Testing Strategy & Test Plan

Project: Sovereign AI Infrastructure
Document Version: 1.0
Date: February 11, 2026
Status: Draft for Review
Owner: QA Lead / Test Engineer

1. Executive Summary

This document defines the comprehensive testing strategy and detailed test plan for the Sovereign AI Infrastructure. The testing approach encompasses **five critical domains**:

1. **Unit Testing** - Component-level verification (Python + Prolog)
2. **Integration Testing** - Cross-component validation
3. **System Testing** - End-to-end pipeline verification
4. **Performance Testing** - Latency, throughput, and resource utilization
5. **Acceptance Testing** - User story validation

Key Testing Principles:

- All code must have corresponding Prolog-encoded specifications
- Test-Driven Development (TDD) is mandatory for domain logic
- Automated testing coverage target: ≥85%
- Every test must include provenance tracking

2. Testing Strategy Overview

2.1 Testing Pyramid

Level	Quantity	Execution Time	Tools
Unit Tests	500+ tests	< 5 minutes	pytest, SWI-Prolog test framework
Integration Tests	100+ tests	< 15 minutes	pytest, Docker Compose
System Tests	50+ scenarios	< 1 hour	Behave (BDD), Selenium
Performance Tests	20+ benchmarks	2-4 hours	Locust, k6, custom benchmarks

Acceptance Tests	30+ stories	As needed	Manual + automated validation

2.2 Test Environment Architecture

Environment	Hardware	Purpose	CI/CD Stage
Local Dev	Developer workstation	Unit testing, TDD	Pre-commit
CI Pipeline	GitHub Actions runners	Automated testing	On PR / Push
Staging	RTX 4090 + 96GB RAM	Integration, System tests	Post-merge
Performance Lab	RTX 4090 + 128GB RAM	Load, stress testing	Nightly
Production Mirror	Production-equivalent	Final validation	Pre-release

3. Unit Testing

3.1 Python Unit Tests

3.1.1 Router Component Tests

Test ID	Description	Input	Expected Output	Priority
ROUTER-001	Domain classification - Coding Architecture	"Design a microservice architecture for e-commerce"	domain: coding_architecture	**High**
ROUTER-002	Domain classification - Coding Implementation	"Implement a Python function to calculate Fibonacci"	domain: coding_implementation	**High**
ROUTER-003	Domain classification - Reasoning	"Explain the trade-offs between SQL and NoSQL"	domain: reasoning	**High**
ROUTER-004	Domain classification - Creative Writing	"Write a short story about AI"	domain: creative_writing	**High**
ROUTER-005	Stakes assessment - High stakes	Medical diagnosis request with patient data	stakes: high	**High**
ROUTER-006			model: qwen_coder_32b	**High**

	Model selection based on domain + stakes	domain: coding_architecture, stakes: high		
ROUTER-007	Tool detection - OCR required	Request with scanned document attachment	tools: ["ocr"]	**Medium**
ROUTER-008	Tool detection - Vision required	Request with image attachment	tools: ["vision"]	**Medium**
ROUTER-009	Confidence scoring calculation	Routing decision with uncertainty	confidence: 0.0-1.0	**High**
ROUTER-010	Fallback routing on low confidence	confidence: < 0.6	Escalate to human or conservative model	**High**

3.1.2 Validator Component Tests

Test ID	Description	Input	Expected Output	Priority
VALID-001	Syntax validation - Python code	Code block with syntax error	status: FAIL, reason: SyntaxError	**High**
VALID-002	Syntax validation - Valid Python	Valid Python function	status: PASS	**High**
VALID-003	Hallucination detection - Non-existent API	Code using fake library	status: FAIL, reason: Hallucinated API	**High**
VALID-004	Grounding validation - Missing citation	Factual claim without source	status: FAIL, reason: Missing grounding	**High**
VALID-005	Grounding validation - Valid citation	Factual claim with proper citation	status: PASS	**High**
VALID-006	Policy compliance - PII detection	Output containing email address	status: FAIL, reason: PII detected	**High**
VALID-007	Retry mechanism on validation failure	Failed block with correction	Regenerate with correction context	**Medium**
VALID-008	Max retry exceeded handling	3 consecutive failures	ValidationError exception	**Medium**

3.1.3 Memory Ledger Tests

Test ID	Description	Input	Expected Output	Priority

MEM-001	Append to scratchpad	Text entry	Entry appended with timestamp	**High**
MEM-002	Commit validated block	Validated code block	Block in project_state.md	**High**
MEM-003	Semantic search retrieval	Query: "database connection"	Relevant code blocks returned	**High**
MEM-004	Context window management	Context exceeding 4K tokens	Relevant context prioritized	**Medium**
MEM-005	Provenance tracking	Any memory operation	Source, timestamp, confidence recorded	**High**

3.2 Prolog Unit Tests

3.2.1 Domain Classification Logic

```
% Test: classify_domain/2 % File: tests/test_domain_classification.pl :- begin_tests(domain_classification). % Test coding_architecture detection test(coding_architecture_design) :- Request = "Design a microservice architecture for e-commerce with API gateway", classify_domain(Request, Domain), assertion(Domain == coding_architecture). % Test coding_implementation detection test(coding_implementation_function) :- Request = "Write a Python function to calculate factorial", classify_domain(Request, Domain), assertion(Domain == coding_implementation). % Test reasoning detection test(reasoning_explanation) :- Request = "Explain the difference between REST and GraphQL", classify_domain(Request, Domain), assertion(Domain == reasoning). % Test creative_writing detection test(creative_writing_story) :- Request = "Write a short story about artificial intelligence", classify_domain(Request, Domain), assertion(Domain == creative_writing). % Test unknown domain fallback test(unknown_domain) :- Request = "Random gibberish xyz123", classify_domain(Request, Domain), assertion(Domain == unknown). :- end_tests(domain_classification).
```

3.2.2 Stakes Assessment Logic

```
% Test: assess_stakes/2 % File: tests/test_stakes_assessment.pl :- begin_tests(stakes_assessment). % Test high stakes - medical context test(high_stakes_medical) :- Request = "Diagnose patient symptoms: chest pain, shortness of breath", assess_stakes(Request, Stakes), assertion(Stakes == high). % Test high stakes - financial test(high_stakes_financial) :- Request = "Calculate tax liability for $1M transaction", assess_stakes(Request, Stakes), assertion(Stakes == high). % Test medium stakes - business logic test(medium_stakes_business) :- Request = "Design database schema for inventory system", assess_stakes(Request, Stakes), assertion(Stakes == medium). % Test low stakes - experimentation test(low_stakes_experiment) :- Request = "Experiment with different sorting algorithms", assess_stakes(Request, Stakes), assertion(Stakes == low). :- end_tests(stakes_assessment).
```

3.2.3 Model Selection Logic

```
% Test: select_model/3 % File: tests/test_model_selection.pl :-
begin_tests(model_selection). % Test architecture specialist selection
test(architecture_specialist) :- Domain = coding_architecture, Stakes = high,
select_model(Domain, Stakes, Model), assertion(Model == qwen_coder_32b). % Test
implementation specialist selection test(implementation_specialist) :- Domain =
coding_implementation, Stakes = high, select_model(Domain, Stakes, Model),
assertion(Model == nemotron_30b). % Test reasoning specialist selection
test(reasoning_specialist) :- Domain = reasoning, Stakes = high,
select_model(Domain, Stakes, Model), assertion(Model == gpt_oss_20b). % Test
creative specialist selection test(creative_specialist) :- Domain =
creative_writing, Stakes = medium, select_model(Domain, Stakes, Model),
assertion(Model == mythomax_13b). % Test fallback on unknown domain
test(unknown_domain_fallback) :- Domain = unknown, Stakes = medium,
select_model(Domain, Stakes, Model), assertion(Model == qwen_coder_32b). %
Conservative default :- end_tests(model_selection).
```

4. Integration Testing

4.1 Router-Worker Integration

Test ID	Scenario	Components	Test Steps	Expected Result
INT-001	End-to-end code generation	Router, Model Loader, Worker	1. Submit coding request 2. Router classifies 3. Model loads 4. Worker generates	Valid code returned
INT-002	Model swap on domain change	Router, Model Loader	1. Architecture request (Qwen) 2. Implementation request (Nemotron) 3. Verify swap	Swap time < 3 seconds
INT-003	Warm pool utilization	Model Loader, Warm Pool	1. Pre-load model to RAM 2. Request that model 3. Measure load time	Load from warm pool < 1s
INT-004	Cold start handling	Model Loader, Disk	1. Clear warm pool 2. Request uncached model 3. Measure load time	Cold load < 15 seconds

4.2 Worker-Validator Integration

Test ID	Scenario	Components	Test Steps	Expected Result
INT-005	Block-by-block validation	Worker, Validator, Memory	1. Generate block 2. Validate 3. Commit if pass	Only valid blocks committed
INT-006	Validation failure retry	Worker, Validator	1. Generate invalid block 2. Validation fails 3. Retry with correction	Max 3 retries, then error
INT-007	Validation policy bypass	Worker, Router	1. Low stakes request 2. Verify no validation	Direct output, no validation

4.3 Multimodal Pipeline Integration

Test ID	Scenario	Components	Test Steps	Expected Result
INT-008	OCR + Generation	OCR Pipeline, Router, Worker	1. Upload scanned document 2. OCR extracts text 3. Router classifies 4. Worker responds	Response references document
INT-009	Vision + Generation	Vision Pipeline, Router, Worker	1. Upload image 2. Vision generates caption 3. Context includes caption	Response describes image
INT-010	Embedding retrieval	Embedding Pipeline, Memory	1. Query with semantic search 2. Retrieve similar memories	Relevant context included

4.4 Memory Ledger Integration

Test ID	Scenario	Components	Test Steps	Expected Result
INT-011	Scratchpad persistence	Memory, File System	1. Append to scratchpad 2. Restart system 3. Verify persistence	Entries preserved
INT-012	Project state accumulation	Memory, Validator	1. Validate multiple blocks 2. Check project_state.md	All blocks in order
INT-013	Vector store indexing	Memory, FAISS	1. Add entries 2. Build index 3. Semantic search	Correct similarity scores

5. System Testing

5.1 End-to-End Scenarios

5.1.1 Use Case 1: Autonomous Software Package Construction

Test ID	SYS-001
Objective	Verify complete package construction from requirements
Preconditions	System running, models loaded, empty project workspace
Input	"Build a Python CLI tool for CSV to JSON conversion with validation"
Steps	1. Submit request via API 2. Router classifies as coding_architecture + high stakes 3. Qwen Coder 32B loaded 4. Architecture design generated 5. Validator confirms structure 6. Implementation blocks generated 7. Each block validated 8. Project committed to memory 9. Final package delivered
Expected Result	- Complete Python package with: • main.py with CLI interface • converter.py with core logic • validator.py with CSV validation • tests/ directory with unit tests • requirements.txt • README.md with usage - All code passes validation - Provenance tracked for all blocks
Success Criteria	Package runs without errors, produces correct JSON output

5.1.2 Use Case 2: Compliant Document Analysis

Test ID	SYS-002
Objective	Verify document analysis with PII protection
Preconditions	System running, OCR pipeline available
Input	Scanned medical record PDF + "Summarize patient history"
Steps	1. Upload PDF via API 2. OCR extracts text with bounding boxes 3. Router detects document + high stakes

	4. GPT-OSS 20B loaded for reasoning 5. Analysis generated 6. Validator checks for PII 7. PII redacted if found 8. Grounding citations added
Expected Result	- Summary without PII (names, SSNs redacted) - Citations reference page/line numbers - Confidence scores for OCR regions - Uncertain regions flagged
Success Criteria	No PII in output, all claims cite source

5.1.3 Use Case 3: Multimodal Research

Test ID	SYS-003
Objective	Verify image + text combined processing
Preconditions	System running, Vision pipeline available
Input	Architecture diagram image + "Generate implementation code for this design"
Steps	1. Upload image via API 2. Vision pipeline generates caption 3. Caption added to context 4. Router classifies as coding_architecture 5. Qwen Coder generates implementation 6. Each block validated 7. Code committed to memory
Expected Result	- Code implements diagram structure - Component names match diagram - Relationships correctly coded - Vision caption in provenance
Success Criteria	Generated code matches diagram intent

5.2 Error Handling & Recovery

Test ID	Scenario	Trigger	Expected Behavior
SYS-ERR-001	Out of Memory recovery	Request exceeds VRAM	Unload current model, load smaller fallback, retry
SYS-ERR-002	Model load failure	Corrupted model file	Error logged, fallback model loaded, alert sent
SYS-ERR-003	Validation timeout	Validator hangs	Timeout after 30s, retry once, then fail gracefully

SYS-ERR-004	OCR failure	Unprocessable image	Error in response, request continues without OCR
SYS-ERR-005	Memory corruption	Invalid memory state	Detect on load, rebuild from git history

6. Performance Testing

6.1 Latency Benchmarks

Metric	Target	Test Method	Acceptance Criteria
Routing Decision	< 100ms	100 requests, measure p50/p95/p99	p95 < 100ms
Model Swap (Warm)	< 3 seconds	Swap between 5 models, measure time	Average < 3s, p95 < 5s
Model Swap (Cold)	< 15 seconds	Clear warm pool, load model	Average < 15s
Token Generation (GPU)	≥ 20 tok/s	Generate 1000 tokens, measure rate	Average ≥ 20 tok/s
Token Generation (CPU)	≥ 5 tok/s	Router/Validator inference	Average ≥ 5 tok/s
Validation Block	< 2 seconds	Validate 100 code blocks	p95 < 2s
OCR Processing	< 5s per page	Process 50 pages	Average < 5s
Vision Encoding	< 3s per image	Process 50 images	Average < 3s
Embedding Retrieval	< 100ms	1000 semantic searches	p95 < 100ms
End-to-End Request	< 30s (simple)	Complete request with validation	p95 < 30s

6.2 Load Testing

Test ID	Scenario	Load Profile	Duration	Success Criteria
LOAD-001	Steady state	10 concurrent users, 1 req/s each	1 hour	Error rate < 1%, latency stable
LOAD-002	Ramp up	0 → 50 users over 10 minutes	30 minutes	No crashes, graceful degradation
LOAD-003	Spike test	100 users instantly for 5 minutes	10 minutes	Recover within 2 minutes

LOAD-004	Model swap stress	Rapid domain switching	30 minutes	Swap times remain < 5s

6.3 Resource Utilization

Resource	Target Utilization	Test Method
GPU VRAM	90-95%	Monitor during load tests
System RAM	< 80%	Monitor during load tests
CPU	< 70% (average)	Monitor during load tests
Disk I/O	< 50 MB/s sustained	Monitor during cold starts

7. Acceptance Testing

7.1 User Story Validation

Story ID	User Story	Acceptance Criteria	Test Method
US-001	As a developer, I want validated code generation so that I can trust the output	• Code compiles without errors • Tests pass (if included) • No hallucinated APIs • Grounding citations present	Generate 50 code samples, verify criteria
US-002	As a compliance officer, I want PII detection so that sensitive data is protected	• 100% of PII detected • False positive rate < 5% • Redaction log maintained	Test with 100 documents containing PII
US-003	As a researcher, I want transparent routing so that I understand model selection	• Routing decision visible • Confidence score shown • Model name disclosed • Reasoning explained	Verify UI/API shows all routing metadata
US-004	As an architect, I want incremental construction so that I can review each component	• Blocks generated one at a time • Each block validated • Project state visible • Rollback capability	Build complex project, verify incremental flow
US-005	As a user, I want multimodal input so that I can include documents and images	• OCR extracts text accurately (>90%) • Vision generates captions • Context includes multimodal data • Provenance tracked	Test with 50 documents and 50 images

US-006	As an operator, I want audit trails so that I can review all decisions	• Every request logged • Routing decisions recorded • Validation results stored • Memory operations tracked	Verify log completeness for 100 requests

7.2 Quality Attributes Validation

Attribute	Target	Measurement
Accuracy - Code	≥ 95% syntactically valid	Automated validation on 1000 samples
Accuracy - Routing	≥ 90% correct model selection	Human review of 200 routing decisions
Accuracy - OCR	≥ 90% character accuracy	Compare OCR output to ground truth
Reliability	≥ 99.5% uptime	Monitor over 30 days
Security	Zero critical vulnerabilities	Penetration testing, code audit
Usability	Task completion > 90%	User testing with 10 participants

8. Test Automation

8.1 CI/CD Pipeline Integration

```
# .github/workflows/test.yml name: Test Suite on: [push, pull_request] jobs: unit-tests: runs-on: ubuntu-latest steps: - uses: actions/checkout@v3 - name: Set up Python uses: actions/setup-python@v4 with: python-version: '3.10' - name: Install dependencies run: | pip install -r requirements.txt pip install -r requirements-test.txt - name: Run Python unit tests run: pytest tests/unit --cov=src --cov-report=xml - name: Set up SWI-Prolog run: | sudo apt-get install swi-prolog - name: Run Prolog unit tests run: | cd tests/prolog swipl -s run_tests.pl -g run_tests -t halt - name: Upload coverage uses: codecov/codecov-action@v3 integration-tests: runs-on: ubuntu-latest needs: unit-tests services: llama-router: image: llama-cpp-server ports: - 8001:8001 llama-validator: image: llama-cpp-server ports: - 8002:8002 steps: - uses: actions/checkout@v3 - name: Run integration tests run: pytest tests/integration -v performance-tests: runs-on: self-hosted-gpu needs: integration-tests if: github.ref == 'refs/heads/main' steps: - uses: actions/checkout@v3 - name: Run performance benchmarks run: | pytest tests/performance --benchmark-only python scripts/analyze_performance.py
```

8.2 Test Data Management

Data Type	Source	Size	Refresh Frequency
Code samples	GitHub public repos	10,000 files	Monthly
OCR test images	Synthetic + public datasets	1,000 images	Quarterly
Vision test images	COCO subset	500 images	Quarterly
PII test documents	Synthetic (no real PII)	200 documents	As needed

8.3 Coverage Requirements

Component	Line Coverage	Branch Coverage	Prolog Clause Coverage
Router	≥ 90%	≥ 85%	≥ 95%
Validator	≥ 90%	≥ 85%	N/A
Memory Ledger	≥ 85%	≥ 80%	N/A
Model Loader	≥ 85%	≥ 80%	N/A
Prolog Domain Logic	N/A	N/A	≥ 95%

9. Test Schedule

Phase	Activity	Duration	Deliverables
Week 1-2	Unit test development	2 weeks	Python + Prolog unit tests
Week 3	Integration test development	1 week	Integration test suite
Week 4	System test development	1 week	BDD scenarios
Week 5	Performance baseline	1 week	Benchmark results
Week 6-7	Test execution (Phase 1)	2 weeks	Test reports, bug reports
Week 8	Bug fixing & retest	1 week	Verified fixes
Week 9	Acceptance testing	1 week	Sign-off

10. Risks & Mitigations

Risk	Impact	Probability	Mitigation
GPU availability for testing	High	Medium	Cloud GPU instances, local RTX 4090 backup
Model output non-determinism	Medium	High	Seed control, statistical validation, retry logic
Test data quality issues	Medium	Medium	Curated datasets, synthetic data generation
Performance variance	Low	High	Multiple runs, statistical analysis, tolerance bands
Prolog test coverage gaps	Medium	Medium	Code review, clause coverage analysis

11. Appendices

Appendix A: Test Tools & Frameworks

Category	Tool	Purpose
Python Unit Testing	pytest	Test execution, fixtures, parametrization
Python Coverage	pytest-cov	Code coverage measurement

Prolog Testing	plunit	SWI-Prolog unit testing framework
BDD	Behave	Gherkin-style acceptance tests
Load Testing	Locust	Python-based load generation
API Testing	HTTPie, curl	Manual API verification
Performance	pytest-benchmark	Benchmark test execution
Mocking	unittest.mock	Component isolation

Appendix B: Test Environment Specifications

Component	Specification
GPU	NVIDIA RTX 4090 (24GB VRAM)
CPU	AMD Ryzen 9 7950X or equivalent
RAM	96GB DDR5
Storage	2TB NVMe SSD
OS	Ubuntu 22.04 LTS
Python	3.10+
SWI-Prolog	9.0+
CUDA	12.1+

Appendix C: Entry/Exit Criteria

Entry Criteria (Testing Can Begin)

- All Phase 1 design documents approved
- Development environment provisioned
- Test data prepared and validated
- CI/CD pipeline configured
- Test tools installed and verified

Exit Criteria (Phase 1 Complete)

- All unit tests passing (≥ 85% coverage)
- All integration tests passing
- All system test scenarios executed
- Performance benchmarks meet targets

- Critical and high bugs resolved
- Acceptance criteria validated
- Test documentation complete
- Stakeholder sign-off obtained

> **Note:** This test plan should be reviewed and updated at the end of each testing phase. Metrics and targets may be adjusted based on findings during execution.

End of Document

Sovereign AI Infrastructure - Testing Strategy & Test Plan v1.0