EDGE AI ENGINEERING SERIES

# The EdgeAI Handoff Checklist

Design-review questions for embedded engineers who are adding machine learning to a product

Companion to **The Embedded Engineer's Guide to Edge AI** by Satheesh Gannoju

Edge2XAI Publications · edge2xai.com

START HERE

# How to use this checklist

A working tool for design reviews. The reasoning behind every question is in the book.

The book follows one idea: an EdgeAI product is not a model, it is a product that contains a model. Six **handoffs** between the embedded side and the ML/data side decide whether it works in the field. Each handoff leaves behind one written artifact, and each artifact has an **exit gate**: a question the team must be able to answer before responsibility moves on.

## Use it in three steps

1. **Pick the handoff you are about to sign off.** Pages 4 to 9 cover H1 to H5, with an integration gate for the runtime on page 8; H6 (field and change) is on pages 12 to 14.
2. **For each line, mark it ☐ open or ☑ done, and write where the evidence is** (a document, a test, a link). A box ticked with no evidence is a guess.
3. **Do not move on until the exit gate is answered with evidence.** If a line does not apply to your product, write “N/A” and why.

## What this checklist is, and is not

It is a set of questions taken from the book, in the order the lifecycle meets them, with a pointer (§) to the section that explains each one. It is not a substitute for your product's safety, security or quality process, and it does not certify anything. Section numbers refer to the first edition of the book.

## Running example

The book uses a teaching reference system, the **Multi-Sensor Monitoring Node** (vibration, current, temperature and RPM/load on a motor-driven machine). Where this checklist says “the product”, substitute your own.

> Tip: copy this page set into your design-review template, one handoff per meeting, and keep the filled pages with the release record.

THE WHOLE METHOD ON ONE PAGE

# The six handoffs

Keep this page next to your project plan. Pages 4 to 14 expand each row into review questions.

| | Handoff and durable question | Artifact it leaves behind | Exit gate | Book |
|---|---|---|---|---|
| H1 | **Product ↔ ML Problem**<br>Are we solving the right product problem with ML? | Product / ML Acceptance Criteria | Can both sides trace the product requirement to measurable learned behavior and product-level acceptance criteria? | §2.4 |
| H2 | **Data ↔ Training**<br>Does the data represent the product world we claim to support? | Data Requirements Specification | Can the team explain which operating conditions are represented in the dataset, and which are intentionally absent? | §2.5 |
| H3 | **Model Input ↔ Firmware**<br>Will production software create exactly what the model expects? | Model Input Specification | Can a known input be processed through both paths and shown to produce equivalent model inputs? | §2.6 |
| H4 | **Model ↔ Deployment Target**<br>Can learned behavior fit the target without unacceptable loss? | Deployment Readiness Report | Can the embedded team integrate a model release without reverse-engineering assumptions from a training notebook? | §2.7 |
| H5 | **Model Output ↔ Product Policy**<br>How does learned evidence become defined product behavior? | Model Output → Product Decision Contract | Can a firmware engineer explain what every model output means and what the application is allowed to do with it? | §2.8 |
| H6 | **Field Behavior ↔ Controlled Evolution**<br>How does field evidence become a traceable, compatible, verifiable change? | Field EdgeAI Diagnostic Card | If an anomaly is reported from the field, can the team identify exactly which configuration produced it and collect enough evidence to investigate? | §2.9 |

A handoff is complete when its artifact exists **and** its exit gate is answered with evidence. The first gate you cannot answer is where your project actually is.

H1 · CHAPTERS 1–3

# Product ↔ ML Problem

Artifact: Product / ML Acceptance Criteria. Evidence to show: operating scenarios, failure-cost assumptions, deterministic-alternative analysis.

| | Check | Book | Owner | Evidence / link |
|---|---|---|---|---|
| ☐ | The product requirement is written without ML terms and would still make sense if no model existed. | §2.4 | | |
| ☐ | A deterministic threshold or rule was tried on paper first, and why it is not enough is written down. | §1.4 | | |
| ☐ | A physics or DSP signature was considered before asking for a learned model. | §1.5 | | |
| ☐ | The requirement traces to measurable learned behavior, and then to application-level acceptance criteria. | §2.4 | | |
| ☐ | “Model accuracy” is not the requirement. The promise to the customer is stated in product terms. | §3.1, §3.3 | | |
| ☐ | The validity envelope is defined: which equipment, sensor configurations and environments the promise covers. | §3.4 | | |
| ☐ | Conditions you will NOT assess are listed (they become NOT ASSESSED, not “normal”). | §3.4 | | |
| ☐ | What happens when required evidence is invalid or missing is defined (ASSESSMENT UNAVAILABLE). | §3.5 | | |
| ☐ | Which mistakes matter to the customer, and what each one costs, is written down before metrics are chosen. | §3.6 | | |
| ☐ | Critical protection stays deterministic; the learned contribution is stated separately. | §3.7 | | |
| ☐ | The EdgeAI Product Definition Card is complete and reviewed. | §3.8 | | |

**Exit gate.** Can both sides trace the product requirement to measurable learned behavior and product-level acceptance criteria?

**Failure signature.** The model team optimizes a metric that looks good while the product generates unacceptable nuisance indications or misses what the customer cares about.

H2 · CHAPTER 4

# Data ↔ Training

Artifact: Data Requirements Specification. Evidence to show: coverage analysis, label definitions, lineage, independence checks, known gaps.

| | Check | Book | Owner | Evidence / link |
|---|---|---|---|---|
| ☐ | The data requirement is derived from the product world, not from what was easy to collect. | §4.2 | | |
| ☐ | Quantity, coverage and quality are tracked as three separate questions. | §4.3 | | |
| ☐ | The capture path (sensor, mounting, configuration) is part of the dataset definition. | §4.4 | | |
| ☐ | Equipment, operating-state, speed/load, environmental and sensor-configuration coverage are each stated. | §4.5, §4.12 | | |
| ☐ | “Normal” is not one condition: normal variation across units and states is represented. | §4.6 | | |
| ☐ | Abnormal data is included only where the evidence is trustworthy; uncertain cases stay marked uncertain. | §4.7 | | |
| ☐ | Metadata travels with every recording, enough to place it in the product world. | §4.8 | | |
| ☐ | Invalid-data cases (dropouts, faults, bad captures) are represented for the unavailable/degraded behavior. | §4.12 | | |
| ☐ | Evaluation data is independent in the way the product must generalize (held-out machines or sessions, not randomly scattered windows). | §4.10 | | |
| ☐ | A coverage matrix exists and the gaps are recorded, not hidden by total hours. | §4.11 | | |
| ☐ | The Data Requirements Specification is complete. | §4.12 | | |

**Exit gate.** Can the team explain which operating conditions are represented in the dataset, and which are intentionally absent?

**Failure signature.** Training and evaluation use steady-state lab data. Production adds transients, variable speed, different mounting and load. Offline results impress; field behavior does not.

H3 · CHAPTER 5

# Model Input ↔ Firmware

Artifact: Model Input Specification. Evidence to show: the specification, golden vectors, preprocessing checks, host-to-target parity.

| | Check | Book | Owner | Evidence / link |
|---|---|---|---|---|
| ☐ | Raw sensor data, features and the model input are distinguished and named. | §5.2 | | |
| ☐ | Observation window and stride are decided as separate choices. | §5.4 | | |
| ☐ | Multi-rate signals are time-aligned by one stated rule, with a freshness limit for slow context. | §5.3 | | |
| ☐ | High-rate feature extraction and slow operating context are each defined. | §5.4 | | |
| ☐ | The feature vector has a fixed, versioned order and layout. | §5.4 | | |
| ☐ | Scaling and normalization constants are versioned; they come from the training partition only. | §5.4 | | |
| ☐ | The Model Input Specification states type, units, timing, ordering, scaling and invalid-input behavior. | §5.5 | | |
| ☐ | Missing or invalid context is handled by rule, never by silent substitution. | §2.6 | | |
| ☐ | The same input pipeline is implemented in development and in firmware, from one written contract. | §5.6 | | |
| ☐ | A golden-vector test runs one known observation through both paths, with intermediate checkpoints and an agreed tolerance. | §5.6 | | |
| ☐ | When the paths disagree, you locate the first stage where values differ before blaming the model. | §5.6 | | |

**Exit gate.** Can a known input be processed through both paths and shown to produce equivalent model inputs?

**Failure signature.** The model passes offline evaluation and firmware passes unit tests, but on-target inference diverges because the two pipelines window, scale, order or compute features differently.

H4 · CHAPTERS 6–8

# Model ↔ Deployment Target

Artifact: Deployment Readiness Report. Evidence to show: reference-to-deployment comparison, operator compatibility, measured memory and latency.

| | Check | Book | Owner | Evidence / link |
|---|---|---|---|---|
| ☐ | The learned task and output meaning were agreed before an architecture was chosen. | §6.4 | | |
| ☐ | The proposed model was challenged against a simpler baseline. | §6.6 | | |
| ☐ | The reference-model package is complete: checkpoint, input-spec version, output specification, training configuration, lineage, evaluation, known limitations. | §6.10 | | |
| ☐ | A real deployment budget exists first: flash, RAM, latency, throughput, power. | §7.3 | | |
| ☐ | You asked whether optimization is needed at all before choosing a technique. | §7.5 | | |
| ☐ | Each technique attacks a gate that actually failed, and you stop when the gate passes. | §7.4, §7.12 | | |
| ☐ | Precision is chosen as a hardware/runtime decision, not just a bit width. | §7.6 | | |
| ☐ | Calibration data covers the activation ranges the product will see. | §7.7 | | |
| ☐ | Reference and deployment models were compared on threshold-sensitive samples. | §7.16 | | |
| ☐ | Every operator resolves intentionally; the kernel path is known and there is no unplanned fallback. | §8.3 | | |
| ☐ | Tensor arena size was measured on the target with a defined margin. | §8.5 | | |
| ☐ | Known-vector inference matches across host, runtime and target. | §8.9 | | |
| ☐ | Invoke() latency is measured on the target; the worst case, not the average, is recorded. | §8.8 | | |
| ☐ | The Deployment Readiness Report and the Runtime Readiness Checklist are complete. | §7.19, §8.12 | | |

**Exit gate.** Can the embedded team integrate a model release without reverse-engineering assumptions from a training notebook?

**Failure signature.** The model is excellent on the development machine but cannot execute within production memory, latency, operator or runtime constraints, and the failure is found at integration.

INTEGRATION GATE · CHAPTER 9

# From inference to embedded product

Artifact: EdgeAI Runtime Integration Contract. Closes the gap between “the model runs” and “the product still works”.

| | Check | Book | Owner | Evidence / link |
|---|---|---|---|---|
| ☐ | A before-and-after audit of the existing product exists; the working baseline is recorded. | §9.2 | | |
| ☐ | The integration resource budget adds the AI cost to the existing CPU, memory and timing budget. | §9.3 | | |
| ☐ | CPU budget uses integrated wall-clock latency, not the standalone Invoke() time. | §9.3, §9.6 | | |
| ☐ | The tensor arena is accounted for alongside every existing memory tenant, with measured high-water marks. | §9.3 | | |
| ☐ | Where inference runs (existing task, dedicated context, core, DSP/NPU) is a reasoned decision. | §9.4 | | |
| ☐ | Priority comes from deadline and interference analysis, not from how “important” the AI is. | §9.4 | | |
| ☐ | The RTOS handoff primitive (event, queue, buffer) matches the semantics: signaling versus ownership. | §9.4 | | |
| ☐ | DMA buffers are never consumed while still owned by DMA; cache coherency rules are followed on shared buffers. | §9.4, §9.5 | | |
| ☐ | Sampling, feature, inference and decision rates are stated separately. | §9.6 | | |
| ☐ | An explicit backpressure policy exists (drop, latest, degrade), proven by overload injection. | §9.6 | | |
| ☐ | Existing-product regression is an acceptance gate, not cleanup. | §9.7 | | |
| ☐ | Stacks are sized for the whole EdgeAI call path; startup reports “AI ready” only after runtime and valid history. | §9.8, §9.10 | | |
| ☐ | A failure of the AI path cannot silently become “normal”; watchdog and recovery are defined. | §9.10 | | |

> Evidence the book asks for: architecture review and trace, stress and ownership tests, RTOS trace, overload injection, memory map and high-water marks, boot trace, fault injection, system regression suite (§9.16).

H5 · CHAPTER 10

# Model Output ↔ Product Policy

Artifact: Model Output → Product Decision Contract. Evidence to show: score and threshold analysis, persistence and hysteresis replay, invalid/rejected cases, state-transition evidence.

| | Check | Book | Owner | Evidence / link |
|---|---|---|---|---|
| ☐ | You know what kind of output the model produces: score, probability, confidence and calibration are not collapsed into one word. | §10.3 | | |
| ☐ | The evidence is validated (sensors valid, fresh, supported operating state) before the score is interpreted. | §10.4 | | |
| ☐ | The product decision vocabulary (states) is defined before any policy code is written. | §10.5 | | |
| ☐ | There is a reject/abstain path for ambiguous or unfamiliar input. | §10.5 | | |
| ☐ | Thresholds come from independent evidence and are versioned. | §10.6 | | |
| ☐ | False positives and false negatives are translated into product consequences. | §10.6 | | |
| ☐ | Persistence is chosen with its cost (detection delay) measured; hysteresis uses separate enter and exit levels. | §10.7 | | |
| ☐ | Operating-state gating is defined: a high score in an unsupported state is not a warning. | §10.7 | | |
| ☐ | The product state machine is explicit; advisory versus action-capable AI is a stated decision. | §10.8 | | |
| ☐ | Behavior when evidence disappears is defined. | §10.8 | | |
| ☐ | Policy is versioned together with the model, and the decision path is instrumented. | §10.9 | | |
| ☐ | The policy was replayed against recorded scores before full system validation. | §10.12 | | |

**Exit gate.** Can a firmware engineer explain what every model output means and what the application is allowed to do with it?

**Failure signature.** The model is statistically sound, but the application turns every transient score into an alert. Customers get nuisance indications and lose trust in the feature.

VERIFICATION · CHAPTER 11

# The EdgeAI verification ladder

One gate per lifecycle stage. Each row names the question it answers and the evidence that answers it. Record where that evidence lives.

| | Gate | Question it answers | Evidence | Where | Evidence link |
|---|---|---|---|---|---|
| ☐ | **0 · Requirement** | Is expected behavior measurable? | Acceptance criteria; NOT ASSESSED / UNAVAILABLE rules | Review | |
| ☐ | **1 · Data** | Is the evidence valid, representative, independent? | Dataset report, coverage and leakage checks | Offline | |
| ☐ | **2 · Model input** | Do development and firmware mean the same thing? | Golden vectors; intermediate checkpoints | Offline + target | |
| ☐ | **3 · Reference model** | Does learned behavior meet the target? | Held-out metrics; error analysis by operating region | Offline | |
| ☐ | **4 · Deployment model** | Did optimization preserve behavior? | Reference ↔ deployment parity on threshold-sensitive samples | Offline | |
| ☐ | **5 · Target runtime** | Does the real target execute correctly? | Known vectors; arena; latency; operator mapping | MCU / SoC | |
| ☐ | **6 · Integration** | Does AI coexist with the existing system? | RTOS trace; stacks; queues; DMA / cache; regression | Target + RTOS | |
| ☐ | **7 · Decision policy** | Does evidence become the correct product state? | Policy replay; boundary scenarios | SIL + target | |
| ☐ | **8 · System** | Does the requirement pass end to end, under stress and failure? | Robustness, fault-injection, HIL, long-duration results | Target + HIL | |

## Before you trust the ladder

| | Check | Book | Owner | Evidence / link |
|---|---|---|---|---|
| ☐ | Each failure is caught at the cheapest valid test environment, not only on the final system. | §11.2 | | |
| ☐ | Verification depth follows the consequence of the failure. | §11.5 | | |
| ☐ | Robustness testing deliberately makes the embedded world messy: noise, drift, sensor faults. | §11.5 | | |
| ☐ | Reliability evidence is more than repeating inference many times. | §11.5 | | |
| ☐ | HIL and fault injection create the conditions you cannot wait for. | §11.5 | | |
| ☐ | Regression proves the new AI feature did not break the old product. | §11.5 | | |

Ladder source: §11.3. Failure analysis: §11.4.

RELEASE · CHAPTER 11

# Production evidence package

“Testing complete” is not a release artifact. A release review should be able to inspect the evidence behind the product claim. §11.6

| | # | Evidence item | Owner | Location / link |
|---|---|---|---|---|
| ☐ | 1 | Product requirements + acceptance criteria | | |
| ☐ | 2 | Data verification / representativeness report | | |
| ☐ | 3 | Model Input Specification + golden-vector evidence | | |
| ☐ | 4 | Reference-model validation + error analysis | | |
| ☐ | 5 | Deployment-model parity | | |
| ☐ | 6 | Target runtime correctness + memory/latency evidence | | |
| ☐ | 7 | RTOS / resource / integration evidence | | |
| ☐ | 8 | Product-decision-policy verification | | |
| ☐ | 9 | EdgeAI FMEA-style analysis + risk-reduction actions | | |
| ☐ | 10 | Fault-tree / top-event analysis where useful | | |
| ☐ | 11 | Robustness + fault-injection results | | |
| ☐ | 12 | HIL / system scenario results | | |
| ☐ | 13 | Existing-product regression evidence | | |
| ☐ | 14 | Long-duration reliability evidence | | |
| ☐ | 15 | Known limitations / unsupported conditions | | |
| ☐ | 16 | Residual risks | | |
| ☐ | 17 | Model / runtime / input-spec / policy versions | | |
| ☐ | 18 | Release recommendation / open actions | | |

Items 2 to 8 are the ladder gates; items 9 to 14 are system-level failure analysis and stress; items 15 to 18 make the release decision honest. A release recommendation with no known limitations listed should be questioned.

H6 · CHAPTER 12

# Field EdgeAI diagnostic card

Artifact: Field EdgeAI Diagnostic Card. Fill one in per field report, before anyone proposes retraining. §12.9

### IDENTITY

☐ Which hardware revision?
☐ Which sensor/configuration revision?
☐ Which Model Input Specification?
☐ Which model / deployment model?
☐ Which runtime / accelerator path?
☐ Which policy/threshold version?

### PHYSICAL EVIDENCE

☐ Raw sensor valid?
☐ Sensor/mounting/calibration changed?
☐ Operating state supported?
☐ RPM/load/temp context fresh?

### EDGEAI PATH

☐ Feature parity valid?
☐ Model score reproducible offline?
☐ Target/deployment parity valid?
☐ Inference on time?
☐ Queue/backpressure healthy?
☐ DMA/cache/NPU ownership healthy?

### PRODUCT DECISION

☐ Score interpreted correctly?
☐ Threshold/persistence/hysteresis version correct?
☐ NOT ASSESSED / UNAVAILABLE / UNKNOWN handled correctly?

### FIELD CHANGE

☐ Machine/process changed?
☐ Measurement changed?
☐ Software/config changed?
☐ Model/policy changed?
☐ Population/site changed?

### CORRECTIVE ACTION

☐ Sensor/installation
☐ Firmware/input pipeline
☐ RTOS/runtime
☐ Policy
☐ Data collection
☐ Model/retraining
☐ Hardware/platform

### BEFORE CLOSURE

☐ Reproduce root cause
☐ Correct at the right lifecycle phase
☐ Rerun downstream verification
☐ Update known limitations / monitoring
☐ Add prevention test to lifecycle

**Exit gate.** If an anomaly is reported from the field, can the team identify exactly which configuration produced it and collect enough evidence to investigate?

**Failure signature.** The team retrains on vaguely described field cases, cannot reproduce the original behavior, and ships an update without knowing whether model, preprocessing, firmware, sensors or policy caused the issue.

H6 · CHAPTER 12

# Find the layer before you retrain

Work down the questions in order. The first check that fails names the layer to fix. §12.7

## The four questions that classify most field problems §12.2

**1** Did the physical system change?
wear, maintenance, load, environment

**2** Did our measurement of it change?
sensor, mounting, calibration, gain, sampling, timestamps

**3** Did the EdgeAI computation path change?
features, model, quantization, runtime, RTOS timing, versions

**4** Did product interpretation change?
threshold, persistence, hysteresis, operating-state gate, policy

## Root-cause order §12.7

| # | Check, in order | If this answer… | …the fix is in | Result |
|---|---|---|---|---|
| 1 | Is the physical / sensor evidence valid? | **NO** | sensor / installation / acquisition fix | |
| 2 | Does production feature data match the Model Input Specification? | **NO** | preprocessing / firmware / configuration fix | |
| 3 | Did the model, deployment or runtime version change? | **YES** | parity / release regression | |
| 4 | Is runtime timing, freshness and queue behavior valid? | **NO** | RTOS / integration fix | |
| 5 | Did product policy interpret the score correctly? | **NO** | policy fix / revalidation | |
| 6 | Is the input distribution outside validated field behavior? | **YES** | collect evidence; data / model lifecycle review | |

Only if all six checks pass is a genuine model limitation suspected, and even then representative evidence is collected before retraining. A version change at check 3 is a lead, not a verdict: run the parity and release-regression checks, and continue down the list if they pass.

## Do not retrain first when the cause is… §12.7

- stale/misaligned context
- sensor failure or installation change
- a preprocessing parity defect
- a quantization/runtime regression
- an RTOS timing/backpressure problem
- wrong operating-state gating
- a threshold/persistence defect
- a configuration/version mismatch

H6 · CHAPTER 13

# Controlled evolution

Artifact: EdgeAI Change & Release Record. Closes the loop and sends the product back around the lifecycle with its history intact.

| | Check | Book | Owner | Evidence / link |
|---|---|---|---|---|
| ☐ | The field issue is traced to a root cause before any change is proposed. | §13.2 | | |
| ☐ | Change impact radius is assessed: the earlier the artifact changed, the more must be reverified. | §13.2 | | |
| ☐ | Field data is not assumed to be training data; its selection, labeling and independence are decided. | §13.3 | | |
| ☐ | A new model is treated as a regression event: better metrics can still be a worse product. | §13.4 | | |
| ☐ | Any change to the product policy that the new model implies is identified and verified. | §13.4 | | |
| ☐ | The unit of change is an AI Configuration (model, input spec, quantization, runtime, policy, thresholds, firmware and hardware compatibility), not a model file. | §13.5 | | |
| ☐ | Update safety is designed: slots, power loss, integrity, self-test, and a recoverable meaning for every update state. | §13.6 | | |
| ☐ | Fleet compatibility and model–policy coupling are checked before a unit is eligible. | §13.7 | | |
| ☐ | Rollout is staged, with shadow operation only where the target can afford it. | §13.7 | | |
| ☐ | Rollback restores a complete, compatible, still-authorized AI Configuration, not just an older model. | §13.7 | | |
| ☐ | Post-update monitoring checks that the original field problem was actually solved. | §13.7 | | |
| ☐ | An EdgeAI Change & Release Record answers: why does this model version exist, and what evidence justified it? | §13.8 | | |

A record entry names: trigger, root cause, dataset / model / input-spec / deployment / runtime versions, resource change, policy change, verification reopened, compatibility, rollback target, and post-release monitoring (§13.8).

THE SHORT VERSION

# Twenty questions for any EdgeAI project

If you only have ten minutes in a review, ask these. Any question the team cannot answer with evidence shows where to look. §14.10

| | | |
|---|---|---|
| 1 | What product problem are we solving? | G A R |
| 2 | Why does it need ML? | G A R |
| 3 | What is inside/outside the assessment envelope? | G A R |
| 4 | What physical evidence reaches the model? | G A R |
| 5 | Is the data representative? | G A R |
| 6 | What exactly is the model input? | G A R |
| 7 | Why this model? | G A R |
| 8 | Why this optimization? | G A R |
| 9 | What does the target hardware really allow? | G A R |
| 10 | What is the runtime/memory cost? | G A R |
| 11 | How do we prove host ↔ target parity? | G A R |
| 12 | What happens if inference is late? | G A R |
| 13 | What happens if a sensor is invalid? | G A R |
| 14 | What does the model output actually mean? | G A R |
| 15 | How does product policy interpret it? | G A R |
| 16 | What can fail across the lifecycle? | G A R |
| 17 | What must be observable in the field? | G A R |
| 18 | What triggers a model/data/policy change? | G A R |
| 19 | How is that change deployed and rolled back? | G A R |
| 20 | What evidence proves the product, not only the model, is ready? | G A R |

Mark each question **G** (answered with evidence), **A** (answered, no evidence yet) or **R** (cannot answer). The Rs are your project risks.

FOR YOU

# Can I own an EdgeAI feature?

A self-assessment from the book. You do not need every box before starting; the gaps show where to practice next. §14.4

☐ I can explain why the product needs ML.
☐ I can challenge data quality and coverage.
☐ I can define or review the exact model input.
☐ I can explain why a model architecture fits the application.
☐ I understand deployment/quantization trade-offs.
☐ I can estimate and measure target resources.
☐ I can integrate inference into real embedded software.
☐ I can distinguish model output from product decision.
☐ I can design host/deployment/target parity tests.
☐ I can analyze EdgeAI-specific failure modes.
☐ I can instrument the field evidence path.
☐ I can explain what should trigger a model change.
☐ I can define what must be reverified after that change.

## Where each chapter fits

| | |
|---|---|
| **Ch 1** | The System Worked. The Machine Still Failed. |
| **Ch 2** | Six Embedded↔ML Handoffs |
| **Ch 3** | The Product Promise |
| **Ch 4** | Data Becomes Part of the Product |
| **Ch 5** | One Input, Two Pipelines |
| **Ch 6** | Who Owns the Model? |
| **Ch 7** | When to Stop Optimizing |
| **Ch 8** | Inside the Inference Runtime |
| **Ch 9** | Inference Meets the RTOS |
| **Ch 10** | The Model Said 0.83 |
| **Ch 11** | The Verification Ladder |
| **Ch 12** | It Passed in the Lab |
| **Ch 13** | A Model Update Is a Product Release |
| **Ch 14** | Can You Own an EdgeAI Feature? |

**About this checklist.** Free companion to The Embedded Engineer's Guide to Edge AI by Satheesh Gannoju (Edge2XAI Publications), an embedded-systems engineer with nearly two decades in real-time firmware and system architecture. The book explains the reasoning behind every line here. Updates: **edge2xai.com**

**What changed.**
**v1.2 · Nov 2026** Section pointers (§) now follow the final book's numbering (about 60% of them changed when the chapters were regrouped). Chapter titles match the final book. Root-cause check 3 now says a version change is a lead, not a verdict.
**v1.0 · Oct 2026** First release. If you printed v1.0, replace it: many of its § pointers no longer match the book.

© 2026 Satheesh Gannoju. You may share this checklist unmodified and use it inside your own organization's design reviews. Examples are illustrative and do not replace your product's safety, security or quality process. Version 1.2.