DRAFT — UNOFFICIAL — NOT FOR OPERATIONAL USE

TECHNICAL MANUAL

SL 5G

USAREUR-AF

TM-50G — ADVANCED OPERATIONS RESEARCH AND SYSTEMS ANALYSIS

Specialist Course Manual

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

DRAFT — NOT FOR OFFICIAL USE. FOR TRAINING PLANNING PURPOSES ONLY.

26 MARCH 2026

DRAFT — UNOFFICIAL — NOT FOR OPERATIONAL USE

TM-50G — ADVANCED OPERATIONS RESEARCH AND SYSTEMS ANALYSIS

Forward: SL 5G prepares senior ORSA practitioners to lead advanced analytical programs on MSS, produce campaign-level decision support, and build persistent OR capability for USAREUR-AF. This is not a survey of methods — it is a practitioner guide for analysts who will own the analytical function at theater level. **Prereqs:** SL 4G, Operations Research/Systems Analysis (required). Completion of, or concurrent enrollment in, a graduate-level OR/MS program or equivalent operational ORSA experience (18+ months) strongly recommended. For AI/ML integration topics, see SL 4H (AI Engineer) and SL 4M (ML Engineer); CONCEPTS_GUIDE_TM50G_ORSA_ADVANCED (read before this manual). *HQ USAREUR-AF · v1.0 · 2026 · DISTRIB: USG only · AUTH: C2DAO/UDRA v1.1*

WARNING: Advanced OR products briefed at the GO/SES level without uncertainty quantification, assumption documentation, and peer review may contribute to decisions with theater-level consequences. These standards are not optional at SL 5G. CAUTION: Agent-based models and stochastic optimization outputs are sensitive to parameter choices and boundary conditions. Small changes in calibration can produce qualitatively different recommendations. Validate all models against historical analogues before operational use. NOTE: This manual assumes mastery of all SL 4G content. Procedures here build directly on Code Workspace configuration, time series methods, linear programming, Monte Carlo simulation, and wargame data architectures covered in SL 4G. Review SL 4G before proceeding if proficiency is uncertain.

CHAPTER 1 — INTRODUCTION AND SCOPE

BLUF: SL 5G prepares senior ORSA practitioners to lead advanced analytical programs on MSS, produce campaign-level decision support, and build persistent OR capability for USAREUR-AF. This is not a survey of methods — it is a practitioner guide for analysts who will own the analytical function at theater level.

1-1. Advanced ORSA Manual

This manual provides advanced-level ORSA procedures for senior analysts, OR team leads, and section chiefs conducting operations research on the Maven Smart System (MSS) within USAREUR-AF and supporting EUCOM requirements. It extends SL 4G into methods, products, and leadership

responsibilities appropriate for senior ORSA practitioners operating at the operational and theater-strategic level.

SL 5G covers advanced optimization (nonlinear programming, multi-objective optimization, stochastic programming, and metaheuristic methods applicable to EUCOM/USAREUR-AF logistics, force allocation, and planning problems); agent-based modeling and simulation (ABMS) for complex adaptive operational systems — urban terrain, logistics networks, and adversary behavior; Bayesian analysis and causal inference (Bayesian networks, DAGs, causal models for operational analysis, and updating prior assessments with operational data); advanced wargame and campaign analysis (designing OR support frameworks for multi-echelon wargames, campaign assessment architectures, and theater-level campaign analysis); decision analysis under deep uncertainty (DAUX) including robustness analysis, scenario-based planning, exploratory modeling, and decision-making when probability distributions cannot be reliably specified; advanced forecasting (ensemble methods, hierarchical models, state space models for multi-echelon readiness and logistics forecasting); OR products for GO/SES audiences; and building persistent OR capability including data architecture for standing analytical environments, team standards, and program management for ORSA on MSS.

SL 5G does NOT cover foundation ORSA methods (regression, ARIMA, LP, Monte Carlo, wargame data collection) — see SL 4G; machine learning model development and deployment — see SL 4M; AI engineering and production pipeline architecture — see SL 4H; or Ontology design and Workshop application development — see SL 3.

1-2. Target Audience

Primary audience: ORSA-coded officers (FA49), senior warrant officers (W4–W5), and Civilian analysts (GS-13/14 or equivalent) with primary ORSA function in USAREUR-AF, EUCOM J5/J8, or subordinate unit analytical cells.

Secondary audience: S/G2 and J2 section chiefs who supervise ORSA cells; C2DAO senior analysts responsible for analytical product governance; senior program analysts (PM/PEO data offices) requiring advanced analytical methodology for Army modernization programs in EUCOM.

Prerequisites (all required before beginning SL 5G):

Prerequisite	Description	Verification
SL 4G complete	All chapters and task procedures complete	Team lead sign-off
18+ months operational ORSA experience	Demonstrated delivery of analytical products at O-4 or above level	Portfolio review
Graduate-level quantitative training	Masters-level statistics, OR/MS, or equivalent	Transcript or demonstrated equivalency
Python or R proficiency	Can independently build, validate, and document analytical models	Code review

Prerequisite	Description	Verification
MSS Code Workspace access	Provisioned and functional	MSS Administrator confirmation

NOTE

Analysts who do not meet all prerequisites should complete SL 4G and accumulate at least one full analytical cycle (readiness assessment, exercise support, or campaign analysis) before proceeding. SL 5G procedures applied without adequate foundation produce models that are technically sophisticated but operationally unreliable.

1-3. Competency Framework

SL 5G develops the following competencies. Use Table 1-1 to self-assess before beginning and again after completing each chapter.

Table 1-1. SL 5G Competency Framework

Competency	Chapter	Beginner Indicator	Practitioner Indicator	Expert Indicator
Nonlinear and stochastic optimization	2	Can formulate LP with constraints	Can formulate NLP; use scipy.optimize	Can design two-stage stochastic programs; apply metaheuristics
Multi-objective optimization	2	Knows single-objective formulations	Can generate Pareto frontier for 2 objectives	Can navigate 3+ objective tradeoff spaces; brief to decision-makers
Agent-based modeling	3	Understands ABMS concepts	Can implement basic Mesa ABMS	Can calibrate to operational data; run scenario ensembles
Bayesian inference	4	Understands Bayes theorem	Can build/query Bayesian network	Can design DAG for operational causal question; perform do-calculus
Causal inference	4	Understands correlation/causation distinction	Can apply regression discontinuity or DiD	Can design observational study with credible causal identification
Campaign analysis	5	Can collect and aggregate wargame data	Can produce multi-turn wargame analytics	Can lead OR framework for multi-echelon campaign assessment

Competency	Chapter	Beginner Indicator	Practitioner Indicator	Expert Indicator
DAUX methods	6	Understands scenario planning concept	Can run scenario analysis; stress test	Can apply XLRM/RDM frameworks; design adaptive strategy
Senior briefing products	7	Can produce GO-level slide	Can conduct uncertainty briefing	Can lead OR program review; peer review others' products
Persistent OR capability	8	Can build single analysis workspace	Can document and hand off analysis	Can architect standing analytical environment; lead ORSA team

1-4. How to Use This Manual

Each chapter provides: - **BLUF** — bottom-line summary of the chapter's content and operational relevance - **Concept section** — the analytical method or framework, explained at implementation depth - **Tasks** — step-by-step procedures in Conditions/Standards/Equipment/Procedure format - **Operational examples** — USAREUR-AF and EUCOM contexts: Grafenwöhr, Hohenfels, Baltic flank, Suwałki Gap, V Corps AOR - **Code** — Python (primary) or R (where standard in the domain) examples suitable for MSS Code Workspaces

Work chapters in order for initial training. Return to individual chapters as reference during operational application. Cross-references to SL 4G procedures are noted inline.

1-5. Relationship to Other TMs in the 40/50 Series

Table 1-2. TM Series Relationship

Manual	Focus	Audience	Key Intersection with SL 5G
SL 4G	Foundation ORSA	ORSA officers, analysts	All methods — direct prerequisite
SL 4H	AI Engineering	Data/AI engineers	Productionizing ORSA models to MSS pipelines
SL 4M	ML Engineering	ML practitioners	ML methods for classification/prediction in ORSA products
SL 4J	Program Management	PMs, APMs	OR support to program analysis and cost modeling
SL 5G (this)	Advanced ORSA	Senior ORSA, FA49 leads	—

Manual	Focus	Audience	Key Intersection with SL 5G
SL 5H	Advanced AI Engineering	Senior AI engineers	Evaluation methodology; productionizing complex ORSA models
SL 5M	Advanced ML Engineering	Senior MLEs	ML pipelines that feed ORSA analytical products
SL 5J	Advanced Program Manager	Senior PMs	Portfolio-level OR program governance
SL 5K	Advanced Knowledge Manager	Senior KMs	Capturing and surfacing ORSA-derived insights
SL 5L	Advanced Software Engineer	Senior SWEs	Platform infrastructure supporting ORSA pipelines
SL 5N	Advanced UI/UX Designer	Senior UX designers	Data visualization design; dashboard UX for ORSA outputs
SL 5O	Advanced Platform Engineer	Senior platform engineers	Infrastructure for analytical workloads; compute scaling

Where SL 5G methods intersect with AI/ML (e.g., using neural networks in ensemble forecasts, or ML classifiers in agent behavior models), cross-references to SL 4H and SL 4M are provided. ORSA analysts are not expected to independently deploy ML models to production — coordinate with SL 4H/M-trained engineers for production integration.

WFF Operational Consumer Note. Advanced ORSA products ultimately serve 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). WFF practitioners are the primary operational consumers of ORSA-derived readiness analyses, campaign assessments, logistics optimizations, and decision support products. When designing analytical products, consider the WFF audience: what decision does this analysis support, and which functional staff section will act on it?

CHAPTER 2 — ADVANCED OPTIMIZATION: NONLINEAR, MULTI-OBJECTIVE, AND STOCHASTIC

BLUF: Linear programming covers the majority of routine resource allocation problems. When objectives are nonlinear, when multiple competing objectives exist simultaneously, or when key parameters are uncertain, advanced optimization methods are required. This chapter covers nonlinear programming (NLP), multi-objective optimization (MOO), stochastic programming, and metaheuristic search — all in the context of USAREUR-AF operational planning problems.

2-1. From Linear to Nonlinear: When LP Is Insufficient

TM-40G Chapter 6 established linear programming as the standard tool for resource allocation. LP requires that both the objective function and all constraints be linear. Many operational problems violate this:

Nonlinear objective cases: - Readiness as a function of training days exhibits diminishing returns — the first ten days of gunnery training contribute more than days eleven through twenty - Logistics cost functions with economies of scale (bulk shipping rates) are concave, not linear - Force lethality as a function of combined arms integration is multiplicative, not additive

Nonlinear constraint cases: - Fuel consumption as a function of vehicle speed is quadratic - Communications range degrades nonlinearly with electronic warfare (EW) environment intensity - Personnel proficiency degrades nonlinearly with time since last qualification

Table 2-1. When to Use Which Optimization Method

Problem Characteristics	Method	MSS Tool
Linear objective, linear constraints	Linear programming (LP)	scipy.optimize.linprog, PuLP
Nonlinear objective or constraints, smooth	Nonlinear programming (NLP)	scipy.optimize.minimize
Multiple competing objectives	Multi-objective optimization (MOO)	pymoo, DEAP
Uncertain parameters, two decision stages	Two-stage stochastic programming	scipy, custom implementation
Large combinatorial search space	Metaheuristics (GA, PSO, SA)	DEAP, pyswarm, custom
Mixed-integer variables	MILP / branch and bound	PuLP with MILP, scipy

2-2. Nonlinear Programming on MSS

2-2a. Problem Formulation

Nonlinear programs take the general form:

```
minimize    f(x)
subject to  g_i(x) ≤ 0   for all i (inequality constraints)
            h_j(x) = 0   for all j (equality constraints)
            x ∈ X         (bounds)
```

where f, g_i, or h_j are nonlinear functions.

USAREUR-AF example — Training Resource Allocation with Diminishing Returns:

V Corps must allocate 1,200 training days across eight BCTs at Grafenwöhr and Hohenfels. Each BCT's readiness as a function of training days follows a log curve (diminishing returns). The objective is to maximize total theater readiness subject to range capacity and instructor constraints.

```
import numpy as np
from scipy.optimize import minimize

# BCT readiness parameters (calibrated to historical C-rating data)
# R_i(t) = a_i * log(1 + t) — diminishing returns model
# a_i = unit-specific scaling factor from regression on historical data
a = np.array([0.42, 0.38, 0.45, 0.41, 0.39, 0.44, 0.37, 0.43])  # 8 BCTs

n_units = len(a)
total_days = 1200
range_capacity = 180     # Grafenwöhr + Hohenfels combined daily capacity
instructor_ratio = 0.08  # instructor-days per training-day

# Objective: maximize total readiness (minimize negative)
def neg_readiness(t):
    return -np.sum(a * np.log(1 + t))

# Constraints
constraints = [
    # Total training days constraint
    {'type': 'eq', 'fun': lambda t: np.sum(t) - total_days},
    # Range capacity: no more than range_capacity days per unit per year
    # (expressed as inequality constraints per unit — handled by bounds below)
]

# Bounds: each BCT gets between 50 and 180 days
bounds = [(50, range_capacity) for _ in range(n_units)]

# Initial guess: equal allocation
t0 = np.full(n_units, total_days / n_units)

result = minimize(
    neg_readiness,
    t0,
    method='SLSQP',        # Sequential Least Squares Programming — handles
constraints
    bounds=bounds,
    constraints=constraints,
    options={'ftol': 1e-9, 'maxiter': 500}
)

if result.success:
    optimal_days = result.x
    optimal_readiness = -result.fun
    print(f"Optimal allocation (days): {np.round(optimal_days, 1)}")
    print(f"Maximum achievable readiness score: {optimal_readiness:.3f}")
    print(f"Readiness gain vs. equal allocation: "
          f"{(optimal_readiness - (-neg_readiness(t0))):.3f}")
else:
```

```
    print(f"Optimization did not converge: {result.message}")
    # DO NOT proceed with unconverged results — re-examine problem formulation
```

CAUTION: Always check result.success before using optimization outputs. An unconverged result is mathematically undefined. Report convergence status alongside results in all products.

NOTE: SLSQP (Sequential Least Squares Programming) is the standard method for smooth NLP with equality and inequality constraints. For problems with integer variables, use mixed-integer methods. For non-smooth objectives (e.g., max-min), consider metaheuristics.

2-2b. Sensitivity Analysis for NLP Results

Nonlinear programs require sensitivity analysis. Unlike LP, there is no built-in simplex sensitivity report. Construct sensitivity by perturbing constraints:

```
def sensitivity_analysis(constraint_value, perturbation_range=0.1):
    """
    Vary the total training day budget by +/- perturbation_range
    and report how optimal readiness changes.
    Returns a table suitable for commander briefing.
    """
    results = []
    base = constraint_value
    for delta in np.linspace(-perturbation_range, perturbation_range, 21):
        perturbed_total = base * (1 + delta)
        # Re-run optimization with perturbed constraint
        constraints_p = [{'type': 'eq', 'fun': lambda t, p=perturbed_total: np.sum(t)
- p}]
        t0_p = np.full(n_units, perturbed_total / n_units)
        r = minimize(neg_readiness, t0_p, method='SLSQP',
                     bounds=bounds, constraints=constraints_p)
        if r.success:
            results.append({'budget_change_pct': delta * 100,
                            'training_days': perturbed_total,
                            'readiness': -r.fun})
    return results

# Shadow price approximation: how much does one additional training day buy?
# Use finite difference on the optimal
```

2-3. Multi-Objective Optimization

2-3a. The Pareto Frontier

Most operational planning problems have multiple, competing objectives. Theater logistics optimization simultaneously minimizes cost, maximizes availability, and minimizes exposure of logistics nodes to threat. You cannot optimize all three simultaneously — tradeoffs are real. Multi-objective optimization (MOO) makes those tradeoffs explicit and rigorous.

The **Pareto frontier** is the set of solutions where no objective can be improved without degrading another. A solution is **Pareto-optimal** if it lies on this frontier. The commander's role is to choose among Pareto-optimal solutions based on their priorities — not to ask the analyst to pick for them.

2-3b. Generating the Pareto Frontier: Weighted Sum and Epsilon-Constraint Methods

Method 1 — Weighted Sum: Combine objectives into a single scalar using weights. Vary weights to trace the frontier.

Method 2 — Epsilon-Constraint (recommended): Fix all objectives except one at constraint bounds (epsilon values), optimize the remaining objective. Systematically vary epsilon values. Produces better coverage of non-convex frontiers.

```
import pandas as pd
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
from scipy.optimize import minimize

# USAREUR-AF example: Baltic Flank logistics pre-positioning
# Two objectives:
#   1. Minimize cost (pre-position supplies at forward nodes vs. main support area)
#   2. Maximize readiness (supplies available within 24hr of need)
# Decision variable: fraction of Class III/V pre-positioned at forward nodes (Suwalki,
Elblag)

# Cost model (linear for illustration; can substitute nonlinear)
def logistics_cost(x):
    # x[i] = fraction pre-positioned at forward node i
    # forward pre-position cost is higher (transport + security)
    forward_cost_rate = np.array([1.8, 2.1, 1.6])  # relative cost vs. MSA storage
    msa_cost_rate = np.ones(3)
    return np.dot(x, forward_cost_rate) + np.dot(1 - x, msa_cost_rate)

def readiness_score(x):
    # Higher pre-positioning = faster availability = higher readiness
    # Diminishing returns: logistics readiness caps out
    return -np.sum(0.95 * (1 - np.exp(-2 * x)))  # negative for minimization

# Epsilon-constraint: fix readiness at epsilon, minimize cost
def pareto_epsilon_constraint(epsilon_values):
    frontier = []
    bounds_x = [(0, 1)] * 3  # fractions between 0 and 1

    for eps in epsilon_values:
        constraints = [
            # Readiness must meet epsilon threshold (readiness_score returns negative)
            {'type': 'ineq', 'fun': lambda x, e=eps: -readiness_score(x) - e}
        ]
        result = minimize(
            logistics_cost,
            x0=np.array([0.5, 0.5, 0.5]),
            method='SLSQP',
```

```
            bounds=bounds_x,
            constraints=constraints
        )
        if result.success:
            frontier.append({
                'cost': result.fun,
                'readiness': -readiness_score(result.x),
                'allocation': result.x
            })
    return pd.DataFrame(frontier).drop_duplicates().sort_values('readiness')

# Generate frontier
epsilons = np.linspace(0.1, 2.5, 30)
pareto_df = pareto_epsilon_constraint(epsilons)

# Plot for commander briefing (saved as PNG — headless environment)
fig, ax = plt.subplots(figsize=(9, 6))
ax.plot(pareto_df['readiness'], pareto_df['cost'], 'b-o', markersize=5)
ax.set_xlabel('Logistics Readiness Score (higher = better)')
ax.set_ylabel('Pre-Positioning Cost (relative units)')
ax.set_title('Baltic Flank Pre-Positioning: Cost vs. Readiness Tradeoff\n'
             'USAREUR-AF V Corps — Pareto Frontier Analysis')
ax.axvline(x=2.0, color='red', linestyle='--', alpha=0.6, label='Minimum readiness
threshold (CO decision)')
ax.legend()
ax.grid(True, alpha=0.3)
fig.tight_layout()
fig.savefig('baltic_pareto_frontier.png', dpi=150, bbox_inches='tight')
plt.close(fig)
```

NOTE: Present the Pareto frontier to commanders as a decision visualization, not a recommendation. Your role is to show the tradeoff curve and clearly mark where the theater minimum requirements (e.g., 24-hour Class V availability) fall on that curve. The commander selects the operating point.

2-3c. Many-Objective Problems (3+ Objectives)

When problems have three or more objectives, 2D Pareto frontier visualization becomes insufficient. Use:

- **Parallel coordinates plots** — each axis represents one objective; each line represents one Pareto solution
- **Radar/spider charts** — compare solutions across objectives
- **Pymoo library** — implements NSGA-II, NSGA-III, and MOEA/D for evolutionary multi-objective optimization

For USAREUR-AF planning problems with 4+ objectives (cost, readiness, risk, sustainability, political considerations), coordinate with C2DAO on approved visualization tools before briefing to senior leaders.

2-4. Stochastic Programming

2-4a. Concept: Decision-Making Under Uncertainty with Recourse

Linear and nonlinear programs assume parameters are known. Stochastic programs model situations where key parameters (demand, threat timing, weather) are uncertain, and decisions must be made before uncertainty resolves — but with the opportunity to adjust after.

The canonical **two-stage stochastic program:** - **Stage 1 (here and now):** Make decisions before uncertainty resolves (pre-position supplies, allocate forces) - **Stage 2 (wait and see):** Make recourse decisions after uncertainty resolves (adjust distribution, reallocate) - **Objective:** Minimize expected total cost across Stage 1 decisions + expected Stage 2 recourse costs

2-4b. Task: Build a Two-Stage Stochastic Program for Theater Logistics

CONDITIONS: You are supporting V Corps G4 planning for a Suwałki Gap reinforcement scenario. Class V (ammunition) demand depends on whether conflict escalates (three scenarios: LOW, MED, HIGH). Pre-positioning decisions must be made now; distribution adjustments can be made after the scenario materializes.

STANDARDS: Stochastic program formulated with at least three scenarios, probability weights sum to 1.0, Stage 1 and Stage 2 decisions clearly separated, expected value of perfect information (EVPI) calculated and reported.

EQUIPMENT: MSS Code Workspace, Python with scipy, numpy, pandas.

PROCEDURE:

1. Define scenarios and probabilities:

```
import numpy as np
import pandas as pd
from scipy.optimize import linprog

# Three demand scenarios for Class V (tons/day) — Suwałki corridor
scenarios = {
    'LOW':  {'prob': 0.50, 'demand': 420},   # steady-state competition
    'MED':  {'prob': 0.35, 'demand': 780},   # elevated tension / limited conflict
    'HIGH': {'prob': 0.15, 'demand': 1400},  # active conflict
}

# Pre-positioning sites
sites = ['Elblag', 'Suwalki_North', 'Augustow']
site_capacity = [600, 400, 350]      # tons capacity
pre_pos_cost = [1.2, 1.8, 1.5]      # cost per ton pre-positioned (relative)
emergency_cost = [4.5, 6.0, 5.2]    # cost per ton emergency delivered (recourse cost)

n_sites = len(sites)
n_scenarios = len(scenarios)
```

1. Formulate and solve the extensive form:

```
# Extensive form: explicitly enumerate all scenarios
# Decision variables: x_i (pre-position at site i, Stage 1)
#                     y_is (emergency delivery at site i in scenario s, Stage 2)
# Minimize: sum_i(cost_i * x_i) + sum_s(prob_s * sum_i(emerg_cost_is * y_is))

# Build LP coefficient vectors
# Variable order: [x_0, x_1, x_2, y_0_LOW, y_1_LOW, y_2_LOW, y_0_MED, ..., y_2_HIGH]
n_vars = n_sites + n_sites * n_scenarios  # 3 stage-1 + 9 stage-2

c = (
    pre_pos_cost +                          # Stage 1 costs
    [p * ec for s, sv in scenarios.items()  # Stage 2 expected costs
     for p, ec in [(sv['prob'], ci) for ci in emergency_cost]]
)

# Constraints: for each scenario, demand must be met (pre-pos + emergency >= demand)
A_ub = []
b_ub = []
for s_idx, (s_name, s_data) in enumerate(scenarios.items()):
    demand = s_data['demand']
    # -sum_i(x_i) - sum_i(y_is) <= -demand  (i.e., supply >= demand)
    row = [0.0] * n_vars
    for i in range(n_sites):
        row[i] = -1.0                               # pre-positioned
        row[n_sites + s_idx * n_sites + i] = -1.0  # emergency recourse
    A_ub.append(row)
    b_ub.append(-demand)

# Site capacity constraints (Stage 1)
for i in range(n_sites):
    row = [0.0] * n_vars
    row[i] = 1.0
    A_ub.append(row)
    b_ub.append(site_capacity[i])

# Bounds: all variables >= 0
bounds = [(0, None)] * n_vars

result = linprog(c, A_ub=A_ub, b_ub=b_ub, bounds=bounds, method='highs')

if result.success:
    x_opt = result.x[:n_sites]
    y_opt = result.x[n_sites:].reshape(n_scenarios, n_sites)

    print("Stage 1 Pre-positioning (tons):")
    for i, site in enumerate(sites):
        print(f"  {site}: {x_opt[i]:.1f} tons")

    print(f"\nExpected total cost: {result.fun:.2f} (relative units)")

    print("\nStage 2 Emergency Delivery by Scenario:")
    for s_idx, (s_name, _) in enumerate(scenarios.items()):
        print(f"  {s_name}: {y_opt[s_idx]}")
```

1. Calculate Expected Value of Perfect Information (EVPI):

```
# EVPI = EV(wait and see) - EV(here and now stochastic)
# Wait-and-see: optimal solution for each scenario independently, then take
expectation

ev_ws = 0.0
for s_idx, (s_name, s_data) in enumerate(scenarios.items()):
    demand = s_data['demand']
    # Single-scenario LP: pre-position optimally knowing the scenario
    c_ws = pre_pos_cost + emergency_cost
    A_ws = [[*([-1.0]*n_sites), *([-1.0]*n_sites)]]
    b_ws = [-demand]
    bounds_ws = [(0, cap) for cap in site_capacity] + [(0, None)] * n_sites
    r_ws = linprog(c_ws, A_ub=A_ws, b_ub=b_ws, bounds=bounds_ws, method='highs')
    if r_ws.success:
        ev_ws += s_data['prob'] * r_ws.fun

evpi = ev_ws - result.fun
print(f"\nExpected Value of Perfect Information (EVPI): {evpi:.2f}")
print(f"Interpretation: Perfect intelligence on escalation scenario is worth "
      f"up to {evpi:.2f} relative cost units in logistics efficiency.")
```

NOTE: EVPI quantifies the value of intelligence. Present this to the G2 as a concrete number: "Perfect intelligence on threat escalation is worth approximately X in logistics cost savings." This connects OR analysis directly to ISR resource justification.

2-5. Metaheuristic Methods

For combinatorial problems where LP/NLP formulations are impractical (complex combinatorial structure, non-differentiable objectives, massive search spaces), metaheuristic methods provide high-quality approximate solutions.

Table 2-2. Metaheuristic Methods for ORSA

Method	Mechanism	When to Use	Python Library
Genetic Algorithm (GA)	Evolution — selection, crossover, mutation	Combinatorial, multi-objective, complex constraints	DEAP, pymoo
Simulated Annealing (SA)	Probabilistic hill-climbing with cooling	Single-objective combinatorial, traveling salesman type	scipy.optimize, simanneal
Particle Swarm Optimization (PSO)	Swarm intelligence, continuous search	Continuous nonlinear, multimodal landscapes	pyswarm, PySwarms
Tabu Search	Memory-guided local search	Scheduling, routing, when local optima are traps	custom implementation

USAREUR-AF application — Force Packaging Optimization:

Assigning brigade-level enablers (aviation, engineer, fires, EW) to operational tasks at Grafenwöhr Exercise Area for a rotational training event. The assignment problem has integer variables (enablers are discrete), multiple objectives (maximize training realism, minimize transit time, balance workload), and non-linear compatibility constraints (certain enabler combinations are incompatible). A genetic algorithm is appropriate.

For GA implementation on MSS, use the DEAP library within Code Workspaces. Consult SL 4H for library installation procedures if DEAP is not in the current approved package list.

2-6. Task: Multi-Objective Optimization for COA Comparison

CONDITIONS: You are supporting G3 during MDMP. Three COAs have been developed for a Suwałki corridor reinforcement operation. Staff has identified five decision criteria. You will formally structure the COA comparison as a multi-objective optimization problem to provide rigorous, defensible analysis.

STANDARDS: COA comparison matrix produced with criteria weights, sensitivity analysis on weights, Pareto analysis identifying dominant and dominated COAs, one-page BLUF product suitable for commander's brief.

EQUIPMENT: MSS Code Workspace, Python, Excel/Contour for visualization.

PROCEDURE:

1. Elicit and document criteria from the staff:

```
# COA Comparison Framework — Suwałki Reinforcement
# Five criteria (from staff elicitation — G3, G4, G2, FSE)
criteria = [
    'Speed to Objective (hrs)',           # minimize
    'Logistics Sustainability (days)',    # maximize
    'Force Protection Exposure',          # minimize (composite score 0-10)
    'Alliance Interoperability Score',    # maximize (0-10, NATO coordination)
    'Strategic Ambiguity Preserved'       # maximize (0-10, escalation control)
]

# COA performance scores (from staff estimates, wargame, or subject matter experts)
# Rows = COAs, Columns = criteria
coa_scores = pd.DataFrame({
    'COA_1_AirAssault':     [18, 7, 6.2, 7.5, 4.0],
    'COA_2_GroundThrust':   [36, 14, 4.1, 8.5, 6.5],
    'COA_3_Hybrid':         [26, 11, 5.0, 9.0, 7.5],
}, index=criteria).T

# Direction of optimization (1 = maximize, -1 = minimize)
direction = [-1, 1, -1, 1, 1]
```

1. Apply normalization and weighted scoring with sensitivity analysis:

```
from itertools import product as iterproduct
```

```
def normalize_coa_matrix(df, direction):
    """Normalize each criterion to [0,1] range, adjusted for direction."""
    normalized = df.copy().astype(float)
    for i, (col, d) in enumerate(zip(df.columns, direction)):
        col_min, col_max = df[col].min(), df[col].max()
        if col_max == col_min:
            normalized[col] = 0.5
        elif d == 1:  # maximize
            normalized[col] = (df[col] - col_min) / (col_max - col_min)
        else:  # minimize
            normalized[col] = (col_max - df[col]) / (col_max - col_min)
    return normalized

norm_scores = normalize_coa_matrix(coa_scores, direction)

# Weight sensitivity: vary weights across plausible range
# G3 nominal weights (from MDMP criteria weighting step)
nominal_weights = np.array([0.25, 0.20, 0.20, 0.20, 0.15])

def weighted_score(normalized_df, weights):
    return normalized_df.values @ weights

# Sensitivity: Monte Carlo weight perturbation
np.random.seed(42)
n_trials = 10000
weight_samples = np.random.dirichlet(alpha=nominal_weights * 10, size=n_trials)
coa_wins = {'COA_1_AirAssault': 0, 'COA_2_GroundThrust': 0, 'COA_3_Hybrid': 0}

for w in weight_samples:
    scores = weighted_score(norm_scores, w)
    winner = norm_scores.index[np.argmax(scores)]
    coa_wins[winner] += 1

print("COA Robustness to Weight Uncertainty (% of trials as top scorer):")
for coa, wins in coa_wins.items():
    print(f"  {coa}: {wins/n_trials*100:.1f}%")
```

1. Report results as a defensible analytical product (see Chapter 7 for briefing standards).

CHAPTER 3 — AGENT-BASED MODELING AND SIMULATION

BLUF: Agent-based modeling and simulation (ABMS) models complex adaptive systems as collections of autonomous agents following behavioral rules. ABMS is uniquely suited for modeling urban terrain operations, adversary adaptation, logistics network dynamics, and scenarios where emergent behavior matters. This chapter covers ABMS design, calibration, validation, and operational application on MSS.

3-1. When to Use Agent-Based Modeling

Traditional simulation (Monte Carlo, ARIMA, system dynamics) models aggregate behavior. ABMS models individual entity behavior and observes emergent collective patterns. Use ABMS when:

- **Heterogeneous actors matter:** Different units, civilian populations, and adversaries behave differently; aggregating them into a single distribution loses critical information
- **Adaptive behavior is central:** Adversaries who change tactics in response to BLUE actions; logistics networks that reroute around damaged nodes
- **Spatial or network structure matters:** Urban terrain, road networks, logistics routes where geography shapes outcomes
- **Emergent effects are the key question:** Asking not "what is average throughput?" but "under what conditions does the logistics network collapse?"

Table 3-1. ABMS vs. Other Simulation Methods

Characteristic	Monte Carlo	System Dynamics	Agent-Based (ABMS)
Unit of analysis	Aggregate distribution	State variables, flows	Individual agents
Adaptive behavior	No	No	Yes
Spatial structure	No	No	Yes
Emergent phenomena	No	Limited	Yes
Computational cost	Low	Low	Medium-High
Calibration complexity	Low	Medium	High
Best for ORSA	Risk analysis, uncertainty	Stocks/flows, logistics	Complex adaptive scenarios

3-2. ABMS Architecture: Agents, Environment, and Rules

An ABMS consists of:

1. **Agents:** Autonomous entities with state, behavioral rules, and the ability to act and interact. In military applications: units, vehicles, supply convoys, civilian actors, adversary cells.
2. **Environment:** The space in which agents operate. May be geographic (grid-based, network-based), abstract (market, information space), or hybrid.
3. **Rules:** Agent decision rules specifying how agents act given their state and environment. Rules may be simple (if-then logic), probabilistic, or learned (using ML — see SL 4M for integration).
4. **Emergence:** Collective patterns that arise from individual interactions and cannot be predicted from agent rules alone.

3-3. Mesa: Python ABMS Framework on MSS

Mesa is the standard Python ABMS library. It runs within Code Workspaces on MSS. For large-scale runs (10,000+ agents, 1,000+ time steps), coordinate with C2DAO on compute allocation.

3-3a. Task: Build a Logistics Network Resiliency ABMS

CONDITIONS: V Corps G4 needs to assess the resiliency of the Baltic flank logistics network under interdiction. The network has five distribution nodes (Elblag, Gdansk, Kaunas, Vilnius, Suwalki); convoys move supplies between nodes; adversary may interdict individual routes. You will model convoy routing behavior and assess throughput degradation under varying interdiction levels.

STANDARDS: ABMS runs successfully for at least 100 time steps; baseline throughput calibrated to known data; interdiction scenarios produce meaningful output differentials; results summarized in a data table suitable for G4 briefing.

EQUIPMENT: MSS Code Workspace, Python with Mesa library (confirm availability with MSS Administrator), pandas, matplotlib.

PROCEDURE:

1. Install and import Mesa:

```
# In Code Workspace — confirm mesa is in approved package list
# pip install mesa  (coordinate with MSS Admin if not pre-installed)

from mesa import Agent, Model
from mesa.time import RandomActivation
from mesa.datacollection import DataCollector
import numpy as np
import pandas as pd
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
import networkx as nx
```

1. Define the logistics network:

```
# Baltic flank logistics network — simplified graph representation
# Nodes: distribution points; Edges: routes with capacity and interdiction
vulnerability

def build_logistics_network():
    G = nx.DiGraph()
    nodes = ['Gdansk', 'Elblag', 'Kaunas', 'Vilnius', 'Suwalki_FWD']
    G.add_nodes_from(nodes)

    # Edges: (from, to, capacity_tons_per_step, interdiction_probability_base)
    edges = [
        ('Gdansk',    'Elblag',      300, 0.05),
        ('Gdansk',    'Kaunas',      250, 0.08),
        ('Elblag',    'Suwalki_FWD', 200, 0.15),  # exposed route
```

```
        ('Kaunas',    'Vilnius',     350, 0.04),
        ('Kaunas',    'Suwalki_FWD', 180, 0.18),  # exposed route
        ('Vilnius',   'Suwalki_FWD', 220, 0.12),
    ]
    for src, dst, cap, p_interdict in edges:
        G.add_edge(src, dst, capacity=cap, p_interdict=p_interdict, active=True)
    return G
```

1. Define convoy agent:

```
class ConvoyAgent(Agent):
    """
    Represents a logistics convoy attempting to move supplies
    from a source node to the forward distribution point.
    """
    def __init__(self, unique_id, model, source, destination, cargo_tons):
        super().__init__(unique_id, model)
        self.source = source
        self.destination = destination
        self.cargo_tons = cargo_tons
        self.current_node = source
        self.delivered = False
        self.interdicted = False
        self.route = None

    def find_route(self):
        """Find shortest active path. Return None if no path exists."""
        G = self.model.network
        # Only traverse active edges
        active_subgraph = nx.DiGraph(
            [(u, v, d) for u, v, d in G.edges(data=True) if d['active']]
        )
        try:
            path = nx.shortest_path(active_subgraph, self.current_node,
self.destination)
            return path
        except nx.NetworkXNoPath:
            return None

    def step(self):
        if self.delivered or self.interdicted:
            return

        route = self.find_route()
        if route is None:
            self.interdicted = True  # no available route — effectively interdicted
            return

        # Move one hop along route
        if len(route) > 1:
            next_node = route[1]
            edge_data = self.model.network.edges[self.current_node, next_node]

            # Check for interdiction on this step
```

```
            if np.random.random() < edge_data['p_interdict'] *
self.model.threat_level:
                self.interdicted = True
                self.model.total_interdicted += self.cargo_tons
                return

            self.current_node = next_node
            if self.current_node == self.destination:
                self.delivered = True
                self.model.total_delivered += self.cargo_tons
```

1. Define the model:

```
class LogisticsNetworkModel(Model):
    """
    Baltic flank logistics network resiliency model.
    Each step represents one operational period (e.g., 4-hour window).
    """
    def __init__(self, n_convoys_per_step=5, threat_level=1.0, n_steps=100):
        super().__init__()
        self.network = build_logistics_network()
        self.threat_level = threat_level  # multiplier on interdiction probabilities
        self.n_convoys_per_step = n_convoys_per_step
        self.schedule = RandomActivation(self)
        self.total_delivered = 0.0
        self.total_interdicted = 0.0
        self.step_count = 0
        self.convoy_id_counter = 0

        # Data collection: record throughput at each step
        self.datacollector = DataCollector(
            model_reporters={
                'TotalDelivered': 'total_delivered',
                'TotalInterdicted': 'total_interdicted',
                'ActiveConvoys': lambda m: len([a for a in m.schedule.agents
                                                if not a.delivered and not
a.interdicted])
            }
        )

    def spawn_convoys(self):
        """Spawn new convoys from rear nodes each step."""
        source_nodes = ['Gdansk', 'Kaunas']  # entry points from west
        for _ in range(self.n_convoys_per_step):
            src = np.random.choice(source_nodes)
            convoy = ConvoyAgent(
                unique_id=self.convoy_id_counter,
                model=self,
                source=src,
                destination='Suwalki_FWD',
                cargo_tons=np.random.uniform(40, 80)  # tons per convoy
            )
            self.convoy_id_counter += 1
            self.schedule.add(convoy)
```

```
    def step(self):
        self.spawn_convoys()
        self.schedule.step()
        self.datacollector.collect(self)
        self.step_count += 1

        # Remove completed and interdicted convoys to manage memory
        to_remove = [a for a in self.schedule.agents if a.delivered or a.interdicted]
        for agent in to_remove:
            self.schedule.remove(agent)
```

1. Run scenarios and compare:

```
def run_scenario(threat_level, n_steps=100, n_replications=30, n_convoys=5):
    """Run multiple replications for a given threat level. Return summary
statistics."""
    delivery_totals = []
    for rep in range(n_replications):
        np.random.seed(rep)  # reproducible replications
        model = LogisticsNetworkModel(
            n_convoys_per_step=n_convoys,
            threat_level=threat_level,
            n_steps=n_steps
        )
        for _ in range(n_steps):
            model.step()
        delivery_totals.append(model.total_delivered)
    return {
        'threat_level': threat_level,
        'mean_delivered': np.mean(delivery_totals),
        'std_delivered': np.std(delivery_totals),
        'p10_delivered': np.percentile(delivery_totals, 10),
        'p90_delivered': np.percentile(delivery_totals, 90)
    }

# Run three threat scenarios
scenarios = [
    ('LOW (Baseline)',       0.5),
    ('MED (Elevated)',       1.0),
    ('HIGH (Active Threat)', 2.0),
]

results = [run_scenario(tl, n_steps=100, n_replications=50) for _, tl in scenarios]
results_df = pd.DataFrame(results)
results_df.insert(0, 'Scenario', [s[0] for s in scenarios])
print(results_df.to_string(index=False))
```

CAUTION: ABMS results are highly sensitive to agent behavioral rules. Before briefing results to commanders, validate model behavior against historical data or subject matter expert assessment. Present the sensitivity of results to key rule assumptions.

3-4. Calibration and Validation of ABMS

Uncalibrated ABMS produces plausible-looking but unreliable outputs. Calibration standards for operational use:

Step 1 — Face Validity: Subject matter experts (G4 logistics officers, transportation NCOs) review agent behavioral rules and confirm they reflect real convoy routing behavior.

Step 2 — Historical Calibration: If historical data is available (exercise movement records, actual convoy data from Defender Europe or IRON WOLF exercises at Grafenwöhr/Hohenfels), calibrate model parameters to reproduce observed throughput distributions.

Step 3 — Extreme-Condition Validation: Verify model produces sensible results at extreme parameter values (threat_level = 0 should give near-perfect delivery; all routes interdicted should give zero delivery).

Step 4 — Sensitivity Analysis: Vary key parameters (p_interdict rates, cargo volumes, convoy count) and confirm results change in the expected direction and magnitude.

Document all four steps in the model description section of the ORSA product.

3-5. Advanced ABMS: Adversary Adaptation

For RED-BLUE scenario modeling, adversary agents must adapt to BLUE actions. This requires behavioral rules with memory and decision logic:

```
class AdversaryCell(Agent):
    """
    Adversary interdiction cell that adapts targeting based on
    observed BLUE logistics patterns.
    Simplified implementation — full version would incorporate
    intelligence-driven targeting rules.
    """
    def __init__(self, unique_id, model, initial_target_edge):
        super().__init__(unique_id, model)
        self.target_edge = initial_target_edge
        self.observation_window = []    # track which routes BLUE uses
        self.adaptation_threshold = 10  # steps before re-targeting

    def observe_blue_routes(self):
        """Record which edges are currently being used by BLUE convoys."""
        active_convoys = [a for a in self.model.schedule.agents
                          if isinstance(a, ConvoyAgent) and not a.delivered]
        for convoy in active_convoys:
            route = convoy.find_route()
            if route and len(route) > 1:
                edge = (route[0], route[1])
                self.observation_window.append(edge)

    def retarget(self):
        """Switch interdiction focus to the most-used active route."""
        if not self.observation_window:
            return
```

```
        from collections import Counter
        most_used = Counter(self.observation_window).most_common(1)[0][0]
        self.target_edge = most_used
        self.observation_window = []

    def step(self):
        self.observe_blue_routes()
        if len(self.observation_window) >= self.adaptation_threshold:
            self.retarget()
        # Apply interdiction to target edge
        if self.target_edge in self.model.network.edges:
            # Temporarily increase interdiction probability on targeted edge
            self.model.network.edges[self.target_edge]['p_interdict'] = min(
                0.8, self.model.network.edges[self.target_edge]['p_interdict'] * 1.5
            )
```

NOTE: Adversary adaptation models require careful subject matter expert review. Oversimplified adversary rules lead to either overconfidence (adversary is too predictable) or overpessimism (adversary is too capable). RED cell personnel should review adversary agent logic before use in senior-level products.

CHAPTER 4 — BAYESIAN ANALYSIS AND CAUSAL INFERENCE

BLUF: Bayesian methods allow analysts to formally incorporate prior knowledge and update assessments as new data arrives. Causal inference methods allow analysts to draw defensible conclusions about cause-and-effect relationships from observational data. Both are critical for operational analysis where controlled experiments are impossible and expert knowledge must be formally incorporated.

NOTE — Palantir Developers reference: *Deep Dive: Advanced Ontology* | *DevCon 5* — Covers advanced Ontology patterns including object type hierarchies, action types, and linking operational entities across complex data models. Relevant here because Bayesian and causal models in ORSA depend on a well-structured Ontology representing the relationships between units, readiness events, logistics flows, and operational outcomes — without this foundation, cross-domain causal analysis cannot be conducted systematically on MSS. Available on the Palantir Developers YouTube channel (@PalantirDevelopers).

4-1. Bayesian Framework for Operational Analysis

4-1a. Bayes Theorem in Operational Terms

Bayes theorem states:

```
P(H|E) = P(E|H) * P(H) / P(E)
```

In operational terms: - **P(H)** = prior probability: your assessment of a hypothesis before seeing new evidence - **P(E|H)** = likelihood: how probable the observed evidence is if the hypothesis is true - **P(H|E)** = posterior probability: updated assessment after incorporating evidence - **P(E)** = normalizing constant (total probability of observing the evidence)

Operational example: G2 assesses probability of adversary cross-border movement in the next 72 hours. Prior assessment based on historical pattern analysis: 15%. New ISR report shows increased vehicle traffic at suspected staging areas. Update the assessment formally.

4-1b. Bayesian Updating in Python

```
import numpy as np
import pandas as pd

# G2 Bayesian updating — force movement assessment
# Hypothesis H: adversary will conduct cross-border movement in 72hr
# Evidence E: ISR detects elevated vehicle traffic at staging areas

# Prior (from historical base rate + current pattern analysis)
p_h_prior = 0.15  # 15% prior probability

# Likelihoods (from G2 intelligence assessment, documented)
# P(elevated traffic | movement in 72hr) — how often is elevated traffic seen before
movement?
p_e_given_h = 0.75     # elevated traffic seen 75% of times prior to movement events

# P(elevated traffic | no movement) — how often is elevated traffic seen with no
movement?
p_e_given_not_h = 0.20 # elevated traffic occurs 20% of the time without movement
(logistics, exercises)

# Bayesian update
p_not_h_prior = 1 - p_h_prior
p_e = p_e_given_h * p_h_prior + p_e_given_not_h * p_not_h_prior

p_h_posterior = (p_e_given_h * p_h_prior) / p_e

print(f"Prior probability of movement: {p_h_prior:.1%}")
print(f"Posterior probability after ISR report: {p_h_posterior:.1%}")
print(f"Update factor (Bayes factor): {p_h_posterior/p_h_prior:.2f}x")

# Sensitivity: how does posterior vary with different likelihood assumptions?
print("\nSensitivity to P(E|H) assumption:")
for p_eh in [0.5, 0.6, 0.75, 0.85, 0.95]:
    p_e_s = p_eh * p_h_prior + p_e_given_not_h * p_not_h_prior
    p_post_s = (p_eh * p_h_prior) / p_e_s
    print(f"  P(E|H)={p_eh:.2f} → Posterior={p_post_s:.1%}")
```

4-2. Bayesian Networks

A Bayesian network (BN) is a directed acyclic graph (DAG) where nodes represent variables and edges represent conditional probabilistic dependencies. BNs compactly represent complex joint probability distributions and support both inference (given evidence, what are probabilities of unobserved variables?) and learning (given data, estimate network parameters).

4-2a. Building a Bayesian Network for Operational Assessment

Use case: USAREUR-AF readiness assessment. Unit readiness depends on equipment status, personnel fill, and training currency — but these are not independent. A BN formalizes these dependencies.

```
# Using pgmpy library for Bayesian networks
# Confirm pgmpy is in approved package list before using
# pip install pgmpy (coordinate with MSS Admin)

from pgmpy.models import BayesianNetwork
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination
import pandas as pd

# Network structure (DAG) — based on Army readiness doctrine (AR 220-1 analogue)
# Equipment Status → Unit Readiness
# Personnel Fill → Unit Readiness
# Training Currency → Unit Readiness
# Logistics Support → Equipment Status

model = BayesianNetwork([
    ('Logistics_Support', 'Equipment_Status'),
    ('Equipment_Status', 'Unit_Readiness'),
    ('Personnel_Fill', 'Unit_Readiness'),
    ('Training_Currency', 'Unit_Readiness')
])

# Conditional probability tables (CPTs)
# States: LOW (0), MED (1), HIGH (2) for all variables

# Logistics Support: unconditional (root node)
cpd_logistics = TabularCPD(
    variable='Logistics_Support',
    variable_card=3,
    values=[[0.20], [0.55], [0.25]]  # P(LOW), P(MED), P(HIGH)
)

# Equipment Status given Logistics Support
cpd_equipment = TabularCPD(
    variable='Equipment_Status',
    variable_card=3,
    values=[
        # P(Equip=LOW | Logistics=LOW, MED, HIGH)
        [0.60, 0.20, 0.05],
```

```
        # P(Equip=MED | Logistics=LOW, MED, HIGH)
        [0.30, 0.55, 0.30],
        # P(Equip=HIGH | Logistics=LOW, MED, HIGH)
        [0.10, 0.25, 0.65],
    ],
    evidence=['Logistics_Support'],
    evidence_card=[3]
)

# Personnel Fill: unconditional (root node)
cpd_personnel = TabularCPD(
    variable='Personnel_Fill',
    variable_card=3,
    values=[[0.15], [0.50], [0.35]]
)

# Training Currency: unconditional (root node)
cpd_training = TabularCPD(
    variable='Training_Currency',
    variable_card=3,
    values=[[0.25], [0.45], [0.30]]
)

# Unit Readiness given Equipment, Personnel, Training (27 combinations)
# This CPT is calibrated to historical C-rating data from USAREUR-AF units
# Values below are illustrative — calibrate to unit-specific data
import itertools

# Simplified: define readiness probability function
def readiness_prob(equip, pers, train):
    """
    Returns [P(LOW), P(MED), P(HIGH)] for unit readiness.
    Calibrate these weights to historical C-rating transitions.
    """
    score = (equip + pers + train) / 6.0  # normalized composite 0-1
    if score < 0.33:
        return [0.70, 0.25, 0.05]
    elif score < 0.67:
        return [0.15, 0.60, 0.25]
    else:
        return [0.05, 0.25, 0.70]

# Build CPT for Unit Readiness
readiness_values = [[], [], []]  # one list per readiness level
for equip, pers, train in itertools.product(range(3), range(3), range(3)):
    probs = readiness_prob(equip, pers, train)
    for i, p in enumerate(probs):
        readiness_values[i].append(p)

cpd_readiness = TabularCPD(
    variable='Unit_Readiness',
    variable_card=3,
    values=readiness_values,
    evidence=['Equipment_Status', 'Personnel_Fill', 'Training_Currency'],
    evidence_card=[3, 3, 3]
```

```
)

# Assemble and validate model
model.add_cpds(cpd_logistics, cpd_equipment, cpd_personnel,
               cpd_training, cpd_readiness)
assert model.check_model(), "BN model structure or CPTs are invalid"

# Inference
infer = VariableElimination(model)

# Query: given low logistics support, what is P(Unit_Readiness)?
result = infer.query(
    variables=['Unit_Readiness'],
    evidence={'Logistics_Support': 0}  # 0 = LOW
)
print("P(Unit Readiness | Logistics_Support=LOW):")
print(result)

# Query: what is the probability readiness is HIGH if we fix both equipment and
training to HIGH?
result2 = infer.query(
    variables=['Unit_Readiness'],
    evidence={'Equipment_Status': 2, 'Training_Currency': 2}
)
print("\nP(Unit Readiness | Equipment=HIGH, Training=HIGH):")
print(result2)
```

4-3. Causal Inference for Operational Analysis

4-3a. The Causal Inference Problem

Correlation is not causation. This is the central challenge for ORSA using observational data. Operational data is almost never produced by controlled experiments. When analysts observe that units with more training days have better readiness, they cannot directly conclude that more training causes better readiness — it may be that well-resourced units both receive more training and have better equipment and personnel, so training is confounded with resource availability.

Causal inference methods allow analysts to draw defensible causal conclusions from observational data by explicitly modeling confounding.

4-3b. Directed Acyclic Graphs for Causal Models

The first step in causal inference is drawing a causal DAG — explicitly stating your causal assumptions. A causal DAG makes confounding visible and allows formal identification of causal effects.

```
Training_Days → Readiness
Resource_Level → Training_Days
Resource_Level → Readiness
```

In this DAG, Resource_Level is a **confounder** — it affects both the treatment (Training_Days) and the outcome (Readiness). A naive regression of Readiness on Training_Days is biased because it does not account for this.

4-3c. Controlling for Confounding: Regression Adjustment

When all confounders are measured, regression adjustment recovers causal effects:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
import statsmodels.formula.api as smf

# Simulated USAREUR-AF readiness data
# In practice: pull from MSS readiness and training datasets
np.random.seed(2026)
n_units = 200

# Generate synthetic data with confounding
resource_level = np.random.normal(0, 1, n_units)      # high = well-resourced
training_days = 60 + 15 * resource_level + np.random.normal(0, 10, n_units)
# True causal effect of training: 1 day training → 0.8 readiness points
readiness = (50
             + 0.8 * training_days
             + 10 * resource_level        # confounding effect
             + np.random.normal(0, 5, n_units))

df = pd.DataFrame({
    'unit_id': range(n_units),
    'training_days': training_days,
    'resource_level': resource_level,
    'readiness': readiness
})

# Naive regression (biased — ignores confounding)
naive = smf.ols('readiness ~ training_days', data=df).fit()
print(f"Naive estimate of training effect: {naive.params['training_days']:.3f}")
print(f"  (True effect is 0.800 — bias from confounding: "
      f"{naive.params['training_days'] - 0.8:+.3f})")

# Adjusted regression (controls for confounder)
adjusted = smf.ols('readiness ~ training_days + resource_level', data=df).fit()
print(f"\nAdjusted estimate of training effect:
{adjusted.params['training_days']:.3f}")
print(f"  95% CI: [{adjusted.conf_int().loc['training_days', 0]:.3f}, "
      f"{adjusted.conf_int().loc['training_days', 1]:.3f}]")
print(f"  Bias after adjustment: {adjusted.params['training_days'] - 0.8:+.3f}")
```

4-3d. Difference-in-Differences for Policy Analysis

When a policy change or intervention affects some units but not others, difference-in-differences (DiD) estimates the causal effect by comparing the change in treated units to the change in control units:

```
# DiD Example: evaluating the effect of a new predictive maintenance
# program implemented at Grafenwöhr units (treatment) vs. Hohenfels units (control)

# Simulated pre/post data
np.random.seed(42)
n = 80

# Pre-period (before program)
readiness_pre_treatment = np.random.normal(72, 8, n//2)
readiness_pre_control   = np.random.normal(71, 8, n//2)

# Post-period (after program, 6 months later)
# True program effect: +6 readiness points
readiness_post_treatment = readiness_pre_treatment + 6 + np.random.normal(0, 4, n//2)
readiness_post_control   = readiness_pre_control + 1 + np.random.normal(0, 4, n//2)  #
trend only

did_df = pd.DataFrame({
    'readiness': np.concatenate([readiness_pre_treatment, readiness_post_treatment,
                                 readiness_pre_control, readiness_post_control]),
    'treated': [1]*(n//2) + [1]*(n//2) + [0]*(n//2) + [0]*(n//2),
    'post':    [0]*(n//2) + [1]*(n//2) + [0]*(n//2) + [1]*(n//2)
})

# DiD regression: coefficient on treated*post = causal effect estimate
did_model = smf.ols('readiness ~ treated + post + treated:post', data=did_df).fit()
did_estimate = did_model.params['treated:post']
did_ci = did_model.conf_int().loc['treated:post']

print(f"DiD estimate of program effect: {did_estimate:.2f} readiness points")
print(f"95% CI: [{did_ci[0]:.2f}, {did_ci[1]:.2f}]")
print(f"True effect: 5.0 | Estimated: {did_estimate:.2f}")
```

NOTE: DiD requires the parallel trends assumption — that treatment and control units would have followed the same trend absent the intervention. Validate this by examining pre-period trends. If pre-period trends differ, DiD estimates are biased. Report parallel trends validation in all DiD products.

4-3e. Do-Calculus and the Backdoor Criterion

For more complex causal structures, use Pearl's do-calculus. The **backdoor criterion** provides a sufficient condition for identifying causal effects from observational data: a set of variables Z satisfies the backdoor criterion for the effect of X on Y if: 1. Z blocks all backdoor paths from X to Y (paths with an arrow into X) 2. Z does not include any descendants of X

If Z satisfies the backdoor criterion, then:

```
P(Y | do(X)) = Σ_z P(Y | X, Z=z) * P(Z=z)
```

For implementation of complex causal identification, use the `dowhy` library (coordinate with MSS Admin for installation). For products involving causal claims from observational data, independent methodological review is required before briefing to senior leaders.

CHAPTER 5 — WARGAME AND CAMPAIGN ANALYSIS

BLUF: Senior ORSA practitioners lead OR support to major exercises and campaign assessment. This chapter covers the design of OR frameworks for multi-echelon wargames (Grafenwöhr, Hohenfels, SHAPE-level CPXs), campaign-level assessment architecture, and the analytical products that feed theater-strategic decision-making.

5-1. OR Support to MDMP: The Analytical Cell

At the operational and theater-strategic level, ORSA support to the Military Decision-Making Process (MDMP) is led by the senior analyst, not individual analysts. The analytical cell provides:

Table 5-1. ORSA Support Framework for MDMP

MDMP Step	ORSA Lead Action	Primary Product	Consumers
Receipt of Mission	Define analytical questions; inventory available data; scope models	Analytical Plan	G3, CoS
Mission Analysis	Force correlation modeling; baseline readiness assessment; threat parametrization	Mission Analysis Brief inputs	G2, G3
COA Development	Optimization model setup; COA feasibility screening	Pre-analysis constraints brief	G3, G4
COA Analysis (Wargame)	Real-time data collection; turn-by-turn aggregation; RED force analysis	Running wargame analytics dashboard	CDST, G3
COA Comparison	COA comparison matrix; sensitivity analysis; EVPI calculation	Decision support brief	CDR, G3
Orders Production	Support logistics annexes; resource requirement models	Logistics appendix inputs	G4, G1
Transition	Establish assessment metrics; design campaign dashboard architecture	Campaign assessment framework	G3, C2DAO

5-2. Designing OR Support to Multi-Echelon Wargames

5-2a. Tiered Data Collection Architecture

Multi-echelon wargames (corps, division, brigade simultaneous play) require a tiered data collection architecture that captures moves at each echelon without creating a data management burden for action officers running the wargame.

Tier 1 (Brigade): Capture unit moves, fires, and logistics expenditures in structured data collection forms. Wargame analysts embed with each brigade cell.

Tier 2 (Division): Aggregate brigade-level data plus division-level decisions (reserve commitment, reinforcement requests, fires allocation). Division ORSA analyst manages this aggregation.

Tier 3 (Corps/Theater): Theater logistics, enabler allocation, sustainment decisions. Senior ORSA analyst manages with MSS pipeline.

5-2b. Task: Design a Campaign Wargame Data Architecture on MSS

CONDITIONS: You are the senior ORSA analyst supporting a V Corps CPX simulating a 30-day Suwałki corridor reinforcement campaign. Four divisions and supporting enablers are playing. The wargame runs for 5 days, representing 30 game-turns (6 turns/day simulated). You must design the MSS data architecture to capture, aggregate, and analyze wargame data in near-real-time.

STANDARDS: Data schema designed and documented before wargame executes; data pipeline built and tested; three analytical products ready for delivery during wargame execution; post-wargame campaign analysis complete within 72 hours of wargame conclusion.

EQUIPMENT: MSS Code Workspace and Foundry Pipeline Builder; wargame control team coordination; G3/G4 staff access for data collection protocols.

PROCEDURE:

1. Design the data schema (pre-wargame, at least 2 weeks prior):

```
# Wargame data schema — define before event
# All data collected into MSS Foundry datasets via structured input forms

# Core schema for wargame turn data
wargame_schema = {
    'turn_data': {
        'turn_id': 'int',         # sequential turn number
        'sim_day': 'int',         # simulated campaign day (1-30)
        'sim_hour': 'int',        # simulated hour within day
        'collection_dtg': 'str',  # real-world collection timestamp (DTG format)
        'collector_id': 'str',    # analyst collecting this turn's data
    },
    'unit_status': {
        'turn_id': 'int',
        'unit_id': 'str',         # e.g., '1-1 ABCT', '3ID'
        'echelon': 'str',         # BDE, DIV, CORPS
```

```
        'grid': 'str',            # MGRS grid (6-digit precision)
        'c_rating': 'int',        # C1-C4
        'personnel_pct': 'float', # % fill
        'equipment_pct': 'float', # % operational
        'class_iii_pct': 'float', # fuel level
        'class_v_pct': 'float',   # ammo level
        'class_ix_pct': 'float',  # spare parts
        'mission_status': 'str',  # ATTACKING, DEFENDING, RESERVE, RECONSTITUTING
    },
    'fires_events': {
        'turn_id': 'int',
        'event_id': 'str',
        'firing_unit': 'str',
        'target_grid': 'str',
        'munition_type': 'str',   # DPICM, EXCALIBUR, ATACMS, etc.
        'rounds_fired': 'int',
        'assessed_effect': 'str', # SUPPRESSED, NEUTRALIZED, DESTROYED
        'ew_environment': 'str',  # CLEAN, DEGRADED, CONTESTED
    },
    'logistics_events': {
        'turn_id': 'int',
        'event_id': 'str',
        'event_type': 'str',      # RESUPPLY, MAINT_RECOVERY, MEDEVAC, etc.
        'origin_unit': 'str',
        'destination_unit': 'str',
        'class_type': 'str',
        'quantity': 'float',
        'success': 'bool',        # was the logistics event completed?
        'failure_reason': 'str',  # if not successful: reason
    }
}

# Document schema in the MSS project wiki before wargame begins
print("Wargame data schema defined.")
print(f"Tables: {list(wargame_schema.keys())}")
```

1. Build real-time aggregation pipeline (pre-wargame):

```
# Pipeline function for wargame analytics
# Runs in Code Workspace, updated each turn

def compute_turn_analytics(unit_status_df, fires_df, logistics_df, turn_id):
    """
    Compute key analytics for a single wargame turn.
    Returns a summary dict for real-time dashboard and turn record.
    """
    turn_data = unit_status_df[unit_status_df['turn_id'] == turn_id]
    fires_data = fires_df[fires_df['turn_id'] == turn_id]
    logistics_data = logistics_df[logistics_df['turn_id'] == turn_id]

    # Force status summary
    corps_c1_c2 = len(turn_data[turn_data['c_rating'].isin([1, 2])]) /
max(len(turn_data), 1)
    mean_class_iii = turn_data['class_iii_pct'].mean()
```

```
    mean_class_v = turn_data['class_v_pct'].mean()

    # Fires effectiveness
    fires_total = len(fires_data)
    fires_effective =
len(fires_data[fires_data['assessed_effect'].isin(['NEUTRALIZED', 'DESTROYED'])])
    fires_pct_effective = fires_effective / max(fires_total, 1)

    # Logistics success rate
    log_total = len(logistics_data)
    log_success = len(logistics_data[logistics_data['success'] == True])
    log_success_rate = log_success / max(log_total, 1)

    return {
        'turn_id': turn_id,
        'corps_c1_c2_pct': corps_c1_c2,
        'mean_class_iii_pct': mean_class_iii,
        'mean_class_v_pct': mean_class_v,
        'fires_effectiveness_pct': fires_pct_effective,
        'logistics_success_rate': log_success_rate,
        'units_reconstituting': len(turn_data[turn_data['mission_status'] ==
'RECONSTITUTING'])
    }
```

5-3. Campaign-Level Assessment Architecture

Campaign assessment differs from wargame analytics. A wargame runs for days and simulates a campaign; real campaign assessment runs for months or years alongside ongoing operations. Campaign assessment architecture on MSS must be:

- **Persistent:** Running continuously, not recreated for each assessment cycle
- **Automated:** Data ingestion and aggregation automated via scheduled transforms
- **Standardized:** Metrics defined before the campaign begins, not revised to show progress
- **Visible:** Accessible to multiple staff sections simultaneously via Workshop dashboards

5-3a. Lines of Effort and Measures of Effectiveness

Campaign assessment centers on Lines of Effort (LOEs) and their associated Measures of Effectiveness (MOEs) and Measures of Performance (MOPs).

Table 5-2. USAREUR-AF Campaign Assessment Framework

Line of Effort	Key MOE	Data Source on MSS	Assessment Frequency
Force Readiness	% BCTs at C1/C2	DRRS integration via ETL	Weekly

Line of Effort	Key MOE	Data Source on MSS	Assessment Frequency
Logistics Sustainment	Days of supply at forward nodes	Property book / logistics feeds	Daily
Training Effectiveness	Gunnery qualification rates; JRTC/NTC rotation outcomes	Training management system	Monthly
Alliance Posture	% NATO partner exercises conducted as planned	SHAPE exercise database	Quarterly
Baltic Flank Presence	Forward-positioned equipment SOR vs. requirement	Army Prepositioned Stocks data	Monthly

5-3b. Task: Build a Campaign Assessment Dashboard on MSS

CONDITIONS: You are designing the persistent campaign assessment architecture for USAREUR-AF's FY26 Campaign Plan. Five LOEs are defined. You will build the MSS data architecture and Workshop dashboard to support quarterly assessment briefs to the CG.

STANDARDS: All five LOEs have automated data pipelines into MSS; Workshop dashboard displays current status with trend lines; dashboard is accessible to G2, G3, G4, and C2DAO without requiring the ORSA analyst to manually update it; quarterly assessment report template is generated from dashboard data.

EQUIPMENT: MSS Foundry (Pipeline Builder, Workshop, Contour), Code Workspace for scheduled transforms, existing operational data feeds.

PROCEDURE:

1. Design the campaign metrics schema (collaborate with G3 and G5 before building)
2. Build ETL pipelines for each data source (see SL 3 and SL 4H for pipeline architecture)
3. Build Contour analysis workbooks for each LOE trend analysis
4. Build Workshop dashboard with LOE status cards, trend charts, and threshold alerts
5. Schedule Code Workspace transform for monthly statistical assessment generation
6. Document metric definitions in the MSS project wiki — definitions must not change during the campaign without CoS approval

CAUTION: Campaign metrics must be defined before the campaign begins and must not be changed to show progress. Retroactively changing metric definitions or thresholds undermines assessment integrity and violates analytical ethics. If metrics become irrelevant, document the rationale for change with CoS approval — do not quietly revise definitions.

5-4. RED Force Modeling

RED force effectiveness is the most uncertain element of campaign analysis. ORSA analysts support G2 with quantitative models for RED force capability, but must communicate uncertainty aggressively.

5-4a. Force Correlation Models

The force correlation model is the quantitative underpinning of COA comparison. It converts unit C-ratings, force structure, terrain factors, and doctrinal factors into a relative combat power ratio for RED vs. BLUE.

Lanchester equations provide the theoretical foundation: under Lanchester's Square Law, the effective combat power of a force is proportional to the square of its size weighted by effectiveness. For massed attrition combat, the force ratio that matters is not 1:1 but √(N_RED) / √(N_BLUE) weighted by effectiveness factors.

For modern, complex operational environments (urban terrain, A2/AD networks, non-contiguous battlespace), Lanchester equations are a starting point — not a complete model. Calibrate against historical data and subject matter expert assessment.

```
import numpy as np
import pandas as pd

def lanchester_square_ratio(n_red, n_blue, attrition_red, attrition_blue):
    """
    Lanchester Square Law force correlation.
    Returns probability BLUE wins using a logistic function of the combat power ratio.

    Parameters:
    - n_red, n_blue: unit counts (or weighted by lethality score)
    - attrition_red, attrition_blue: attrition rate coefficients
    - Returns: P(BLUE wins) based on combat power ratio
    """
    # Lanchester square law: b_power = attrition_red * n_blue^2
    #                        r_power = attrition_blue * n_red^2
    blue_power = attrition_red * (n_blue ** 2)
    red_power  = attrition_blue * (n_red ** 2)

    # Log ratio as input to logistic function for probability
    if red_power == 0:
        return 1.0
    log_ratio = np.log(blue_power / red_power)
    p_blue_wins = 1 / (1 + np.exp(-log_ratio))
    return p_blue_wins

# Suwałki scenario — illustrative
scenarios = [
    ('BLUE Advantage', 12, 8, 1.2, 1.0),
    ('Parity',         10, 10, 1.0, 1.0),
    ('RED Advantage',   8, 14, 0.9, 1.1),
]
```

```
print("Force Correlation Analysis — Suwałki Corridor")
print(f"{'Scenario':<20} {'BLUE N':>8} {'RED N':>8} {'P(BLUE wins)':>14}")
print("-" * 55)
for name, n_blue, n_red, a_red, a_blue in scenarios:
    p = lanchester_square_ratio(n_red, n_blue, a_red, a_blue)
    print(f"{name:<20} {n_blue:>8} {n_red:>8} {p:>14.1%}")
```

5-9. Advanced Operations Assessment on MSS (FM 5-0 Ch 8)

BLUF: SL 4G introduced MOE/MOP frameworks and CARVER scoring. SL 5G extends these into multi-variable sensitivity analysis, Bayesian updating, and longitudinal campaign assessment — the quantitative methods that transform static assessments into living decision support products.

5-9a. Multi-Variable Sensitivity Analysis for MOE/MOP

Static MOE/MOP assessment answers the question "are we achieving the effect?" Multi-variable sensitivity analysis answers the harder question: "which variables most influence whether we achieve the effect, and how confident are we in the assessment given uncertainty in each?"

The senior ORSA analyst conducts sensitivity analysis by:

1. **Identify assessment variables.** For each MOE, enumerate the MOPs and indicators that feed the assessment. Map the causal chain from indicator data through MOP aggregation to MOE determination.
2. **Vary inputs systematically.** Hold all inputs at baseline except one; vary that input across its plausible range. Record the resulting MOE change. Repeat for each input variable.
3. **Rank by influence.** Variables that produce the largest MOE swing across their plausible range are the assessment's critical drivers. These variables require the most rigorous data collection and the tightest update cadence.
4. **Present tornado diagrams.** Brief the Operations Assessment Working Group (OAWG) using tornado charts that show which variables drive the assessment and which are noise.

5-9b. Bayesian Updating of Assessments

Campaign assessments degrade when they rely solely on point-in-time snapshots. Bayesian updating provides a principled method for incorporating new data into existing assessments without discarding prior knowledge.

Apply Bayesian updating to operations assessment as follows:

1. **Establish prior.** Use the initial assessment (from wargame analysis, historical analogue, or subject-matter estimate) as the prior probability distribution for each MOE.
2. **Define likelihood function.** For each new data collection (indicator report, ISR collection, logistics status), define how the data relates to the MOE — what is the probability of observing this data if the effect is being achieved vs. not achieved?

3. **Update posterior.** Apply Bayes' theorem to produce the updated assessment. The posterior becomes the new prior for the next update cycle.
4. **Track convergence.** As data accumulates, the posterior should narrow. If it does not, the indicators may lack discriminating power — flag this to the OAWG for indicator redesign.

5-9c. Longitudinal Trend Analysis for Campaign Assessment

Campaign assessment requires tracking MOE/MOP trajectories over time to distinguish signal from noise. Use time-series methods (covered in SL 4G) applied to assessment data: decompose trends, identify inflection points, and forecast trajectory under current operational approach.

Produce longitudinal assessment products at the following cadences: weekly for active operations, bi-weekly for shaping operations, monthly for steady-state theater posture.

NOTE (FM 6-0): The ORSA produces the quantitative assessment products that feed the commander's assessment during the Operations Assessment Working Group. The OAWG is the primary venue where ORSA analysis meets command judgment. The ORSA does not make the assessment — the commander does. The ORSA provides the quantitative foundation, uncertainty bounds, and trend analysis that enable an informed assessment. Prepare products for the OAWG with this role distinction in mind: present evidence and uncertainty, not conclusions.

CHAPTER 6 — DECISION ANALYSIS UNDER DEEP UNCERTAINTY

BLUF: Deep uncertainty (DAUX) occurs when analysts cannot reliably specify probability distributions over future states of the world. DAUX methods — robustness analysis, scenario planning, and adaptive strategy — replace the quest for a single optimal plan with the search for strategies that perform adequately across a wide range of futures. This is the appropriate framework for most theater-strategic planning problems.

6-1. The Deep Uncertainty Problem

Classical optimization and probabilistic analysis assume you can specify: - The set of possible futures (the state space) - The probability of each future (probability distribution over states) - A utility function that correctly values outcomes

At the theater-strategic level, none of these assumptions reliably hold: - **Unknown unknowns** — the 2022 Russian ORBAT assumptions failed not because analysts used the wrong probabilities, but because the relevant factors (logistics dysfunction, command culture) were not in the state space at all - **Non-stationarity** — historical base rates for adversary behavior are poor predictors in novel geopolitical conditions - **Value complexity** — commanders may not be able to specify a single utility function across cost, casualties, political effects, and alliance cohesion

When to apply DAUX methods: - Planning horizons > 24 months - Scenarios involving novel adversary capabilities or new operational domains - Alliance commitments where partner behavior is uncertain - Multi-decade resource programming decisions (force structure, infrastructure)

6-2. XLRM Framework

The XLRM framework structures DAUX analysis:

- **X (Exogenous uncertainties):** Factors outside the planner's control — adversary actions, weather, political conditions, partner reliability
- **L (Levers):** Policy decisions available to the planner — force structure, pre-positioning levels, training programs, alliance arrangements
- **R (Relationships):** Models relating X and L to outcomes
- **M (Measures):** How to assess whether a strategy succeeded — MOEs, MOPs, thresholds

Table 6-1. XLRM for USAREUR-AF Baltic Posture Decision

Category	Variable	Range/Description
X — Uncertainty	Adversary modernization pace	Slow, nominal, accelerated
X — Uncertainty	NATO solidarity under Article 5	Full, partial (1-2 allies hesitant), fragmented
X — Uncertainty	US force structure availability	Current, -15%, -30% (INDOPACOM demand)
X — Uncertainty	Baltic state infrastructure capacity	As planned, delayed 2yr, partially failed
L — Lever	Prepositioned equipment sets	1, 2, or 3 brigade sets forward
L — Lever	Rotational presence level	Battalion, brigade, or division equivalent
L — Lever	Integrated air/missile defense posture	Current, enhanced, forward-deployed
R — Model	Force correlation at M-Day+3	Lanchester-based model (Chapter 5)
R — Model	Logistics sustainability at D+30	Stochastic logistics model (Chapter 2)
M — Measure	P(hold key terrain at D+30)	≥ 0.70 required
M — Measure	Logistics days-of-supply at D+30	≥ 5 days required

6-3. Robustness Analysis

Robustness analysis asks: which strategies perform adequately across the widest range of scenarios? Rather than optimizing for the expected case, robustness analysis identifies strategies that avoid catastrophic failure in bad scenarios while maintaining reasonable performance in good scenarios.

```
import numpy as np
import pandas as pd
```

```
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt

# Robustness analysis — USAREUR-AF Baltic posture
# Three strategies (levers), many uncertain scenarios (X combinations)

strategies = {
    'Strategy_A_Minimal':   {'prepos_sets': 1, 'presence': 'battalion', 'IAMD':
'current'},
    'Strategy_B_Moderate':  {'prepos_sets': 2, 'presence': 'brigade',   'IAMD':
'enhanced'},
    'Strategy_C_Forward':   {'prepos_sets': 3, 'presence': 'division',  'IAMD':
'forward'},
}

# Scenario parameters (simplified for illustration)
np.random.seed(2026)
n_scenarios = 500

scenarios_df = pd.DataFrame({
    'adv_modernization': np.random.choice(['slow', 'nominal', 'fast'],
                                          n_scenarios, p=[0.25, 0.50, 0.25]),
    'nato_solidarity':   np.random.choice(['full', 'partial', 'fragmented'],
                                          n_scenarios, p=[0.55, 0.35, 0.10]),
    'us_availability':   np.random.choice(['full', 'minus15', 'minus30'],
                                          n_scenarios, p=[0.50, 0.35, 0.15]),
})

def evaluate_strategy(strategy_params, scenario):
    """
    Simplified performance model.
    Returns P(hold terrain at D+30) and logistics_days.
    In practice: connect to full force correlation and logistics models.
    """
    # Base performance (higher prepos + presence = better)
    base_p = 0.40 + 0.15 * strategy_params['prepos_sets']
    if strategy_params['presence'] == 'brigade':
        base_p += 0.08
    elif strategy_params['presence'] == 'division':
        base_p += 0.18

    # Scenario modifiers
    if scenario['adv_modernization'] == 'fast':
        base_p -= 0.15
    elif scenario['adv_modernization'] == 'slow':
        base_p += 0.08

    if scenario['nato_solidarity'] == 'fragmented':
        base_p -= 0.20
    elif scenario['nato_solidarity'] == 'partial':
        base_p -= 0.08

    if scenario['us_availability'] == 'minus30':
        base_p -= 0.18
```

```
    elif scenario['us_availability'] == 'minus15':
        base_p -= 0.08

    # IAMD bonus
    if strategy_params['IAMD'] == 'forward':
        base_p += 0.06
    elif strategy_params['IAMD'] == 'enhanced':
        base_p += 0.03

    p_hold = np.clip(base_p + np.random.normal(0, 0.05), 0, 1)
    log_days = 3 + strategy_params['prepos_sets'] * 3.5 + np.random.normal(0, 1)
    return p_hold, max(0, log_days)

# Evaluate each strategy across all scenarios
results = []
for strat_name, strat_params in strategies.items():
    for _, scen in scenarios_df.iterrows():
        p_hold, log_days = evaluate_strategy(strat_params, scen)
        results.append({
            'strategy': strat_name,
            'p_hold': p_hold,
            'log_days': log_days,
            'meets_p_hold_threshold': p_hold >= 0.70,
            'meets_log_threshold': log_days >= 5.0,
            'meets_all_thresholds': (p_hold >= 0.70) and (log_days >= 5.0),
            **scen.to_dict()
        })

results_df = pd.DataFrame(results)

# Robustness: % of scenarios meeting all thresholds
robustness = (results_df.groupby('strategy')['meets_all_thresholds']
              .mean()
              .rename('robustness_score')
              .reset_index()
              .sort_values('robustness_score', ascending=False))

print("Strategy Robustness Analysis — Baltic Posture")
print("(% of 500 scenarios meeting P(hold)≥0.70 AND logistics≥5 days)")
print(robustness.to_string(index=False))

# Scenario-conditional robustness: where does each strategy fail?
print("\nRobustness by adversary modernization pace:")
print(results_df.groupby(['strategy', 'adv_modernization'])['meets_all_thresholds']
      .mean().unstack().round(3))
```

NOTE: Robustness analysis does not tell commanders which strategy is "best" — it tells them which strategies are most resilient to surprise. Present robustness analysis alongside expected-value analysis. Commanders may rationally prefer a lower-expected-value strategy if it is more robust to bad scenarios.

6-4. Scenario-Based Planning and Adaptive Strategy

When deep uncertainty makes point estimates unreliable, scenario-based planning replaces single forecasts with a structured set of plausible futures. The key principles:

Principle 1 — Scenarios are not forecasts. Scenarios represent the range of plausible futures, not predictions. A scenario is not assigned a probability in DAUX — all are treated as plausible.

Principle 2 — Scenarios should span key axes of uncertainty. Use the XLRM framework to identify the most consequential uncertainties. Design scenarios to span the range of those uncertainties.

Principle 3 — Adaptive strategy identifies trigger conditions. Rather than committing to a single plan, an adaptive strategy identifies decision points and trigger conditions: "If we observe [indicator], we will shift from Strategy B to Strategy C."

Table 6-2. Baltic Flank Scenario Set

Scenario	Narrative	Key Conditions
S1 — Constrained Competition	Long-term strategic competition; no escalation to conflict	NATO solidarity high; US availability constrained by INDOPACOM; slow adversary modernization
S2 — Threshold Crisis	Adversary hybrid operations; alliance tested; risk of escalation	NATO solidarity partially fractured; US availability at -15%; fast modernization
S3 — Rapid Escalation	Rapid conventional escalation; limited warning	NATO solidarity uncertain; US availability full but delayed; adversary takes initiative
S4 — Protracted Conflict	Prolonged conventional campaign; coalition management paramount	NATO solidarity stressed over time; sustainment dominates; modernization less relevant

For each scenario, assess strategy performance against MOEs. Design the adaptive strategy to specify which indicators trigger transitions between postures.

CHAPTER 7 — SENIOR-LEVEL OR PRODUCTS AND BRIEFINGS

BLUF: GO/SES-level ORSA products require different design choices than working-level analysis. Senior leaders need the finding, the confidence level, the key uncertainty, and the recommended decision — in that order, in two minutes. This chapter covers the structure, review process, and delivery of ORSA products for senior leader audiences.

7-1. Understanding the GO/SES Audience

General Officers and Senior Executive Service members face specific constraints that shape how ORSA products must be designed:

Time: A 4-star's analytical product review window is typically 2-5 minutes. A 2-star's is 5-10 minutes. Design products to deliver the critical finding in the first 30 seconds.

Abstraction level: Senior leaders need the operational implication, not the method. "Our model estimates with 80% confidence that logistics sustainability falls below the 5-day threshold by D+18 under the MED threat scenario" is the right level. "The ARIMA (2,1,0) model with seasonal differencing produced a RMSE of 4.3 on the holdout set" is not what they need in the executive brief.

Decision relevance: Every product must answer "so what" and "what do you recommend?" Senior leaders are not evaluating your analysis — they are making decisions. If your product does not specify a recommended commander action or decision point, it is incomplete.

Uncertainty: Senior leaders are experienced with uncertainty. Presenting false precision destroys credibility faster than acknowledging uncertainty. "Our estimate ranges from X to Y depending on adversary intent" is more credible than a single point that later proves wrong.

7-2. The OR Product Structure for Senior Audiences

Table 7-1. Standard OR Product Structure (GO/SES)

Slide / Section	Content	Length
1 — BLUF	Finding, confidence level, recommended decision	3-5 bullet points
2 — Decision Context	What question this answers; why it matters now	1 slide
3 — Key Finding	Main result with uncertainty range; visual (chart, map)	1-2 slides
4 — Sensitivity	How the finding changes under key assumptions; what would change the recommendation	1 slide
5 — Recommended Actions	Specific, actionable decisions with decision points	1 slide
Backup — Methodology	For technical challenge; not briefed unless asked	As needed
Backup — Data Quality	Data sources, limitations, recency	As needed

NOTE: The methodology goes in backup. If a senior leader asks "how did you derive this?", you need to answer fluently — but you don't brief the method in the main deck unless the method is itself the decision point (e.g., "should we trust this model to feed automated tasking?").

7-3. Uncertainty Communication Standards

7-3a. Language Standards

Use standardized probability language consistently. Table 7-2 translates probability ranges to verbal descriptors aligned with intelligence community standards (ODNI ICD 203 analogue adapted for ORSA).

Table 7-2. Probability Language Standards

Probability Range	Verbal Descriptor	Usage in Products
0-05%	Remote / Almost certainly not	Use sparingly; requires very strong evidence
06-15%	Unlikely / Low probability	
16-35%	Somewhat unlikely	
36-55%	Roughly even / Uncertain	Note: broad range — be explicit
56-70%	Likely / Probable	
71-85%	Highly likely / High confidence	
86-95%	Almost certain	
96-100%	Virtually certain	Use sparingly; implies near certainty

7-3b. Confidence Levels

Distinguish probability (likelihood of an event) from confidence (how much you trust the estimate):

- **High confidence:** Strong data, validated model, multiple corroborating sources
- **Moderate confidence:** Adequate data, reasonable assumptions, some validation
- **Low confidence:** Limited data, significant model uncertainty, assumptions not well-validated

Always state both: "We assess with **high confidence** that **there is a 72% probability** the logistics network will fall below threshold by D+18."

7-3c. Visualizing Uncertainty for Senior Leaders

Recommended charts for uncertainty communication:

- **Fan charts:** Forecast with confidence intervals widening over time — clear visual of uncertainty growth
- **Probability bars:** Horizontal bars showing probability ranges for discrete outcomes
- **Scenario performance tables:** Strategy × scenario matrices showing how performance varies
- **Tornado diagrams:** Horizontal bar chart showing which inputs drive the most output uncertainty (sensitivity analysis visualization)

```
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt
import numpy as np

def plot_tornado_diagram(base_case, sensitivities, output_metric, save_path):
    """
    Generate a tornado diagram for sensitivity analysis.

    Parameters:
    - base_case: float, the baseline model output
    - sensitivities: list of dicts with keys: 'variable', 'low_val', 'high_val'
    - output_metric: str, name of the output metric for labeling
    - save_path: str, output file path
    """
    # Sort by impact magnitude (largest bar at top)
    sensitivities = sorted(
        sensitivities,
        key=lambda x: abs(x['high_val'] - x['low_val']),
        reverse=True
    )

    fig, ax = plt.subplots(figsize=(10, max(4, len(sensitivities) * 0.7 + 1.5)))

    y_positions = range(len(sensitivities))

    for i, sens in enumerate(sensitivities):
        low_delta = sens['low_val'] - base_case
        high_delta = sens['high_val'] - base_case
        # Bar from low to high, centered at base_case
        bar_left = min(low_delta, high_delta)
        bar_width = abs(high_delta - low_delta)
        color = '#c0392b' if high_delta > 0 else '#2980b9'
        ax.barh(i, bar_width, left=base_case + bar_left,
                color='#2980b9', alpha=0.7, height=0.6)
        # Label bars
        ax.text(base_case + high_delta + 0.01 * abs(base_case),
                i, f"+{high_delta:.1f}", va='center', fontsize=9, color='#c0392b')
        ax.text(base_case + low_delta - 0.01 * abs(base_case),
                i, f"{low_delta:.1f}", va='center', ha='right', fontsize=9,
color='#27ae60')

    ax.set_yticks(list(y_positions))
    ax.set_yticklabels([s['variable'] for s in sensitivities], fontsize=10)
    ax.axvline(x=base_case, color='black', linewidth=1.5, linestyle='--')
    ax.set_xlabel(output_metric, fontsize=11)
    ax.set_title(f'Sensitivity Analysis — {output_metric}\n'
                 f'(Bars show output range when input varied ±1 std dev)',
                 fontsize=12)
    ax.grid(True, axis='x', alpha=0.3)
    fig.tight_layout()
    fig.savefig(save_path, dpi=150, bbox_inches='tight')
    plt.close(fig)

# Example usage
```

```
sensitivities_example = [
    {'variable': 'Class V consumption rate',   'low_val': 5.8, 'high_val': 3.2},
    {'variable': 'Route interdiction rate',     'low_val': 5.1, 'high_val': 3.8},
    {'variable': 'Pre-positioned stocks level', 'low_val': 3.5, 'high_val': 6.1},
    {'variable': 'Convoy success rate',         'low_val': 4.2, 'high_val': 5.5},
    {'variable': 'Surge resupply availability', 'low_val': 4.4, 'high_val': 5.2},
]
plot_tornado_diagram(
    base_case=4.8,
    sensitivities=sensitivities_example,
    output_metric='Logistics Days-of-Supply at D+30',
    save_path='logistics_sensitivity_tornado.png'
)
```

7-4. Peer Review Standards

No analytical product developed by a single analyst should be briefed at GO/SES level without independent peer review. Peer review at SL 5G level covers:

Table 7-3. Peer Review Checklist (Summary — see Appendix A for full)

Review Area	Reviewer Focus	Pass/Fail Standard
Data provenance	Are sources documented? Are data quality issues noted?	All sources cited, limitations stated
Model assumptions	Are assumptions explicitly stated? Are they defensible?	No hidden or implausible assumptions
Uncertainty quantification	Are confidence intervals present? Is language calibrated?	Point estimates accompanied by ranges
Validation evidence	Is there out-of-sample or historical validation?	Performance metrics reported
Sensitivity analysis	Are key inputs varied? Does recommendation change?	Tornado/sensitivity table present
Plain-language BLUF	Can a non-technical reader understand the finding?	BLUF clear to O-4 staff officer
Recommendation specificity	Is there a clear decision recommendation?	Specific action or decision point stated

7-5. The OR Program Review

At the program level, senior ORSA practitioners are responsible for conducting periodic OR program reviews — assessments of the entire analytical program, not just individual products.

An OR program review covers: - Portfolio of active and planned analytical products - Team capacity assessment (skills inventory against requirements) - Data pipeline health and data quality status - Model inventory — which models are in production, when last validated, who owns them - Alignment to commander's decision-making requirements for the current planning horizon - Lessons learned from recently completed products

Conduct OR program reviews quarterly, or following major exercises (post-DEFENDER EUROPE, post-IRON WOLF). Brief results to C2DAO and the G3.

CHAPTER 8 — BUILDING PERSISTENT OR CAPABILITY ON MSS

BLUF: Individual analytical excellence is necessary but insufficient. USAREUR-AF requires a persistent, institutionalized OR capability that survives personnel turnover, grows analytical capacity over time, and integrates with theater decision-making processes. This chapter covers the data architecture, team standards, and program management required to build and sustain that capability.

NOTE — Palantir Developers reference: *Applied AI: Scaling AI Workflows and Task Execution with AIP* — Covers how Palantir AIP orchestrates multi-step analytical workflows, automates task routing, and integrates AI-generated outputs into operational decision products. Relevant to senior ORSA practitioners designing persistent analytical environments where recurring analytical cycles (readiness forecasts, campaign assessments) should be partially automated rather than rebuilt manually each cycle. Available on the Palantir Developers YouTube channel (@PalantirDevelopers).

8-1. The Persistent Analytical Environment

A persistent analytical environment (PAE) differs from an ad hoc analytical setup:

Ad hoc: An analyst receives a question, accesses MSS, builds a model in a Code Workspace, delivers a product, and moves on. The next analyst who receives a similar question starts over from scratch.

Persistent: The analytical team maintains a documented library of validated models, calibrated to current data, ready to answer recurring question types on short notice. Data pipelines run automatically. Products can be regenerated in hours, not weeks.

Table 8-1. Elements of a Persistent Analytical Environment

Element	Description	Responsible
Data architecture	Foundry datasets, pipelines, and ontology structure for ORSA inputs	Senior ORSA + C2DAO
Model library	Documented, versioned, validated models in Code Workspace repositories	Each model owner

Element	Description	Responsible
Product templates	Standard analysis notebooks that pull current data and regenerate products	ORSA team lead
Metadata and documentation	Model cards, data lineage, assumption logs	All analysts
Review and recertification schedule	Calendar for periodic model validation and data freshness checks	Team lead
Succession planning	All models can be operated by at least two analysts	Team lead

8-2. Data Architecture for ORSA on MSS

8-2a. The ORSA Data Layer

ORSA on MSS consumes data from operational systems (readiness feeds, logistics systems, training management) and produces analytical datasets (model outputs, assessment records, scenario results). Both layers must be managed.

Input data management: - All input datasets ingested through documented ETL pipelines (see SL 4H for pipeline architecture) - Data quality checks run at ingestion — automated alerts for anomalies (missing data, out-of-range values, schema changes) - Data lineage tracked through Foundry — every analytical dataset can be traced to source systems - Access controls enforced at dataset level — ORSA datasets are not accessible to all MSS users

NOTE — Palantir Developers reference: *Deep Dive: Optimizing Data Pipelines with Iceberg Tables and Lightweight Compute* | *DevCon 4* — Covers Foundry's Iceberg table format and lightweight compute options for optimizing pipeline performance at scale — directly relevant to the recurring ETL pipelines (readiness feeds, logistics feeds) that underpin the persistent analytical environment. Watch when designing or troubleshooting high-volume ORSA data layers. Available on the Palantir Developers YouTube channel (@PalantirDevelopers).

Output data management: - Model outputs stored as Foundry datasets, not only in Code Workspace notebooks - Each output dataset tagged with: model version, run date, input data version, analyst ID - Campaign assessment results stored in persistent datasets that accumulate over time (not overwritten each cycle) - Historical model predictions stored alongside actuals for ongoing validation

8-2b. Task: Design the ORSA Data Layer on MSS

CONDITIONS: You are the senior ORSA analyst standing up a new analytical cell for V Corps. You have MSS access, a Foundry project folder provisioned by C2DAO, and three analysts on your team. You will design the data architecture for the cell's persistent analytical environment.

STANDARDS: Data layer design documented; input and output datasets identified; pipeline architecture sketched; access controls specified; documentation plan established; design reviewed and approved by C2DAO before implementation begins.

EQUIPMENT: MSS access, Foundry project folder, team coordination with C2DAO data architects.

PROCEDURE:

1. Inventory analytical requirements (before designing architecture):

```
ORSA Data Requirements Inventory — V Corps Analytical Cell

Recurring Questions:
1. Corps readiness forecast (weekly, 8-week horizon) — inputs: DRRS data
2. Logistics sustainment assessment (biweekly) — inputs: property book, supply status
3. Training pipeline analysis (monthly) — inputs: training management system
4. Exercise/wargame analysis (event-driven) — inputs: wargame data collection
5. Campaign assessment (quarterly) — inputs: all of above

New Analysis Requests (event-driven, varied):
- COA analysis during MDMP
- Red force modeling
- Special studies for CoS / CG

For each: document data sources, refresh frequency, sensitivity, distribution
```

1. Design the Foundry folder structure:

```
V_Corps_ORSA/
├── 00_raw/
│   ├── DRRS_readiness_feed/          ← automated ETL from DRRS
│   ├── logistics_status_feed/        ← automated ETL from property systems
│   ├── training_management_feed/     ← automated ETL from training system
│   └── wargame_data/                 ← manual input during exercises
├── 01_processed/
│   ├── readiness_cleaned/            ← validated, normalized readiness data
│   ├── logistics_cleaned/            ← validated logistics data
│   └── training_cleaned/             ← validated training data
├── 02_analytical/
│   ├── readiness_forecast/           ← model outputs: readiness forecasts
│   ├── logistics_assessment/         ← model outputs: logistics assessments
│   ├── campaign_assessment/          ← persistent campaign assessment records
│   └── wargame_analysis/             ← wargame analytics outputs
├── 03_products/
│   ├── weekly_readiness_brief/       ← dashboard-ready products
│   ├── quarterly_campaign_assessment/← formal assessment products
│   └── special_studies/              ← ad hoc analytical products
└── 99_code/
    ├── models/                       ← versioned model code
    ├── pipelines/                    ← ETL pipeline code
    └── notebooks/                   ← analysis notebooks
```

1. Define access controls in coordination with C2DAO:

Folder	Read Access	Write Access	Rationale
00_raw	ORSA team	ETL pipeline service account only	Protect source data integrity
01_processed	ORSA team, C2DAO	ORSA team	Processed data for analytical use
02_analytical	ORSA team, G3, G4, G2	ORSA team	Outputs accessible to consumers
03_products	All MSS users with need	ORSA team	Finished products widely shared
99_code	ORSA team, C2DAO	ORSA team	Code repository access

8-3. The Model Library and Model Cards

Every model in production use must have a **model card** — a structured document summarizing the model's purpose, inputs, assumptions, validation status, and limitations. Model cards are stored in the `99_code/models/` folder alongside the model code.

Model Card Template:

```
## MODEL CARD — [Model Name]

**Version:** [v1.0, v1.1, etc.]
**Owner:** [Analyst name and contact]
**Last Validated:** [Date]
**Next Recertification Due:** [Date — typically 12 months after validation or after significant input data change]

### Purpose
What question does this model answer? What decisions does it inform?

### Inputs
| Input | Source | Refresh Frequency | Last Validated |
|---|---|---|---|
| [dataset name] | [source system] | [daily/weekly/monthly] | [date] |

### Method
Brief description of analytical method. Reference TM chapter and specific technique.

### Key Assumptions
- [Assumption 1 — with justification]
- [Assumption 2 — with justification]

### Validation Evidence
- Backtesting period: [dates]
- Out-of-sample RMSE / accuracy: [metric]
- Historical analogue comparison: [result]
```

```
### Known Limitations
- [Limitation 1]
- [Limitation 2]

### Output
What the model produces. Where output is stored on MSS.

### Peer Review
Reviewed by: [name], [date]
Review findings: [summary]
```

8-4. Team Standards and Succession Planning

A persistent OR capability requires team standards — conventions that make every analyst's work legible to every other analyst.

8-4a. Code Standards for ORSA on MSS

```
# Standard ORSA Code Workspace header — use in every analysis notebook

"""
============================================================
ORSA Analysis: [Short description]
Analyst: [Name, grade/GS level]
Date: [YYYYMMDD]
Model version: [v1.x]
Input data: [Dataset names and date ranges]
Purpose: [The operational question this analysis addresses]

Peer reviewed by: [Name, date] — or PENDING REVIEW
C2DAO coordination: [Confirmed / Not required / Pending — specify]
============================================================
"""

# All code should:
# 1. Set random seeds for reproducibility
# 2. Include data validation before modeling
# 3. Check optimization convergence before using results
# 4. Save outputs to designated Foundry datasets, not only notebook output
# 5. Close all matplotlib figures after saving (prevents memory issues)
# 6. Be runnable from top to bottom without modification

import numpy as np
np.random.seed(2026)  # Set seed at top of every notebook
```

8-4b. Succession Planning Requirements

Before any analyst departs:

1. All model code committed to `99_code/models/` repository — no analysis in personal workspaces only

2. Model cards complete and current for all production models
3. At least one other analyst can run each production model end-to-end
4. Input data pipeline documentation complete (who to contact if a feed breaks)
5. Handoff brief conducted with incoming analyst and team lead

CAUTION: ORSA capability gaps discovered during a crisis — when a key analyst departs and no one can maintain a production model — can affect commander decision support at the worst possible time. Succession planning is not optional. Treat it as an operational readiness requirement.

8-5. Integrating ML and AI with ORSA

As MSS evolves with AI capabilities under Army CIO Memorandum (April 2024), ORSA analysts will increasingly work alongside SL 4H (AI Engineers) and SL 4M (ML Engineers). Key integration principles:

ORSA owns the question and the interpretation. ML engineers build and deploy models. ORSA analysts define what question the model should answer, what data is analytically valid, what validation is required, and how the output translates to commander decisions.

ML outputs are model inputs, not final products. A SL 4M ML model that classifies equipment maintenance risk is an input to an ORSA logistics analysis — not a standalone product for the G4. ORSA adds the analytical layer: uncertainty quantification, sensitivity analysis, decision framing.

Human-in-the-loop gates are the ORSA analyst's responsibility. If an ML model feeds a semi-automated decision process (scheduling, prioritization), the ORSA analyst must design and document the review gate, specify the conditions under which automated recommendations are overridden, and brief those gates to the commander who approves the system.

Coordinate on model governance. Model cards for integrated ORSA/ML products require co-authorship by the ORSA analyst (analytical validity) and the ML engineer (technical implementation). Both sign off on the model card before the product goes to senior leaders.

APPENDIX A — OR PRODUCT REVIEW CHECKLIST

Use this checklist before delivering any ORSA product to a commander or senior staff officer. The reviewer must be a different analyst from the product author.

A-1. Data Quality and Provenance

- ☐ All input datasets documented by name, source system, and date range
- ☐ Data validation results documented — no anomalies unaccounted for
- ☐ Missing data handling approach documented

- ☐ Data classification and handling appropriate — C2DAO coordination confirmed if required
- ☐ Recency of data appropriate for decision — not using stale data for time-sensitive decision

A-2. Model Validity

- ☐ Model type appropriate for the question (e.g., causal model for causal question; simulation for distributional question)
- ☐ All model assumptions explicitly stated in the product or backup
- ☐ Assumptions reviewed by subject matter expert (not just the analyst who made them)
- ☐ Out-of-sample or historical validation evidence documented
- ☐ No extrapolation beyond the range of calibration data without flagging

A-3. Uncertainty Quantification

- ☐ No bare point estimates — all estimates accompanied by ranges or confidence intervals
- ☐ Probability language calibrated to Table 7-2 standards
- ☐ Confidence level (high/moderate/low) stated separately from probability estimate
- ☐ Forecast products include prediction intervals
- ☐ Simulation products report distributional results (not just mean)

A-4. Sensitivity Analysis

- ☐ Key input assumptions varied through plausible range
- ☐ Sensitivity results presented visually (tornado diagram, scenario table, or equivalent)
- ☐ Identifies which assumption most affects the recommendation
- ☐ States whether recommendation changes under plausible alternative assumptions

A-5. Operational Translation

- ☐ BLUF present: finding, confidence, recommended action in first paragraph
- ☐ Technical language accessible to intended audience level
- ☐ Explicit "so what" — operational implication of finding stated
- ☐ Recommended decision or action is specific, not vague
- ☐ Decision timeline specified — when does commander need to act?

A-6. Reproducibility and Documentation

- ☐ All code committed to project repository
- ☐ Any analyst on the team can re-run analysis from stored code and data
- ☐ Model card created or updated for production models
- ☐ Product archived in `03_products/` with standard naming convention

A-7. Peer Review Record

Field	Value
Product title	
Author	
Reviewer	
Review date	
Issues found	
Resolution	
Reviewer sign-off	

APPENDIX B — ANALYTICAL STANDARDS REFERENCE

B-1. Validation Metric Reference

Table B-1. Model Performance Metrics by Model Type

Model Type	Primary Metric	Secondary Metric	Threshold for Operational Use
Regression (continuous)	RMSE, MAE	R^2, MAPE	MAPE < 15% on holdout; document residual structure
Classification (binary)	AUC-ROC	Precision, Recall, F1	AUC > 0.75 on holdout; class-specific rates reviewed
Time series forecast	RMSE, MAPE	Winkler score (PI coverage)	MAPE < 20% on holdout; PI coverage ≥ 80% at 80% nominal
Simulation (ABMS/MC)	Distributional calibration	Face validity (SME review)	Historical mean ± 1 std dev reproduced; SME sign-off
Optimization	Convergence status	Constraint feasibility	result.success == True; all constraints verified
Bayesian network	Log-likelihood, Brier score	Calibration curve	Calibration plot reviewed; significant deviation documented

B-2. Sample Size Reference for ORSA

Table B-2. Minimum Sample Sizes for Common ORSA Analyses

Analysis Type	Minimum N	Preferred N	Notes
Linear regression (k predictors)	10 * k	20-30 * k	Rule of thumb; check VIF, residuals
Logistic regression (binary)	10 events per predictor	20+ EPP	Events = positive cases in outcome
Time series (ARIMA)	50 periods	100+ periods	More for seasonal models
DiD (difference-in-differences)	30 per group	80+ per group	Plus parallel trends validation
ABMS validation	10-20 replications	30-50 replications	Check convergence of statistics
Monte Carlo simulation	1,000 iterations	10,000+	Check stability of tail percentiles

B-3. Effect Size Reference

For USAREUR-AF operational contexts, Table B-3 translates statistical effect sizes to operational significance thresholds:

Table B-3. Effect Size Operational Significance Thresholds

Domain	Metric	Operationally Meaningful Threshold	Basis
Readiness	C-rating change	0.5 C-level or ≥ 10 percentage points	Command reporting standards
Logistics	Days of supply	≥ 1 day change at forward nodes	Planning threshold
Training effectiveness	Gunnery qualification rate	≥ 5 percentage point change	Training management
Force correlation	Combat power ratio	≥ 0.10 ratio change	Wargame experience
Forecast accuracy	MAPE reduction	≥ 3 percentage point improvement	Resource planning value

GLOSSARY

Agent-Based Modeling and Simulation (ABMS): A simulation approach that models individual agents with behavioral rules and observes emergent collective behavior. Used for complex adaptive systems — logistics networks, urban terrain, adversary adaptation. Distinguished from Monte Carlo simulation by its individual-level, adaptive nature.

Bayes Factor: The ratio of the likelihood of evidence under the hypothesis to the likelihood of evidence under the null hypothesis. A Bayes factor > 10 provides strong evidence; > 100 provides decisive evidence. A concise summary of how much a piece of evidence should change a prior assessment.

Bayesian Network (BN): A directed acyclic graph (DAG) where nodes represent variables and edges represent conditional probabilistic dependencies. Supports inference (probability queries given evidence) and learning (parameter estimation from data).

Causal DAG: A directed acyclic graph representing causal assumptions. Nodes are variables; edges are directed causal arrows. Used to identify confounders, mediators, and colliders for causal inference. First step in any causal analysis.

Campaign Assessment: The systematic, ongoing measurement of progress toward campaign objectives using pre-defined Measures of Effectiveness (MOEs) and Measures of Performance (MOPs). Distinguished from after-action analysis by its prospective design and persistent operation throughout the campaign.

Confounder: A variable that causes both the treatment (independent variable) and the outcome (dependent variable), creating the appearance of a causal relationship when none exists (or masking one that does). The central challenge of causal inference from observational data.

Decision Analysis Under Deep Uncertainty (DAUX): A family of methods for decision-making when probability distributions over future states cannot be reliably specified. Includes robustness analysis, scenario planning, exploratory modeling, and adaptive strategy. Appropriate when planning horizons are long, situations are novel, or key uncertainties are not amenable to probabilistic characterization.

Difference-in-Differences (DiD): A causal inference method that estimates treatment effects by comparing changes in treated units to changes in control units. Requires the parallel trends assumption. Used for evaluating policy changes or interventions that affect some units but not others.

Expected Value of Perfect Information (EVPI): The maximum value of obtaining perfect information about uncertain parameters, measured as the improvement in objective value vs. the best decision under uncertainty. Used to bound the value of ISR, intelligence, and data collection investments.

Extensive Form: In stochastic programming, the complete specification of a stochastic program with all scenarios explicitly enumerated. Converts a stochastic program into a deterministic equivalent LP or NLP.

Lanchester Equations: A set of differential equations (Lanchester's linear law and square law) modeling combat attrition. The square law: $dB/dt = -\alpha \cdot R$, $dR/dt = -\beta \cdot B$, predicting that combat power is proportional to the square of force size weighted by effectiveness. A theoretical foundation for force correlation, not a complete operational model.

Lines of Effort (LOE): In campaign planning, the logical groupings of tasks that, together, accomplish a campaign objective. LOEs structure campaign assessment by organizing MOEs around coherent operational themes.

Measure of Effectiveness (MOE): An indicator used to assess whether a campaign or operation is achieving its objectives. Defined before the operation begins. Distinguished from Measures of Performance (MOPs), which measure execution of tasks rather than achievement of effects.

Mesa: A Python agent-based modeling framework. Used in MSS Code Workspaces for ABMS implementation. Provides agent scheduling, data collection, and visualization components.

Metaheuristic: An optimization method that searches large solution spaces using probabilistic rules rather than gradient-based methods. Includes genetic algorithms, simulated annealing, and particle swarm optimization. Used when exact optimization methods are impractical due to combinatorial complexity or non-differentiable objectives.

Model Card: A structured documentation artifact for analytical models, recording purpose, inputs, assumptions, validation evidence, limitations, and review status. Required for all production models in the USAREUR-AF persistent analytical environment.

Multi-Objective Optimization (MOO): Optimization with two or more competing objectives. Produces a Pareto frontier rather than a single optimal solution. Used when operational problems have genuine tradeoffs among objectives (cost vs. readiness, speed vs. sustainability, risk vs. capability).

Nonlinear Programming (NLP): Mathematical optimization where the objective function or constraints are nonlinear. Solved with gradient-based methods (SLSQP, L-BFGS-B) or metaheuristics. Applies when operational relationships exhibit diminishing returns, economies of scale, or nonlinear system interactions.

Parallel Trends Assumption: The identifying assumption for difference-in-differences analysis. States that treated and control units would have followed the same trend in the outcome variable absent the intervention. Must be validated using pre-period data before DiD results can be interpreted causally.

Pareto Frontier (Pareto Front): The set of Pareto-optimal solutions in a multi-objective optimization problem. A solution is Pareto-optimal if no objective can be improved without degrading another. The Pareto frontier defines the achievable tradeoff surface and is the primary output of multi-objective optimization.

Persistent Analytical Environment (PAE): An institutionalized analytical capability characterized by documented model libraries, automated data pipelines, standard procedures, and succession planning. Distinguished from ad hoc analysis by its continuity across personnel turnover and planning cycles.

Robustness Analysis: In DAUX, the evaluation of strategies across a large ensemble of plausible scenarios to identify strategies that perform adequately across the widest range of futures. Replaces expected-value optimization when probability distributions over scenarios are unreliable.

Scenario-Based Planning: A structured approach to planning under deep uncertainty that replaces single-point forecasts with a set of plausible future scenarios. Scenarios span key axes of uncertainty and are used to stress-test strategies rather than predict specific outcomes.

Sequential Least Squares Programming (SLSQP): A gradient-based nonlinear programming method that handles equality and inequality constraints. The default NLP solver in scipy.optimize for ORSA applications on MSS.

Stochastic Programming: A mathematical programming framework where some parameters are uncertain, modeled as random variables with known (or assumed) distributions. In two-stage stochastic programming, Stage 1 decisions are made before uncertainty resolves; Stage 2 (recourse) decisions are made after.

Two-Stage Stochastic Program: A stochastic programming formulation with two decision stages: here-and-now decisions (Stage 1, before uncertainty resolves) and wait-and-see recourse decisions (Stage 2, after uncertainty resolves). Minimizes expected total cost across both stages.

XLRM Framework: A structured DAUX framework organizing analysis around Exogenous uncertainties (X), policy Levers (L), system Relationships (R), and performance Measures (M). Used to systematically design exploratory modeling experiments and scenario-based planning exercises.

SL 5G — ADVANCED OPERATIONS RESEARCH AND SYSTEMS ANALYSIS — MAVEN SMART SYSTEM

HEADQUARTERS, UNITED STATES ARMY EUROPE AND AFRICA Wiesbaden, Germany

Distribution: Distribution authorized to U.S. Government agencies and their contractors only. Other requests must be referred to Headquarters, C2DAO, Wiesbaden, Germany.

Implements: Army CIO Memorandum, Data and Analytics Policy (April 2024); Unified Data Reference Architecture (UDRA) v1.1 (February 2025) See: learn-data.armydev.com for current platform documentation Prerequisite: SL 4G, Operations Research/Systems Analysis

ORSA-specific governing references:

Publication	Title	Relevance
DA PAM 600-3	Officer Professional Development	Defines FA 49 (ORSA) career progression and qualifications
AR 5-11	Management of Army M&S	M&S governance policy
DA PAM 5-11	VV&A of Army Models and Simulations	Model credibility and accreditation standards
ATP 5-0.3	Multi-Service TTP for Operation Assessment	Assessment methodology, MOE/MOP analytical frameworks
AR 71-9	Warfighting Capabilities Determination	Capabilities-based assessments — core ORSA analytical work

Publication	Title	Relevance
ADP 5-0	The Operations Process	Analytical framework for plan/prepare/execute/assess
Army CIO Memorandum	Data and Analytics Policy (April 2024)	Data governance authority

Strategic Guidance:

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

Document	Authority	Relevance
UDRA v1.1	Unified Data Reference Architecture (February 2025)	Technical reference architecture
JADC2 Strategy Summary	Joint All-Domain Command and Control (March 2022)	Cross-domain data integration strategy

Supplementary Reading — Advanced ODT Use Cases:

- **First Army ODT Mobilization Simulation (2026)** — First Army's 13-person ODT, partnering with Georgia Tech Research Institute, built a simulation modeling thousands of RC units simultaneously for LSCO mobilization planning on the Army Data Platform. Integrates CCMD requirements, unit status, training schedules, weapons platforms, range capacity, and transportation logistics. Enables planners to assess changes in minutes instead of weeks. This represents the most advanced publicly documented ODT analytical product — demonstrating campaign-level ORSA at echelon. Directly relevant to SL 5G competencies in advanced optimization, simulation, and decision analysis under deep uncertainty.
- **Army Reserve ODT at Mojave Falcon (June 2025)** — First Army Reserve ODT deployment supporting 79th TSC/311th ESC. Used Palantir, Tableau, and Power BI for logistics optimization. Key finding for ORSA practitioners: data quality in simulations poorly mimicked real-world conditions — ammunition lacked granularity compared to actual TAMIS data; food and fuel information was oversimplified. Validates SL 5G emphasis on model calibration and validation against operational data.

End of SL 5G