Technical Risk Assessment

Phase 1: Design & Specification

Sovereign AI Infrastructure Project

Document:	Technical Risk Assessment (Phase 1)
Version:	1.0
Date:	February 11, 2026
Status:	Draft for Review
Owner:	Solutions Architect / Technical Lead
Standard:	ISO/IEC 25010:2011; IEEE 1540-2001; SEI CMMI RSKM; NIST SP 800-30
Dependencies:	System Architecture Document v1.0, Data Architecture Document v1.0

1. Executive Summary

This Technical Risk Assessment provides a deep-dive analysis of architectural and technical risks specific to the Sovereign AI Infrastructure solution design. The assessment evaluates risks across technology maturity, integration complexity, scalability, performance, dependencies, and technical debt.

> **Key Findings:**
>
> - **Architecture Novelty:** Bicameral GPU/CPU separation with Prolog-based constitutional routing represents unproven integration patterns
> - **Hardware Constraints:** 16GB VRAM imposes severe limitations on model selection and concurrent operations
> - **Integration Complexity:** 6+ specialist models, 3 multimodal pipelines, and novel validation ladder create combinatorial integration challenges
> - **Technical Debt Risk:** Novel patterns may require significant refactoring as understanding evolves

2. Assessment Scope and Methodology

2.1 Scope

This assessment covers technical risks arising from:

- System Architecture (bicameral GPU/CPU design, model orchestration)
- Data Architecture (memory ledger, multimodal pipelines)
- Routing Logic (Prolog-based constitutional routing)
- Validation Architecture (block-by-block validation, Wiggum Loop)

- Build Pipeline (Constitutional Build Pipeline, 9-phase workflow)
- Technology Stack (llama.cpp, Prolog, Python, Markdown)

2.2 Methodology

Following NIST SP 800-30 and ISO/IEC 25010:2011:

1. **Risk Identification:** Architecture review, threat modeling, assumption analysis
2. **Risk Analysis:** Likelihood and impact assessment based on technical factors
3. **Risk Evaluation:** Comparison against risk criteria from Risk Management Plan
4. **Risk Treatment:** Mitigation strategies and proof-of-concept requirements

3. Technology Maturity Risks

3.1 Prolog-Based Constitutional Routing

Risk Statement: The use of Prolog for production routing logic introduces risks related to team expertise, debugging difficulty, and integration complexity.

Factor	Assessment
Technology Maturity	Prolog is mature (50+ years) but niche; limited modern tooling and libraries
Team Expertise	Low - No team members have production Prolog experience
Integration Risk	High - Python-Prolog bridge (pyswip) adds dependency and potential failure points
Debugging Risk	High - Declarative logic debugging differs from imperative; traceability challenges
Performance Risk	Low - SWI-Prolog is performant for rule-based classification; latency target (<1s) achievable

Mitigation Requirements:

- **PoC Required:** Validate Python-Prolog integration with representative routing rules (Phase 0)
- **Training:** 2-3 day Prolog intensive for core developers (Phase 0-1)
- **Consulting:** Architecture review by Prolog expert (Phase 1)
- **Fallback:** Python rule engine implementation identified if Prolog proves intractable
- **Testing:** 200+ prompt routing test suite with >85% accuracy target

3.2 llama.cpp Model Serving

Risk Statement: Reliance on llama.cpp for model inference introduces risks related to quantization quality, memory management, and version stability.

Factor	Assessment

Technology Maturity	High - llama.cpp is actively maintained, widely used, production-ready
Quantization Risk	Medium - Q4_K_M quality varies by model; may not preserve required accuracy
Memory Management	High - Manual VRAM management required; OOM crashes possible
Version Stability	Low - Active development may introduce breaking changes

Mitigation Requirements:

- **PoC Required:** Validate each model's Q4_K_M quality on representative tasks (Phase 0)
- **Version Pinning:** Lock llama.cpp version; test updates in staging
- **Monitoring:** Real-time VRAM monitoring with pre-emptive OOM prevention
- **Fallback Quantization:** Q3_K_M and Q2_K options identified for each model

3.3 Multimodal Pipeline Technologies

Risk Statement: OCR, Vision, and Embedding pipelines depend on technology choices not yet finalized.

Pipeline	Options	Risk Assessment
OCR	Tesseract, PaddleOCR	Tesseract is mature but accuracy may fall short of 90% target; PaddleOCR has better accuracy but higher complexity
Vision	CLIP, BLIP, LLaVA	Model selection impacts VRAM usage; CLIP is lightweight but less descriptive; LLaVA is accurate but VRAM-heavy
Embeddings	FAISS, ChromaDB, numpy	FAISS is performant but adds dependency; ChromaDB is simpler but less scalable; numpy is lightweight but requires custom indexing

Mitigation Requirements:

- **Decision Required:** Finalize technology choices by Phase 4 start
- **Benchmarking:** Accuracy and performance benchmarks for each option
- **Abstraction Layer:** Design pluggable interface to allow technology swaps

4. Integration Risks

4.1 Component Integration Matrix

Integration Point	Components	Risk Level	Key Risks
Router-Worker	Prolog Router → llama.cpp Worker	High	Model swap latency, routing accuracy, JSON parsing failures
Worker-Validator	GPU Worker → CPU Validator	High	Bicameral synchronization, validation latency, format mismatches
Router-Memory	Prolog Router → Markdown Ledger	Medium	File I/O performance, atomic write failures, corruption
Multimodal-OCR	OCR Pipeline → Router/ Worker	Medium	Accuracy degradation, format conversion, provenance tracking
Build Pipeline	9 Phases → Wiggum Loop	Critical	Phase gate failures, infinite loops, artifact corruption
Memory-Version Control	Markdown → Git	Low	Merge conflicts (deferred to v2.0 multi-user)

4.2 Critical Integration: Build Pipeline

Risk: The Constitutional Build Pipeline's 9-phase workflow with Wiggum Loop represents the highest integration complexity. Phase dependencies, gate validations, and retry logic create multiple failure modes:

- **Phase Gate Failures:** Any phase failing validation blocks entire pipeline
- **Infinite Retry Loops:** Wiggum Loop without proper termination could consume infinite resources
- **Artifact Corruption:** Partial writes or failed commits could corrupt memory ledger
- **State Inconsistency:** Mismatches between project_state.md and actual implementation

Mitigation Requirements:

- **Hard Limits:** Maximum 3 retries per Wiggum iteration; maximum 5 minutes per phase
- **Atomic Writes:** File locking for all memory ledger operations
- **State Validation:** Checksum verification between project_state.md and filesystem
- **Escalation:** Automatic escalation to user after max retries
- **Recovery:** Rollback capability to last committed state

5. Scalability Risks

5.1 Hardware Scaling Limitations

Constraint	Current (v1.0)	Future Need	Risk
VRAM	16GB (Tesla A2)	32-48GB (multi-model)	Single Worker only; no concurrent models
RAM	128GB	256GB+ (warm pool)	Limited warm pool size (2-3 models)
Concurrency	Single-user	10+ concurrent users	Architecture requires redesign for v2.0
Storage	1TB NVMe	2TB+ (model vault)	Model storage limits specialist diversity

> **Scalability Assumption:** The architecture is designed to be extensible to multi-GPU and distributed deployments in v2.0, but this requires significant refactoring. Current design decisions (shared memory ledger, sequential execution) may create technical debt for horizontal scaling.

5.2 Technical Debt Risks

Design Decision	Technical Debt	Mitigation
Sequential task execution	Requires redesign for concurrent processing	Document extension points; design for future parallelism
Shared Markdown ledger	Becomes bottleneck with multi-user	Abstract storage interface; database backend option
Prolog routing	Team expertise gap for maintenance	Comprehensive documentation; training investment
Single GPU constraint	Worker model contention	Warm pool strategy; predictive loading

6. Performance Risks

6.1 Latency Risk Analysis

Operation	Target	Risk	Probability
Router classification	≤1s	Prolog query optimization; JSON parsing	Low
Model swap (RAM→VRAM)	≤3s	PCIe bandwidth; model size	Medium
Block validation	≤5s	CPU inference speed; prompt length	High
High-stakes task (end-to-end)	≤60s	Multiple blocks × validation + retries	High
OCR (per page)	≤5s	Image quality; OCR engine performance	Medium

> **Performance Risk:** High-stakes tasks with block-by-block validation are at highest risk of exceeding latency targets. A 100-line code generation with 10-line blocks requires 10 validations × 5s = 50s minimum, plus generation time and potential retries.

6.2 Throughput Risk Analysis

Current Design Throughput: ~30 tasks/hour (sequential)

Risk: Throughput is fundamentally limited by single-GPU, sequential execution. Even with perfect optimization, bicameral validation adds inherent latency that cannot be eliminated without architecture changes.

7. Dependency Risks

7.1 External Dependencies

Dependency	Type	Risk	Mitigation
llama.cpp	Open Source	Breaking changes; security vulnerabilities	Version pinning; security monitoring
SWI-Prolog	Open Source	Compatibility; maintenance	Stable release track; minimal feature use
Model Weights	HuggingFace	Availability; licensing changes	Local mirrors; license verification
Python Ecosystem	PyPI	Dependency conflicts; supply chain	Lock files; vulnerability scanning

7.2 Internal Dependencies

Dependency	Risk	Mitigation
Router accuracy	Incorrect routing cascades to all downstream components	≥85% accuracy target; fallback mechanisms
Validator accuracy	False positives block good outputs; false negatives pass errors	<5% FP, <3% FN targets; calibration testing
Memory ledger integrity	Corruption affects all model interactions	Atomic writes; Git versioning; backups

8. Proof-of-Concept Requirements

To mitigate technical risks, the following PoCs are required before phase progression:

PoC	Phase	Success Criteria	Risks Addressed
Python-Prolog Integration	0	≤1s latency; JSON I/O working; 100+ rules	R-003 (Prolog complexity)
Model Quality Validation	0	All models <5% hallucination on test set	R-002 (Model quality)
VRAM Management	0	No OOM with 16GB; graceful fallback	R-001 (GPU memory)
Validator Speed	0	≥3 tokens/sec on CPU	R-007 (CPU inference)
Wiggum Loop	2	Converges in ≤3 retries; no infinite loops	Build pipeline integration
OCR Pipeline	4	≥90% accuracy on test documents	R-008 (OCR accuracy)
End-to-End Latency	2	High-stakes task ≤60s p95	R-005 (Validation latency)

9. Risk Treatment Recommendations

9.1 Design Phase Actions

Risk	Treatment	Owner	Due Date
Prolog complexity	PoC; training; consulting	Solutions Architect	Phase 1 end
Integration complexity	Interface contracts; error handling spec	Technical Lead	Phase 1 end
Performance uncertainty	Benchmarking plan; load testing strategy	Performance Engineer	Phase 1 end
Technical debt	Extension point documentation; refactoring plan	Solutions Architect	Phase 1 end

9.2 Implementation Phase Monitoring

- **Weekly:** PoC progress review; risk register updates
- **Per Sprint:** Integration test results; performance metrics
- **Phase Gates:** Technical risk clearance before progression

10. Assumptions and Constraints Validation

10.1 Technical Assumptions

Assumption	Validation Method	Status
Q4_K_M quantization preserves quality	Phase 0 model quality PoC	Pending
CPU sustains ≥3 tokens/sec for Validator	Phase 0 inference benchmark	Pending

PCIe bandwidth sufficient for model swaps	Phase 0 swap time measurement	Pending
Prolog routing latency ≤1s	Phase 0 integration PoC	Pending
Markdown ledger performance adequate	File I/O benchmark	Pending

10.2 Constraint Impact Analysis

Constraint	Impact	Mitigation in Design
16GB VRAM	Single Worker; frequent swaps	Warm pool; predictive loading; model quantization
128GB RAM	Limited warm pool (2-3 models)	LRU eviction; domain-based priority
Single GPU	No parallel Workers	Sequential execution; bicameral separation
Sequential execution	Throughput limited	Quality over speed positioning

11. Conclusion

The Sovereign AI Infrastructure design introduces significant technical risks due to its novel architecture (bicameral GPU/CPU, Prolog routing, constitutional validation). The highest risks are:

1. **GPU Memory Exhaustion (R-001):** Critical - requires immediate PoC validation
2. **Model Quality (R-002):** Critical - system unusable if models hallucinate excessively
3. **Prolog Complexity (R-003):** High - team learning curve and integration risk
4. **Build Pipeline Integration:** Critical - 9-phase workflow is highest complexity point

> **Recommendation:** Proceed with Phase 1 design only if Phase 0 PoCs validate key assumptions (GPU memory management, model quality, Prolog integration). If any PoC fails, reconsider architecture or hardware requirements before committing to implementation.