Threat Modeling Report for an Edge-AI Deployment Pipeline

STRIDE, PASTA, and MITRE ATLAS Mapping for a CIFAR-10 CNN trained on RTX 5080 and deployed on Jetson Orin Nano

Prepared for: Brojogopal Sapui
Version: 1.0 | Date: 08 May 2026
Status: Baseline threat model for research and security learning workflow

Scope: RTX 5080 CUDA/PyTorch training, ONNX export, SCP transfer, Jetson Orin Nano TensorRT FP16 deployment, and a reference inference microservice design.

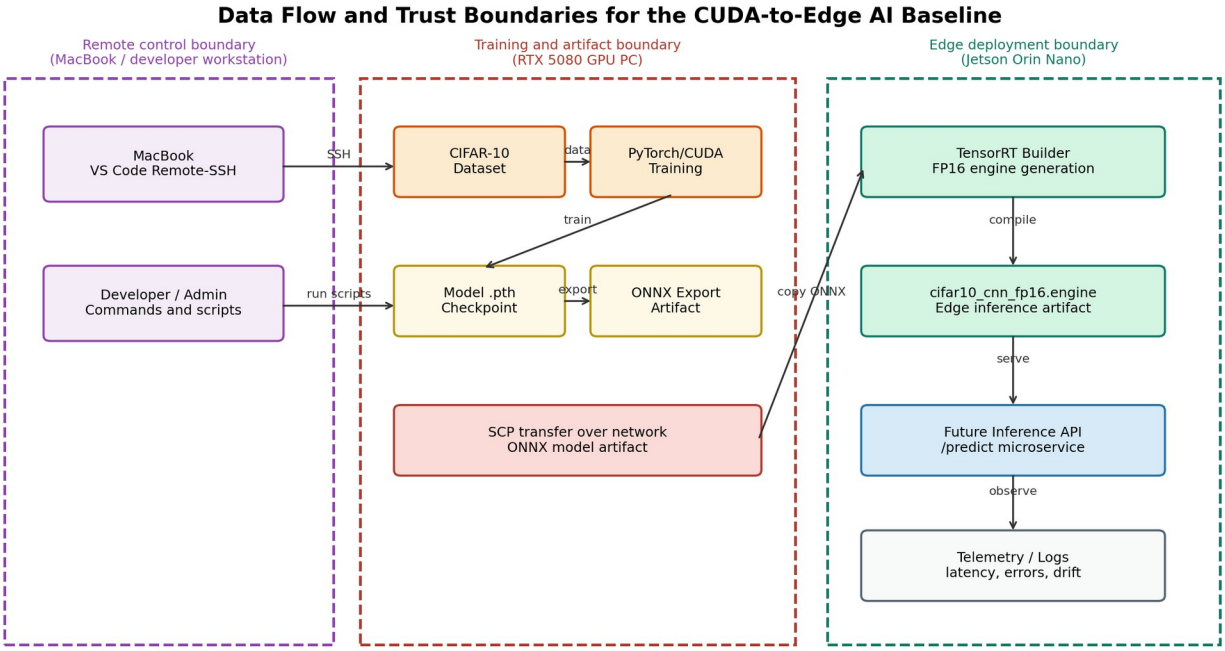

Figure 1. In-scope data flow and trust boundaries for the completed CUDA-to-edge baseline.

1. Executive Summary

Purpose. This document turns the completed CUDA-to-edge experiment into an industry-style threat modeling artifact. The technical baseline trained a CIFAR-10 CNN on an RTX 5080 GPU using PyTorch/CUDA, exported the trained model to ONNX, transferred the ONNX artifact to a Jetson Orin Nano, and compiled it into a TensorRT FP16 engine for optimized edge inference.

Security interpretation. The value of this baseline is not the CIFAR-10 model itself. The value is that the end-to-end model lifecycle is now concrete enough to analyze as an attack surface: training data, training code, model checkpoints, exported ONNX artifacts, file transfer, device runtime, TensorRT engine generation, inference API behavior, telemetry, and hardware-observable execution.

Observed result	Value from experiment	Security relevance
Training platform	RTX 5080 GPU desktop; PyTorch/CUDA; CUDA 12.8 visible from PyTorch	Separates training-time trust boundary from deployment-time trust boundary.
Model accuracy	80.16% test accuracy after 8 epochs	Baseline for integrity checks; future adversarial or poisoning tests can be measured against this reference.
Model artifacts	cifar10_cnn_cuda.pth, cifar10_cnn_cuda.onnx, cifar10_cnn_fp16.engine	Each artifact is a security-sensitive object that requires provenance, integrity, and access control.
RTX inference	PyTorch GPU inference approx. 0.4455 ms for one test sample loop	Framework-level reference behavior before edge compilation.
Jetson deployment	TensorRT 10.3.0; Orin GPU; FP16 engine generation succeeded	Hardware-specific optimized artifact; may differ from the training framework in behavior, precision, timing, and observability.
Jetson benchmark	TensorRT mean latency approx. 0.212 ms; throughput approx. 5012 qps	Runtime profile provides a baseline for anomaly detection, DoS testing, and side-channel-aware characterization.

Threat modeling frameworks used. STRIDE is used to systematically enumerate security properties by component; PASTA is used to structure risk-driven analysis from business objectives through attack simulation and impact; MITRE ATLAS is used to map AI/ML-specific adversarial behavior such as model access, poisoning, backdooring, inference API abuse, model extraction, and adversarial input generation. Microsoft describes STRIDE as a model that categorizes threats into spoofing, tampering, repudiation, information disclosure, denial of service, and elevation of privilege [R3]. OWASP frames threat modeling as a process of scoping the system, determining threats, identifying countermeasures, and validating the work [R4][R5]. MITRE describes ATLAS as a living knowledge base of adversary tactics and techniques against AI-enabled systems [R1].

2. System Scope and Architecture

2.1 In-scope system

- MacBook used as the remote control workstation through VS Code Remote-SSH.
- RTX 5080 GPU desktop used for CUDA/PyTorch training, ONNX export, and local inference testing.
- Jetson Orin Nano used for edge deployment, TensorRT FP16 engine generation, and benchmarking.
- Model artifacts: PyTorch checkpoint (.pth), ONNX model (.onnx), TensorRT engine (.engine).
- Commands and transfer path: scp from RTX desktop to Jetson user directory /home/brojonano/Documents/ai_lab.
- Future sample microservice: an edge inference service exposing a /predict endpoint that calls the TensorRT engine and returns the predicted CIFAR-10 class.

2.2 Out-of-scope items for this baseline

- A production-grade cloud model registry is not yet deployed; this report provides the control requirements that such a registry should satisfy.
- No real customer data or personally identifiable information is used; CIFAR-10 is a public research dataset, so privacy risk is lower than for enterprise datasets.
- A hardened production API gateway, certificate infrastructure, SIEM integration, and secure boot validation are future extensions.

- The current benchmark uses trtexec random input benchmarking; application-level inference with real image input on Jetson is the next validation step.

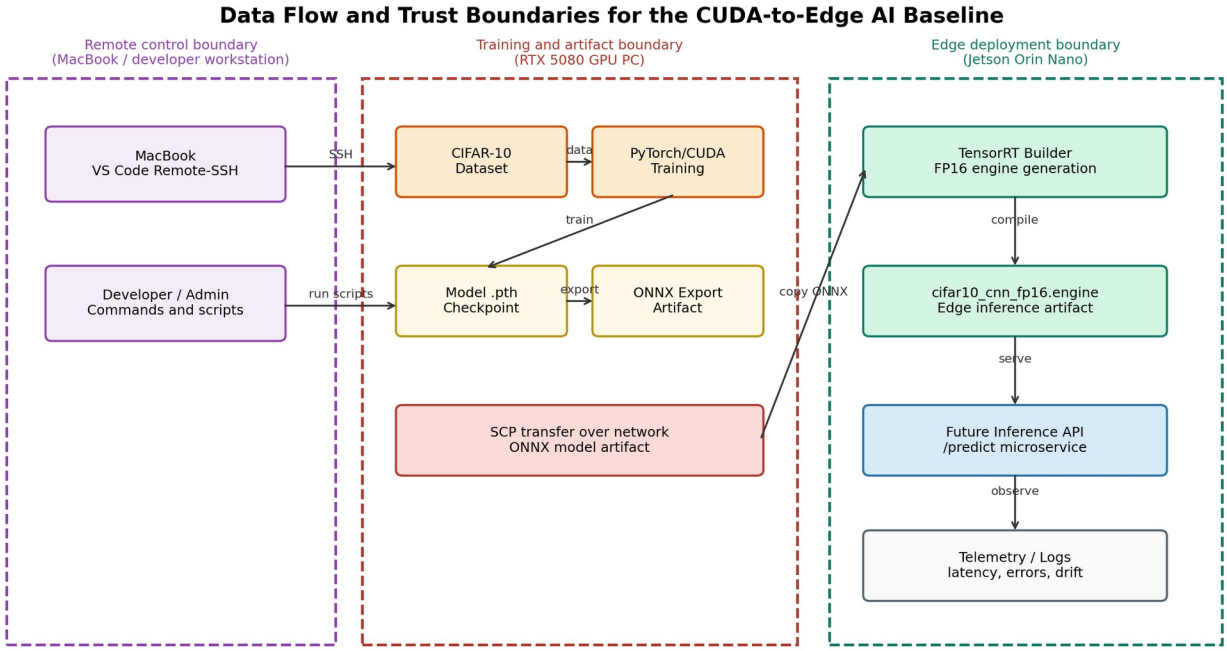

Figure 2. Threat modeling DFD: components, data flows, artifacts, and trust boundaries.

2.3 Critical assets

ID	Asset	Location	Primary security property	Why it matters
A1	Training data and labels	./data/CIFAR-10 on RTX system	Integrity, provenance, reproducibility	Poisoned labels or samples can alter model behavior.
A2	Training code and environment	train_cifar10_cuda.py; Miniconda ml environment	Integrity, reproducibility	Malicious dependency or code change can silently alter model.
A3	PyTorch checkpoint	cifar10_cnn_cuda.pth	Integrity, confidentiality	Checkpoint can be replaced, backdoored, or stolen.
A4	ONNX artifact	cifar10_cnn_cuda.onnx	Integrity, authenticity	ONNX graph is the handoff artifact between training and deployment.
A5	TensorRT engine	cifar10_cnn_fp16.engine	Integrity, hardware binding	Engine can be replaced on Jetson or rolled back to vulnerable version.
A6	Inference API inputs/outputs	Future /predict microservice	Availability, integrity, abuse resistance	Query interface enables extraction, evasion, and DoS.
A7	Telemetry and logs	Benchmark logs, latency traces, future service logs	Integrity, confidentiality, non-repudiation	Needed for detection, audit, and incident reconstruction.
A8	Device credentials and SSH access	MacBook to RTX; RTX to Jetson	Confidentiality, authorization	Credential compromise enables direct artifact tampering.

2.4 Trust boundaries

Boundary	Crossing flow	Risk introduced	Minimum control
B1: MacBook to RTX GPU desktop	VS Code Remote-SSH commands and file edits	Credential theft, command misuse, remote code execution on training host	SSH keys, strong local device security, MFA where possible, command audit.
B2: RTX training environment to model artifacts	Training produces .pth and ONNX export	Silent tampering of code, data, or dependency changes model	Pinned dependencies, hash logs, Git commit IDs, model lineage

		artifact	metadata.
B3: RTX to Jetson transfer	scp cifar10_cnn_cuda.onnx to /home/brojonano/Documents/ai_lab	Artifact replacement in transit or wrong destination permissions	SSH host key validation, checksum verification, signed artifacts.
B4: Jetson filesystem to TensorRT runtime	TensorRT compiles ONNX to engine and loads engine	Malicious model/engine executed on edge device	Read-only deployment directory, signature verification before build/load, controlled runtime user.
B5: Future client to inference microservice	Image upload and prediction response	Evasion, extraction, request flooding, malicious payloads	Authentication, input validation, rate limits, monitoring, output policy.

3. Reference Inference Microservice Design

Design intent. The sample microservice is a lightweight edge inference service that exposes a /predict endpoint. It accepts an image, validates and preprocesses it to CIFAR-10 tensor shape, calls the TensorRT FP16 engine, and returns the predicted label and confidence score. Even though the current experiment used trtexec for benchmarking, this reference service is the appropriate target for STRIDE and ATLAS analysis because it represents how the model would be consumed in practice.

Component	Role	Data handled	Security responsibility
Client	Sends image to /predict	Image input, request metadata	Authenticate, prevent replay, obey rate limits.
API Gateway / Reverse Proxy	Terminates TLS and routes requests	HTTP headers, tokens, request body	TLS/mTLS, request size limit, WAF rules, rate limiting.
Inference Service	Validates input and invokes TensorRT engine	Normalized tensor, prediction output	Input validation, least privilege, structured logging, exception isolation.
Model Artifact Store	Stores approved .onnx and .engine files	Model files, hashes, signatures, metadata	Integrity, version control, signing, access control.
Telemetry Agent	Exports latency, errors, confidence drift	Logs, metrics, GPU utilization	Tamper-resistant logging, privacy filtering, alerting.
Admin / DevOps	Deploys model and config	SSH keys, deployment scripts, model versions	RBAC, change approval, audit trail.

4. STRIDE Threat Model for the Sample Microservice

Method. STRIDE is applied per component and per data flow. The goal is not to produce one generic checklist, but to ask concrete questions: who can impersonate the client, what can be modified, which actions cannot be proven, what may be exposed, what can be made unavailable, and where privilege can be increased. This follows the practical DFD-driven approach recommended by threat modeling guidance [R3][R4][R5].

STRIDE category	Main target	Threat scenario	Impact	Primary mitigations
S: Spoofing	Client/API identity	Attacker uses stolen token or SSH key to submit requests or deploy artifacts as a legitimate user.	Unauthorized inference use, extraction attempts, or model replacement.	mTLS/JWT for API; SSH keys over passwords; key rotation; identity-bound deployment approvals.
T: Tampering	ONNX model, TensorRT engine, preprocessing code	Adversary modifies cifar10_cnn_cuda.onnx or cifar10_cnn_fp16.engine before deployment.	Backdoored model, incorrect predictions, hidden behavior under trigger input.	SHA-256 verification; signed artifacts; read-only deployment directory; Git commit and model lineage log.
R: Repudiation	Training, export, transfer, deployment actions	A user denies replacing a model or running an export/build command because there is no audit record.	Poor incident response and inability to attribute compromise.	Append-only audit logs; command logging; deployment ticket ID; artifact manifest with timestamp and actor.
I: Information Disclosure	Model weights, output confidence, logs, telemetry	Attacker downloads model or uses API outputs to infer model behavior or training membership.	Model IP theft, privacy leakage, competitive intelligence loss.	Restrict file access; confidence rounding; rate limits; log redaction; encrypt backups and transfers.
D: Denial of Service	Inference endpoint and	High-rate image requests,	Inference unavailable;	Input size limits; queue

	Jetson resources	oversized files, malformed inputs, or GPU memory pressure.	thermal throttling; degraded real-time behavior.	limits; GPU health monitoring; circuit breaker; nvpmodel/thermal alerting.
E: Elevation of Privilege	Service user, deployment user, device OS	Inference service vulnerability leads to shell access or model directory write access.	Attacker moves from API abuse to persistent edge-device control.	Run service as non-root; container isolation; AppArmor/seccomp; restricted sudo; patched JetPack and dependencies.

4.1 STRIDE-by-data-flow analysis

Data flow	STRIDE concerns	Threat example	Controls to implement
F1: Developer edits training/export scripts over Remote-SSH	Spoofing, Tampering, Repudiation	Wrong SSH identity or compromised account changes code and exports a malicious ONNX.	SSH key-based access, Git status check, signed commits, command audit.
F2: Dataset downloaded and loaded by torchvision	Tampering, Information Disclosure	Dataset or labels are poisoned or replaced with a malicious variant.	Dataset checksum, trusted source, dataset versioning, outlier checks.
F3: .pth checkpoint generated from training	Tampering, Repudiation	Checkpoint is overwritten after training but before export.	Immutable artifact folder, checkpoint hash immediately after training, artifact manifest.
F4: ONNX exported from .pth	Tampering, Elevation of Privilege	ONNX graph contains unexpected operations or is exported from wrong model architecture.	ONNX validation, graph diff, operator allowlist, export script review.
F5: ONNX transferred by SCP to Jetson	Spoofing, Tampering, Repudiation	Wrong Jetson user/path or replaced artifact on edge device.	Known_hosts validation, correct user brojonano, checksum before and after transfer.
F6: TensorRT builds FP16 engine	Tampering, Information Disclosure	Engine file replaced after build or build logs leak environment details.	Build in protected directory, engine hash, minimal logging in production.
F7: Future /predict API returns label/confidence	Information Disclosure, DoS	Query-based model extraction or high-rate request flood.	Rate limiting, output minimization, confidence clipping, anomaly detection.

5. PASTA Analysis

Method. PASTA is used here as a risk-centric methodology: it connects the technical pipeline to security objectives, attack simulation, and business/research impact. Public PASTA references describe seven stages: define objectives, define technical scope, decompose the application, analyze threats, analyze vulnerabilities, analyze attacks, and assess risk/impact [R6].

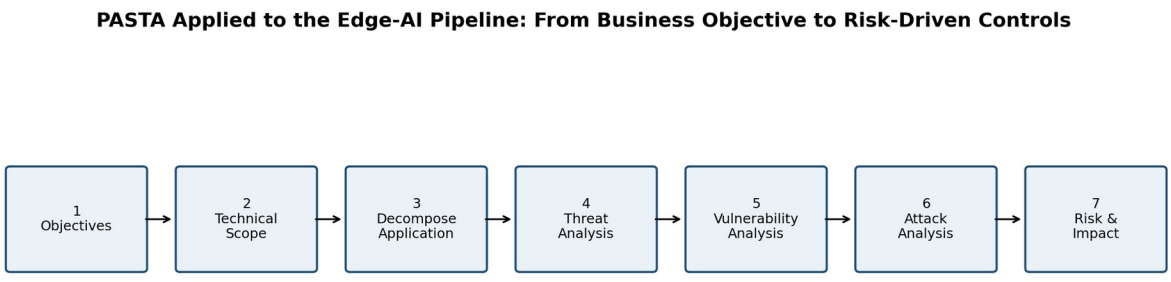

Figure 3. PASTA stages applied to the CUDA-to-edge AI workflow.

PASTA stage	Application to this project	Artifacts produced	Key conclusion
1. Define objectives	Create a reproducible edge AI	Model accuracy, artifact integrity,	Security objective: protect model

	baseline that can later support secure AI research.	deployment reproducibility, low-latency inference, security observability.	lineage and runtime behavior, not only model accuracy.
2. Define technical scope	RTX training host, Miniconda ml environment, PyTorch/CUDA, ONNX, SCP, Jetson Orin Nano, TensorRT.	Training code, dataset, .pth, .onnx, .engine, SSH identities, Jetson filesystem.	Scope includes both ML artifacts and infrastructure access paths.
3. Decompose application	Break pipeline into data, training, export, transfer, deployment, inference, monitoring.	DFD and trust boundaries from Section 2.	Primary trust boundary is between training host and edge device.
4. Analyze threats	Identify attacker goals: model tampering, poisoning, extraction, evasion, DoS, credential abuse.	STRIDE and ATLAS mapping.	Threats differ before and after deployment.
5. Vulnerability analysis	Look for missing controls in current baseline.	No artifact signing, no formal model registry, no API auth yet, no runtime drift monitor.	Baseline is acceptable for learning but not production.
6. Attack analysis	Simulate viable attack paths using attack trees.	Replace ONNX during transfer; poison data; abuse inference API; compromise SSH.	Attack analysis defines what to test next.
7. Risk and impact analysis	Prioritize controls based on likelihood, impact, and feasibility.	Risk register in Section 8.	Immediate priority: artifact integrity, SSH hardening, and deployment auditability.

6. MITRE ATLAS Mapping for the Trained Model and Deployment Artifacts

Use of ATLAS. MITRE ATLAS is used as the AI/ML-specific adversary behavior reference. MITRE describes ATLAS as a globally accessible living knowledge base of tactics and techniques against AI-enabled systems [R1]. The official atlas-data repository contains the tactics, techniques, mitigations, and case-study data used by the ATLAS website [R2]. Because ATLAS is a living framework, exact technique IDs should be verified against the live matrix when converting this document into a formal compliance or red-team plan.

Model lifecycle element	Representative ATLAS mapping	Attack scenario	Potential impact	Recommended controls
Training data	Poison Training Data (AML.T0020); Publish Poisoned Datasets (AML.T0019)	Attacker changes CIFAR-10-like samples or labels before training.	Lower accuracy, targeted misclassification, hidden trigger behavior.	Dataset hash, trusted source, data validation, label audits, holdout tests.
Training code/environment	ML Supply Chain Compromise; Develop Capabilities	Malicious package or modified train script produces compromised checkpoint.	Backdoored checkpoint with normal-looking accuracy.	Pinned dependencies, isolated environment, code review, SBOM, reproducible builds.
PyTorch checkpoint .pth	Backdoor ML Model (AML.T0018); Inject Payload style artifact tampering	Checkpoint is replaced or altered before ONNX export.	Attacker controls model behavior while preserving artifact name.	Checkpoint signing, access control, hash manifest, immutable storage.
ONNX artifact .onnx	Model artifact tampering; ML supply-chain compromise	ONNX graph modified during export or SCP transfer.	TensorRT compiles a malicious or degraded model.	ONNX graph validation, operator allowlist, signature before transfer and before build.
TensorRT engine .engine	Persistence / defense evasion via optimized binary artifact	Attacker replaces engine after build; engine differs from approved ONNX.	Persistent compromised runtime artifact on Jetson.	Engine hash, read-only deployment, build provenance, rollback protection.
Inference API	Exfiltration via ML Inference API (AML.T0024); Infer Training Data Membership (AML.T0024.000); Extract ML Model (sub-technique under inference API exfiltration)	High-volume queries used to learn decision boundaries or extract model behavior.	Model IP theft, privacy leakage, unplanned API cost.	Rate limiting, output minimization, confidence clipping, query anomaly detection.
Input image stream	Craft Adversarial Data (AML.T0043); Evasion-style attacks	Perturbed image causes wrong prediction while appearing normal.	Safety and reliability failure in downstream task.	Adversarial robustness tests, input distribution monitoring, confidence thresholds.
Telemetry/logging	Defense evasion; collection/exfiltration support	Attacker disables logs or removes evidence of model replacement.	Undetected attack and weak forensics.	Append-only logs, remote log shipping, integrity-protected telemetry.

6.1 Personal ATLAS reference for this edge-AI baseline

Interpretation. The table below is a personal ATLAS-oriented reference for the user's workflow. It converts ATLAS-style tactics into concrete questions and telemetry for the RTX-to-Jetson project.

ATLAS-oriented tactic	Question for this project	Relevant assets/signals	Actionable control
Reconnaissance / Discovery	What can an attacker learn about model type, runtime, TensorRT version, device, paths, and exposed APIs?	System banners, model names, public posts, API metadata, logs.	Remove unnecessary version exposure; use generic API errors; restrict directory listing.
Resource Development	Could an attacker prepare poisoned datasets, adversarial images, or proxy CNNs before touching the device?	Public CIFAR-10 pipeline, training script, known architecture.	Adversarial test suite; model watermarking; training data validation.
Initial Access	How can the attacker first reach RTX, Jetson, Git repo, model folder, or API?	SSH, VS Code Remote-SSH, scp, future REST endpoint.	SSH hardening, firewall rules, keys, least privilege, separate deploy user.
ML Model Access	Can the attacker query, download, copy, or modify the model or engine?	.pth, .onnx, .engine, /predict endpoint.	Artifact access control; authenticated API; no public model files.
Execution	Can the attacker cause malicious code or model behavior to execute on Jetson?	Malformed model artifact, unsafe dependency, vulnerable parser.	Operator allowlist, TensorRT/JetPack patching, container isolation.
Persistence	Can the attacker survive reboot or redeploy by replacing the engine or startup service?	cifar10_cnn_fp16.engine, systemd service, deployment folder.	Read-only engine path, signed engine manifest, deployment attestation.
Defense Evasion	Can the attacker hide a model replacement or unusual latency pattern?	Missing logs, overwritten artifacts, dynamic clocks.	Remote log shipping, checksums at startup, baseline latency profiles.
Credential Access	Can SSH keys, tokens, or deployment secrets be stolen?	MacBook, RTX user profile, Jetson user home.	Credential vault, passphrases, key rotation, no secrets in scripts.
Collection / Exfiltration	What can be extracted through queries, files, or logs?	Model weights, confidence scores, telemetry, directory contents.	Output minimization, monitoring, file permissions, egress restrictions.
Impact	What is the business/research impact if predictions or runtime are manipulated?	Wrong classification, latency DoS, benchmark falsification.	Safety checks, fallback mode, version rollback, incident response playbook.

7. Attack Scenarios and Attack Tree

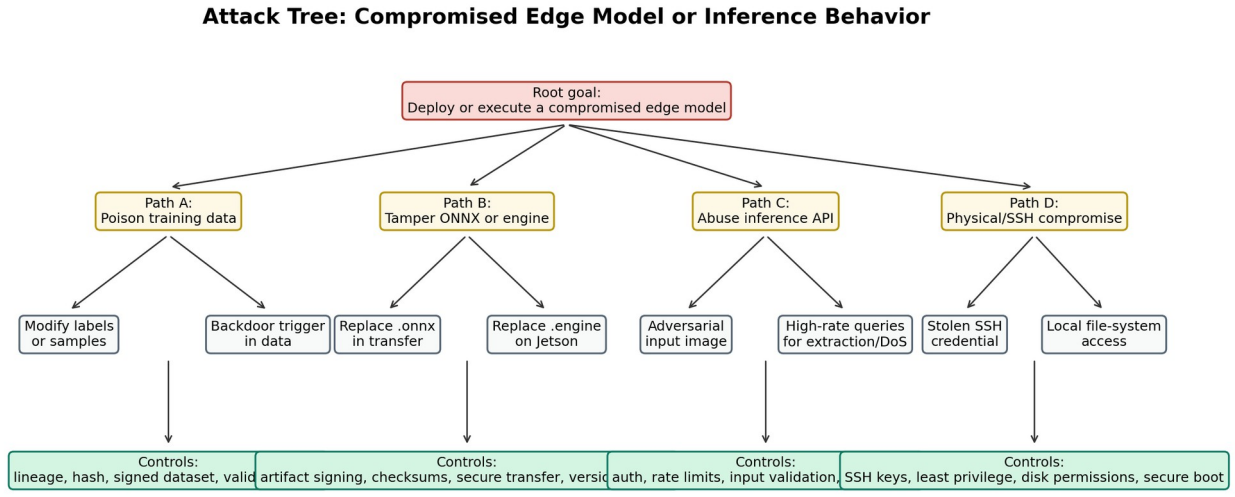

Figure 4. Attack tree for compromising the deployed edge model or inference behavior.

ID	Scenario	Attack path	Impact	Priority	Primary response
AS-1	Model artifact replacement during	Attacker gains access to RTX account or	TensorRT compiles compromised ONNX;	High	Check hash on RTX before transfer and on

	transfer	network path and replaces cifar10_cnn_cuda.onnx before or after SCP.	behavior differs from approved model.		Jetson after transfer; sign ONNX; build only from approved artifact folder.
AS-2	Training data poisoning	Attacker modifies samples or labels before training or introduces a poisoned dataset variant.	Model learns hidden trigger or targeted class bias while test accuracy may remain acceptable.	High	Dataset source pinning, checksums, label audit, data augmentation review, separate poisoned-data tests.
AS-3	Query-based model extraction	Attacker repeatedly calls future /predict endpoint and uses outputs to train a substitute model.	Loss of model IP and stronger adversarial attack preparation.	Medium	Authentication, rate limiting, confidence rounding, per-client anomaly detection.
AS-4	Adversarial image evasion	Attacker crafts an input image that causes wrong prediction under TensorRT FP16 runtime.	Reliability failure, possible downstream security decision error.	Medium	Adversarial robustness testing, input normalization checks, confidence threshold, human review for low confidence.
AS-5	Denial of service on Jetson inference	Large request volume or malformed images consume CPU/GPU resources and trigger thermal throttling.	Loss of availability, degraded latency, missed real-time deadlines.	Medium	Rate limits, request body limits, queue backpressure, GPU/thermal monitoring.
AS-6	Side-channel-aware observation	Attacker with local or physical access observes timing/power/thermal behavior for model or input inference.	Possible leakage of model behavior or sensitive input characteristics.	Research risk	Baseline timing/power traces, randomization/batching where acceptable, physical security, telemetry correlation.

8. Risk Register and Prioritized Controls

Risk ID	Risk	Likelihood	Impact	Rating	Rationale	Treatment
R1	Unsigned model artifacts	High	High	Critical	Anyone with filesystem/transfer access could replace ONNX or TensorRT engine.	Implement artifact signing and SHA-256 manifest; verify before TensorRT build and before service start.
R2	Weak deployment audit trail	Medium	High	High	Difficult to prove who deployed which model and when.	Append-only deployment logs, artifact manifest, Git commit hash, timestamped release note.
R3	Future inference API extraction	Medium	Medium	High	Repeated queries can expose decision boundaries or model behavior.	API auth, rate limits, output minimization, anomaly detection.
R4	Training data poisoning	Medium	High	High	Poisoned data can compromise model while accuracy remains plausible.	Data lineage, checksums, label audit, robustness tests.
R5	Jetson filesystem tampering	Medium	High	High	Local access can replace .engine file or service config.	Least privilege, read-only model directory, file integrity monitoring.
R6	DoS/thermal throttling	Medium	Medium	Medium	High request volume or bad inputs degrade latency.	Rate limit, request size limit, GPU/thermal monitoring, backpressure.
R7	Side-channel leakage	Low to Medium	Medium	Medium	Physical/local observation of timing/power may reveal input/model behavior.	Profiling baseline, access control, randomized scheduling if practical.
R8	Dependency compromise	Low to Medium	High	High	Malicious dependency can alter export or inference path.	Pinned package versions, SBOM, dependency scanning, offline cache for releases.

8.1 Control implementation roadmap

Phase	Action	Security value

Immediate (next 1-2 sessions)	Create artifact manifest with SHA-256 hashes for .pth, .onnx, .engine; verify hashes before and after SCP; document exact commands and paths.	Reduces accidental and malicious artifact confusion.
Short term	Use SSH keys instead of password authentication; restrict Jetson deployment folder permissions; create a non-root inference service user.	Reduces credential and privilege risk.
Short term	Build a simple FastAPI/Flask /predict microservice with input validation, request size limit, structured logs, and rate limiting.	Creates realistic target for STRIDE and ATLAS testing.
Medium term	Add model signing with GPG/Sigstore-style workflow; block service start if signature/hash check fails.	Prevents unapproved model or engine execution.
Medium term	Implement telemetry baseline: latency distribution, confidence distribution, error rate, GPU temperature and power/clock state.	Supports anomaly detection and DoS/side-channel experiments.
Research extension	Add adversarial image tests, poisoning tests, FP32 vs FP16 comparison, quantization effects, and power/thermal profiling.	Turns the baseline into a publishable security-aware edge AI evaluation workflow.

9. Experimental Evidence and Security-Relevant Interpretation

The following plots are included as evidence from the completed baseline. They are not claims of state-of-the-art model performance. They establish reference accuracy, loss, latency, and latency decomposition values that future security tests can compare against.

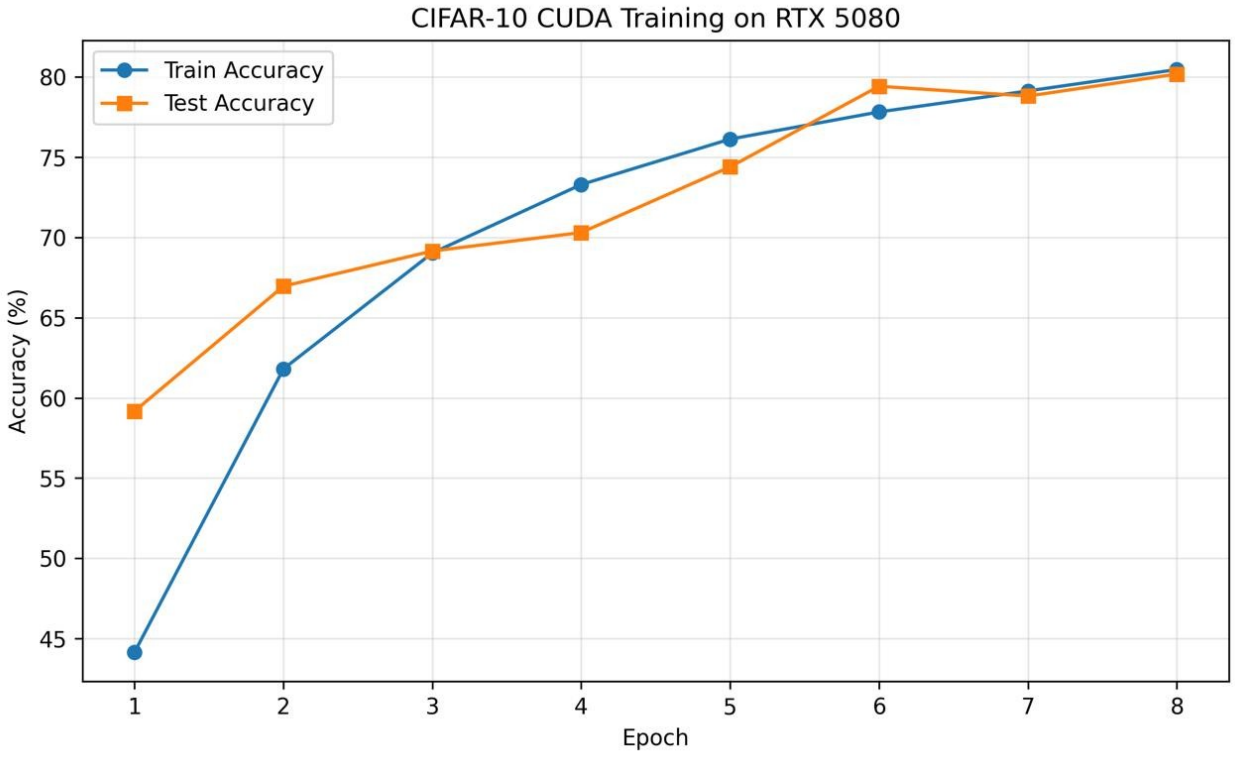

Figure 5. Training/test accuracy over 8 epochs on RTX 5080. Security use: clean baseline for poisoning/evasion experiments.

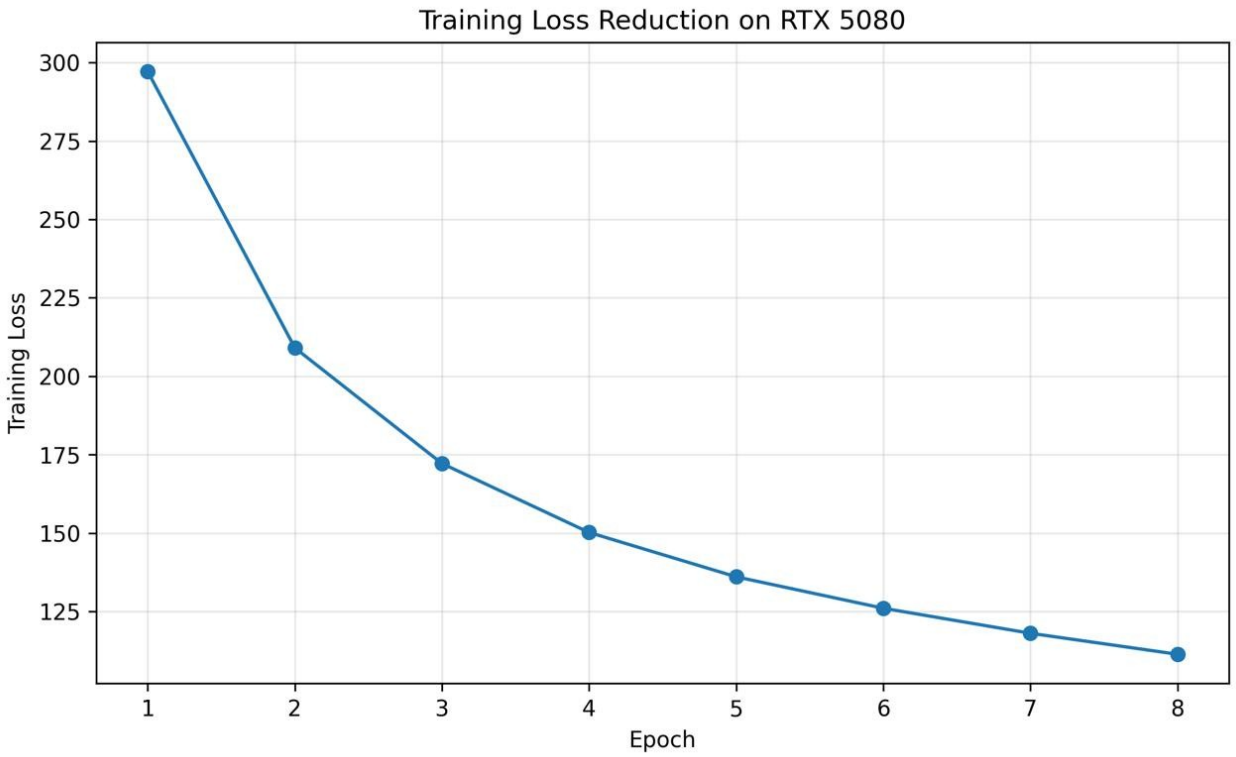

Figure 6. Training loss reduction on RTX 5080. Security use: detects abnormal learning behavior in poisoned or tampered training runs.

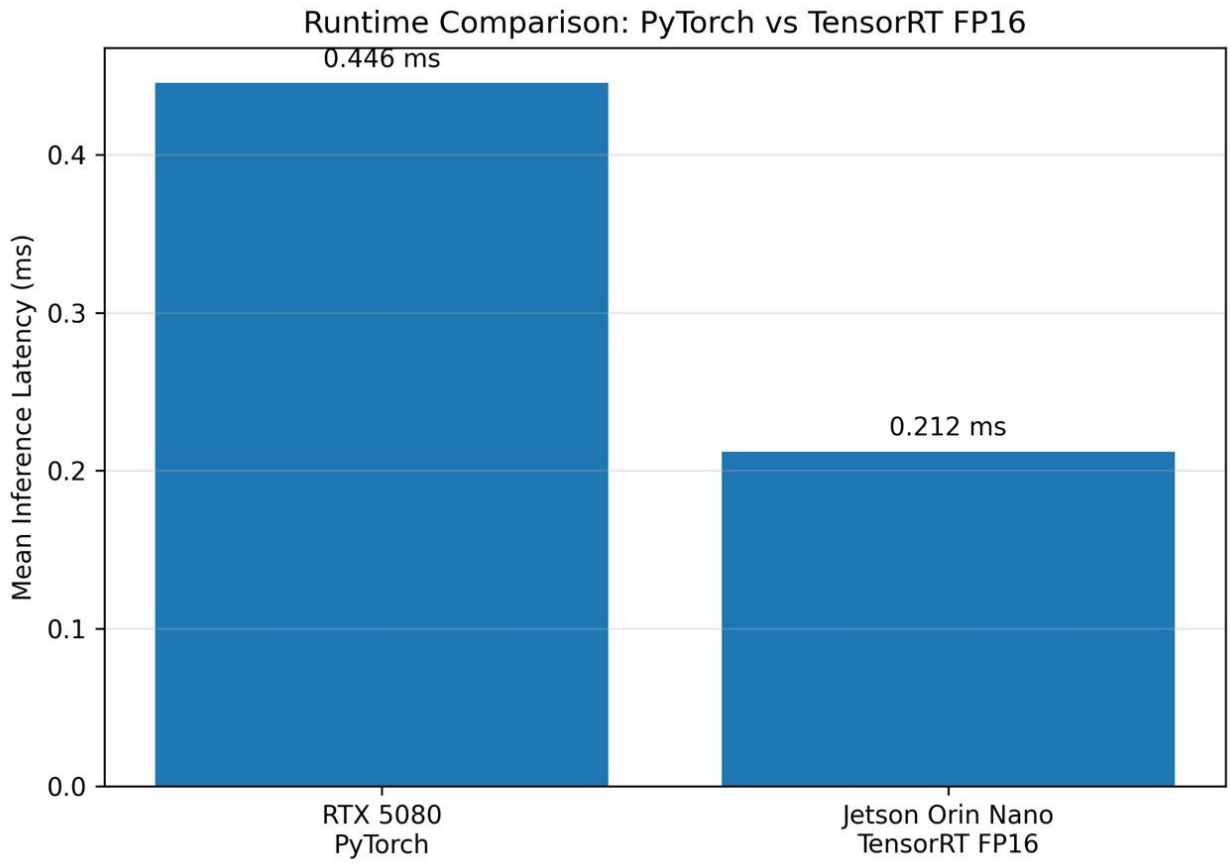

Figure 7. Runtime comparison between PyTorch GPU inference and TensorRT FP16 edge runtime. Interpretation must be careful: this compares framework/runtime paths, not raw device capability.

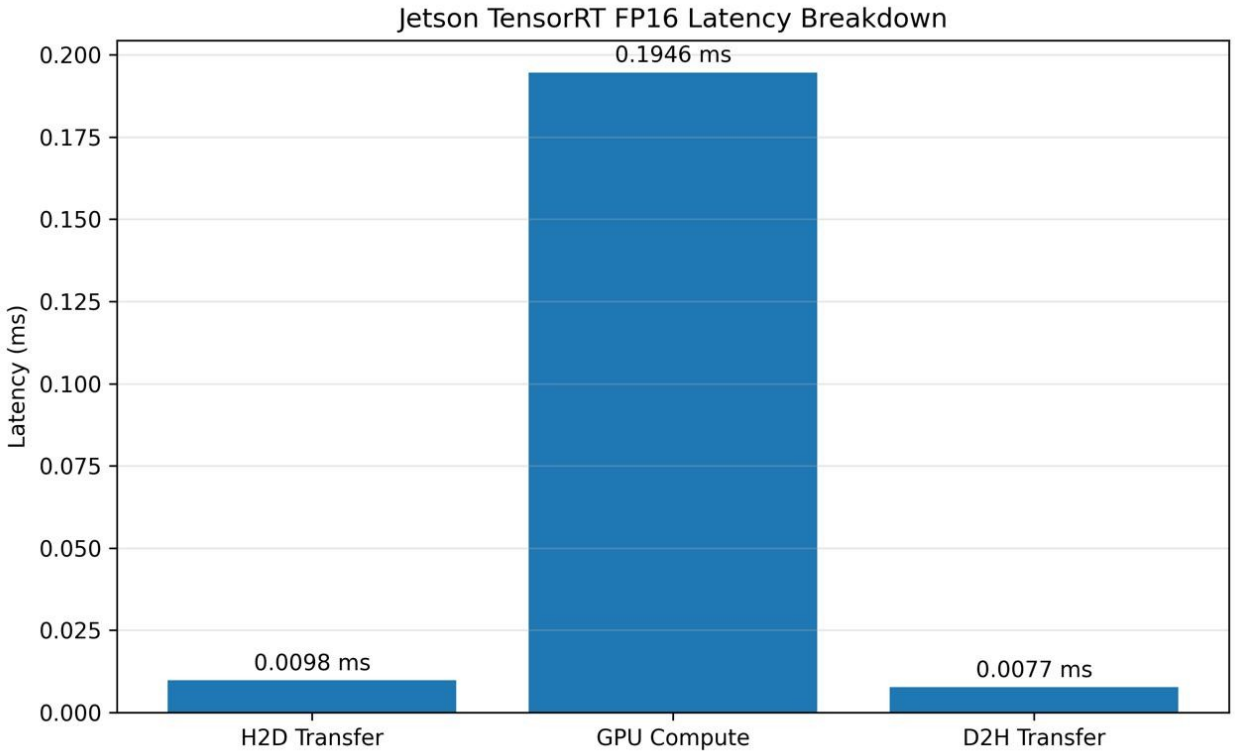

Figure 8. Jetson TensorRT FP16 latency breakdown. Security use: baseline for DoS, anomaly detection, and side-channel-aware profiling.

10. Secure Edge-AI Baseline Checklist

Control domain	Checklist item	Priority/notes
Artifact integrity	Hash .pth, .onnx, and .engine after creation; verify hashes before deployment.	Required before production-like testing.
Artifact authenticity	Sign model artifacts and reject unsigned artifacts in deployment scripts.	Critical for supply-chain control.
Provenance	Record Git commit, Python package versions, PyTorch version, CUDA version, TensorRT version, device ID.	Required for reproducibility and incident response.
Access control	Use SSH keys; avoid shared accounts; restrict model directory write permissions.	Prevents simple artifact replacement.
Runtime isolation	Run inference service as non-root in a container or restricted service account.	Limits blast radius of API vulnerabilities.
Input hardening	Validate file type, dimensions, size, normalization path, and batch limits.	Prevents malformed input and DoS.
Output governance	Return only necessary class/score information; avoid excessive confidence detail in public APIs.	Reduces model extraction signal.
Monitoring	Log latency, confidence distribution, error rates, model hash, engine hash, and device telemetry.	Supports anomaly detection and forensics.
Robustness testing	Add adversarial examples, corrupted images, data drift, and FP32/FP16 divergence tests.	Connects ML behavior to security assurance.
Incident response	Define rollback, revoke keys, rebuild engine from trusted ONNX, and compare to baseline metrics.	Turns model security into an operational process.

Appendix A: Commands from the Completed Baseline

These commands are included for traceability and to connect the threat model to the actual tested workflow.

RTX training:
& "C:\Users\brojo\miniconda3\envs\ml\python.exe" train_cifar10_cuda.py

ONNX export:
& "C:\Users\brojo\miniconda3\envs\ml\python.exe" export_onnx_cifar10.py

RTX inference test:
& "C:\Users\brojo\miniconda3\envs\ml\python.exe" infer_cifar10_cuda.py

Transfer ONNX to Jetson:
scp cifar10_cnn_cuda.onnx brojonano@100.74.213.67:/home/brojonano/Documents/ai_lab

Jetson TensorRT FP16 build and benchmark:
cd ~/Documents/ai_lab
/usr/src/tensorrt/bin/trtexec --onnx=cifar10_cnn_cuda.onnx --saveEngine=cifar10_cnn_fp16.engine --fp16
/usr/src/tensorrt/bin/trtexec --loadEngine=cifar10_cnn_fp16.engine

References

ID	Source	URL
R1	MITRE ATLAS. Official site: a living knowledge base of adversary tactics and techniques against AI-enabled systems.	https://atlas.mitre.org/
R2	MITRE ATLAS Data GitHub repository. Contains tactics, techniques, mitigations, case studies, and data used by the ATLAS website.	https://github.com/mitre-atlas/atlas-data
R3	Microsoft Learn. Threat Modeling Tool threats and STRIDE model categories.	https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats
R4	OWASP. Threat Modeling Process.	https://owasp.org/www-community/Threat_Modeling_Process
R5	OWASP Cheat Sheet Series. Threat Modeling Cheat Sheet.	https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html
R6	PASTA threat modeling overview and seven-stage process reference.	https://threat-modeling.com/pasta-threat-modeling/
R7	MITRE ATLAS SAFE-AI report: defensive framework for securing AI-enabled systems.	https://atlas.mitre.org/pdf-files/SAFEAI_Full_Report.pdf