Whitepaper #01: Sovereign AI Infrastructure

Subtitle: *Architecting Intelligent Systems for Disconnected Environments* **Author:** Dustin J. Ober, PMP **Version:** 2.0 | January 2025

1. Executive Summary

The Bottom Line Up Front (BLUF): While the commercial AI revolution is driven by cloud-hosted APIs (e.g., OpenAI, Anthropic, Google), organizations in defense, healthcare, finance, and critical infrastructure face a stark reality: they cannot send their data to the cloud. For these sectors, the future of AI is not "Cloud First"—it is **"Sovereign First."**

The Core Problem: Strict regulatory frameworks (ITAR, HIPAA, CUI, CMMC) and Zero Trust security mandates often prohibit the use of external inference endpoints. Data leakage risks, however small, are unacceptable when the data involves national security, protected health information, or sensitive intellectual property. Furthermore, mission-critical systems often operate in "air-gapped" or disconnected environments where internet connectivity is physically impossible or operationally prohibited.

The Strategic Imperative: Organizations that master Sovereign AI gain a decisive advantage: the ability to leverage cutting-edge intelligence capabilities without compromising security posture, regulatory compliance, or operational continuity. In contested environments—whether cyber, kinetic, or regulatory—the organization with local AI capability maintains decision advantage.

The Solution: This comprehensive guide outlines the reference architecture for a **Sovereign Inference Unit (SIU)**—a fully offline, self-contained AI pipeline. By repatriating workloads to local infrastructure, organizations achieve:

- **100% Data Sovereignty**: No data leaves your network perimeter
- **Zero External Dependencies**: Operations continue during internet outages or adversary actions

- **Regulatory Compliance**: Meet the strictest security frameworks by design
- **Predictable Performance**: Dedicated resources with guaranteed latency
- **Intellectual Property Protection**: Models and prompts remain internal assets

This document serves as a technical guide for architects, engineers, and decision-makers sizing the hardware (VRAM), network topology, security controls, and software stack required to deploy Large Language Models (LLMs) behind the firewall.

Document Roadmap:

1. Strategic context and regulatory drivers
2. Operational challenges unique to disconnected environments
3. Hardware sizing methodology and calculations
4. Network architecture for zero-trust deployment
5. Security hardening best practices
6. Software stack recommendations
7. Implementation case studies
8. Operational procedures and troubleshooting

---

2. Strategic Imperatives for Sovereign AI

Before diving into technical architecture, organizations must understand the strategic forces driving the shift toward sovereign AI capabilities.

2.1 The Geopolitical Landscape

The global AI landscape is increasingly fragmented by export controls, data localization laws, and strategic competition:

Export Controls & Restrictions:

- **ITAR (International Traffic in Arms Regulations)**: Defense-related AI applications cannot use foreign-hosted services

- **EAR (Export Administration Regulations)**: Advanced AI hardware and software face export restrictions
- **Semiconductor Controls**: GPU export restrictions to certain nations affect cloud service availability
- **EU AI Act**: Imposes requirements on AI systems used within the European Union

Strategic Implications: Organizations operating globally must navigate a patchwork of regulations that may prohibit cloud AI usage in certain contexts. A sovereign AI capability provides regulatory flexibility and ensures operational continuity regardless of geopolitical shifts.

2.2 Data Sovereignty Laws

Data localization requirements are expanding globally:

Jurisdiction	Regulation	Key Requirements
European Union	GDPR	Data processing restrictions, transfer limitations
United States	HIPAA	PHI cannot be processed by unauthorized entities
United States	CMMC	Defense contractor data protection requirements
China	PIPL	Data localization for sensitive information
Russia	Federal Law No. 242-FZ	Personal data must be stored on Russian servers

The Common Thread: Sensitive data increasingly cannot leave organizational or national boundaries. Cloud AI services, by definition, require data egress—creating an irreconcilable conflict with these requirements.

2.3 Supply Chain Vulnerabilities

Cloud AI dependency creates strategic vulnerabilities:

- **Provider Outages**: Major cloud AI services have experienced significant outages, disrupting dependent operations
- **API Deprecation**: Models are retired without warning, breaking production systems

- **Pricing Changes**: Cloud AI costs can escalate unpredictably, breaking budgets
- **Terms of Service Shifts**: Providers may change data usage policies post-deployment
- **Third-Party Exposure**: Your prompts and data may be accessible to the cloud provider

Risk Mitigation: Sovereign AI eliminates single points of failure and ensures long-term operational control.

2.4 Mission Continuity Requirements

For mission-critical applications, AI availability cannot depend on external connectivity:

- **Military Operations**: Contested electromagnetic environments may deny internet access
- **Emergency Response**: Natural disasters may sever connectivity precisely when AI is most needed
- **Industrial Control**: Manufacturing and energy systems require deterministic, always-available AI
- **Healthcare**: Clinical decision support must function during network outages

The Sovereign Advantage: Local AI continues operating when cloud services are unavailable, ensuring mission continuity in degraded conditions.

---

3. The Operational Challenge: Why "Local" is Hard

Deploying AI in a connected enterprise is relatively simple: provision an API key, install a Python library, and start sending JSON requests. Deploying AI in a disconnected environment is a fundamental systems engineering challenge. It forces architects to solve problems that cloud providers usually abstract away.

3.1 The Cloud Disconnect

Modern AI development assumes ubiquitous connectivity. Toolchains rely on "lazy loading" resources from the internet:

- `pip install` fetches libraries from PyPI
- `docker pull` fetches containers from Docker Hub

- `AutoTokenizer.from_pretrained()` fetches weights from Hugging Face
- `langchain` and similar frameworks phone home for updates and telemetry

Dependency Mapping Example: Consider a typical RAG (Retrieval-Augmented Generation) pipeline. A single `requirements.txt` may specify 20 direct dependencies, but the full dependency tree often includes 200+ packages, each requiring internet access for initial installation.

```
langchain==0.1.0
├── openai (API calls to external service)
├── tiktoken (downloads encoding files on first use)
├── chromadb
│   └── onnxruntime (may download model files)
├── sentence-transformers
│   └── huggingface_hub (downloads models from HF)
└── ... 180+ additional transitive dependencies
```

In a sovereign environment, **all of these commands fail immediately**. The "Operational Challenge" is not just running the model; it is rebuilding the entire supply chain of dependencies that allows the model to exist.

The Solution Framework:

1. **Dependency Audit**: Map every external call in your toolchain
2. **Asset Collection**: Download all packages, models, and data on the open side
3. **Integrity Verification**: Hash and sign all assets before transfer
4. **Internal Hosting**: Mirror all repositories within the secure boundary
5. **Configuration Override**: Point all tools to internal resources

3.2 The "No-Egress" Mandate

The defining constraint of a secure facility is the **No-Egress Policy**. Data cannot leave the network. This creates specific friction points:

Ingress Difficulty: Getting open-source innovation *into* the environment requires rigorous process:

1. **Asset Identification**: Determine exactly what files are needed

2. **Download to Ingest Station**: Collect assets on an internet-connected system
3. **Malware Scanning**: Run multiple AV/EDR scans on all assets
4. **Integrity Hashing**: Generate SHA256 hashes for verification
5. **Manual Approval**: Security review and authorization
6. **Physical Transfer**: USB, optical media, or data diode
7. **Integrity Verification**: Confirm hashes match on the receiving side
8. **Internal Deployment**: Publish to internal repositories

Typical Transfer Timeline: 2-6 weeks for new software packages in high-security environments.

No Telemetry: You cannot use cloud monitoring tools like Datadog, LangSmith, or Weights & Biases. You must build your own observability stack to answer critical questions:

- Is the model generating coherent outputs or hallucinating?
- What is the actual token throughput under load?
- Are users experiencing acceptable latency?
- When will VRAM constraints cause failures?

3.3 The Throughput Reality Check

Stakeholders often expect "ChatGPT-level speed." This expectation must be actively managed.

Cloud Reality:

- Massive clusters of H100/H200 GPUs
- Auto-scaling to handle thousands of concurrent users
- Aggressive batching and optimization
- Speculative decoding and other advanced techniques
- Result: 50-100+ tokens/second generation

Local Reality:

- Single workstation or small rack
- Fixed GPU capacity serving a specific team

- Limited optimization resources
- Result: 10-40 tokens/second typical for 70B models

The Trade-off Advantage: While local hardware cannot match raw cloud throughput, it offers superior trade-offs:

Metric	Cloud AI	Sovereign AI
Peak Throughput	Higher	Lower
Latency Consistency	Variable	Predictable
Queue Competition	Shared globally	Dedicated
Availability	Dependent on internet	Independent
Cost Predictability	Variable	Fixed
Data Exposure	Required	Zero

Setting Expectations: Document and communicate realistic performance benchmarks before deployment. A slower response that maintains security is infinitely more valuable than a fast response that violates compliance.

3.4 The Human Factor: Skills Gap

Sovereign AI operations require skills that differ from cloud AI consumption:

Cloud AI Skills	Sovereign AI Skills
API integration	Systems administration
Prompt engineering	GPU driver management
Vendor management	Container orchestration
Cost optimization	Capacity planning

Cloud AI Skills	Sovereign AI Skills
SDK usage	Dependency management

Training Requirements:

- Linux systems administration (RHEL/Ubuntu)
- NVIDIA driver and CUDA toolkit management
- Container technologies (Docker, Apptainer)
- Python environment management
- Observability stack operation (Prometheus/Grafana)
- Security hardening procedures

Recommendation: Budget for training or hire personnel with HPC (High-Performance Computing) or data center experience in addition to AI/ML skills.

---

4. Regulatory & Compliance Deep Dive

Different sectors face distinct regulatory requirements that drive sovereign AI adoption.

4.1 Defense & Intelligence (ITAR, CUI, CMMC)

ITAR (International Traffic in Arms Regulations):

- Covers defense articles, services, and related technical data
- Prohibits sharing with foreign persons without license
- Cloud providers with foreign employees may not meet ITAR requirements
- Sovereign AI ensures all processing occurs by authorized personnel

CUI (Controlled Unclassified Information):

- Requires safeguarding of sensitive government information
- NIST SP 800-171 specifies 110 security controls

- Cloud AI may not meet all CUI handling requirements
- Local processing provides full control over data handling

CMMC (Cybersecurity Maturity Model Certification):

- Required for defense contractors
- Level 2+ requires demonstration of cybersecurity practices
- Level 3+ adds additional requirements for CUI
- Sovereign AI supports compliance demonstration

Impact Levels:

Level	Description	Cloud Acceptable?
IL2	Public data	Yes
IL4	CUI	FedRAMP High only
IL5	CUI + mission data	Limited
IL6	Classified	No - air-gapped required

4.2 Healthcare (HIPAA)

Protected Health Information (PHI) Requirements:

- PHI cannot be disclosed to unauthorized entities
- Business Associate Agreements (BAAs) required for all data processors
- Many cloud AI providers do not offer healthcare-specific agreements
- Patient data sent to general-purpose AI constitutes unauthorized disclosure

Sovereign AI Healthcare Benefits:

- All PHI remains within covered entity's control
- No BAA complexity with AI providers
- Full audit trail for compliance demonstration

- De-identification can occur locally before any external processing

Use Cases:

- Clinical decision support
- Medical documentation assistance
- Patient communication drafting
- Research data analysis

4.3 Financial Services (SOX, PCI-DSS, GLBA)

Regulatory Framework:

- **SOX**: Audit trail and control requirements for financial reporting
- **PCI-DSS**: Payment card data protection
- **GLBA**: Customer financial information privacy

AI-Specific Concerns:

- Model decisions affecting financial reporting must be auditable
- Customer financial data cannot be exposed to third parties
- Algorithmic trading systems require deterministic behavior

Sovereign AI Advantages:

- Complete audit trail of all AI interactions
- No exposure of financial data to cloud providers
- Deterministic, reproducible inference results
- Full control over model versions and behavior

---

5. Hardware Sizing: "Sizing the Iron"

In a disconnected environment, you cannot simply "scale up" an instance class when a model crashes. Hardware is fixed, procurement cycles are long (often 6-12 months), and the cost of

under-provisioning is a failed deployment. Therefore, calculating Video Random Access Memory (VRAM) requirements is the single most critical step in architecting a sovereign inference unit.

5.1 The VRAM Equation

The total VRAM required (M_total) is the sum of three distinct components:

```
M_total = M_weights + M_kv_cache + M_activations + O_system
```

Component Breakdown:

Component	Formula	Description
M_weights	P × B_p	Static model parameters
M_kv_cache	2 × L × H × C_ctx × B_kv × B_batch	Dynamic context storage
M_activations	Variable	Temporary computation buffers
O_system	~500MB - 2GB	CUDA overhead, framework buffers

Variable Definitions:

Variable	Description	Example Values
P	Number of parameters	7B, 13B, 70B, 405B
B_p	Bytes per parameter	FP16=2, Q8=1, Q4=0.5
L	Number of layers	32, 40, 80
H	Hidden dimension	4096, 5120, 8192
C_ctx	Context length	4096, 8192, 32768, 128K
B_kv	Bytes per KV entry	2 (FP16), 1 (FP8)
B_batch	Batch size	1-32 typically

5.2 Worked Example: Llama-3-70B

Model Specifications:

- Parameters (P): 70 billion
- Layers (L): 80
- Hidden dimension (H): 8192
- Context length (C_ctx): 8192 tokens

Calculation at Q4_K_M Quantization:

```
M_weights = 70B × 0.5 bytes = 35 GB

M_kv_cache = 2 × 80 × 8192 × 8192 × 2 bytes × 1 batch
           = 2 × 80 × 8192 × 8192 × 2
           = 21.5 GB (at 8K context, single user)

M_activations ≈ 3-5 GB (varies by framework)

O_system ≈ 1.5 GB

M_total ≈ 35 + 21.5 + 4 + 1.5 = 62 GB
```

Hardware Requirement: Single A100-80GB (tight) or dual A6000-48GB

Critical Insight: The KV cache grows linearly with context length and batch size. A model that fits comfortably at 4K context may OOM at 32K context.

5.3 Quick Reference Sizing

Tier	Example Model	Quantization	VRAM Required	Hardware Example	Users
Edge	Llama-3.2-3B	Q4_K_M	~2.5 GB	RTX 3060, Jetson Orin	1-2
Small	Llama-3-8B	Q4_K_M	~6 GB	RTX 3060-12GB, RTX 4070	1-5

Tier	Example Model	Quantization	VRAM Required	Hardware Example	Users
Medium	Mixtral 8x7B	Q4_K_M	~26 GB	RTX 4090, A5000-24GB	1-10
Large	Llama-3-70B	Q4_K_M	~42 GB	A6000-48GB, Dual 3090	1-20
XL	Llama-3-70B	Q8_0	~75 GB	A100-80GB	1-30
Enterprise	Llama-3.1-405B	Q4_K_M	~200 GB	8× A100-80GB	Varies

Embedding Models (for RAG):

Model	Parameters	VRAM	Use Case
all-MiniLM-L6	22M	<1 GB	Fast, lower quality
bge-base	110M	~1 GB	Balanced
bge-large	335M	~2 GB	High quality
E5-mistral-7b	7B	~14 GB	Best quality

5.4 GPU Architecture Comparison

NVIDIA Consumer vs Professional:

Aspect	Consumer (RTX)	Professional (A-series)
VRAM	Up to 24GB	Up to 80GB
ECC Memory	No	Yes (critical for reliability)
Virtualization	Limited	Full support

Aspect	Consumer (RTX)	Professional (A-series)
Support	Limited	Enterprise
Warranty	3 years	5 years
Multi-GPU	Limited NVLink	Full NVLink/NVSwitch
Price	$1,500-2,000	$15,000-30,000

When to Use Consumer GPUs:

- Development and prototyping
- Small team deployments (< 10 users)
- Cost-constrained environments
- Models under 24GB

When to Require Professional GPUs:

- Production mission-critical systems
- High availability requirements
- Multi-user (10+) deployments
- Models requiring 48GB+
- Clustered deployments

AMD ROCm Considerations: AMD GPUs (MI250, MI300) offer compelling price/performance but require:

- Different software stack (HIP vs CUDA)
- Less mature ecosystem
- Fewer pre-built containers
- More manual optimization

Recommendation: Use NVIDIA for sovereign deployments unless organization has existing AMD expertise and can accept additional integration effort.

5.5 Multi-GPU Configurations

NVLink vs PCIe:

Interconnect	Bandwidth	Use Case
PCIe 4.0 x16	32 GB/s	Independent models per GPU
PCIe 5.0 x16	64 GB/s	Independent models per GPU
NVLink 3	600 GB/s	Single model across GPUs
NVLink 4	900 GB/s	Single model across GPUs

Configuration Strategies:

1. **Model Parallelism (NVLink required):**
    - Split single large model across GPUs
    - Required for models exceeding single GPU VRAM
    - Example: 405B model across 8× A100
2. **Data Parallelism (PCIe sufficient):**
    - Same model replicated on each GPU
    - Different users served by different GPUs
    - Simple load balancing
3. **Hybrid:**
    - Multiple model replicas, each spanning multiple GPUs
    - Maximum throughput for large models

5.6 Storage & I/O Considerations

Model loading speed affects startup time and failover:

Storage Type	Read Speed	70B Model Load Time
HDD	150 MB/s	~4 minutes
SATA SSD	550 MB/s	~70 seconds
NVMe Gen3	3.5 GB/s	~12 seconds
NVMe Gen4	7 GB/s	~6 seconds
NVMe RAID0	14+ GB/s	~3 seconds

Recommendations:

- Minimum: NVMe SSD for model storage
- Optimal: NVMe Gen4 in RAID0 for fast failover
- Enterprise: Dedicated high-speed NAS for model registry

Capacity Planning:

- Reserve 2× model size for swap/staging
- Plan for multiple model versions
- Budget for embedding model storage
- Account for log and trace storage

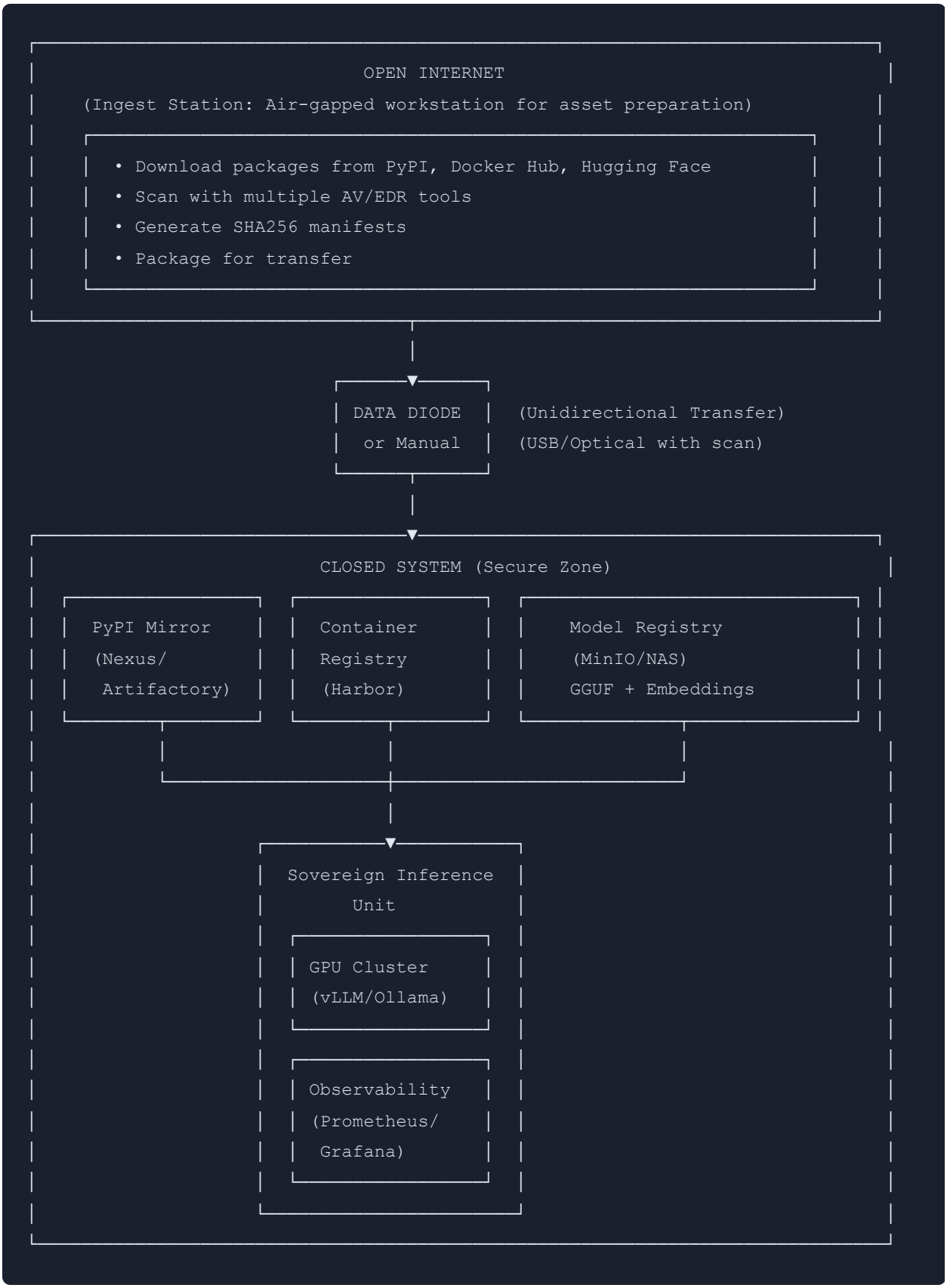
---

6. Network Architecture: The "Zero Trust" Setup

In a sovereign environment, the network is a constraint. We achieve functionality by building a "Local Internet"—a mirrored ecosystem that lives entirely within the secure boundary.

6.1 The Physical Air-Gap Topology

The Sovereign Inference Unit (SIU) resides on a dedicated subnet with physically severed uplinks.



6.2 Internal Repository Stack

PyPI Mirror (Python Packages):

Use **Sonatype Nexus Repository** or **JFrog Artifactory** to host Python wheels internally.

Nexus Configuration:

```
# Create PyPI proxy repository (for initial population on open side)
# Then create hosted repository for closed side

# Client pip.conf (on closed systems):
[global]
index-url = https://nexus.internal.lab/repository/pypi-hosted/simple
trusted-host = nexus.internal.lab
timeout = 120
```

Initial Population Process:

```
# On open-side ingest station:
pip download -d ./packages -r requirements.txt

# Generate manifest:
sha256sum ./packages/* > manifest.sha256

# Transfer packages via approved method
# On closed side:
twine upload --repository-url https://nexus.internal.lab/repository/pypi-hosted/
./packages/*
```

Container Registry:

Use **Harbor** for enterprise container management:

```
# Harbor provides:
# - Vulnerability scanning
# - Image signing
# - Replication
# - RBAC

# Docker daemon configuration (/etc/docker/daemon.json):
{
  "insecure-registries": [],
```

```
  "registry-mirrors": ["https://harbor.internal.lab"]
}
```

Model Registry:

Use **MinIO** or shared NAS for model file management:

```
# MinIO bucket structure:
models/
├── llm/
│   ├── llama-3-70b-q4_k_m.gguf
│   ├── llama-3-70b-q4_k_m.gguf.sha256
│   └── llama-3-8b-q4_k_m.gguf
├── embedding/
│   ├── bge-large-en-v1.5/
│   └── e5-mistral-7b-instruct/
└── manifests/
    └── approved-models.json
```

6.3 Containerization: Docker vs. Apptainer

Feature	Docker	Apptainer (Singularity)
Primary Use	Development, open internet	Production, HPC, secure
Privilege Model	Requires root daemon	Rootless execution
Image Format	Layered OCI images	Single `.sif` file
Security Audit	Complex (layers)	Simple (single file hash)
GPU Support	nvidia-docker runtime	Native `--nv` flag
HPC Integration	Limited	Native (Slurm, PBS)
File Transfer	`docker save` (tar)	Single file copy
Signing	Docker Content Trust	Built-in SIF signing

Recommended Workflow:

```
# 1. Build with Docker on open side
docker build -t my-inference:v1.0 .
docker save -o my-inference-v1.0.tar my-inference:v1.0

# 2. Generate hash
sha256sum my-inference-v1.0.tar > my-inference-v1.0.tar.sha256

# 3. Transfer via approved method

# 4. Convert to Apptainer on closed side
apptainer build my-inference-v1.0.sif docker-archive://my-inference-v1.0.tar

# 5. Sign the SIF
apptainer sign my-inference-v1.0.sif

# 6. Run with GPU access
apptainer run --nv my-inference-v1.0.sif
```

6.4 Cross-Domain Solutions

For environments requiring data flow between classification levels:

Data Diodes:

- Unidirectional network devices
- Physically prevent reverse data flow
- Used for ingesting updates to high-side systems
- Vendors: Owl Cyber Defense, Waterfall Security, BAE Systems

Guards:

- Bi-directional but controlled
- Content inspection and filtering
- Policy-based transfer decisions
- Higher risk than diodes

Multi-Level Security (MLS) Considerations:

- SELinux MLS mode for label-based access control
- Trusted Platform Modules (TPM) for attestation
- Separate inference instances per classification level
- No model or prompt sharing across levels

---

7. Security Hardening Best Practices

Sovereign AI systems require defense-in-depth security across all layers.

7.1 Operating System Hardening

Base Configuration:

- Start with minimal installation (no GUI)
- Apply all security updates before air-gapping
- Configure automatic security updates from internal mirror

STIG Compliance (DoD environments):

```
# Apply DISA STIG using OpenSCAP
oscap xccdf eval --profile stig \
  --results results.xml \
  --report report.html \
  /usr/share/xml/scap/ssg/content/ssg-rhel9-ds.xml

# Common STIG requirements:
# - Disable unnecessary services
# - Configure audit logging
# - Set password policies
# - Enable SELinux enforcing
# - Configure firewall rules
```

SELinux for AI Workloads:

```
# Set enforcing mode
setenforce 1
```

```
sed -i 's/SELINUX=permissive/SELINUX=enforcing/' /etc/selinux/config

# Create custom policy for inference service
# Allow GPU access while restricting network
```

7.2 Model Security

Verified Provenance:

```
# Always verify model hashes before use
sha256sum -c model-manifest.sha256

# For critical environments, require cryptographic signatures
gpg --verify model.gguf.sig model.gguf
```

Prompt Injection Defenses:

- Input sanitization before model processing
- Output filtering for sensitive patterns
- System prompt hardening
- Rate limiting per user

Output Sanitization:

```
def sanitize_output(response: str, sensitive_patterns: list) -> str:
    """Remove or mask sensitive information from model output."""
    for pattern in sensitive_patterns:
        response = re.sub(pattern, "[REDACTED]", response)
    return response
```

7.3 Audit & Logging

Comprehensive Logging Requirements:

```
# Log structure for compliance
log_entry = {
    "timestamp": "2025-01-09T10:30:00Z",
    "user_id": "user123",
```

```
    "session_id": "sess_abc123",
    "model": "llama-3-70b-q4",
    "prompt_hash": "sha256:abc123...",  # Hash, not raw prompt if sensitive
    "response_hash": "sha256:def456...",
    "tokens_in": 150,
    "tokens_out": 500,
    "latency_ms": 2500,
    "gpu_id": 0,
    "vram_used_gb": 45.2
}
```

Log Retention:

- Determine retention requirements by regulation
- HIPAA: 6 years
- SOX: 7 years
- Defense: Often indefinite
- Plan storage accordingly

Audit Trail Integrity:

- Write logs to append-only storage
- Regular log hashing/signing
- Forward to central SIEM (internal)

---

8. Software Stack: The "Sovereign Inference" Layer

8.1 Operating System

Recommended Distributions:

Distribution	Use Case	Support
Ubuntu LTS	General production	5 years standard

Distribution	Use Case	Support
RHEL	Enterprise/DoD	10 years
Rocky Linux	RHEL alternative	Community
Ubuntu Pro	Extended security	10 years

NVIDIA Driver Installation (Offline):

```
# Download driver on open side
wget https://us.download.nvidia.com/XFree86/Linux-x86_64/550.54.14/NVIDIA-Linux-
x86_64-550.54.14.run

# Transfer to closed side and install
chmod +x NVIDIA-Linux-x86_64-550.54.14.run
./NVIDIA-Linux-x86_64-550.54.14.run --silent

# Pin driver version to prevent auto-updates
apt-mark hold nvidia-driver-550

# Verify installation
nvidia-smi
```

CUDA Toolkit Installation:

```
# Download offline installer on open side
# Transfer and install
sh cuda_12.4.0_550.54.14_linux.run --silent --toolkit

# Add to PATH
echo 'export PATH=/usr/local/cuda-12.4/bin:$PATH' >> ~/.bashrc
echo 'export LD_LIBRARY_PATH=/usr/local/cuda-12.4/lib64:$LD_LIBRARY_PATH' >>
~/.bashrc

# Verify
nvcc --version
```

Critical: Document exact driver and CUDA versions. Mismatches between driver, CUDA, and PyTorch cause silent inference failures.

8.2 Inference Engine Comparison

Engine	Best For	Throughput	Ease of Use	API
Ollama	Single-user, development	Moderate	Excellent	OpenAI-compatible
vLLM	Production, high throughput	Excellent	Moderate	OpenAI-compatible
llama.cpp	Edge, resource constrained	Good	Good	Custom/OpenAI
TGI	Enterprise multi-model	Excellent	Moderate	HuggingFace
LocalAI	Multi-model, flexibility	Good	Good	OpenAI-compatible

Recommendation Decision Tree:

```
Is this for production with multiple users?
├── Yes: Use vLLM
│   └── Need multi-model serving? Consider TGI
└── No:
    ├── Development/prototyping? Use Ollama
    ├── Edge/resource constrained? Use llama.cpp
    └── Need maximum flexibility? Use LocalAI
```

vLLM Deployment Example:

```
# Run with Docker/Apptainer
apptainer run --nv vllm.sif \
  --model /models/llama-3-70b-q4_k_m.gguf \
  --tensor-parallel-size 2 \
  --gpu-memory-utilization 0.90 \
```

```
    --max-model-len 8192 \
    --port 8000
```

8.3 Quantization Deep Dive

Quantization reduces model size by using fewer bits per parameter, enabling larger models on limited hardware.

Quantization Comparison:

Method	Bits	Size vs FP16	Perplexity Impact	Speed Impact
FP16	16	100% (baseline)	None	Baseline
FP8	8	50%	Minimal (<0.1%)	~10% faster
Q8_0	8	50%	Minimal	Varies
Q5_K_M	5	31%	Very small (~0.5%)	~15% faster
Q4_K_M	4	25%	Small (~1%)	~20% faster
Q4_K_S	4	25%	Small-moderate	~20% faster
Q3_K_M	3	19%	Moderate (~2-3%)	~25% faster
Q2_K	2	12.5%	Significant (5%+)	~30% faster

Recommendation by Use Case:

Use Case	Recommended Quantization
Maximum quality, VRAM available	FP16 or Q8_0
Production general use	Q4_K_M
Memory constrained + quality	Q5_K_M

Use Case	Recommended Quantization
Edge deployment	Q4_K_S or Q3_K_M
Experimentation only	Q2_K

Quality Validation: Always test quantized models against your specific use cases. Generic perplexity benchmarks may not reflect domain-specific performance.

8.4 Observability Stack

Without cloud monitoring, build internal observability:

Prometheus + Grafana Stack:

```
# docker-compose.yml for observability
version: '3.8'
services:
  prometheus:
    image: prom/prometheus:v2.48.0
    volumes:
      - ./prometheus.yml:/etc/prometheus/prometheus.yml
    ports:
      - "9090:9090"

  grafana:
    image: grafana/grafana:10.2.0
    ports:
      - "3000:3000"
    volumes:
      - grafana-storage:/var/lib/grafana

volumes:
  grafana-storage:
```

Key Metrics to Monitor:

Metric	Description	Alert Threshold
`gpu_memory_used_bytes`	VRAM consumption	>90%

Metric	Description	Alert Threshold
`inference_tokens_per_second`	Generation speed	<10 tok/s
`inference_queue_depth`	Waiting requests	>10
`inference_latency_p99`	99th percentile latency	>30s
`model_load_time_seconds`	Startup time	>120s
`inference_errors_total`	Error count	Any increase

Alerting Configuration:

```
# Prometheus alerting rules
groups:
  - name: sovereign-ai
    rules:
      - alert: HighVRAMUsage
        expr: gpu_memory_used_bytes / gpu_memory_total_bytes > 0.95
        for: 5m
        labels:
          severity: critical
        annotations:
          summary: "GPU VRAM critically high"
```

---

9. Implementation Case Studies

9.1 Case Study: Defense Intelligence Analysis System

Environment:

- Classification: IL6 (Top Secret)
- Air-gapped facility
- 50 analysts requiring AI assistance

Architecture:

- 4× Dell PowerEdge R760xa with 4× A100-80GB each (16 GPUs total)
- Model: Llama-3-70B Q4_K_M replicated across GPU pairs
- Inference: vLLM with tensor parallelism
- Storage: 50TB NAS for documents, 10TB NVMe for models

Implementation Challenges:

1. **6-month procurement cycle** for classified hardware
2. **Driver compatibility** issues with RHEL 8 STIG baseline
3. **Model validation** required security review of training data provenance

Results:

- 40 concurrent users supported
- 25 tokens/second average generation
- 99.5% uptime over 12 months
- Zero data spillage incidents

Lessons Learned:

- Start procurement 12 months before needed deployment
- Maintain driver/CUDA compatibility matrix
- Build relationships with security reviewers early

9.2 Case Study: Hospital Clinical Decision Support

Environment:

- Regulation: HIPAA, state privacy laws
- Network: Isolated clinical VLAN
- Users: 200 physicians, nurses

Architecture:

- Single Dell Precision 7920 with 2× RTX A6000-48GB
- Model: Llama-3-8B Q4_K_M (fast responses)
- Embedding: bge-large for document search
- Vector DB: ChromaDB (local SQLite backend)

Use Cases:

- Medication interaction checking
- Clinical documentation drafting
- Patient history summarization

Implementation Challenges:

1. **PHI sanitization** - Built custom de-identification pipeline
2. **EHR integration** - Created FHIR adapter for Epic
3. **Physician adoption** - Required extensive training

Results:

- 15% reduction in documentation time
- Zero PHI breaches
- High physician satisfaction (4.2/5.0)

9.3 Case Study: Manufacturing Quality Control

Environment:

- Location: Factory floor (edge deployment)
- Connectivity: Isolated OT network
- Uptime requirement: 99.99%

Architecture:

- NVIDIA Jetson AGX Orin (64GB)
- Model: Llama-3.2-3B Q4_K_M
- Vision: Custom fine-tuned defect detection

Use Cases:

- Real-time defect classification
- Quality report generation
- Operator guidance

Implementation Challenges:

1. **Environmental factors** - Required industrial enclosure
2. **Power reliability** - Added UPS backup
3. **Model updates** - Quarterly update cycle via USB

Results:

- 99.97% uptime
- 50ms average inference latency
- 30% reduction in escaped defects

---

10. Operational Runbooks

10.1 Model Update Procedure

Pre-Update Checklist:

- [ ] New model validated on test system
- [ ] Rollback plan documented
- [ ] Maintenance window scheduled
- [ ] Users notified

Update Steps:

```
# 1. Download new model on ingest station
wget https://huggingface.co/.../new-model.gguf
```

```
# 2. Generate hash
sha256sum new-model.gguf > new-model.gguf.sha256

# 3. Transfer via approved method

# 4. Verify hash on closed side
sha256sum -c new-model.gguf.sha256

# 5. Stage model
cp new-model.gguf /models/staging/

# 6. Test in staging environment
./test-model.sh /models/staging/new-model.gguf

# 7. During maintenance window:
systemctl stop inference-service
cp /models/production/current.gguf /models/rollback/
cp /models/staging/new-model.gguf /models/production/current.gguf
systemctl start inference-service

# 8. Validate
./validate-inference.sh

# 9. If validation fails:
systemctl stop inference-service
cp /models/rollback/current.gguf /models/production/current.gguf
systemctl start inference-service
```

10.2 Disaster Recovery

Backup Strategy:

Component	Backup Frequency	Retention	Method
Model files	On change	3 versions	NAS replication
Configuration	Daily	30 days	Git + encrypted backup
Vector DB	Hourly	7 days	SQLite backup
Logs	Real-time	Per policy	Central log server

Recovery Time Objectives:

Failure Scenario	RTO	Procedure
Single GPU	15 min	Failover to replica
Full server	2 hours	Restore from backup server
Storage failure	4 hours	Restore from NAS backup
Facility loss	24 hours	Activate DR site

10.3 Capacity Planning

Monitoring for Scaling Triggers:

Metric	Warning	Critical	Action
Queue depth avg	>5	>15	Add GPU capacity
P99 latency	>20s	>60s	Add GPU capacity
VRAM utilization	>85%	>95%	Reduce context or batch
Daily requests	>80% capacity	>95% capacity	Plan expansion

Scaling Options:

1. **Vertical (same server):** Add GPUs if slots available
2. **Horizontal (more servers):** Add inference nodes with load balancer
3. **Model optimization:** Use smaller quantization or model

---

11. Deployment Checklist

Phase 1: Pre-Deployment (2-4 weeks before)

Hardware:

- [ ] VRAM calculated and validated for target model + context + batch
- [ ] Server racked and cabled
- [ ] Power and cooling verified
- [ ] Redundancy configured (RAID, dual PSU)

Network:

- [ ] Air-gap topology established
- [ ] Internal DNS configured
- [ ] Firewall rules implemented (no egress)
- [ ] Internal repositories populated (PyPI, containers, models)

Security:

- [ ] OS hardened per STIG/CIS benchmark
- [ ] SELinux/AppArmor enforcing
- [ ] Audit logging configured
- [ ] User accounts provisioned

Phase 2: Deployment (1-2 days)

Software:

- [ ] NVIDIA drivers installed offline and version pinned
- [ ] CUDA toolkit installed and verified
- [ ] Container runtime configured
- [ ] Inference engine deployed
- [ ] Observability stack operational

Models:

- [ ] Models transferred and integrity verified (SHA256)
- [ ] Model loaded successfully

- [ ] Warm-up inference completed

Phase 3: Validation (1-3 days)

Testing:

- [ ] Functional tests passed
- [ ] Load testing completed within VRAM constraints
- [ ] Latency benchmarks documented
- [ ] Error handling validated
- [ ] Rollback procedure tested

Documentation:

- [ ] Runbooks reviewed and approved
- [ ] Contact escalation list verified
- [ ] User training completed

Phase 4: Go-Live

Final Checks:

- [ ] All stakeholders signed off
- [ ] Monitoring dashboards accessible
- [ ] On-call rotation scheduled
- [ ] Communication channels established

Sign-off Matrix:

Role	Name	Signature	Date
System Owner			
Security Officer			
Network Admin			

Role	Name	Signature	Date
Operations Lead			

---

12. Troubleshooting & Common Pitfalls

12.1 CUDA/Driver Mismatches

Symptoms:

- `CUDA error: no kernel image is available for execution on the device`
- `CUDA driver version is insufficient for CUDA runtime`
- Silent inference failures with garbage output

Diagnosis:

```
# Check driver version
nvidia-smi

# Check CUDA version
nvcc --version

# Check PyTorch CUDA
python -c "import torch; print(torch.version.cuda)"
```

Resolution:

```
# Match versions using compatibility matrix:
# https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/

# For PyTorch, use matching CUDA version:
pip install torch==2.2.0+cu121
```

12.2 Out-of-Memory (OOM) Errors

Symptoms:

- `CUDA out of memory`
- Process killed by OOM killer
- Inference suddenly stops

Diagnosis:

```
# Monitor VRAM in real-time
watch -n 1 nvidia-smi

# Check system OOM events
dmesg | grep -i oom
```

Resolution Options:

1. Reduce batch size
2. Reduce context length
3. Use more aggressive quantization
4. Enable GPU memory fraction limit
5. Add more VRAM (hardware upgrade)

```
# In vLLM, limit memory usage:
--gpu-memory-utilization 0.85  # Leave 15% headroom
```

12.3 Silent Failures & Quality Degradation

Symptoms:

- Model outputs become nonsensical
- Response quality suddenly drops
- Increased hallucination rate

Potential Causes:

1. **Corrupted model file** - Re-verify hash

2. **Temperature drift** - Check GPU thermals
3. **Memory corruption** - ECC errors (if available)
4. **Context overflow** - Inputs exceeding max length

Monitoring:

```
# Implement quality scoring
def check_response_quality(response: str) -> float:
    """Score response coherence 0-1."""
    # Check for repetition
    # Check for coherent sentences
    # Check for expected format
    return quality_score

# Alert if quality drops
if check_response_quality(response) < 0.7:
    alert("Response quality degraded")
```

12.4 Performance Regression

Symptoms:

- Inference speed suddenly decreases
- Latency increases over time
- Queue depth grows unexpectedly

Investigation:

```
# Check for thermal throttling
nvidia-smi -q -d PERFORMANCE

# Check GPU utilization patterns
nvidia-smi dmon -s u

# Check for competing processes
nvidia-smi pmon
```

Common Causes:

1. Thermal throttling - Improve cooling
2. Memory fragmentation - Restart service
3. Competing workloads - Isolate GPU
4. Driver issues - Check for updates/downgrades

---

13. Conclusion

Building a Sovereign AI capability is a systems engineering challenge that marries hardware constraints, network security, and operational discipline. While the upfront architectural cost is significantly higher than cloud AI consumption, the strategic value—**complete data sovereignty**—is non-negotiable for mission-critical sectors.

Key Principles to Remember

1. **VRAM is King:** Size your hardware for the context window and concurrent users, not just the model weights. Under-provisioning leads to failures; over-provisioning provides operational margin.

2. **Mirror Everything:** Your internal environment must replicate the open-source supply chain. Every PyPI package, every container image, every model weight must be available locally.

3. **Quantize Intelligently:** Q4_K_M offers the optimal balance of size and quality for most production use cases. Validate against your specific domain before deployment.

4. **Containerize for Portability:** Build with Docker for familiar tooling, deploy with Apptainer for security and reproducibility. The single SIF file simplifies security auditing.

5. **Monitor Internally:** Without cloud observability, build your own. Prometheus/Grafana provides enterprise-grade monitoring without external dependencies.

6. **Plan for the Long Cycle:** Procurement, security reviews, and testing take months, not days. Start early and maintain relationships with stakeholders.

7. **Train Your Team:** Sovereign AI requires different skills than cloud AI. Invest in systems administration, GPU management, and operational training.

8. **Document Relentlessly:** In disconnected environments, you cannot search Stack Overflow. Comprehensive internal documentation is essential.
9. **Test Failure Modes:** Regularly exercise rollback procedures, disaster recovery, and failover. The time to discover problems is during testing, not production incidents.
10. **Stay Current Strategically:** Balance the security cost of updates against the capability gains of newer models and tools. Establish a regular update cadence.

Strategic Recommendations for Leadership

- **Budget for True Cost:** Sovereign AI requires 3-5× the initial investment of cloud AI, but eliminates recurring API costs and data exposure risks.
- **Accept Performance Trade-offs:** Local inference is slower than cloud but provides consistency and availability that cloud cannot match.
- **Invest in People:** The limiting factor is often skilled personnel, not hardware. Training and retention are critical.
- **Plan Multi-Year:** Hardware refresh cycles, model updates, and capability expansion should be planned 2-3 years ahead.

The Path Forward

The AI landscape will continue evolving rapidly. Models are becoming more capable while requiring fewer resources. Inference optimization techniques are improving. The gap between cloud and local capability is narrowing.

Organizations that invest in sovereign AI infrastructure today will:

- Maintain decision advantage in contested environments
- Ensure compliance as regulations tighten
- Protect intellectual property and sensitive data
- Build internal expertise that compounds over time

The future of AI for mission-critical sectors is not in the cloud—it is **sovereign**.

Next in this Series:

- **Whitepaper #02:** *The Disconnected Pipeline: Solving Dependency Management & Containerization in Secure Facilities.*
- **Whitepaper #03:** *Private Knowledge Retrieval: Architecting Local RAG Systems.*
- **Whitepaper #04:** *Verifiable Intelligence: DSPy, Governance, and Hallucination Control.*

Appendix A: Glossary of Terms

Term	Definition
Air-Gap	Physical separation of a network from the internet
Apptainer	Container platform for HPC/secure environments (formerly Singularity)
CUI	Controlled Unclassified Information
Data Diode	Hardware enforcing unidirectional data flow
FedRAMP	Federal Risk and Authorization Management Program
GGUF	Quantized model format for llama.cpp/Ollama
IL4/IL5/IL6	DoD Impact Levels for cloud computing
ITAR	International Traffic in Arms Regulations
KV Cache	Key-Value cache storing attention context
NVLink	High-bandwidth GPU interconnect
OOM	Out Of Memory error
Quantization	Reducing model precision to decrease size

Term	Definition
SIF	Singularity Image Format (Apptainer container)
SIU	Sovereign Inference Unit
STIG	Security Technical Implementation Guide
vLLM	High-throughput LLM inference engine
VRAM	Video Random Access Memory (GPU memory)
Zero Trust	Security model assuming no implicit trust

Appendix B: Quick Reference Cards

VRAM Quick Calculator

```
Model VRAM (Q4) ≈ Parameters (B) × 0.5 GB
KV Cache (8K ctx) ≈ 10-25 GB for 70B models
Total ≈ Model + KV + 5GB overhead

Examples:
- 8B Q4: ~6 GB total
- 70B Q4: ~45-60 GB total
- 405B Q4: ~200 GB total
```

Essential Commands

```
# Check GPU status
nvidia-smi

# Monitor GPU in real-time
watch -n 1 nvidia-smi

# Check CUDA version
```

```
nvcc --version

# Verify model hash
sha256sum -c model.sha256

# Run Apptainer with GPU
apptainer run --nv container.sif

# Check inference service
systemctl status inference

# View logs
journalctl -u inference -f
```

Port Reference

Service	Default Port
vLLM API	8000
Ollama API	11434
Prometheus	9090
Grafana	3000
MinIO	9000
Nexus	8081
Harbor	443

Appendix C: Further Reading & Resources

Technical Documentation:

- NVIDIA CUDA Toolkit Documentation

- vLLM Documentation
- Apptainer User Guide
- NIST SP 800-171 (CUI Security)

Model Sources (for open-side download):

- Hugging Face Hub
- Ollama Model Library
- TheBloke GGUF Quantizations

Security Frameworks:

- DISA STIGs
- CIS Benchmarks
- NIST Cybersecurity Framework

About the Author

Dustin J. Ober, PMP, M.Ed. *AI Developer & Technical Instructional Systems Designer*

Dustin J. Ober is a specialist in the intersection of Artificial Intelligence, Instructional Strategy, and secure systems architecture. With a background spanning over two decades in the United States Air Force and defense contracting, he focuses on deploying high-impact technical solutions within mission-critical environments.

Unlike traditional developers who focus solely on code, Dustin bridges the gap between **technical capability** and **operational reality**. His expertise lies in architecting "Sovereign AI" systems—designing offline, air-gapped inference pipelines that allow organizations to leverage state-of-the-art intelligence without compromising data security or compliance.

He holds a Master of Education in Instructional Design & Technology and is a certified Project Management Professional (PMP). He actively develops open-source tools for the AI community, focusing on DSPy implementation, neuro-symbolic logic, and verifiable agentic workflows.

Connect:

- **Web:** aiober.com
- **LinkedIn:** linkedin.com/in/dustinober
- **Email:** dustinober@me.com

Suggested Citation: Ober, D. J. (2025). *Sovereign AI Infrastructure: Architecting Intelligent Systems for Disconnected Environments* (Whitepaper No. 01, Version 2.0). AIOber Technical Insights.