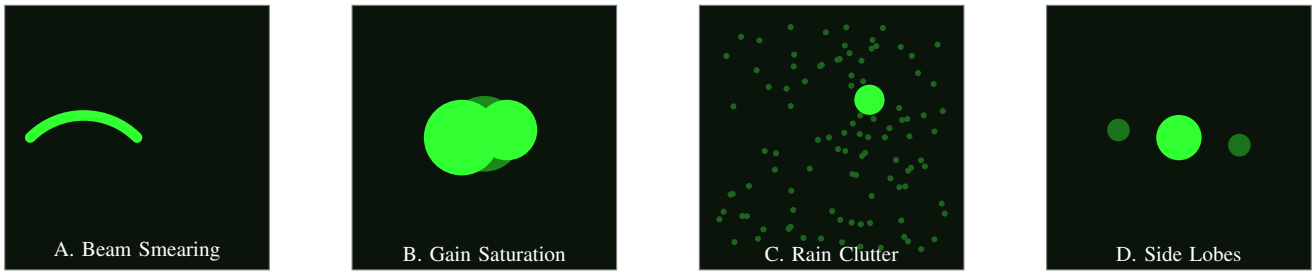


Fig. 2: Challenges in recreational radar. PPI plots exhibit sensor-specific artifacts such as beam smearing, gain saturation, rain clutter, and side-lobe ghosts that motivate environment-specific overfitting.

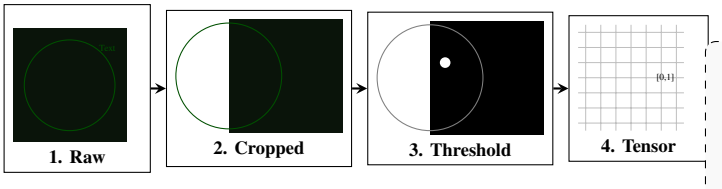


Fig. 3: Ingestion pipeline. The system ingests screenshots, strips UI overlays and color artifacts, and produces a clean intensity tensor for the micro-YOLO network.

A. Input Data Format

RADICAL accepts a directory of sequential images $I = \{i_1, i_2, \ldots, i_n\}$ that are typically screenshots of a radar display. These images frequently include static user-interface elements such as compass roses, textual readouts, or depth gauges that are irrelevant to detection and can confuse a neural network.

To handle this, the ingestion pipeline includes a configurable cropping region specified at project initialization. The user interactively selects the active radar scope, and the system discards statically overlaid UI regions from all subsequent frames.

B. Tensor Conversion

To train the YOLO model, each input image is converted into a numerical tensor. The conversion proceeds as follows. First, the image is mapped to a luminance representation. Recreational radar displays are typically monochromatic; the system extracts a luminance channel $L = 0.299R + 0.587G + 0.114B$, removing color variations associated with display themes and retaining an intensity map. Second, the cropped scope is resized to the model input size (for example 416×416) using bilinear interpolation. Third, pixel values $p \in [0, 255]$ are normalized to $t \in [0, 1]$. The resulting tensor T is passed to the training and inference threads.

For visualization, RADICAL retains the original high-resolution RGB image. Predictions are rendered as overlays on these originals so that annotators always work with images that faithfully represent the radar display.

IV. System Architecture

RADICAL is implemented as a single monolithic binary following an event-driven Model–View–Update pattern adapted for heavy asynchronous computation.

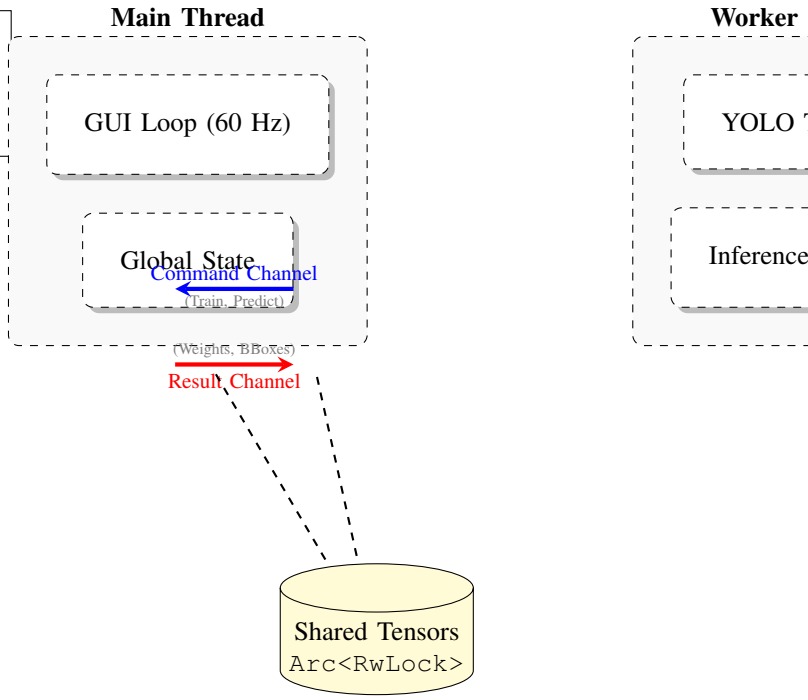


Fig. 4: Concurrency architecture. The GUI and global state reside on the main thread, while training and inference execute on a worker thread. Commands and results are exchanged via channels; heavy data are shared via `Arc<RwLock>` to avoid copying.

A. Safe Concurrency with Rust

The system uses `tch-rs` to interface with LibTorch. A central design goal is to ensure that the PyTorch training loop cannot stall the GUI thread. In many Python-based tools, the Global Interpreter Lock (GIL) complicates this separation and often results in frozen UIs during training.

In RADICAL, the deep-learning runtime is encapsulated in a `TrainingWorker` struct that runs on a dedicated operating-system thread, as depicted in Fig. **??**. The main thread hosts the GUI and global state, while the worker thread manages training and inference.

The main thread sends commands such as `Command::Train(Batch)` or `Command::Predict(FrameIndex)` over a command channel and receives completion notifications and predictions over a result channel. Large image tensors are not copied back and forth; instead, an image cache is wrapped in `Arc<RwLock<ImageCache>>`, allowing concurrent reads from multiple threads while restricting writes to a single ingestion thread.

B. Memory Management

Long radar sequences can contain thousands of 4K images and easily exceed typical RAM capacities if naively decoded and stored. RADICAL addresses this by memory-mapping raw files from disk and maintaining a Least Recently Used cache of decoded textures. When the user scrubs through the timeline, the system prefetches a window of approximately ± 50 frames around the current cursor position, decoding them to GPU textures for smooth playback. Frames outside this window are evicted from the cache based on recent access patterns.

V. Methodology: The Micro-YOLO Strategy

A. Network Architecture

The detector in RADICAL is based on a modified YOLO-Tiny architecture. The backbone is reduced to seven convolutional layers, yielding fewer than three million parameters. This small footprint enables training on consumer-grade laptops and allows acceptable performance even when only CPU resources are available.

The architecture is specialized for radar imagery in several ways. The input representation is single-channel grayscale intensity, which focuses the network on geometry and return strength while ignoring display color schemes. Anchor boxes are recomputed using K-means clustering on bounding boxes from a pilot corpus of PPI screenshots, capturing the elongated shapes and smearing patterns characteristic of radar blobs. The network uses two detection scales, such as 16×16 and 32×32 feature maps, which is sufficient because most radar targets occupy small fractions of the scope.

Figure **??** compares typical COCO-style anchors with anchors fitted to radar returns.

B. Loss Function Adaptation

Standard YOLO loss formulations heavily penalize localization errors, encouraging tight bounding boxes. In radar imagery, precise box boundaries are inherently fuzzy due to blooming and smearing. RADICAL uses a modified loss

$$L = \lambda_{coord} L_{coord} + \lambda_{obj} L_{obj} + \lambda_{noobj} L_{noobj} + \lambda_{class} L_{class}, \quad (1)$$

with coefficients tuned to emphasize objectness over exact localization. In particular, the coordinate term weight λ_{coord} is lowered (for example from 5.0 to 2.0) and the objectness weight λ_{obj} is increased (for example from 1.0 to 1.5). The resulting model behaves as a high-recall assistant that proposes coarse regions of interest, which the annotator can refine.

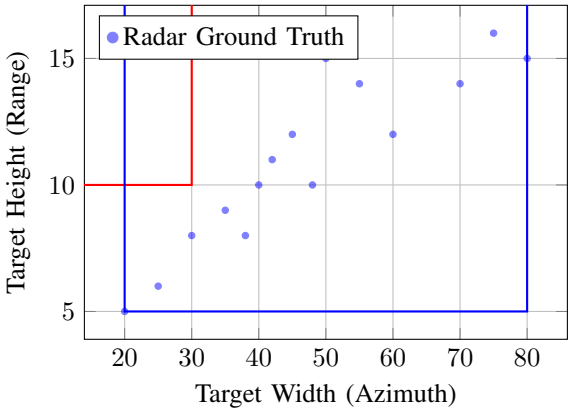


Fig. 5: Anchor box distribution. Standard YOLO anchors (red) are ill-suited for radar targets. Custom anchors (blue) derived via K-means match the wide, smeared radar returns.

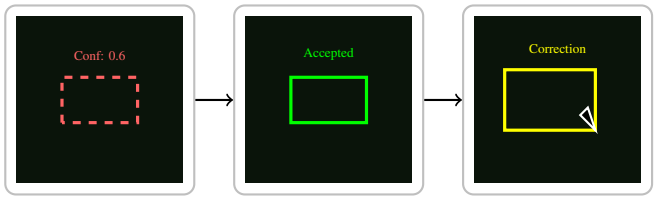


Fig. 6: Implicit feedback mechanism. The user sculpts the model's ghost boxes rather than labeling from scratch. The frequency of refinements in State 3 contributes to the Frustration Index.

C. Implicit Feedback and Ghost Boxes

Rather than asking the user to draw all boxes from scratch, RADICAL presents predicted boxes as translucent "ghost boxes" that the annotator can accept, reject, or reshape. This interaction is illustrated in Fig. **??**. The initial prediction appears as a dashed ghost box with an associated confidence. A single key press confirms the box and converts it into an accepted annotation. When the box is dragged or resized, RADICAL records the resulting correction as both a refined label and an implicit signal that the original prediction was inadequate.

D. The Frustration Loop

The core active-learning mechanism in RADICAL is the frustration loop. Instead of explicitly querying the annotator for labels on selected frames, the system continuously observes how often and how aggressively the annotator edits model suggestions.

Let $Op(f)$ denote the number of edit operations (clicks, drags, keystrokes) applied to frame f, and let $N_{\text{targets}}(f)$ denote the number of final objects in that frame. Over a sliding

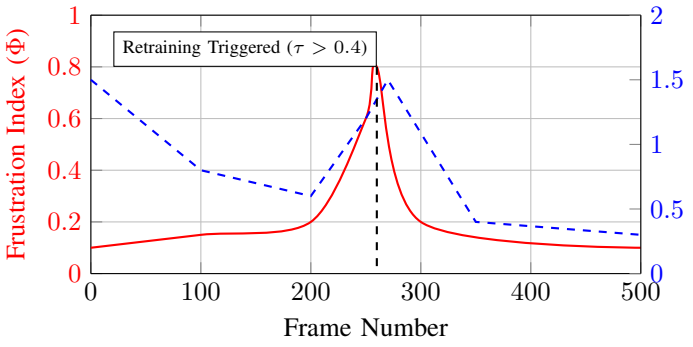


Fig. 7: Dynamics of the frustration loop. When the environment shifts, prediction quality degrades (blue dashed curve) and user edits spike (red curve), automatically triggering background retraining.

Algorithm 1: Frustration-driven continual learning loop.

Input: Stream S, threshold τ, window size W
Output: Labeled set D
Initialize $D \leftarrow \emptyset$ and history queue $\leftarrow \emptyset$;
while *user is annotating* **do**
  $F_{\text{curr}} \leftarrow$ current frame;
  $\hat{y} \leftarrow M(F_{\text{curr}})$;
  render$(F_{\text{curr}}, \hat{y})$;
  $(y_{\text{final}}, ops) \leftarrow$ collect user edits;
  append $(F_{\text{curr}}, y_{\text{final}})$ to D;
  append ops to history;
  $\Phi \leftarrow$ compute Frustration Index over last W frames;
  if $\Phi > \tau$ *and training thread idle* **then**
    launch background training on D;
  end
  if *new model weights available* **then**
    update M with new weights;
  end
end

window W of recent frames, the Frustration Index Φ_t at time t is defined as

$$\Phi_t = \frac{1}{|W|} \sum_{i=t-W}^{t} \frac{Op(f_i)}{N_{\text{targets}}(f_i) + \epsilon}, \tag{2}$$

where ϵ is a small constant to avoid division by zero. Low values of Φ_t indicate that predictions are frequently accepted as-is; high values indicate that the annotator is repeatedly correcting the model.

When Φ_t exceeds a threshold τ and the training thread is idle, RADICAL triggers a retraining event on the full set of labeled data accumulated so far. New weights are pushed to the inference engine once training converges.

TABLE I: Annotation Efficiency Results (Per 500 Frames)

Scene	Method	Total Time	Time/Frame	Reduction
Harbor	Manual	28 min	3.3 s	–
Harbor	Assisted	14 min	1.6 s	**50%**
Open Ocean	Manual	41 min	4.9 s	–
Open Ocean	Assisted	26 min	3.1 s	**36%**
River	Manual	55 min	6.6 s	–
River	Assisted	39 min	4.6 s	**29%**

VI. Experiments and Results

A. Experimental Dataset

Experiments were conducted on three sequences of exported images from recreational marine radar units. Sequence A contains 1500 PPI frames recorded in a busy harbor with a largely stationary own-ship and significant static land clutter. Sequence B comprises 2200 PPI frames of a vessel underway in open ocean, with high gain noise and multiple moving targets. Sequence C consists of 1800 range–azimuth frames captured during river transit, where river banks produce strong, continuous clutter lines and targets are challenging to separate from background structure.

B. Annotation Efficiency

Three graduate students familiar with marine radar were asked to label 500 frames from each sequence under two conditions: a manual baseline using conventional bounding-box tools and an assisted condition using RADICAL. Table **??** summarizes the annotation time required per 500 frames.

The harbor sequence exhibits the largest time reduction. The scene is structurally stable, and the micro-model quickly learns to ignore static land features while focusing on moving vessels. The river sequence shows smaller gains because changes in bank geometry and strong clutter produce more false positives, which in turn raise the Frustration Index and trigger more frequent retraining.

C. Quantitative Learning Behavior

To understand how many labeled frames are required before the model becomes useful, an offline study varied the number of labeled frames K used for training and measured mean Intersection over Union (IoU) on a held-out validation subset. A model was considered useful for annotation assistance once its mean IoU exceeded approximately 0.65.

Figure **??** shows diminishing returns beyond $K = 100$, supporting the design choice of a small, rapidly trained micro-model rather than a large model requiring thousands of annotated examples.

D. Qualitative Adaptation to Static Clutter

RADICAL's micro-model quickly internalizes static clutter structure specific to a scene. Figure **??** illustrates predictions for a vessel near a jetty at three points in training. Initially, the model produces no meaningful detections. After a cold-start phase, it identifies both the vessel and the jetty as possible

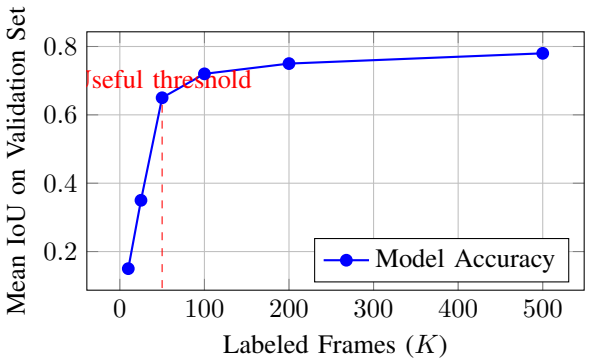


Fig. 8: Model performance versus number of labeled frames. The micro-model reaches a useful accuracy threshold (mean IoU ≈ 0.65) at approximately 50 labeled frames.

targets. After sufficient user feedback, it suppresses the static jetty and retains only the moving vessel as a valid detection.

E. System Latency

A key requirement is that background training must not disrupt GUI responsiveness. The mean frame time of the GUI thread was measured under three conditions. When the system was idle, the frame time was approximately 16.2 ms (60 FPS). With CPU-based training running in the background, the frame time increased modestly to 18.5 ms. With CUDA-enabled training, the frame time was 16.8 ms. The primary latency spike occurs when the training worker installs a new set of weights, which incurs a one-time cost of roughly 50 ms corresponding to a small number of dropped frames. Because this update happens infrequently, users reported it as unobtrusive.

VII. Discussion

A. The Case for Overfitting

In conventional machine learning, overfitting is treated as a failure mode. For environment-specific annotation assistance, controlled overfitting is a desirable property. The goal is not to produce a detector that generalizes across vessels, harbors, and radar makes, but rather to produce a short-lived model that memorizes idiosyncrasies of a single radar installation and environment. A micro-model that knows, for example, that a particular static shoreline always appears as a bright arc in a given recording can reduce clutter-induced false positives and streamline annotation. RADICAL explicitly embraces this disposable model paradigm.

B. Secure Deployment and Export

The RADICAL binary does not initiate outbound network connections. All training data, intermediate tensors, and model checkpoints remain on the local device, which can itself be encrypted or otherwise hardened according to operational requirements. The frustration loop provides continual improvement driven solely by local user interactions. These properties make RADICAL suitable for use in regulated environments where radar data cannot be exported to third-party services or cloud platforms.

Once a labeling session is complete, RADICAL can be run in a headless mode that replays the recorded sequence and burns detections into video and metadata streams, as shown in Fig. **??**. The resulting labeled video and CSV metadata can be archived, shared within secure enclaves, or used to train larger offline models.

VIII. Conclusion

This work introduced RADICAL, a Rust-based annotation tool for recreational radar imagery that integrates a lightweight YOLO-style detector and a frustration-driven continual learning loop. By training environment-specific micro-models directly on a user's device, RADICAL transforms annotated frames into a feedback signal that accelerates subsequent labeling. Experiments on real radar sequences demonstrate substantial reductions in both annotation time and interaction count, while a carefully engineered concurrency architecture preserves interactivity during background training.

Future work will extend RADICAL beyond 2D radar displays to multimodal settings that include 3D point clouds or camera streams and will explore federated variants of the micro-model strategy in which local models can contribute to shared priors without exposing raw sensor data.

References

[1] Y. Wang, Z. Jiang, X. Gao, et al., "RodNet: Radar Object Detection using Cross-Modal Supervision," *Proc. IEEE/CVF Winter Conf. on Applications of Computer Vision (WACV)*, 2021.

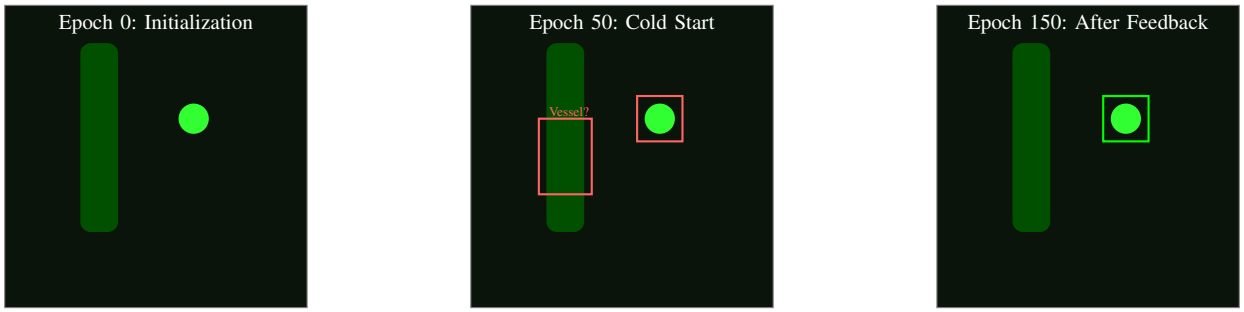
[2] J. Rebut, A. Ouaknine, W. Malik, et al., "RAW: A Robust and Lightweight End-to-End Radar-Camera Sensor Fusion," *IEEE Robotics and Automation Letters*, 2022.

[3] S. Amrehn et al., "UI-Net: Interactive Artificial Neural Networks for Segmenting Medical Images," *Eurographics Workshop on Visual Computing for Biology and Medicine*, 2017.

[4] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[5] B. Settles, "Active Learning Literature Survey," University of Wisconsin–Madison Department of Computer Sciences, Tech. Rep., 2009.

[6] N. Matsunaga et al., "Rust for Scientific Computing," *Journal of Open Source Software*, 2021.



(a) No predictions. (b) False positive on jetty. (c) Jetty suppressed, vessel retained.

Fig. 9: Evolution of on-device learning. Within 150 frames of interaction, the model learns to suppress static jetty clutter specific to this location, a distinction a generic model failed to make.

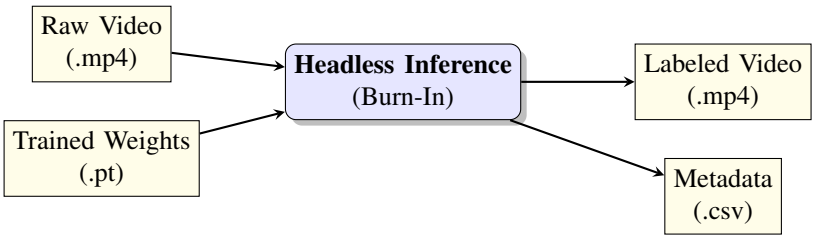


Fig. 10: Archival workflow. After a session, the tool can run in headless mode to re-play footage and burn the session-specific model's predictions into video and metadata files.