TECHNICAL MANUAL

SL 5M

TM-50M — ADVANCED MACHINE LEARNING ENGINEERING

Specialist Course Manual

HEADQUARTERS
UNITED STATES ARMY EUROPE AND AFRICA
(USAREUR-AF)
Wiesbaden, Germany

26 MARCH 2026

DRAFT — UNOFFICIAL — NOT FOR OPERATIONAL USE

TM-50M — ADVANCED MACHINE LEARNING ENGINEERING

Forward: SL 5M qualifies senior machine learning engineers to design, build, and lead production ML platform capability on MSS. This manual extends SL 4M into automated MLOps, advanced neural architectures, federated learning, real-time inference, ML security, and platform architecture leadership.
Prereqs: SL 4M, Machine Learning Engineer (required — Go evaluation on file). SL 4H (AI Engineer) recommended. Data Literacy Technical Reference (recommended); CONCEPTS_GUIDE_TM50M_ML_ENGINEER_ADVANCED (read before this manual). *HQ USAREUR-AF · v1.0 · 2026 · DISTRIB: USG only · AUTH: C2DAO/UDRA v1.1*

WARNING: AUTOMATED MODEL DEPLOYMENT IS A PRIVILEGED CAPABILITY. Automated pipelines that promote model versions to production without human review are classified as high-risk automation under UDRA v1.1. Implementing this pattern without a documented, C2DAO-approved automation design and explicit mission owner sign-off constitutes an unauthorized production change. The senior MLE who designs the pipeline is accountable for every automated promotion it executes. WARNING: CROSS-DOMAIN FEDERATED LEARNING. Federated learning patterns that coordinate gradient updates across classification-separated environments may be subject to EUCOM data sharing agreements, coalition partner protocols, and USAREUR-AF C2DAO architecture review. Do not initiate cross-domain federated training without written authorization from the data steward and C2DAO. "No data moves" is not a sufficient authorization argument — the aggregated model weights can encode information from the source domains. CAUTION: MODEL COMPRESSION AND PRODUCTION EQUIVALENCE. A quantized or pruned model is a different model from the full-precision original. Performance metrics from the full-precision model do not transfer. Every compressed model variant must be independently evaluated on an operationally representative holdout set before deployment. Failure to re-evaluate may result in silent performance degradation in production. NOTE: SL 5M targets senior MLEs with production responsibility — personnel designing platform-level infrastructure, leading ML capability for a major MSS program, or serving as the ML technical authority for a command. This is not a follow-on to SL 4M for general practitioners. It is a specialization track for personnel who will own the platform, not just operate it.

CHAPTER 1 — INTRODUCTION AND SCOPE

1-1. Advanced ML Engineer Manual

BLUF: SL 5M qualifies senior machine learning engineers to design, build, and lead production ML platform capability on MSS. This manual extends SL 4M into automated MLOps, advanced neural architectures, federated learning, real-time inference, ML security, and platform architecture leadership.

SL 4M produced MLEs who can build and deploy models. SL 5M produces ML platform engineers — practitioners who design the infrastructure others build on. The distinction matters. A SL 4M MLE asks, "How do I deploy this model?" A SL 5M MLE asks, "What deployment infrastructure should this team use, and what are the failure modes?" SL 5M personnel operate at the level of ML system design: retraining architectures, shared feature stores, model registry standards, inference optimization, and adversarial robustness programs.

USAREUR-AF requires this level of capability because MSS is not a collection of individual models — it is an ML platform supporting dozens of use cases across III Corps, V Corps, 21st TSC, 10th AAMDC, 56th MDC-E, SETAF-AF, USAREUR-AF G2, and subordinate commands. As that platform scales, it requires practitioners who can design it, secure it, and lead it as a discipline.

1-2. Scope

SL 5M covers:

Topic	Description
Automated MLOps	Full retraining pipelines, A/B deployment, canary releases, production gates
Advanced neural architectures	Transformers for operational text, graph neural networks for C2 network analysis
Federated learning	Cross-domain coordination, coalition scenarios, privacy-preserving gradient aggregation
Advanced interpretability	SHAP at scale, LIME, global vs. local explanations, DoD AI ethics compliance auditing
Bias auditing	Fairness metrics, disaggregated evaluation, bias remediation for personnel models
Real-time inference	Latency optimization, request batching, model compression, quantization, benchmarking
ML security	Adversarial robustness, data poisoning detection, model extraction defense, supply chain

Topic	Description
Platform architecture	Feature stores, model registries, experiment tracking, shared ML infrastructure design
Cross-domain ML	AIP Logic integration (SL 4H interface), Agent Studio model consumption patterns
ML leadership	Code review standards, reproducibility requirements, production readiness gates, team capability development
Model documentation	DoD RAIMTF model cards, AI accountability documentation, Army policy compliance

SL 5M does NOT cover SL 4M prerequisite material (foundational MLE skills are assumed mastered); basic Foundry navigation, Transform authoring, or Ontology consumption — see SL 3; AIP Logic application design or Agent Studio development — see SL 4H; statistical optimization and operations research — see SL 4G / SL 5G; program management of ML programs of record — see SL 4J; or Foundry platform administration (enrollment, permissions, environment configuration) — contact C2DAO.

1-3. The SL 5M MLE in the USAREUR-AF Data Chain

```
USAREUR-AF COMMANDER / C2DAO
(Policy, governance, architecture authority)
       |
       v
  SL 5M SENIOR MLE                    <- You are here
  (Platform design, automation, security, leadership)
       |
       v
  SL 4M MLE
  (Model build, train, evaluate, deploy)
       |
       v
  DEPLOYED MODEL / FEATURE PIPELINE
  (Production Ontology, inference endpoints, monitoring)
       |
       v
  SL 4H (AI Eng) / SL 3 (Workshop)
  (Downstream consumers: AIP Logic, dashboards, apps)
       |
       v
  SL 1 USER
  (Operational decision supported)
```

The SL 5M MLE is a force multiplier — not because they write the most code, but because the infrastructure they design determines what the SL 4M practitioners beneath them can build and how safely they can build it. A poorly designed feature store creates incorrect features for every model that uses it. A misconfigured model registry loses experiment reproducibility across the platform. A missing adversarial robustness gate exposes every deployed model to the same attack class.

1-3A. Relationship to Other SL 5 Publications

Publication	Track	Key Overlap with SL 5M
SL 5G	ORSA Advanced	Statistical validation methods; uncertainty quantification for ML outputs
SL 5H	AI Engineer Advanced	AIP Logic integration; fine-tuning infrastructure (Chapter 3); adversarial robustness
SL 5M	ML Engineer Advanced	THIS DOCUMENT
SL 5J	Program Manager Advanced	ML program lifecycle; governance documentation requirements
SL 5K	Knowledge Manager Advanced	Feature data governance; training corpus design
SL 5L	Software Engineer Advanced	Platform SDK infrastructure; OSDK model-serving integrations
SL 5N	UI/UX Designer Advanced	Feature engineering UI; model monitoring dashboards
SL 5O	Platform Engineer Advanced	ML platform infrastructure; training pipeline orchestration

WFF Operational Consumer Note. The six Warfighting Function (WFF) tracks — Intelligence (SL 4A), Fires (SL 4B), Movement and Maneuver (SL 4C), Sustainment (SL 4D), Protection (SL 4E), and Mission Command (SL 4F) — are the primary operational consumers of ML models built and maintained by SL 5M engineers. Readiness prediction, logistics demand forecasting, personnel risk models, and anomaly detection all feed WFF staff sections. When designing ML systems, account for the WFF end-user: calibration, interpretability, and latency requirements vary significantly by WFF function and decision context.

1-4. Prerequisites and Entry Standards

SL 5M entry requires demonstrated proficiency at SL 4M level. The following are entry conditions, not goals.

Prerequisite	Verification Standard
SL 4M completion	Certification on file, confirmed by supervisor
Production model deployment	Has deployed at least one model to MSS Ontology in a production use case

Prerequisite	Verification Standard
Drift monitoring implementation	Has implemented and operated a drift detection pipeline on a production model
Model governance completion	Has independently completed SL 4M Appendix A governance checklist for a production model
Python proficiency (advanced)	Can write multi-class sklearn pipelines, custom PyTorch training loops, and Foundry Transform pipelines without reference
MLOps tooling	Familiar with experiment tracking, model versioning, and pipeline orchestration concepts
Git / code review	Has conducted peer code reviews; understands branching strategy for production ML repositories

Personnel who do not meet these entry standards must complete SL 4M production experience requirements first. There is no accelerated path.

1-5. Governing References

Document	Relevance
DoD Responsible AI Implementation Toolkit and Framework (RAIMTF)	Model documentation, risk assessment, accountability chain for AI systems
DoD AI Ethics Principles (2020)	Responsible, equitable, traceable, reliable, governable — required properties of all DoD AI
Army AI/ML Governance Framework (current)	Army-level model lifecycle, oversight, and incident reporting requirements
USAREUR-AF C2DAO Architecture Guidance	Theater ML architecture standards, automation approval tiers, production deployment authority
NATO Architecture Framework v4 (NAFv4)	Coalition architecture standards for federated or MPE-accessible ML outputs
EUCOM Data Sharing Agreements	Applicable for any model consuming partner-nation data or participating in coalition federated learning
learn-data.armydev.com	Authoritative reference implementations, approved patterns, and policy updates
Army DIR 2024-03	Digital Engineering Policy — Army-wide digital engineering adoption directive
FM 3-12	Cyberspace Operations and Electromagnetic Warfare — operational context for ML systems

Document	Relevance
DA PAM 25-2-5	Software Assurance — software security standards

1-5a. Strategic Guidance

The following are strategic guidance documents — not doctrine — that inform MSS training design and operational context.

Document	Authority	Relevance
Army CIO Data Stewardship Policy (April 2, 2024)	Army CIO	Foundational data governance; ML models are data products; stewardship accountability
UDRA v1.1 (February 2025)	Army Enterprise	Domain ownership, federated governance, ML automation authorization tiers

1-6. Advanced Governance Tiers

SL 4M covered the baseline six-gate governance process. SL 5M introduces automation authorization tiers that apply to the more powerful capabilities in this manual.

Automation Tier	Description	Additional Approval Required
Tier 0 (Manual)	All model promotions require human review and explicit approval	None (SL 4M standard)
Tier 1 (Monitored Automation)	Automated retraining; human approval gate before any production promotion	C2DAO design review of automation architecture
Tier 2 (Gated Automation)	Automated retraining; automated promotion if all production readiness gates pass	C2DAO + mission owner sign-off; automatic rollback required
Tier 3 (Full Automation)	Automated retraining and promotion with no human approval step	Not authorized on MSS without USAREUR-AF CIO exception

NOTE

Most MSS ML automation should target Tier 1 or Tier 2. Tier 3 is not authorized for any production MSS model without explicit written exception from the USAREUR-AF CIO. Document the tier in every pipeline design document.

CHAPTER 2 — ADVANCED MLOPS AND AUTOMATED RETRAINING

2-1. Overview

BLUF: Advanced MLOps on MSS means automating the full model lifecycle — data ingestion through production deployment — with explicit governance gates, rollback capability, and comprehensive monitoring. This chapter covers automated retraining pipeline design, A/B model deployment, canary releases, and production readiness gates.

SL 4M introduced drift detection and manual retraining. SL 5M builds the infrastructure that makes retraining automated, auditable, and safe. The distinction is architectural: where SL 4M MLEs respond to drift, SL 5M MLEs design systems that detect drift, validate a new model, and promote it with defined human oversight requirements — consistently, every time, across every model on the platform.

2-2. Automated Retraining Pipeline Architecture

NOTE — Palantir Developers reference: *Deep Dive: Optimizing Data Pipelines with Iceberg Tables and Lightweight Compute* | *DevCon 4* — The primary DevCon deep dive on Iceberg table optimization for production ML pipelines, covering time-travel, snapshot isolation, efficient incremental reads, and lightweight compute patterns. These capabilities underpin the data validation and feature pipeline stages of the five-stage retraining architecture described below. Available on the Palantir Developers YouTube channel (@PalantirDevelopers).

An automated retraining pipeline on MSS consists of five stages. Each stage must be implemented as a discrete, independently monitorable unit. Do not collapse stages — a monolithic retraining job is not debuggable when it fails in production.

The Five-Stage MSS Retraining Pipeline:

```
Stage 1: TRIGGER
  - Drift detection threshold exceeded
  - Scheduled cadence (weekly/monthly)
  - Data volume trigger (N new records since last training)
  - Manual trigger (MLE or data steward)

Stage 2: DATA VALIDATION
  - Schema validation against expected feature schema
  - Distribution check: new training data vs. previous training data
  - Data quality gates: null rate, outlier rate, referential integrity
  - FAIL FAST: if data quality gate fails, halt and alert — do not train on bad data

Stage 3: FEATURE PIPELINE
  - Execute registered feature pipeline (version-pinned)
  - Validate feature output schema and distribution
  - Log feature statistics to experiment tracker
```

```
Stage 4: TRAINING AND EVALUATION
  - Train candidate model with registered hyperparameters or HPO search
  - Evaluate against holdout set using registered evaluation suite
  - Compare candidate vs. current production model on all registered metrics
  - Bias/fairness evaluation (required for personnel models)
  - PROMOTION GATE: candidate must exceed production model on primary metric
  - ALERT: if candidate does not improve, alert MLE — do not silently retain incumbent

Stage 5: DEPLOYMENT
  - Package model artifact with metadata (training timestamp, data version, metric
snapshot)
  - Tier 1: Halt for human approval
  - Tier 2: Auto-promote if all gates pass; notify MLE and data steward
  - Rollback register: record previous model version for immediate rollback
  - Post-deployment: activate monitoring on new model version
```

2-3. Task: Design and Implement an Automated Retraining Pipeline

CONDITIONS: You are the senior MLE responsible for a production model on MSS. The model has been in production for at least one retraining cycle. Drift monitoring is operational. You have C2DAO approval for your target automation tier.

STANDARDS: A compliant automated retraining pipeline runs end-to-end without manual intervention, produces a complete audit trail for every run, enforces all five stages with independent failure modes, and either halts for human approval (Tier 1) or auto-promotes with rollback capability (Tier 2) as authorized.

EQUIPMENT: Foundry Code Workspace, Foundry Transforms (pipeline orchestration), Foundry Checks (data quality gates), MSS model registry (versioned Foundry dataset + governance tracking system — not a native Foundry platform product; this is MSS-built infrastructure), experiment tracker, alerting integration.

PROCEDURE:

1. **Define the trigger conditions.** Document in the pipeline design document: What triggers a retraining run? Specify at minimum: (a) drift metric threshold, (b) scheduled cadence, (c) data volume threshold. At least two triggers are required — relying on a single trigger creates a single point of failure in the monitoring system.
2. **Implement the data validation stage as a separate Foundry Transform.** Write a dedicated validation Transform that reads the candidate training dataset and executes the following checks:
3. Schema check: all required feature columns present, correct dtype
4. Null rate check: null rate per column does not exceed registered threshold
5. Distribution check: KL divergence or PSI (Population Stability Index) vs. training distribution reference
6. Volume check: minimum record count for a valid training run Write each check as a Foundry Check (`@check` decorator). Tag failures as `Severity.ERROR` — a data validation error must halt the pipeline. Do not proceed to training on a dataset that fails data validation.

7. **Version-pin the feature pipeline.** The automated retraining pipeline must execute a specific, registered version of the feature pipeline — not the current-HEAD version. This prevents an unreviewed feature pipeline change from silently altering a production model's input distribution during an automated run. Record the feature pipeline version in the experiment run metadata.
8. **Implement the promotion gate as an explicit comparison step.** The candidate model does not auto-promote simply because training succeeded. The promotion gate compares the candidate to the current production model on a held-out, representative evaluation set. Define: (a) the primary metric, (b) the minimum improvement threshold required for promotion (e.g., +0.5% F1), (c) the secondary metrics that must not degrade. Log all metric comparisons to the experiment tracker.
9. **Implement human approval integration (Tier 1 / Tier 2).** For Tier 1, the pipeline halts after the promotion gate and sends an alert to the responsible MLE and data steward with: candidate metric summary, comparison vs. production, training data version, and a direct link to the experiment run. For Tier 2, the pipeline auto-promotes if all gates pass, sends the same notification as confirmation, and registers the previous model version in the rollback register.
10. **Implement rollback.** Every production deployment must register the previous model version before overwriting the production pointer. Rollback must be executable in under 15 minutes by the on-call MLE. Test rollback on every major pipeline version. Document the rollback procedure in the pipeline runbook.
11. **Implement post-deployment monitoring activation.** After promotion, the pipeline must activate drift monitoring on the new model version. Do not rely on pre-existing monitoring from the previous version — the new model may have different feature importances and a different decision boundary. Re-baseline all drift monitors against the new model's training distribution.
12. **Write and register the pipeline runbook.** Document: trigger conditions, expected run time, data dependencies, escalation path on failure, rollback procedure, and contact information for the responsible MLE and data steward. Register the runbook in the MSS documentation repository. The runbook is a required artifact — a pipeline without a runbook is not production-ready.

CAUTION: DO NOT PIN HYPERPARAMETERS WITHOUT DOCUMENTED JUSTIFICATION. An automated pipeline that retrains with fixed hyperparameters assumes those hyperparameters remain optimal as the data distribution shifts. Either perform a limited hyperparameter search on each retraining run (document the search space and budget) or schedule a periodic full HPO run to validate that pinned values remain appropriate. Document your choice and its rationale in the pipeline design document.

NOTE

Population Stability Index (PSI) is the preferred distribution shift metric for tabular features on MSS. PSI < 0.1 indicates stable; 0.1–0.25 indicates moderate shift warranting investigation; > 0.25 indicates significant shift requiring data steward review before proceeding. These thresholds are defaults — calibrate to your specific operational use case.

2-4. A/B Model Deployment

A/B deployment routes a defined fraction of inference traffic to a candidate model version while the incumbent model serves the remainder. On MSS, A/B deployment is the standard mechanism for validating a new model version against live operational data before full promotion.

A/B deployment on MSS requires:

Requirement	Detail
Traffic split definition	What percentage of requests routes to candidate? Start at 5–10% for high-stakes models.
Assignment consistency	A given input (e.g., a unit's readiness record) must consistently route to the same model version for the duration of the experiment. Use hash-based assignment on a stable identifier.
Metric tracking	Both model versions log predictions and outcomes to the same evaluation dataset. Automated comparison runs on a defined schedule.
Duration specification	Define minimum experiment duration before a promotion decision is made. Duration must be sufficient to accumulate statistical significance on the primary metric.
Rollout authority	Define who can increase traffic split, trigger full promotion, or terminate the experiment.
Fallback behavior	If the candidate produces errors above a defined threshold, traffic automatically reverts 100% to incumbent.

A/B deployment is NOT appropriate when:

- The two model versions use different feature schemas (A/B requires identical inputs)
- The use case cannot tolerate any prediction inconsistency between versions for the same entity
- The evaluation metric requires a minimum observation period longer than the operational tempo allows

In these cases, use a staged rollout with a full validation period between stages rather than simultaneous A/B traffic.

2-5. Canary Releases

A canary release is a production deployment pattern where the new model version is promoted to a small, defined subset of the production population before full rollout. It differs from A/B deployment in intent: A/B is an experiment to determine which version is better; a canary release is a production safety mechanism to detect failures before they affect the full population.

Canary release procedure for MSS:

1. Identify a canary population: a defined subset of the full operational population (e.g., one brigade's readiness records, one logistics node's demand forecast data). The canary population must be representative but bounded — large enough to detect real failures, small enough to limit blast radius.
2. Deploy the new model version to the canary population only. The incumbent model continues to serve all non-canary records.
3. Monitor the canary population for a defined observation window (minimum 48 hours for daily-refreshed operational data; minimum two operational cycles for lower-frequency data).
4. If canary metrics remain within tolerance, expand to 25%, then 50%, then full rollout on a defined schedule.
5. If canary metrics degrade, halt rollout, notify the data steward and mission owner, and initiate root-cause analysis before any further promotion.

NOTE

Canary releases require that the inference pipeline support simultaneous routing to two model versions by population key. Implement this as a routing layer in the inference Transform — not as two separate pipelines that each read the full dataset. Two full-dataset pipelines create double load and do not enforce the population boundary correctly.

2-6. Production Readiness Gates

Every automated retraining pipeline must enforce a defined set of production readiness gates before any model version reaches production. These gates are not the same as the governance approval gates in SL 4M — they are automated technical checks that run as part of the pipeline.

Required production readiness gates (automated):

Gate	Check	Failure Action
Data quality	All data validation checks pass	Halt pipeline; alert MLE and data steward
Schema compatibility	Candidate model feature schema matches production inference schema	Halt pipeline; alert MLE
Performance threshold	Candidate exceeds incumbent on primary metric by defined margin	Halt promotion; alert MLE with comparison report
Secondary metric floor	No secondary metric degrades below defined floor	Halt promotion; alert MLE
Bias gate (personnel models)	Disaggregated metrics within defined disparity threshold	Halt promotion; alert MLE and mission owner

Gate	Check	Failure Action
Inference latency	P99 latency within defined SLA on benchmark workload	Halt promotion; alert MLE
Serialization test	Model loads from serialized artifact without error	Halt pipeline; alert MLE
Rollback register	Previous model version registered before promotion	Halt promotion if rollback register fails

Required production readiness gates (human-reviewed, Tier 1 and Tier 2):

Gate	Reviewer	Documentation
Evaluation acceptance	Mission owner	Signed evaluation acceptance form
Bias/fairness acceptance (if applicable)	Mission owner + G1 (personnel models)	Signed bias acceptance form
Deployment authorization	C2DAO	Logged in MSS governance tracker

Automated gates must pass before the human-reviewed gates are presented. Do not route a candidate to human review if it has failed automated gates — this creates approval fatigue and degrades the quality of human oversight.

CHAPTER 3 — ADVANCED NEURAL ARCHITECTURES FOR OPERATIONAL DATA

3-1. Overview

BLUF: Two neural architectures are operationally relevant at SL 5M level on MSS: transformer-based models for operational text (SITREPs, maintenance records, intelligence summaries) and graph neural networks (GNNs) for C2 network analysis. This chapter covers architecture selection, implementation, operational validation, and deployment considerations for both.

The majority of MSS ML use cases are served by well-tuned gradient boosting models and classical neural networks — the practitioner who reaches for a transformer for a tabular readiness prediction problem is choosing complexity without operational justification. SL 5M covers advanced neural architectures because there are genuine operational problems that require them, not because they are technically sophisticated.

3-2. Transformers for Operational Text

Operational text sources on MSS include:

Source	Data Type	Typical Use Case
SITREPs	Structured narrative	Classification, information extraction, anomaly detection
Maintenance work orders (GCSS-Army)	Semi-structured text	Fault code prediction, parts demand, readiness impact
Intelligence summaries (at appropriate classification)	Structured narrative	Pattern recognition, named entity extraction
Unit historical AARs	Unstructured narrative	Capability assessment, lesson extraction
Logistics exception reports	Semi-structured	Root cause classification, escalation prediction

When to use a transformer model (vs. simpler text approaches):

Use a transformer when the downstream task requires understanding of context across the full text span, not just keyword presence. Transformers are appropriate for: - Semantic similarity: "Is this SITREP describing the same class of event as these historical SITREPs?" - Multi-label classification where labels are semantically entangled - Information extraction that requires co-reference resolution

Do not use a transformer when: - A TF-IDF + logistic regression achieves the required performance on a validated benchmark - The operational text vocabulary is highly domain-specific and a pre-trained model's vocabulary alignment is poor without fine-tuning - The inference latency requirement cannot accommodate transformer inference times at your scale

NOTE — Palantir Developers reference: *xAI x TWG for Model Tuning Infrastructure | DevCon 2* — A DevCon deep dive on the infrastructure decisions required to support model fine-tuning at scale, including compute allocation, checkpoint management, and training-serving integration. Directly relevant to the fine-tuning infrastructure design decisions in this task and the model registry requirements in Chapter 8. Available on the Palantir Developers YouTube channel (@PalantirDevelopers).

3-3. Task: Fine-Tune a Transformer for SITREP Classification

CONDITIONS: You have a labeled dataset of SITREPs with operational category labels (e.g., logistics exception, personnel readiness, equipment maintenance, force protection). The dataset has at minimum 500 labeled examples per class. You have an authorized Code Workspace with GPU-backed compute allocated. Pre-trained model weights are available in the MSS-approved model repository (do not download external model weights from the internet — use the approved repository).

STANDARDS: The fine-tuned model achieves F1 ≥ 0.80 on the held-out test set per class. The model runs inference on a single SITREP in ≤ 200ms on the production compute profile. The model is registered in the MSS model registry with a complete model card.

EQUIPMENT: Foundry Code Workspace (GPU-backed), MSS-approved pre-trained transformer checkpoint, Hugging Face `transformers` library (authorized version, pinned in workspace requirements), labeled SITREP dataset (Foundry dataset).

PROCEDURE:

1. **Load and inspect the labeled dataset.** Read the SITREP dataset from Foundry. Examine label distribution. If any class has fewer than 200 examples, flag the imbalance before proceeding — class imbalance in text classification creates predictable precision/recall tradeoffs that must be documented and accepted by the mission owner before deployment.
2. **Establish a baseline with a simpler model.** Before fine-tuning a transformer, train a TF-IDF + logistic regression baseline. Record F1 per class. This baseline has two purposes: (a) it establishes whether the transformer's added complexity is justified by a measurable performance gain, and (b) it provides a production fallback if the transformer fails latency gates.
3. **Preprocess operational text.** SITREPs contain military abbreviations, DTG strings, grid coordinates, and unit designations. Apply a preprocessing step that: (a) normalizes DTG strings to a consistent format, (b) expands known military abbreviations using the approved MSS abbreviation dictionary (do not rely on the pre-trained tokenizer to handle these correctly), (c) removes PII if present (SSNs, names in non-aggregate contexts — coordinate with data steward).
4. **Load the approved pre-trained checkpoint.** Instantiate the model from the MSS-approved checkpoint path. Do not download weights from external sources. Verify the checkpoint hash against the registry entry before loading.
5. **Configure fine-tuning.** Use the following as default starting parameters — adjust based on dataset size and validation performance:
6. Learning rate: 2e-5 (smaller than typical supervised learning; transformer fine-tuning is sensitive to LR)
7. Batch size: 16 (adjust down if GPU OOM; adjust up with gradient accumulation rather than increasing LR)
8. Epochs: 3–5 (more epochs risk catastrophic forgetting of pre-trained representations)
9. Warmup steps: 10% of total training steps
10. Weight decay: 0.01
11. Max sequence length: 256 tokens for most SITREPs; increase only if truncation is causing classification errors on validation set

2. **Train with early stopping on the validation F1.** Monitor per-class F1 on the validation set at each epoch. Stop training if validation F1 does not improve for two consecutive epochs. Save the checkpoint at best validation F1 — do not use the final epoch checkpoint.
3. **Evaluate on the held-out test set.** Compute precision, recall, F1, and confusion matrix per class. Compare against the TF-IDF baseline. If the transformer improvement is < 3 percentage points F1 per class, document the trade-off — the additional inference complexity may not be operationally justified.
4. **Benchmark inference latency.** Run inference on a benchmark set of 1000 SITREPs and measure P50, P95, and P99 latency. If P99 exceeds the production SLA, implement one of the optimization techniques in Chapter 6 (quantization, batching) before proceeding to deployment.
5. **Register the model.** Register the fine-tuned checkpoint in the MSS model registry with: training dataset version, base checkpoint identifier, training configuration, evaluation metrics (test set), latency benchmark results, and a complete model card (see Appendix B).

CAUTION: PRE-TRAINED MODELS AND OPERATIONAL VOCABULARY. Pre-trained transformer checkpoints are trained on general-domain text. Military operational vocabulary — unit designations, equipment nomenclature, tactical terminology — is often out-of-vocabulary or misrepresented in pre-trained embeddings. Evaluate embedding quality on a sample of operational text before committing to a checkpoint. If the tokenizer fragments common military terms (e.g., "HMMWV" → ["H", "##MM", "##WV"]), consider domain-adaptive pre-training (continued pre-training on unlabeled operational text before task-specific fine-tuning).

NOTE

The MSS-approved transformer checkpoint library is maintained by the C2DAO ML team. New checkpoints require approval review before addition to the library. If you need a checkpoint not in the library, submit a request through the C2DAO architecture review process — do not work around the approval process by downloading external weights.

3-4. Graph Neural Networks for C2 Network Analysis

Command and control (C2) networks, logistics dependency graphs, and unit relationship hierarchies are graph-structured data. Classical ML models that treat each unit or node as an independent record cannot capture dependency effects: a readiness problem at an OPCON unit propagates through the command relationship graph in ways that per-row tabular models cannot detect.

Graph Neural Networks (GNNs) operate on graph-structured data, passing messages between connected nodes to produce node-level, edge-level, or graph-level predictions. On MSS, the most operationally relevant GNN applications are:

Application	Graph Structure	Prediction Task
C2 readiness propagation	Nodes: units; Edges: command relationships (OPCON, TACON, DS)	Node-level: predicted readiness impact of degradation at any node
Logistics dependency analysis	Nodes: units + depots; Edges: supply relationships	Node-level: supply chain disruption risk; graph-level: theater sustainment risk
Threat network analysis	Nodes: entities; Edges: observed associations	Node-level: threat actor classification; edge-level: relationship type prediction
Coalition coordination graph	Nodes: NATO/partner units; Edges: coordination relationships	Graph-level: coordination bottleneck detection

3-5. Task: Build a GNN for Unit Readiness Propagation

CONDITIONS: You have an MSS Ontology with Unit objects and command relationship links. You have unit readiness data (C-ratings, equipment fill, personnel strength) as node features. You have a labeled dataset of historical readiness cascades — instances where a readiness event at one unit measurably impacted a connected unit's reported readiness within a defined time window.

STANDARDS: The GNN model predicts readiness impact propagation at ≥ 0.75 AUC on the held-out test set. The model produces node-level explanations (influence scores from neighbor nodes) for every prediction. The model is registered in the MSS model registry with complete documentation.

EQUIPMENT: Foundry Code Workspace, PyTorch Geometric (authorized version), MSS Ontology graph export (units and command relationships as edge list), readiness dataset (Foundry dataset).

PROCEDURE:

1. **Construct the graph from the Ontology.** Export unit nodes and command relationship links from the Foundry Ontology. Represent as a node feature matrix (one row per unit, columns = readiness features) and an edge index (two-column tensor of source/target node indices for each command relationship). Validate: (a) the graph is connected — isolated nodes indicate missing Ontology relationships, not an absence of real relationships; (b) the edge direction is consistent with command hierarchy (edges flow from subordinate to superior in OPCON relationships for readiness propagation modeling).
2. **Define node features.** Include: C-rating (normalized 0–1), equipment fill percentage, personnel strength percentage, days since last readiness report, number of subordinate units, relationship type (OPCON/TACON/DS as one-hot). Feature engineering for graph models follows the same principles as tabular models — domain knowledge determines which features carry signal.
3. **Select GNN architecture.** For readiness propagation on a hierarchical C2 graph:
4. Use a Graph Convolutional Network (GCN) or Graph Attention Network (GAT) for initial implementation

5. GAT is preferred if you expect neighbor node importance to vary (e.g., a critical OPCON relationship should carry more weight than a DS relationship) — attention heads learn these weights from data
6. Start with 2–3 message-passing layers; depth should not exceed the typical graph diameter of the command hierarchy being modeled (generally 4–6 for USAREUR-AF corps-level)
7. Hidden dimension: 64–128 for a unit-scale graph; larger dimensions do not consistently improve performance on small graphs
8. **Implement training with mini-batch sampling.** Full-graph training is only feasible for small graphs. For graphs with > 1000 nodes, implement neighborhood sampling (e.g., GraphSAGE sampling or cluster-based mini-batch) to make training scalable. Use neighbor sample sizes of [15, 10, 5] for 3-layer models as a starting point.
9. **Evaluate with a time-aware split.** Do not split the labeled cascade dataset randomly — cascades are temporally ordered. Use a temporal split: train on cascades before a defined cutoff date, validate and test on later cascades. Random splitting creates temporal leakage (the model learns future readiness states to predict past ones) and will produce unrealistically high metrics that fail on live data.
10. **Generate node-level explanations.** For every production prediction, the model must produce an explanation attributing the predicted readiness impact to specific neighbor nodes and their contributions. Use GNNExplainer or IntegratedGradients on graph inputs (PyTorch Geometric provides both). The explanation must be surfaced in the Workshop dashboard that consumes this model — operational users must be able to understand why a unit is flagged as at-risk, not just that it is.
11. **Register and document.** Register in the MSS model registry. The model card must include: graph structure description (node types, edge types, feature schema), evaluation metrics (AUC, precision at operational threshold), explanation method, known limitations (e.g., model assumes static graph topology — dynamic OPCON relationships require graph refresh).

WARNING: GRAPH TOPOLOGY AS SENSITIVE INFORMATION. The command relationship graph — who is OPCON to whom, at what echelons, with what force structure — is sensitive operational information. The GNN model artifact encodes this graph structure in its trained weights. Treat trained GNN model artifacts with the same handling requirement as the source command relationship data. Do not store model artifacts in unprotected repositories.

CHAPTER 4 — FEDERATED AND PRIVACY-PRESERVING MACHINE LEARNING

4-1. Overview

BLUF: Federated learning enables model training across data sources that cannot be centralized — due to classification boundaries, coalition data sharing restrictions, or data sovereignty requirements. This chapter covers federated learning architecture for MSS cross-domain scenarios, coalition partner scenarios, and privacy-preserving techniques that reduce the risk of sensitive information leakage through model weights or gradients.

Federated learning does not eliminate data access controls — it provides a pattern for coordinating training without centralizing raw data. The gradient updates that federated learning communicates can still encode sensitive information. This chapter treats federated learning as a security-constrained engineering problem, not a workaround for data governance.

4-2. Federated Learning Scenarios on MSS

Scenario	Description	Authorization Path
Cross-domain (different classification levels)	Training data exists at multiple classification levels; cannot be combined	C2DAO + data steward for each domain; ISSM coordination
Cross-command (different USAREUR-AF subordinate commands)	Training data is geographically distributed across III Corps, V Corps, 21st TSC, 10th AAMDC, 56th MDC-E, SETAF-AF nodes	C2DAO architecture review; data steward coordination per command
Coalition partner (MPE environment)	Partner-nation data participates in joint model training without sharing raw data	C2DAO + EUCOM J6 + applicable data sharing agreement
Privacy-sensitive operational data	Personnel data used for readiness models; privacy-preserving training required	C2DAO + G1 + JAG coordination for applicable privacy framework

4-3. Federated Learning Architecture on MSS

The standard MSS federated learning architecture uses a central aggregator pattern (FedAvg protocol) with the following components:

```
CENTRAL AGGREGATOR (MSS/Foundry)
  - Holds: global model, aggregation logic, round coordination
  - Does NOT hold: local training data from participants
  - Responsibilities: initialize model, distribute weights, collect gradients/updates,
aggregate, distribute updated global model

PARTICIPANT NODES (each MSS domain / command node / partner node)
  - Holds: local training data (stays local, never transmitted)
  - Responsibilities: receive global model weights, train on local data for N local
epochs, transmit weight updates (not raw data) to aggregator

SECURE CHANNEL
  - Weight updates transmitted over encrypted channel
  - Aggregator authenticates each participant before accepting updates
  - Differential privacy noise applied to updates before transmission (see 4-5)
```

FedAvg protocol (simplified):

```
Round r:
  1. Aggregator distributes current global model weights W_r to all participants
  2. Each participant k trains on local data for E epochs, producing W_r^k (updated
weights)
  3. Each participant transmits W_r^k - W_r (the weight delta, not full weights) to
aggregator
  4. Aggregator computes W_{r+1} = W_r + (1/K) * Σ_k (W_r^k - W_r)
     (weighted average if participant dataset sizes differ)
  5. Aggregator distributes W_{r+1} to all participants
  Repeat for R rounds
```

4-4. Task: Implement a Federated Retraining Coordination Protocol

CONDITIONS: You are coordinating a federated retraining run for a readiness prediction model across three V Corps brigade combat teams, each holding local readiness data that cannot be centralized due to operational security restrictions. The global model is registered in the MSS model registry. You have C2DAO authorization for the federated training design.

STANDARDS: The federated training protocol completes a defined number of rounds with ≥ 2/3 of participants providing updates in each round (fault tolerance). The global model achieves performance within 5% of a hypothetical centralized baseline on a held-out evaluation set. Gradient updates are transmitted over an authenticated, encrypted channel. Differential privacy is applied to participant updates per the design document.

EQUIPMENT: Central Foundry environment, Foundry Code Workspace at aggregator node, authenticated Foundry connections to participant environments, differential privacy library (authorized version), encrypted communication channel (provided by USAREUR-AF IT/J6).

PROCEDURE:

1. **Define the participation agreement.** Before writing code, document: (a) which participants are included and their data steward contacts, (b) the model architecture (must be identical at all participants), (c) the number of local training epochs per round, (d) the number of global rounds, (e) the minimum number of participating nodes required to proceed with aggregation in each round, (f) the differential privacy budget (ε, δ) if applicable. This document requires data steward sign-off from each participating domain before the protocol begins.
2. **Distribute the initial global model.** Package the current model weights as a serialized artifact. Distribute to each participant node via the authorized secure channel. Verify receipt at each participant before proceeding. Record distribution timestamp and model artifact hash.
3. **Coordinate local training.** Send the round initiation signal to all participants. Each participant: loads the global model weights, trains for the defined number of local epochs on their local dataset, applies differential privacy noise to the weight update (see 4-5), and transmits the noisy weight update to the aggregator. The aggregator waits for the defined minimum number of updates before proceeding. Log: which participants responded, response latency, any transmission errors.
4. **Aggregate updates.** Apply the FedAvg aggregation at the central aggregator. Weight the participant updates by participant dataset size if sizes differ significantly (weighted FedAvg reduces the influence of participants with very small local datasets). Record the aggregated update norm — a very large aggregated update norm may indicate a rogue or compromised participant update (see Chapter 7, Byzantine robustness).
5. **Evaluate the global model.** After each round, evaluate the updated global model on a held-out global evaluation set (data that is held at the aggregator, representative of the full distribution). Track: global evaluation metric per round, participant-specific evaluation metric if accessible (for detecting data heterogeneity effects), update norm per participant. Log all metrics to the experiment tracker.
6. **Terminate and register the final model.** After R rounds or when the global evaluation metric converges (improvement < defined threshold for two consecutive rounds), terminate the protocol. Register the final global model in the MSS model registry with: federated training run ID, participant list (anonymized if required by data sharing agreement), number of rounds completed, final evaluation metrics, differential privacy budget consumed, participation agreement document reference.

WARNING: GRADIENT UPDATE INSPECTION. Gradient updates from participant nodes can be inspected by the aggregator. In cross-domain federated learning where participants are at different classification levels, the aggregator must not be at a lower classification level than any participant's data. The aggregator's classification level determines what information can be safely aggregated. Coordinate with the ISSM and C2DAO before deploying any cross-domain federated architecture.

CAUTION: NON-IID DATA. In operational federated learning, participant data is almost never identically distributed (IID). Each brigade has its own readiness profile, operational tempo, and reporting patterns. Non-IID data causes the FedAvg algorithm to converge to a suboptimal global model that performs well for participants with large datasets and poorly for participants with small or atypical datasets. Mitigation options: (a) FedProx (adds a proximal term to local training to prevent drift from the global model), (b)

per-participant model fine-tuning after global training, (c) stratified global evaluation to detect per-participant performance disparities. Document the non-IID mitigation strategy in the participation agreement.

4-5. Differential Privacy for MSS

Differential privacy (DP) provides a mathematical guarantee that the presence or absence of any single training record cannot be inferred from the model's weight updates with probability greater than e^ε (the privacy budget). On MSS, DP is required for all federated models that process personnel data and recommended for any model trained on data with individual-level records.

DP implementation on MSS uses the Gaussian mechanism for gradient perturbation:

```
Noisy gradient update = Clipped gradient + Gaussian noise
where:
  - Gradient clipping norm C: clips the L2 norm of each gradient to ≤ C
    (prevents any single record from having outsized influence)
  - Gaussian noise σ: standard deviation proportional to C/ε
    (larger ε = less noise = less privacy; smaller ε = more noise = more privacy)
  - DP budget (ε, δ): ε is the privacy loss bound; δ is the failure probability
    (for MSS personnel models: target ε ≤ 8.0, δ ≤ 1/N where N is training set size)
```

DP parameter selection guidance for MSS:

Use Case	Recommended ε	Notes
Personnel readiness (individual-level)	ε ≤ 3.0	Strongest protection; expect 2–5% performance penalty
Equipment readiness (individual-vehicle records)	ε ≤ 8.0	Moderate protection
Logistics demand (aggregate unit-level)	ε ≤ 10.0 or not required	Individual records have lower sensitivity
Threat pattern (aggregated intelligence)	Coordinate with data steward	Classification-specific requirements

NOTE

Differential privacy incurs a performance cost. A model trained with DP will perform worse than the same model trained without DP on the same data. This is expected and unavoidable — the privacy guarantee requires adding noise. Document the DP-induced performance delta in the model card. The mission owner must accept the DP-constrained performance level explicitly.

CHAPTER 5 — ADVANCED INTERPRETABILITY AND BIAS AUDITING

5-1. Overview

BLUF: Advanced interpretability at SL 5M level means operating SHAP and LIME at production scale, designing interpretability as a platform capability (not a per-model afterthought), and conducting bias audits that satisfy DoD AI Ethics Principles, Army CIO policy (April 2024), and RAIMTF requirements. This chapter covers SHAP at scale, LIME for local explanations, fairness metrics for operational models, bias remediation, and the audit documentation trail required for DoD compliance.

5-2. SHAP at Production Scale

SL 4M introduced SHAP for model explanation. At SL 5M level, the challenge is not computing SHAP — it is computing SHAP at the volume, frequency, and latency that production operational models require.

SHAP computational complexity by explainer type:

Explainer	Complexity	Best For
TreeExplainer	O(TLD) — fast for tree ensembles	All gradient boosting models on MSS; preferred
DeepExplainer	O(B * F) — batch-based neural network	Neural networks; significantly slower than TreeExplainer
KernelExplainer	O(2^F * N) — model-agnostic	Avoid in production; use only for validation of other explainers
LinearExplainer	O(F) — linear models	Linear models only; very fast
PermutationExplainer	O(F * N)	Model-agnostic; slower than TreeExplainer but feasible for moderate F

Production SHAP strategies for MSS:

1. **Precompute SHAP values on the evaluation dataset.** For batch inference models (daily readiness scoring, weekly logistics forecasts), compute SHAP values for every prediction as part of the inference Transform. Store SHAP values as additional columns in the output dataset. This makes explanation available in downstream dashboards without adding latency to the serving path.
2. **Use background dataset compression.** SHAP explainers require a background dataset to define the baseline expectation. For production, compress the background dataset to 100–500 representative samples using k-means clustering on the feature space. A smaller, representative background dataset produces nearly identical SHAP values to the full training set at a fraction of the computational cost.

3. **Cache explanations for repeated queries.** Operational dashboards often request the same explanation multiple times (a user refreshing a readiness view). Implement explanation caching with a TTL (time-to-live) aligned to the model's refresh cadence — cache SHAP values until the next model refresh clears them.
4. **Use waterfall plots for individual-record explanations in Workshop.** For Workshop dashboards serving individual readiness records or logistics exception reports, the SHAP waterfall plot (showing per-feature contribution from the expected value to the prediction) is the most operationally interpretable format. The -30 builder implementing the Workshop application consumes the SHAP values column from the inference output dataset — they do not need to run the explainer themselves.

5-3. Task: Implement Platform-Level Interpretability Infrastructure

CONDITIONS: You are designing the interpretability infrastructure for a new MSS model platform serving 10+ ML models with different architectures (tree ensembles, neural networks, linear models). Each model produces predictions consumed by Workshop dashboards. The mission owner requires explanation for every prediction in the dashboard.

STANDARDS: Explanation is available for every model prediction with latency impact ≤ 10% on the total inference pipeline runtime. SHAP values are stored per-prediction in the model output dataset. Explanation is surfaced to end-users in a Workshop dashboard without requiring Workshop builders to implement SHAP themselves.

EQUIPMENT: Foundry Code Workspace, SHAP library (authorized version, pinned in workspace requirements), model output datasets (Foundry datasets), Foundry Ontology (for registering explanation schema as a standard property set), Workshop (for surfacing explanations to end-users).

PROCEDURE:

1. **Define a standard explanation output schema.** Establish a platform-wide schema for model explanation outputs:
2. `prediction_id`: unique identifier linking explanation to prediction record
3. `feature_names`: array of feature names used in this prediction
4. `shap_values`: array of SHAP values, one per feature, same order as feature_names
5. `base_value`: SHAP base value (expected model output)
6. `prediction_value`: model output for this record
7. `top_features`: pre-computed array of (feature_name, shap_value) sorted by |shap_value| descending, top 5 — pre-computation reduces dashboard rendering latency Registering this schema in the MSS Ontology as a standard property set enables -30 builders to consume explanations without knowing the underlying model.

8. **Implement explainer selection logic.** Write a utility function that, given a trained model artifact, selects the appropriate explainer type (TreeExplainer for tree models, LinearExplainer for linear models, DeepExplainer for neural networks) and instantiates it with a compressed background dataset. This function is the platform library that SL 4M MLEs call — they should not need to implement explainer selection themselves.
9. **Integrate explanation into the inference Transform.** Modify the standard inference Transform template to include an explanation step after prediction. For batch models: compute SHAP values for all predictions in the batch, append as columns to the output dataset. For models where TreeExplainer is applicable, this adds < 5% to total inference time at typical MSS batch sizes (10k–100k records).
10. **Publish the explanation dataset as an Ontology property set.** Register the explanation output dataset as a data product. Link it to the model's primary prediction output via `prediction_id`. Workshop applications that consume model predictions can join to the explanation dataset to surface per-feature contributions without additional computation.
11. **Document the interpretation guide.** For each deployed model, write a one-page interpretation guide that tells operational users: (a) how to read a SHAP waterfall plot, (b) what the top features mean in operational terms (not statistical terms), and (c) what a large positive / large negative SHAP value means for the specific prediction task. This guide is mandatory — explanation outputs without interpretation guides create false confidence in users who misread the charts.

5-4. LIME for Local Explanations

LIME (Local Interpretable Model-agnostic Explanations) generates local explanations by fitting a simple interpretable model (typically linear) to the neighborhood of a specific prediction. LIME is slower than SHAP for tree models but is more appropriate when:

- The model is a complex neural network or black-box ensemble where SHAP computational cost is prohibitive
- The end-user requires an explanation in terms of input text segments (for transformer models, LIME-based text explanations highlight which tokens drove the classification)
- The explanation must be model-agnostic (e.g., comparing explanations across model versions with different architectures during an A/B test)

LIME vs. SHAP selection guidance:

Situation	Recommended Explainer
Production tree ensemble model, batch inference	SHAP (TreeExplainer)
Production neural network, batch inference	SHAP (DeepExplainer) if feasible; LIME as fallback

Situation	Recommended Explainer
Transformer text classification, individual query	LIME (text explainer)
Real-time inference, latency-constrained	Pre-computed SHAP (batch) or LIME with sample size reduced to 50
Debugging individual mispredictions	LIME preferred (neighborhood-local; easier to interpret edge case behavior)
Global feature importance for audit report	SHAP (global mean absolute SHAP values) preferred

5-5. Bias Auditing for Operational Models

DoD AI Ethics Principles require that DoD AI systems be equitable — they must not exhibit bias that creates disparate treatment across groups. Army CIO Memo (April 2024) extends this to data products including ML models. On MSS, bias auditing is mandatory for any model that:

- Produces outputs used in personnel decisions (readiness assessments, duty assignment, evaluation support)
- Differentiates between groups of personnel in its feature inputs or prediction outputs
- Is used to allocate resources across units in ways that may have disparate impact

Bias auditing is recommended (not mandatory) for logistics and equipment models, but should be conducted when the mission owner identifies a potential equity concern.

Fairness metrics for MSS operational models:

Metric	Definition	Use Case
Demographic parity	P(prediction = positive	group A) = P(prediction = positive
Equal opportunity	P(prediction = positive	actual = positive, group A) = same for group B
Equalized odds	Equal opportunity + equal false positive rates across groups	Strictest; requires both TPR and FPR equality
Calibration	P(actual = positive	predicted probability = p) is equal across groups
Individual fairness	Similar individuals receive similar predictions	Requires a domain-specific similarity metric

Group definitions for MSS personnel model bias audits:

The following groups require disaggregated evaluation on all personnel models. The data steward must confirm that group membership data is available and authorized for use in bias evaluation (it is an audit tool, not a training feature — it must not be included as a model input feature).

Protected Attribute	MSS Data Source	Notes
Gender	IPPS-A records via authorized data product	Use only for audit evaluation, not as model input
Race/ethnicity	IPPS-A records via authorized data product	Use only for audit evaluation, not as model input
MOS/career field	IPPS-A	MOS may be legitimate model input; audit for MOS-correlated disparities
Grade/rank	IPPS-A	May be legitimate model input; audit separately
Component (Active/Reserve/Guard)	IPPS-A	Component may be correlated with data collection patterns; audit carefully

5-6. Task: Conduct a Bias Audit for a Personnel Readiness Model

CONDITIONS: You have a trained readiness prediction model that produces C-rating probability scores for individual soldiers. The model is a candidate for production deployment. You have access to the held-out test set with ground truth labels and model predictions. You have access to authorized demographic data for the test set population (via approved IPPS-A data product, joined on masked individual identifier).

STANDARDS: The audit report documents demographic parity, equal opportunity, and calibration for all required protected attributes. Disparities exceeding defined thresholds are flagged and remediation options are documented. The audit report is reviewed and accepted by the mission owner and G1 before any deployment proceeds.

EQUIPMENT: Foundry Code Workspace, trained model artifact (candidate for production), held-out test set with ground truth labels and model predictions (Foundry dataset), authorized IPPS-A demographic data product (Foundry dataset, accessible via approved join on masked individual identifier), bias audit report template, fairness metric computation library (scikit-learn or equivalent authorized version).

PROCEDURE:

1. **Join the evaluation dataset with demographic data.** Join the test set predictions to the authorized demographic data product using the masked individual identifier. This join must occur inside Foundry — do not export the joined dataset. Verify that the joined demographic columns are marked as audit-only and are not included in the model's training feature set.
2. **Compute overall model performance.** Record: accuracy, precision, recall, F1, AUC, calibration curve. These are the baseline metrics against which disaggregated metrics are compared.

3. **Compute disaggregated metrics per protected attribute.** For each required protected attribute (gender, race/ethnicity, component, rank group): compute precision, recall, F1, and AUC for each group value. Use a minimum group size of 30 records for a stable estimate — report groups below this threshold as "insufficient sample size" rather than computing a potentially misleading metric.
4. **Apply disparity thresholds.** The following thresholds trigger a required remediation action:
5. Recall disparity (equal opportunity gap): | TPR_group_A - TPR_group_B | > 0.05 — REQUIRES REMEDIATION
6. Precision disparity: | Precision_group_A - Precision_group_B | > 0.08 — REQUIRES INVESTIGATION
7. AUC disparity: | AUC_group_A - AUC_group_B | > 0.05 — REQUIRES INVESTIGATION These thresholds are MSS defaults. The mission owner may set stricter thresholds. Do not set looser thresholds without C2DAO approval.
8. **Document findings in the bias audit report.** The bias audit report must include: model identifier and version, evaluation dataset description, disaggregated metrics for all required attributes, disparity calculations, flags for any threshold exceedances, analysis of likely causes (e.g., "lower recall for Reserve component soldiers is likely driven by infrequent reporting cadence in the training data"), and proposed remediation options.
9. **Implement and re-evaluate remediation if required.** Remediation options include: reweighting the training loss by group membership (cost-sensitive learning), threshold calibration per group (different decision thresholds by group to equalize recall), collecting additional training data for underrepresented groups, or feature removal (removing features that encode protected attributes through proxy variables). Re-run the full bias audit after any remediation. Document: which remediation was implemented, the metric change before/after, and the overall performance impact.
10. **Obtain mission owner and G1 acceptance.** The mission owner reviews the bias audit report and explicitly accepts or rejects the model for deployment. For personnel models, G1 coordination is required before acceptance. Document the acceptance decision. A deployment that proceeds without this step is a governance violation.

NOTE

Bias auditing is not a one-time activity. The bias audit must be repeated after every retraining event. Include the bias audit as a required stage in the automated retraining pipeline (see Chapter 2). If the automated pipeline detects a bias threshold exceedance, the pipeline must halt and alert the MLE — bias exceedances are not auto-remediable; they require human review.

CHAPTER 6 — REAL-TIME INFERENCE AND MODEL OPTIMIZATION

6-1. Overview

BLUF: Real-time inference on MSS requires explicit latency SLAs, defined fallback behavior, and systematic optimization. This chapter covers the MSS inference architecture, latency measurement methodology, request batching, model compression (pruning and quantization), and the production readiness benchmarking process.

Most MSS ML models run as batch inference — daily or weekly scoring jobs. Real-time inference is required when operational use cases need prediction latency < 10 seconds from input to output. Real-time inference is operationally appropriate for: anomaly detection on live sensor feeds, interactive readiness queries in Workshop applications, and real-time logistics exception scoring. It is not appropriate as a default — batch inference is simpler, cheaper, and easier to monitor.

NOTE

Batch inference is one deployment pattern. Foundry also supports near-real-time inference via streaming transforms, on-demand inference via OSDK function calls integrated into applications, and scheduled incremental inference via @incremental transforms. Select the pattern based on operational latency requirements.

6-2. MSS Inference Architecture Options

Architecture	Latency Target	Use Case	Notes
Batch Transform (scheduled)	Hours	Daily/weekly scoring	Default for most use cases; highest reliability
On-Demand Transform (triggered)	Minutes	Event-driven scoring on new records	Appropriate for operational event-triggered workflows
Foundry Model Endpoint (real-time)	Seconds (< 10s P99)	Interactive Workshop, AIP Logic integration	Requires endpoint health monitoring and fallback
Streaming inference (experimental)	Sub-second	Real-time sensor feeds	Not generally available on MSS; coordinate with C2DAO

6-3. Latency Measurement and SLA Definition

Before optimizing latency, measure it correctly. A latency measurement that does not reflect the production workload profile will produce misleading optimization decisions.

Latency measurement protocol for MSS inference endpoints:

1. **Measure end-to-end latency, not model latency.** End-to-end latency includes: input preprocessing (feature extraction from raw Ontology data), model inference, output postprocessing (explanation computation, threshold application), and response serialization. Model-only latency is a useful diagnostic but not the SLA metric.
2. **Measure under production-representative load.** Issue requests at the expected production request rate for a sustained period (minimum 10 minutes). Transient load does not reveal steady-state latency degradation from resource contention.
3. **Record the full latency distribution, not just mean.** Record P50, P90, P95, P99 latency. Mean latency is misleading for SLA purposes — operational users experience the tail, not the average. Define the SLA as a P99 constraint: "P99 latency ≤ 8 seconds under production load."
4. **Test with production-representative input distribution.** Use a sample of actual operational records as the benchmark payload. Synthetic inputs may not exercise the same code paths as production data, particularly for preprocessing that handles edge cases.

6-4. Request Batching

Request batching accumulates multiple inference requests and processes them together as a single batch. For most ML frameworks, batch inference is significantly faster per-request than single-record inference because it amortizes fixed overhead (model loading, preprocessing) across multiple records.

Batching configuration for MSS endpoints:

Parameter	Recommended Starting Value	Notes
Max batch size	32 for neural networks; 256–1024 for tree models	Tune based on memory and latency budget
Max batch wait time	50–100ms for interactive use; 500ms for event-driven	Wait time adds to P99 but reduces average latency under load
Dynamic batching	Enabled	Accept batches smaller than max size if wait time exceeded

Batching is not appropriate when:

- Individual requests require different model versions (A/B routing is incompatible with naive batching)
- The request payload includes large variable-length sequences that cause excessive padding overhead (common with transformers on variable-length SITREPs)

- The fallback path (see 6-6) cannot be batched — partial batch failures require careful error handling

6-5. Task: Optimize a Production Inference Pipeline

CONDITIONS: An existing production inference endpoint is exceeding its P99 latency SLA of 5 seconds under peak operational load. The model is a gradient boosting classifier (XGBoost) predicting equipment maintenance requirements from GCSS-Army data. Baseline P99 latency is 9.2 seconds. Batch inference of 500 records per request.

STANDARDS: After optimization, P99 latency ≤ 4 seconds under the production load profile. Model performance (F1, AUC) does not degrade below 2% of the pre-optimization baseline.

EQUIPMENT: Foundry Code Workspace, existing production inference Transform and model artifact, latency profiling instrumentation (Python time/logging or equivalent), production-representative benchmark payload (sample of operational GCSS-Army records), XGBoost library (authorized version), MSS model registry entry for the model under optimization.

PROCEDURE:

1. **Profile the latency breakdown.** Instrument the inference pipeline to record latency per stage: data ingestion (Foundry dataset read), preprocessing, model inference, postprocessing, output write. In most cases, one stage dominates. This step prevents optimizing the wrong bottleneck. In operational practice, data ingestion and preprocessing account for 60–80% of end-to-end latency for batch inference pipelines — the model itself is rarely the bottleneck.
2. **Optimize data ingestion.** If Foundry dataset read is the bottleneck: (a) ensure the Transform is reading only the required columns (projection pushdown), not full-width reads; (b) verify that filter predicates are pushed down to the Foundry storage layer, not applied post-read; (c) check whether the dataset is partitioned by a column aligned with the query predicate (e.g., partitioned by unit_id when querying by unit_id).
3. **Optimize preprocessing.** If preprocessing is the bottleneck: (a) vectorize all preprocessing operations — eliminate Python loops over rows; use pandas vectorized operations or numpy; (b) precompute stable features (features that do not change with new data, such as unit historical statistics) in a separate scheduled pipeline and join at inference time rather than recomputing; (c) cache frequently reused lookup tables (reference data, unit mappings) in memory for the duration of the inference job.
4. **Apply model-level optimization if preprocessing is not the bottleneck.** For XGBoost models: (a) verify the model is loaded from a compiled/serialized artifact, not re-fitted at inference time; (b) use XGBoost's native prediction on numpy arrays rather than pandas DataFrames (avoids dtype conversion overhead); (c) for very large models (> 500 trees), evaluate tree pruning: remove trees with below-threshold contribution to the ensemble; re-evaluate performance after pruning.

5. **Apply quantization for neural network models.** If the model is a neural network: (a) apply post-training dynamic quantization (weights quantized to INT8; activations quantized at runtime) — this reduces model size ~4x and accelerates CPU inference 2–3x at minimal performance cost; (b) evaluate INT8 quantized model on the full evaluation suite; confirm performance degradation is within the ≤ 2% tolerance; (c) benchmark quantized model latency under production load profile.
6. **Re-benchmark end-to-end latency.** After each optimization step, re-measure P99 latency under production-representative load. Continue until the SLA is met or all feasible optimizations are exhausted. If optimization cannot achieve the SLA, escalate to C2DAO for a compute allocation review — the current endpoint may be under-resourced.
7. **Re-evaluate model performance.** Run the full model evaluation suite on the optimized model. Confirm F1 and AUC are within 2% of the pre-optimization baseline. For quantized models, also confirm that the SHAP explanation output is numerically consistent with the pre-quantization baseline (quantization can alter explanation magnitudes).
8. **Document and register the optimized model.** Update the model registry entry: add optimized artifact hash, optimization techniques applied, latency benchmark results (pre- and post-optimization), performance delta, and the compute profile under which the benchmark was run.

CAUTION: QUANTIZATION AND EXPLANATION CONSISTENCY. INT8 quantization can alter SHAP values by ±5–10% relative to the full-precision model. If the model card commits to specific SHAP ranges for operational decision support, re-validate and update the interpretation guide after quantization. Misleading SHAP values in a quantized model can cause incorrect operational interpretations.

6-6. Fallback Behavior and Graceful Degradation

Every real-time inference endpoint on MSS must define and implement fallback behavior. An endpoint that fails silently — returning no prediction, returning a default value, or crashing — without notifying consumers is an operational hazard.

Required fallback behaviors:

Failure Mode	Required Behavior
Endpoint timeout	Return fallback signal (configurable: last known prediction, rule-based default, "prediction unavailable" flag) and log timeout
Model artifact load failure	Alert MLE and data steward; revert to previous registered model version within 15 minutes
Input schema mismatch	Return validation error with specific field description; do not produce a prediction from a malformed input
Feature pipeline failure	Alert MLE and data steward; do not serve stale predictions older than the defined staleness threshold without explicit operator acknowledgment

Failure Mode	Required Behavior
Latency SLA breach (sustained)	Alert on-call MLE if P99 exceeds SLA for > 5 minutes sustained; log all breaching requests for root-cause analysis

The fallback behavior must be documented in the pipeline runbook and tested in a staging environment before production deployment. Test each failure mode explicitly by injecting the failure condition.

CHAPTER 7 — ML SECURITY AND ADVERSARIAL ROBUSTNESS

7-1. Overview

BLUF: ML models on MSS face real security threats: adversarial inputs designed to cause misprediction, data poisoning attacks that corrupt the training process, and model extraction attacks that reverse-engineer proprietary model behavior. This chapter covers the threat model for MSS ML systems, defensive techniques for adversarial robustness, data poisoning detection, model extraction defense, and supply chain security for ML artifacts.

ML security is not a research concern on MSS — it is an operational requirement. USAREUR-AF MSS models inform decisions in a contested information environment. A threat actor with knowledge of the operational models used for readiness assessment, logistics forecasting, or anomaly detection has an operational incentive to subvert those models. The SL 5M MLE is responsible for ensuring that MSS models are hardened against these threat classes.

7-2. Threat Model for MSS ML Systems

Threat Class	Attack Vector	Target	Potential Operational Impact
Adversarial examples	Crafted inputs at inference time	Readiness classifiers, anomaly detectors	Misprediction causes incorrect operational decisions
Data poisoning	Corrupted data injected into training pipeline	Any model trained on external or multi-source data	Model trained on poisoned data behaves incorrectly for specific inputs or globally
Model extraction	Systematic querying of model endpoint	Any production endpoint	Attacker reconstructs model; identifies evasion inputs
Training data leakage	Model memorizes sensitive training records	Personnel models, intelligence models	Sensitive information extractable from model weights via membership inference

Threat Class	Attack Vector	Target	Potential Operational Impact
Supply chain compromise	Malicious pre-trained model weights or library	Transformer models, any model using external checkpoints	Backdoor triggers cause specific mispredictions
Byzantine attack (federated)	Malicious participant submits corrupted gradient updates	Federated learning aggregator	Global model corrupted; performance degrades or specific inputs misclassified

7-3. Adversarial Robustness

BLUF: Adversarial robustness requires both evaluation (measuring the model's vulnerability to adversarial inputs) and hardening (reducing that vulnerability through defensive techniques). Both are required before production deployment of any MSS model that consumes inputs from untrusted sources.

MSS adversarial threat surface by use case:

Use Case	Input Source Trust Level	Adversarial Threat Level	Required Defense
Internal readiness prediction (IPPS-A/GCSS-Army)	Trusted (authoritative Army systems)	Low	Anomaly detection on feature distribution; no adversarial hardening required
Logistics demand (internal + partner data)	Mixed (partner feeds partially trusted)	Moderate	Input validation; distribution monitoring; adversarial evaluation required
Threat pattern recognition (intelligence feeds)	Untrusted (external/adversary-influenced)	High	Full adversarial hardening required
SITREP classification (free-text input)	Partially trusted	Moderate-High	Input sanitization; adversarial text evaluation required

7-4. Task: Conduct Adversarial Robustness Evaluation

CONDITIONS: You have a trained production candidate model with at least moderate adversarial threat level (see 7-3 threat surface table). The model consumes tabular features or text inputs from external or partner data sources. You have access to the model and the evaluation dataset.

STANDARDS: The adversarial evaluation report documents model accuracy under defined adversarial attack conditions. Robustness is quantified as the relative accuracy degradation under attack vs. clean inputs. If degradation exceeds the defined threshold, adversarial hardening is implemented and the evaluation is repeated before deployment.

EQUIPMENT: Foundry Code Workspace, trained model artifact, evaluation dataset (Foundry dataset), adversarial example generation library (TextFooler or BERT-Attack for text models, authorized versions; gradient computation library for tabular models), adversarial evaluation report template.

PROCEDURE:

1. **Establish clean input baseline.** Record model accuracy on the clean, unperturbed evaluation set. This is the robustness baseline. All adversarial degradation is measured relative to this baseline.
2. **Generate adversarial examples — tabular models.** For tabular feature inputs, apply the following perturbation types:
3. Feature-bounded perturbation: shift each feature independently within a realistic operational range (e.g., equipment fill percentage ±10%, not ±100%). This tests robustness to minor data quality errors or precision manipulation.
4. Worst-case perturbation (FGSM analog for gradient-accessible models): compute the gradient of the loss with respect to input features and shift features in the direction that maximizes loss, constrained by the feature's valid range. This tests worst-case adversarial vulnerability.
5. Realistic manipulation: define operationally plausible input manipulations (e.g., reporting a unit at C-2 instead of C-3 for one inspection period). Test whether these manipulations consistently evade detection.
6. **Generate adversarial examples — text models.** For text classification models (SITREP classifiers), apply:
7. Synonym substitution: replace key terms with synonyms that preserve meaning but change token representations
8. Character-level perturbations: typos, abbreviation variations (realistic in operational text)
9. Paraphrase attacks: restate the SITREP content in different wording while preserving operational meaning Use the TextFooler or BERT-Attack framework (authorized versions) for systematic adversarial text generation.
10. **Measure robustness degradation.** Compute accuracy on each adversarial test set. Calculate: robustness degradation = (clean accuracy - adversarial accuracy) / clean accuracy. Thresholds:
11. Degradation < 5%: Acceptable for low-threat use cases; document
12. Degradation 5–15%: Requires hardening or mission owner acceptance with documented risk
13. Degradation > 15%: Not deployable without adversarial hardening; hardening required
14. **Implement adversarial training if hardening is required.** Adversarial training augments the training set with adversarial examples and retrains the model to be correct on both clean and adversarial inputs. Implementation:
15. Generate adversarial training examples using the same perturbation techniques as the evaluation (Steps 2/3)

16. Mix adversarial examples into the training set at a ratio of 1:3 (adversarial:clean) as a starting point
17. Retrain the model on the augmented dataset
18. Re-evaluate on both clean and adversarial evaluation sets
19. Adversarial training trades a small amount of clean-input performance for robustness. Document this tradeoff.
20. **Document in the model card.** The adversarial evaluation section of the model card (see Appendix B) must include: adversarial attack types tested, robustness degradation per attack type, hardening applied (if any), and residual risk characterization.

7-5. Data Poisoning Detection

Data poisoning occurs when a threat actor inserts or modifies training records to corrupt the resulting model. On MSS, the highest-risk data poisoning vectors are: partner-nation data feeds ingested into joint models, open-source intelligence (OSINT) feeds processed by text models, and any data pipeline that ingests records without human review.

Data poisoning detection techniques for MSS:

Technique	Description	MSS Application
Distribution monitoring	Compare new training batch distribution to established baseline; flag statistical outliers	Apply to all external/partner data feeds at ingestion; use PSI > 0.25 as alert threshold
Label consistency checking	Flag records where the label is statistically inconsistent with the feature values	Apply to labeled training data; anomalous label-feature combinations warrant manual review
Influence function analysis	Identify training records with disproportionate influence on model weights	Apply when model behavior change is suspected; compute before and after retraining
Provenance validation	Verify data provenance chain from source to training dataset	Apply to all data pipeline stages; log provenance in Transform metadata
Gradient-based detection	Identify training examples with anomalous gradient norms during training	Apply during training; log per-record gradient norms; flag top 0.1% for review

WARNING: DATA POISONING IN PARTNER DATA FEEDS. Any data pipeline that ingests records from coalition partner systems, contractor-provided feeds, or external databases represents a potential data poisoning attack surface. Apply distribution monitoring at every ingestion boundary. Anomalous statistical shifts in partner data feeds require data steward review and EUCOM J2 notification before the affected data is included in any model training run.

7-6. Model Extraction Defense

Model extraction attacks work by systematically querying a model endpoint and using the (input, output) pairs to train a surrogate model that approximates the original. On MSS, model extraction is a concern for endpoints accessible to users who should not have visibility into the model's decision logic — for example, a model used to score anomalous logistics patterns that could be reverse-engineered to identify evasion strategies.

Defenses against model extraction on MSS:

Defense	Implementation	Tradeoff
Query rate limiting	Enforce maximum request rate per authenticated user; alert on high-volume systematic queries	Minimal operational impact if threshold set above legitimate use patterns
Output perturbation	Add calibrated noise to prediction probabilities before serving (does not change the binary classification output, only the confidence score)	Reduces utility of confidence scores for downstream consumers
Prediction rounding	Round confidence scores to two decimal places; reduce information density in output	Low operational impact; moderately effective against extraction
Query logging and anomaly detection	Log all inference requests; apply anomaly detection to identify systematic probing patterns	Reactive defense; detection may lag extraction attempt
Access control	Restrict endpoint access to authorized users/applications only	Most effective defense; implement as the primary control

Access control is the primary defense. Output perturbation, rate limiting, and query logging are defense-in-depth measures. Implement them all for high-sensitivity models, but do not use them as a substitute for proper access control.

7-7. Supply Chain Security for ML Artifacts

The MSS ML artifact supply chain includes: pre-trained model checkpoints, third-party Python libraries, training datasets from external sources, and shared platform components. A compromise at any point in this chain can introduce backdoors, performance degradation, or data leakage.

Supply chain security requirements for MSS:

Artifact Type	Security Requirement
Pre-trained model checkpoints	Load only from the MSS-approved checkpoint registry; verify SHA-256 hash against registry entry before loading; do not download from external sources
Python dependencies	Pin all dependency versions in requirements.txt; use only approved packages from the MSS PyPI mirror; periodically re-audit pinned versions against known vulnerability databases
Training datasets (external source)	Verify provenance documentation; apply data distribution monitoring at ingestion; data steward must authorize inclusion in training data before first use
Feature pipelines (shared)	Code review required for all changes; version-controlled; any change to a shared feature pipeline requires re-evaluation of all models that consume it
Model artifacts (production)	Store in MSS model registry with immutable versioning; SHA-256 hash recorded at registration; verify hash before loading in any inference pipeline

Backdoor detection in pre-trained checkpoints:

When adopting a pre-trained checkpoint, apply the following checks before deploying any model fine-tuned on that checkpoint:

1. Test the base checkpoint on a benchmark dataset — record baseline performance
2. Test with a set of known "canary" inputs that should not trigger any special behavior — verify output distributions are consistent with the benchmark
3. Examine activation patterns on the canary set for unusual spikes or saturation in specific layers
4. Review the checkpoint provenance documentation — who produced it, how it was trained, what data it was trained on

If any anomaly is detected, quarantine the checkpoint, notify the C2DAO, and do not proceed with fine-tuning until the anomaly is resolved.

CHAPTER 8 — ML PLATFORM ARCHITECTURE AND LEADERSHIP

8-1. Overview

BLUF: SL 5M MLEs design and lead the ML platform that SL 4M practitioners build on. This chapter covers shared feature store design, model registry architecture, experiment tracking at scale, cross-track integration (SL 4H, SL 4G), code review standards for production ML, reproducibility requirements, and the production readiness framework for leading an ML team.

8-2. Shared Feature Store Design

NOTE — Palantir Developers reference: *Chad & Xander | Lightweight Transforms in Pipeline Builder —* Covers lightweight transform patterns that reduce compute cost and complexity in Foundry Pipeline Builder. Lightweight transforms are a practical design option for the canonical feature pipeline Transforms in a shared feature store, particularly for features that can be computed efficiently without full Spark overhead. Available on the Palantir Developers YouTube channel (@PalantirDevelopers).

A feature store is a centralized system for storing, serving, and managing ML features. On MSS, a shared feature store prevents the feature engineering redundancy problem: 10 different models computing the same "unit equipment fill percentage 30-day rolling average" feature 10 different ways, with 10 different bugs and 10 different staleness profiles.

Feature store requirements for MSS:

Requirement	Implementation on Foundry
Feature definitions as code	Each feature defined in a versioned Foundry Transform with a registered schema
Feature versioning	Transform version controls the feature definition; model registry records which feature version each model was trained on
Training / serving consistency	Same Transform used for both training data generation and inference-time feature computation — eliminates training-serving skew
Feature freshness metadata	Each feature dataset records last-updated timestamp; models consuming stale features generate a warning
Feature-to-model lineage	Foundry dataset lineage graph tracks which models consume which feature datasets
Access control per feature	Column-level permissions applied at the feature dataset level; models can only access features they are authorized for

Training-serving skew is the most common and most damaging feature store failure mode. It occurs when the feature computation logic used during training is subtly different from the logic used during inference — producing a systematic performance gap between offline evaluation and production. Prevent it by design: the feature Transform that generates training data must be the same Transform that computes inference features. Never write separate "training preprocessing" and "inference preprocessing" code.

8-3. Task: Design a Shared Feature Store for an MSS ML Program

CONDITIONS: You are the senior MLE leading an ML program with 5+ production models, all consuming overlapping sets of readiness, logistics, and personnel features from the MSS Ontology. Feature engineering is currently duplicated across model codebases, with inconsistent definitions and variable

staleness.

STANDARDS: The feature store design document specifies: feature catalog structure, feature definition standards, versioning protocol, training-serving consistency mechanism, freshness SLAs, and the migration plan for existing models.

EQUIPMENT: Foundry Code Workspace, access to all existing model codebases and their feature pipeline code (MSS Git repository), Foundry dataset lineage graph (to identify upstream data sources for each feature), MSS Ontology (to map features to authoritative source datasets), feature store design document template.

PROCEDURE:

1. **Conduct a feature inventory.** Enumerate all features across all existing models. Identify duplicates (same operational concept, different implementations), near-duplicates (same concept, slightly different aggregation window or normalization), and unique features. Quantify: for duplicate features, do the implementations agree? Compute the correlation between duplicate feature implementations on the same data; correlations < 0.99 indicate implementation divergence requiring resolution.
2. **Define the feature catalog structure.** Organize features into domains aligned with Foundry Ontology domains:
3. Personnel features: individual and unit-level personnel readiness metrics
4. Equipment features: equipment fill, deadline rates, maintenance backlogs
5. Logistics features: supply fill rates, demand forecasts, distribution node status
6. Operational features: training completion, METL progress, deployment history Each domain has a designated feature steward (the MLE most familiar with the domain's data sources) who owns the canonical definition for that domain's features.
7. **Write canonical feature definitions.** For each deduplicated feature, write a canonical Foundry Transform that: (a) reads from the authoritative source dataset (registered in the Ontology), (b) computes the feature using explicit, commented logic (no ambiguous business rules), (c) outputs a dataset with a registered schema (column names, dtypes, units, null handling), and (d) records the last-updated timestamp in the output metadata. Version-control all canonical feature Transforms in the MSS Git repository.
8. **Implement training-serving consistency.** Register each canonical feature Transform as the single source of truth. All model training pipelines import features from the canonical feature dataset — they do not recompute features locally. Inference pipelines read the same feature dataset (for batch inference) or call the same Transform (for on-demand inference). Any MLE who recomputes a canonical feature in their own pipeline is creating a training-serving skew risk; code review should flag this.
9. **Define freshness SLAs.** For each feature domain, define the maximum acceptable feature staleness for production model inference. Example: "Equipment deadline rates must be refreshed within 24 hours of a GCSS-Army feed update." Register these SLAs in the feature catalog. Implement

automated alerts if a feature dataset is not refreshed within its SLA window — do not serve predictions from stale features without alerting the responsible MLE.

10. **Write the migration plan.** Existing models that compute features locally must be migrated to the shared feature store. The migration plan specifies: (a) migration order (start with the feature most shared across models), (b) equivalence validation procedure (confirm the shared feature and the local feature agree on a test dataset before cutover), (c) rollback procedure for each migrated model, and (d) the timeline with model-by-model cutover dates.

> **NOTE**
>
> Feature store migration is a breaking change for every model migrated. Each migrated model requires re-evaluation against the shared feature to confirm no performance regression. Even a "correct" feature reimplementation may have different edge case handling that alters model behavior on specific records. Do not migrate multiple models simultaneously — migrate one, validate, then proceed to the next.

8-4. Model Registry Architecture

> **NOTE**
>
> Foundry has no native "Model Registry" product. The MSS model registry described in this section is MSS-built infrastructure: a versioned Foundry dataset serving as the authoritative model catalog, combined with the governance tracking system and documentation artifacts defined by the C2DAO. The deployment pattern for a registered model is: batch inference Transform writing to a Foundry dataset → output dataset surfaced as a computed Object property via the Ontology (or consumed via AIP Logic integration). SL 5M MLEs design and maintain this MSS-internal construct — it is not a Foundry platform capability that is provisioned or configured through the Foundry UI.

The MSS model registry is the authoritative record of all ML models in the platform. At SL 5M level, the registry is not just a storage system — it is the governance backbone of the ML platform.

Model registry required fields for MSS:

Field	Description	Required
Model identifier	Unique name; format: `{use_case}_{architecture}_{version}`	Yes
Version	Semantic version (major.minor.patch)	Yes
Status	draft / evaluated / approved / production / deprecated	Yes
Training dataset	Foundry dataset path + version hash	Yes
Feature pipeline	Canonical feature Transform identifier + version	Yes

Field	Description	Required
Hyperparameters	Full training configuration (JSON serialized)	Yes
Evaluation metrics	Primary metric, secondary metrics, per-class metrics, bias metrics	Yes
Evaluation dataset	Foundry dataset path + version hash (separate from training)	Yes
Model artifact hash	SHA-256 of serialized model artifact	Yes
Governance artifacts	Links to: bias audit report, evaluation acceptance, deployment authorization	Required for production status
Model card	Link to RAIMTF-compliant model card (see Appendix B)	Required for production status
Responsible MLE	Name, contact, team	Yes
Data steward	Name, contact	Yes
Inference endpoint	URL or Transform path for production inference	Production only
Monitoring configuration	Drift detector configuration, alerting contacts	Production only
Deprecation date	When the model version will be retired	Production models approaching end of life

Version numbering convention for MSS models:

- **Major version** (X.0.0): Change in model architecture, task definition, or fundamental feature schema — requires full governance re-approval
- **Minor version** (X.Y.0): Retraining with same architecture on updated data — requires evaluation acceptance; bias audit if personnel model
- **Patch version** (X.Y.Z): Bug fix in inference code, documentation update, minor configuration change — requires MLE review and data steward notification; no full re-evaluation required

8-5. Experiment Tracking at Scale

Experiment tracking at scale means that every training run is logged, reproducible, and searchable — not just the most recent one. The ML platform historian is essential: when a production model degrades, the ability to roll back to a previous experiment run requires that run to be fully reproducible from the registry.

Minimum required experiment tracking metadata for MSS:

Category	Fields to Log
Inputs	Training dataset path + version, feature pipeline version, random seed
Configuration	Full hyperparameter configuration (JSON), library versions (requirements.txt snapshot)
Training process	Loss per epoch (train and validation), training time, compute profile used
Outputs	Model artifact location + hash, serialization format
Evaluation	All metrics on validation and test sets; per-class breakdown; bias metrics
Environment	Python version, CUDA version (if GPU), Foundry workspace version

Reproducibility requirement: given an experiment run ID, any MLE on the team must be able to reproduce the training run and obtain identical (within floating-point tolerance) model weights. This requires: the exact training dataset version, the exact feature pipeline version, the exact hyperparameter configuration, and the exact random seed — all recorded in the experiment tracker.

8-6. Cross-Track Integration: SL 4H (AI Engineer) and SL 4G (ORSA)

SL 5M MLEs design integration interfaces between the ML platform and adjacent capability tracks. The two most common integration points are:

Integration with SL 4H (AI Engineer): AIP Logic and Agent Studio

NOTE — Palantir Developers reference: *Applied AI: Scaling AI Workflows and Task Execution with AIP* — Covers how AIP workflows scale across complex, multi-step task execution — directly relevant to the design of AIP Logic integrations that consume ML model outputs from the MSS platform. Provides the SL 5M MLE with the AI Engineer's perspective on workflow orchestration, clarifying the interface design requirements for model endpoints. Available on the Palantir Developers YouTube channel (@PalantirDevelopers).

AIP Logic enables AI-powered workflows that can call ML model endpoints as part of a broader automated pipeline. Agent Studio enables autonomous agents that can query model outputs, trigger Actions, and orchestrate multi-step workflows.

Integration Pattern	SL 4H Responsibility	SL 5M MLE Responsibility
Agent calls model endpoint	AI Eng designs the agent workflow and the endpoint call	MLE ensures endpoint is documented, schema-stable, and latency-compliant; model card is readable by AI Eng
AIP Logic uses model output as condition	AI Eng implements the Logic workflow	MLE registers the model output schema in the Ontology; ensures prediction confidence scores are calibrated

Integration Pattern	SL 4H Responsibility	SL 5M MLE Responsibility
Agent triggers model retraining	AI Eng designs the trigger mechanism	MLE defines the triggerable API, the required authorization, and the automated retraining pipeline (Chapter 2)

The MLE-AI Eng interface contract: the MLE publishes a stable inference interface (schema, latency SLA, versioning policy) and a readable model card. The AI Eng is a consumer, not a model expert. The MLE cannot expect the AI Eng to debug model behavior — that is the MLE's responsibility.

Integration with SL 4G (ORSA): Statistical Validation and Uncertainty Quantification

ORSA practitioners on MSS are the statistical domain experts. They are the most qualified reviewers of model evaluation methodology, uncertainty quantification approaches, and the interpretation of model outputs in operational decision frameworks.

Integration Pattern	SL 4G ORSA Responsibility	SL 5M MLE Responsibility
Validation of evaluation methodology	Reviews evaluation metrics for statistical validity; flags methodological weaknesses	Presents methodology for ORSA review; incorporates feedback; documents disagreements
Uncertainty quantification	Advises on calibration, confidence interval construction, decision thresholds	Implements UQ outputs; surfaces them in model card and dashboard
Simulation-backed evaluation	Runs operational simulations to evaluate model impact on decisions	Provides model output distributions for ORSA to use as simulation inputs
Statistical significance testing	Designs significance tests for A/B model experiments	Implements A/B logging; provides paired samples for ORSA to analyze

Do not skip ORSA review of evaluation methodology for any model with high operational consequence. The MLE builds the model; the ORSA validates that the evaluation is a sound basis for the deployment decision. These are distinct competencies and both are required.

8-7. Code Review Standards for Production ML

Code review in production ML must check correctness properties that standard software code review does not address. Establish and enforce the following ML-specific code review standards for all MSS ML production code:

ML code review checklist (reviewer responsibilities):

Category	Check
Data leakage	Is there any possibility that test or validation data influenced training? (Check: preprocessing steps, imputation with population statistics, feature scaling — all computed on training data

Category	Check
	only, then applied to validation/test)
Training-serving skew	Does the training preprocessing code match the inference preprocessing code exactly? (Check: any difference is a bug)
Evaluation methodology	Is the evaluation dataset representative of the production population? Is the split methodology appropriate (temporal split for time-series; stratified split for imbalanced classes)?
Random seeds	Are all random seeds set and recorded for reproducibility?
Experiment logging	Is the experiment tracker called at the start of the run with full configuration? Are all metrics logged at the end?
Model serialization	Is the model serialized with a stable format (joblib for sklearn, torch.save with full state_dict for PyTorch)?
Feature schema validation	Is the feature schema validated before training and inference? Will the pipeline raise a clear error if features are missing or incorrectly typed?
Bias audit	Is the bias evaluation included in the evaluation step, not deferred to "later"?
Resource cleanup	Are large objects (training datasets, intermediate tensors) explicitly deleted after use? (Foundry workspace memory is shared; leaking memory affects other jobs)
Governance integration	Are all governance artifacts (evaluation metrics, model card, deployment authorization) generated by the pipeline, not manually assembled post-hoc?

NOTE

Code review is not optional for production ML code on MSS. Every pull request that touches a production feature pipeline, model training script, or inference Transform must have at least one reviewer who is a SL 4M qualified MLE. Senior code (automated retraining pipelines, shared feature store Transforms, model registry integration) requires a SL 5M qualified reviewer.

8-8. Leading ML Capability: Production Readiness Gates and Team Development

The SL 5M MLE's leadership responsibilities extend beyond technical design. They include: setting and enforcing production readiness standards, developing SL 4M practitioners on the team, and building a culture of operational ML accountability.

Production readiness framework for MSS ML teams:

Stage	Artifact Required	Reviewer
Use case authorization	Use case brief; data availability assessment	C2DAO

Stage	Artifact Required	Reviewer
Design review	Model design document; feature pipeline spec; governance plan	SL 5M senior MLE + C2DAO
Pre-training review	Data quality report; feature schema validation; bias plan	SL 5M senior MLE
Evaluation review	Evaluation report; bias audit; ORSA methodology sign-off	Mission owner + SL 5M senior MLE
Pre-deployment review	Full production readiness checklist (Appendix A); model card (Appendix B); rollback procedure	SL 5M senior MLE + C2DAO
Post-deployment review (30 days)	Monitoring report; drift status; operational user feedback	SL 5M senior MLE + data steward

Team development responsibilities for SL 5M MLEs:

- Conduct structured code review for all SL 4M practitioners, providing documented, specific, actionable feedback — not just approval/rejection
- Lead post-incident reviews for all production ML incidents; publish findings to the team
- Maintain the team's shared knowledge base: approved patterns, anti-patterns with documented failure modes, lessons from past incidents
- Review and update production readiness gate criteria annually; incorporate lessons from incidents and from platform evolution
- Identify knowledge gaps in the SL 4M team; recommend or design targeted training to address them
- Represent ML capability requirements to C2DAO in architecture reviews; advocate for platform investments that unblock the team

APPENDIX A — ML PRODUCTION READINESS CHECKLIST (ADVANCED)

Purpose: This checklist extends the SL 4M governance checklist for models operating under SL 5M scope: automated retraining pipelines, real-time inference endpoints, federated models, or models with advanced neural architectures. Complete all items. Mark N/A only if the item is genuinely inapplicable — document the justification.

Model Identifier: ____ **Version:** ____ **Responsible SL 5M MLE:** ____ **Date of Review:** ____

Section 1: Automated Retraining Pipeline (if applicable)

#	Requirement	Status	Notes
1.1	Automation tier documented and C2DAO-approved (Tier 1 / Tier 2 / N/A)	☐	
1.2	All five pipeline stages implemented as discrete, independently monitorable units	☐	
1.3	Trigger conditions defined and documented (minimum two triggers)	☐	
1.4	Data validation stage implemented with Foundry Checks; all failures tagged ERROR	☐	
1.5	Feature pipeline version-pinned in pipeline configuration	☐	
1.6	Promotion gate implemented with primary metric threshold and secondary metric floors	☐	
1.7	Human approval integration implemented per authorized automation tier	☐	
1.8	Rollback register implemented; rollback tested and documented	☐	
1.9	Post-deployment monitoring activation implemented	☐	
1.10	Pipeline runbook written and registered in MSS documentation repository	☐	
1.11	Bias audit included as required pipeline stage for personnel models	☐	

Section 2: Real-Time Inference Endpoint (if applicable)

#	Requirement	Status	Notes
2.1	Latency SLA defined (P99 latency target specified)	☐	
2.2	P99 latency benchmark conducted under production-representative load	☐	
2.3	Latency SLA met in benchmark; benchmark results recorded	☐	
2.4	Request batching configured and validated	☐	
2.5	Fallback behavior defined and implemented for all failure modes	☐	
2.6	Fallback behavior tested by explicit failure injection	☐	
2.7	Endpoint health monitoring configured; alerting contacts registered	☐	
2.8	Input schema validation implemented; malformed inputs return structured errors	☐	
2.9	Model artifact hash verified at endpoint load time	☐	
2.10	Explanation output available per-prediction (SHAP or LIME per architecture)	☐	

Section 3: Advanced Neural Architecture (if applicable)

#	Requirement	Status	Notes
3.1	Architecture selection documented with justification (why transformer/GNN required; baseline comparison)	☐	
3.2	Pre-trained checkpoint loaded from MSS-approved registry; hash verified	☐	
3.3	Operational vocabulary coverage evaluated; domain-adaptive pre-training applied if required	☐	
3.4	Inference latency benchmarked; quantization applied if required to meet SLA	☐	
3.5	Quantized model re-evaluated on full evaluation suite; performance within tolerance	☐	
3.6	Node/local explanations implemented for GNN models	☐	
3.7	Graph topology sensitivity assessed; model artifact handling requirements documented	☐	

Section 4: Federated Learning (if applicable)

#	Requirement	Status	Notes
4.1	Participation agreement written and signed by all data stewards	☐	
4.2	C2DAO and (if cross-domain) ISSM authorization obtained	☐	
4.3	EUCOM J6 and applicable data sharing agreement reviewed (coalition scenarios)	☐	
4.4	Differential privacy budget (ε, δ) defined and documented	☐	
4.5	DP-constrained performance delta documented; mission owner acceptance recorded	☐	
4.6	Non-IID mitigation strategy documented and implemented	☐	
4.7	Byzantine robustness mechanism implemented (aggregation anomaly detection)	☐	
4.8	Communication channel authenticated and encrypted	☐	

Section 5: Interpretability and Bias

#	Requirement	Status	Notes
5.1	SHAP or LIME implemented; explainer type appropriate for model architecture	☐	

#	Requirement	Status	Notes
5.2	Explanation output available per-prediction in model output dataset	☐	
5.3	Interpretation guide written for operational end-users	☐	
5.4	Bias audit conducted for all required protected attributes (personnel models)	☐	
5.5	Disparity thresholds applied; exceedances documented and remediated or accepted	☐	
5.6	Mission owner and G1 (personnel models) bias audit acceptance recorded	☐	
5.7	Bias audit integrated into automated retraining pipeline (if pipeline exists)	☐	

Section 6: Security

#	Requirement	Status	Notes
6.1	Adversarial threat level assessed per 7-3 table	☐	
6.2	Adversarial evaluation conducted for models at moderate or high threat level	☐	
6.3	Robustness degradation documented; adversarial hardening applied if > 15% threshold	☐	
6.4	Data poisoning detection implemented for external/partner data feeds	☐	
6.5	Model extraction defenses implemented per model sensitivity level	☐	
6.6	All dependencies version-pinned; no unapproved packages	☐	
6.7	Model artifact stored in MSS model registry with SHA-256 hash	☐	
6.8	Pre-trained checkpoint provenance and backdoor screening documented	☐	

Section 7: Platform and Governance

#	Requirement	Status	Notes
7.1	Features sourced from canonical shared feature store; no local feature recomputation	☐	

#	Requirement	Status	Notes
7.2	Training-serving consistency verified; identical feature Transform used for training and inference	☐	
7.3	Experiment run fully reproducible from registry (dataset version, feature version, seed, config)	☐	
7.4	Model registered in MSS model registry with all required fields complete	☐	
7.5	Model card complete per Appendix B; all RAIMTF-required sections present	☐	
7.6	Code review completed by SL 5M qualified reviewer for all production-scope code	☐	
7.7	SL 4M governance checklist also completed (this checklist does not replace it)	☐	
7.8	C2DAO deployment authorization obtained and logged	☐	

Section scores: - Section 1: ***/11 Section 2:*** /10 Section 3: ***/7 Section 4:*** /8 - Section 5: ***/7 Section 6:*** /8 Section 7: ___/8

Decision: All items checked (or N/A with justification) required before production deployment. Any unchecked item without documented N/A justification is a STOP condition.

SL 5M Senior MLE Signature: ____ Date: ____ C2DAO Authorization Reference: ____

APPENDIX B — MODEL DOCUMENTATION STANDARDS (DOD RAIMTF)

Purpose: This appendix defines the model card standard for all MSS ML models, aligned to DoD Responsible AI Implementation Toolkit and Framework (RAIMTF) requirements. A model card is a required governance artifact. No model may be promoted to production status in the MSS model registry without a complete model card. The model card is a living document — update it after every significant retraining event or governance review.

B-1. Model Card Template

MODEL CARD [Model Identifier] Version: [X.Y.Z] **Status:** [draft / evaluated / approved / production / deprecated] **Date:** [YYYYMMDD] **Responsible MLE:** [Name, Team, Contact] **Data Steward:** [Name, Team, Contact] **Mission Owner:** [Name, Unit, Contact] **C2DAO Authorization Reference:** [Reference number or memo citation]

SECTION 1: MODEL OVERVIEW

Field	Value
Model Name	
Use Case	[One-sentence description of the operational decision this model supports]
Prediction Task	[Classification / Regression / Ranking / Anomaly Detection]
Model Architecture	[e.g., XGBoost gradient boosting classifier, 300 trees; or BERT-base fine-tuned for 3-class classification]
Primary Consumer	[Who uses this model's output: Workshop dashboard, AIP Logic workflow, ORSA analysis]
Operational Consequence Level	[Low / Medium / High] — defined as: Low = informational; Medium = informs resource allocation; High = informs personnel decisions or force posture
DoD AI Risk Category	[Minimal / Low / Medium / High / Critical] per RAIMTF risk classification

SECTION 2: INTENDED USE

2-1. **Intended Use Cases.** [Describe the specific operational contexts for which this model is designed and validated. Be explicit about scope — what units, data sources, operational conditions, and decision types.]

2-2. **Out-of-Scope Use Cases.** [Explicitly list uses for which this model is NOT validated: different units, different operational conditions, different time horizons, different decision types. Unauthorized out-of-scope uses are the responsibility of the using unit — the MLE is responsible for documenting this boundary clearly.]

2-3. **Human-in-the-Loop Requirements.** [Describe the required human oversight for this model's outputs. Which outputs require human review before any action? What is the documented fallback if the model is unavailable?]

SECTION 3: TRAINING DATA

Field	Value
Training Dataset	[Foundry dataset path + version hash]

Field	Value
Data Sources	[List all upstream sources: GCSS-Army, IPPS-A, etc.]
Training Period	[Date range of training data]
Record Count	[Number of training records]
Feature Count	[Number of input features]
Known Biases in Training Data	[Document known data collection biases: reporting gaps, systemic missingness, historical policy artifacts]
Data Steward	[Name and contact]
Authorization	[Reference to data access authorization]

SECTION 4: MODEL PERFORMANCE

4-1. **Evaluation Dataset.** [Foundry dataset path + version hash. Must be distinct from training data. Describe how the evaluation set was constructed and why it is representative of the production population.]

4-2. **Primary Metric.** [Metric name; rationale for selection; operational interpretation of the metric value]

4-3. **Performance Summary.**

Metric	Value	Notes
Primary metric		
[Secondary metric 1]		
[Secondary metric 2]		
[Additional metrics as applicable]		

4-4. **Baseline Comparison.** [How does this model compare to the pre-existing method (rule-based system, previous model, human judgment baseline)? If no baseline exists, document that explicitly.]

4-5. **Known Performance Limitations.** [Where does the model perform below acceptable thresholds? Are there specific data regimes, operational conditions, or unit types where performance is degraded? Document all known limitations.]

SECTION 5: BIAS AND FAIRNESS (required for personnel models; recommended for all models)

5-1. **Protected Attributes Evaluated.** [List all protected attributes evaluated per Section 5-5 of SL 5M.]

5-2. **Disaggregated Performance.**

Group	Primary Metric	Disparity vs. Overall	Status
[Group 1]			[Within threshold / Flagged]

Group	Primary Metric	Disparity vs. Overall	Status
[Group 2]			[Within threshold / Flagged]
[Continue for all groups]			

5-3. **Remediation Applied.** [If any disparity exceeded threshold: describe the remediation implemented, the metric change before/after remediation, and any residual disparity with mission owner acceptance documentation.]

5-4. **Mission Owner Acceptance.** [Reference to signed bias audit acceptance form. For personnel models: G1 coordination reference.]

SECTION 6: ADVERSARIAL ROBUSTNESS (required for moderate/high threat level models)

Attack Type	Robustness Degradation	Status	Hardening Applied
[Attack type 1]	[% degradation]	[Pass / Fail]	[None / Adversarial training / Input validation]
[Attack type 2]			

6-1. **Residual Risk.** [After hardening: what adversarial attack scenarios remain a concern? What is the operational impact if a successful adversarial attack occurs?]

SECTION 7: EXPLAINABILITY

7-1. **Explanation Method.** [SHAP TreeExplainer / SHAP DeepExplainer / LIME / Other]

7-2. **Explanation Availability.** [Per-prediction explanations available in model output dataset: Yes / No. If yes: output dataset path.]

7-3. **Top Features.** [List the 5–10 features with highest mean |SHAP| value. For each: feature name, operational meaning, and directional interpretation (higher feature value → higher/lower prediction).]

| Feature | Mean |SHAP| | Operational Meaning | Direction | |---|---|---|---| | [Feature 1] | | | | [Feature 2] | | | | [Continue] | | | |

7-4. **Interpretation Guide Reference.** [Link to the interpretation guide for operational end-users.]

SECTION 8: OPERATIONAL REQUIREMENTS AND DEPLOYMENT

8-1. **Inference Architecture.** [Batch Transform / On-Demand Transform / Real-Time Endpoint]

8-2. **Refresh Cadence.** [How often are predictions refreshed? What event triggers a refresh?]

8-3. **Latency SLA.** [If real-time: P99 latency target and measured benchmark result.]

8-4. **Fallback Behavior.** [What happens when the model is unavailable? What is the fallback and who is notified?]

8-5. **Feature Dependencies.** [List all canonical feature datasets consumed; freshness SLA for each.]

8-6. **Downstream Consumers.** [List all Workshop applications, AIP Logic workflows, or Agent Studio agents that consume this model's output.]

SECTION 9: MAINTENANCE AND MONITORING

9-1. **Retraining Trigger.** [What triggers a retraining event? What drift metric and threshold?]

9-2. **Monitoring Configuration.** [What drift monitors are active? What alerting thresholds are configured?]

9-3. **Responsible MLE On-Call.** [Contact for production incidents.]

9-4. **Incident Response Reference.** [Link to the incident response procedure for this model.]

9-5. **Deprecation Plan.** [What conditions trigger model deprecation? What is the replacement plan?]

SECTION 10: GOVERNANCE AND ACCOUNTABILITY

Artifact	Reference	Date	Status
Use Case Authorization			
Design Review			
Evaluation Acceptance			
Bias Audit Acceptance		N/A if not applicable	
C2DAO Deployment Authorization			
RAIMTF Risk Assessment			
Army CIO Policy Compliance Review			

10-1. **DoD AI Ethics Principles Compliance.**

Principle	Compliance Description
Responsible	[How is human accountability maintained for this model's outputs?]
Equitable	[How is fairness ensured across groups? Reference bias audit.]
Traceable	[How can a specific prediction be traced back to its inputs and training data?]
Reliable	[How is model reliability measured and maintained in production?]
Governable	[How can this model be modified, constrained, or shut down if required?]

B-2. RAIMTF Risk Classification Guidance

DoD AI Risk Category	Definition	MSS Examples
Minimal	Informational outputs; no direct operational consequence; human decision not influenced	Summary statistics, trend visualizations

DoD AI Risk Category	Definition	MSS Examples
Low	Informs operational decisions; human in the loop; low consequence of error	Maintenance scheduling suggestions, logistics demand forecasts
Medium	Directly informs personnel or resource allocation decisions; moderate consequence of error	C-rating prediction, readiness scoring, training prioritization
High	Informs force posture, mission planning, or personnel decisions with significant consequence	Intelligence pattern recognition, threat anomaly detection
Critical	Directly informs lethal or near-lethal decisions, or decisions with strategic consequence	Not authorized for MSS production without SecArmy-level approval and rigorous red team

NOTE

USAREUR-AF has no currently authorized Critical-category AI systems on MSS. If a proposed use case is assessed at Critical risk, stop and escalate to C2DAO and USAREUR-AF CIO before any development work proceeds.

GLOSSARY

A/B Deployment — A model deployment pattern that simultaneously routes a defined fraction of inference traffic to a candidate model version and the remainder to the incumbent, enabling live performance comparison on production data.

Adversarial Training — A model hardening technique that augments the training dataset with adversarially perturbed examples, teaching the model to produce correct outputs on both clean and adversarial inputs.

Automated Retraining Pipeline — An MLOps construct that automates the full model lifecycle from drift detection trigger through training, evaluation, promotion gating, and deployment, with defined human oversight requirements.

Backdoor Attack — A supply chain attack in which a model checkpoint is modified to behave correctly on most inputs but produce specific attacker-desired outputs when a trigger pattern is present in the input.

Byzantine Robustness — The property of a federated learning aggregation algorithm that produces a correct result even when a fraction of participant nodes submit corrupted or malicious gradient updates.

Canary Release — A production deployment pattern that limits initial rollout of a new model version to a small, bounded subset of the production population to detect failures before full rollout.

C2DAO — Command and Control Data Architecture Office; the USAREUR-AF theater-level architecture authority for MSS. All ML governance approvals, automation tier authorizations, and production deployment authorizations route through C2DAO.

Calibration — The property of a probabilistic model where the predicted confidence scores match empirical frequencies — a model that predicts 70% confidence is correct approximately 70% of the time.

Data Poisoning — An attack in which malicious records are inserted into or existing records are modified in a training dataset to corrupt the resulting model's behavior.

Differential Privacy (DP) — A mathematical framework that provides provable bounds on the information that can be inferred about any individual training record from a model or its outputs, by adding calibrated noise during training.

Equalized Odds — A fairness metric requiring that a model has equal true positive rates and equal false positive rates across demographic groups.

Experiment Tracking — The systematic recording of all inputs, configurations, and outputs of a model training run, sufficient to reproduce the run and compare it to other runs.

FedAvg — Federated Averaging; the standard federated learning aggregation algorithm that averages participant weight updates, weighted by participant dataset size.

Feature Store — A centralized platform for storing, versioning, and serving ML features, ensuring consistency between features used in training and features used in production inference.

Graph Neural Network (GNN) — A class of neural network architectures that operate on graph-structured data by passing messages between connected nodes to produce node-level, edge-level, or graph-level predictions.

LIME — Local Interpretable Model-agnostic Explanations; a method that generates local explanations by fitting a simple interpretable model to the neighborhood of a specific prediction.

MLOps — Machine Learning Operations; the set of practices for reliably and efficiently deploying and maintaining ML models in production.

Model Card — A standardized documentation artifact that describes a model's intended use, training data, performance, biases, limitations, and governance status. Required for all MSS production models per DoD RAIMTF.

Model Extraction — An attack in which an adversary systematically queries a model endpoint to reconstruct a surrogate model that approximates the original model's behavior.

Population Stability Index (PSI) — A metric measuring the distributional shift between two datasets for a given feature. PSI < 0.1: stable; 0.1–0.25: moderate shift; > 0.25: significant shift.

Production Readiness Gate — An automated or human-reviewed checkpoint that a model candidate must pass before promotion to a more advanced deployment stage.

RAIMTF — Responsible AI Implementation Toolkit and Framework; the DoD framework for AI governance, risk assessment, and accountability documentation. Model cards on MSS follow RAIMTF structure.

Rollback Register — A mechanism in the automated deployment pipeline that records the previous production model version immediately before promotion, enabling rapid reversion if the new version fails in production.

SHAP — SHapley Additive exPlanations; a game-theory-based method for computing per-feature contribution to individual model predictions, providing consistent and locally accurate model explanations.

Training-Serving Skew — A performance gap between offline model evaluation and production model behavior caused by differences between the feature preprocessing logic used during training and the logic used during inference.

Transformer — A neural network architecture based on self-attention mechanisms, primarily used for sequence modeling tasks including natural language processing. On MSS: used for operational text classification and information extraction from SITREPs and maintenance records.

UDRA — Unified Data Reference Architecture; the Army enterprise data architecture standard (v1.1, February 2025) governing data domain ownership, federated data governance, and ML model classification requirements.

Quantization — A model compression technique that reduces the numerical precision of model weights (e.g., from 32-bit float to 8-bit integer), reducing model size and accelerating CPU inference at a small performance cost.

SL 5M — Advanced Machine Learning Engineering — Maven Smart System HEADQUARTERS, UNITED STATES ARMY EUROPE AND AFRICA Wiesbaden, Germany

DoD and Army Strategic References:

- **DoD Responsible AI Strategy & Implementation Pathway (June 2024 update)** — DoD framework for responsible AI development, testing, and fielding
- **DoD AI Cybersecurity Risk Management Guide (CDAO)** — Risk management guidance for AI systems in DoD environments
- **DoD Directive 3000.09, Autonomy in Weapon Systems (January 2023 update)** — Policy on autonomous and semi-autonomous functions in weapon systems