Domain Logic Specification

Project: Sovereign AI Infrastructure
Document Version: 1.0
Date: February 11, 2026
Status: Draft for Review
Owner: Domain Logic Architect

1. Executive Summary

This document specifies the **domain logic** governing the Sovereign AI Infrastructure. The domain logic is implemented primarily in **SWI-Prolog**, serving as the "constitutional law" of the system. It defines:

- **Domain Classification** - Categorizing requests into semantic domains
- **Stakes Assessment** - Evaluating risk and complexity levels
- **Model Selection** - Mapping domain/stakes to optimal models
- **Validation Policy** - Determining validation depth and requirements
- **Tool Detection** - Identifying required multimodal tools
- **Confidence Scoring** - Quantifying routing decision certainty

> **Core Principle:** The Router is the Constitution; models are civil servants. No Python code may override a Prolog decision. If Prolog says validator=true, the pipeline fails if validation is skipped.

2. Domain Taxonomy

2.1 Domain Categories

Domain	Description	Keywords (Examples)	Primary Model
coding_architecture	High-level software design, system architecture, refactoring planning	design, architecture, refactor, structure, pattern, microservice	qwen_coder_32b
coding_implementation			nemotron_30b

	Code writing, function implementation, debugging	implement, write code, function, class, debug, fix	
reasoning	Analysis, explanation, comparison, planning	explain, compare, analyze, why, how, trade-off	gpt_oss_20b
creative_writing	Stories, narratives, creative content, tone adjustment	story, write, creative, narrative, poem, tone	mythomax_13b
documentation	Technical docs, comments, README generation	document, readme, comment, explain code	qwen_coder_32b
unknown	Unclassifiable or ambiguous requests	N/A (fallback)	qwen_coder_32b

2.2 Domain Classification Rules

```
classify_domain(Request, Domain)
```

Rule 1: Keyword Threshold

A request is classified into a domain if it contains ≥2 keywords associated with that domain.

Rule 2: Precedence Order

If multiple domains match, apply precedence:

1. coding_architecture
2. coding_implementation
3. reasoning
4. creative_writing
5. documentation
6. unknown (fallback)

Rule 3: Embedding Fallback

If no domain matches by keywords, use semantic similarity search against domain exemplars.

```
% Domain Classification Predicates
% File: router.pl

% Domain keyword definitions
domain_keywords(coding_architecture, [
    'design', 'architecture', 'refactor', 'structure', 'pattern',
```

```
    'microservice', 'system design', 'class diagram', 'module'
]).

domain_keywords(coding_implementation, [
    'implement', 'write code', 'function', 'class', 'debug',
    'fix', 'method', 'algorithm', 'optimize code'
]).

domain_keywords(reasoning, [
    'explain', 'compare', 'analyze', 'why', 'how', 'trade-off',
    'difference between', 'advantages', 'disadvantages'
]).

domain_keywords(creative_writing, [
    'story', 'write', 'creative', 'narrative', 'poem',
    'tone', 'style', 'imagine', 'fiction'
]).

domain_keywords(documentation, [
    'document', 'readme', 'comment', 'explain code',
    'api doc', 'tutorial', 'guide'
]).

% Main classification predicate
classify_domain(Request, Domain) :-
    string_lower(Request, LowerRequest),
    findall(D, matching_domain(LowerRequest, D), Domains),
    sort(Domains, UniqueDomains),
    select_by_precedence(UniqueDomains, Domain).

% Check if request matches domain (>=2 keywords)
matching_domain(Request, Domain) :-
    domain_keywords(Domain, Keywords),
    count_matching_keywords(Request, Keywords, Count),
    Count >= 2.

% Count matching keywords
count_matching_keywords(Request, Keywords, Count) :-
    include(keyword_in_request(Request), Keywords, Matches),
    length(Matches, Count).

keyword_in_request(Request, Keyword) :-
    sub_string(Request, _, _, _, Keyword).

% Precedence-based selection
select_by_precedence([], unknown).
select_by_precedence(Domains, Domain) :-
    precedence_order(Ordered),
    member(Domain, Ordered),
    member(Domain, Domains), !.

precedence_order([
    coding_architecture,
    coding_implementation,
    reasoning,
    creative_writing,
```

```
    documentation
]).
```

3. Stakes Assessment

3.1 Stakes Levels

Level	Score Range	Description	Validation Required
High	≥ 5	Production code, medical, financial, legal, safety-critical	Block-by-block
Medium	2-4	Business logic, internal tools, documentation	End-stage
Low	< 2	Experiments, prototypes, creative exploration	None

3.2 Stakes Calculation

```
assess_stakes(Request, Stakes)
```

Formula: Stakes Score = Sum(Complexity Indicators) + Sum(Risk Indicators)

Complexity Indicators (+1 each):

- Multi-file changes implied
- External dependencies mentioned
- Database schema involved
- API integration required
- Complex algorithm requested

Risk Indicators (+2 each):

- Medical/health context
- Financial/monetary impact
- Legal/compliance requirements
- PII/sensitive data handling
- Safety-critical system

> **Override Rule:** coding_architecture is always at least High Stakes (Score ≥ 3), regardless of calculated score.

```
% Stakes Assessment Predicates
% File: router.pl
```

```
% Complexity indicators (+1 each)
complexity_indicator(Request, 'multi_file') :-
    sub_string(Request, _, _, _, 'multiple files').
complexity_indicator(Request, 'multi_file') :-
    sub_string(Request, _, _, _, 'across modules').

complexity_indicator(Request, 'dependencies') :-
    sub_string(Request, _, _, _, 'import').
complexity_indicator(Request, 'dependencies') :-
    sub_string(Request, _, _, _, 'library').

complexity_indicator(Request, 'database') :-
    sub_string(Request, _, _, _, 'database').
complexity_indicator(Request, 'database') :-
    sub_string(Request, _, _, _, 'schema').

complexity_indicator(Request, 'api') :-
    sub_string(Request, _, _, _, 'api').
complexity_indicator(Request, 'api') :-
    sub_string(Request, _, _, _, 'endpoint').

complexity_indicator(Request, 'algorithm') :-
    sub_string(Request, _, _, _, 'algorithm').
complexity_indicator(Request, 'algorithm') :-
    sub_string(Request, _, _, _, 'complex').

% Risk indicators (+2 each)
risk_indicator(Request, 'medical') :-
    sub_string(Request, _, _, _, 'patient').
risk_indicator(Request, 'medical') :-
    sub_string(Request, _, _, _, 'diagnosis').
risk_indicator(Request, 'medical') :-
    sub_string(Request, _, _, _, 'medical').

risk_indicator(Request, 'financial') :-
    sub_string(Request, _, _, _, 'money').
risk_indicator(Request, 'financial') :-
    sub_string(Request, _, _, _, 'transaction').
risk_indicator(Request, 'financial') :-
    sub_string(Request, _, _, _, 'payment').

risk_indicator(Request, 'legal') :-
    sub_string(Request, _, _, _, 'legal').
risk_indicator(Request, 'legal') :-
    sub_string(Request, _, _, _, 'compliance').
risk_indicator(Request, 'legal') :-
    sub_string(Request, _, _, _, 'regulation').

risk_indicator(Request, 'pii') :-
    sub_string(Request, _, _, _, 'personal data').
risk_indicator(Request, 'pii') :-
    sub_string(Request, _, _, _, 'ssn').
risk_indicator(Request, 'pii') :-
    sub_string(Request, _, _, _, 'email').

risk_indicator(Request, 'safety') :-
```

```
    sub_string(Request, _, _, _, 'safety').
risk_indicator(Request, 'safety') :-
    sub_string(Request, _, _, _, 'critical system').

% Main stakes assessment
assess_stakes(Request, Stakes) :-
    string_lower(Request, LowerRequest),

    % Count complexity indicators (+1 each)
    findall(C, complexity_indicator(LowerRequest, C), Complexity),
    length(Complexity, ComplexityScore),

    % Count risk indicators (+2 each)
    findall(R, risk_indicator(LowerRequest, R), Risks),
    length(Risks, RiskCount),
    RiskScore is RiskCount * 2,

    % Calculate total
    TotalScore is ComplexityScore + RiskScore,

    % Map to stakes level
    score_to_stakes(TotalScore, Stakes).

% Score to stakes mapping
score_to_stakes(Score, high) :- Score >= 5, !.
score_to_stakes(Score, medium) :- Score >= 2, Score < 5, !.
score_to_stakes(_, low).

% Stakes override for architecture domain
stakes_with_override(Domain, BaseStakes, high) :-
    Domain = coding_architecture,
    (BaseStakes = high ; BaseStakes = medium ; BaseStakes = low), !.
stakes_with_override(_, Stakes, Stakes).
```

4. Model Selection Logic

4.1 Model Proficiency Matrix

Model	coding_architecture	coding_implementation	reasoning	creative_writing	docun
qwen_coder_32b	0.95	0.85	0.70	0.40	0.90
nemotron_30b	0.75	0.95	0.65	0.35	0.70
gpt_oss_20b	0.70	0.60	0.90	0.75	0.80
mythomax_13b	0.30	0.40	0.60	0.95	0.65

4.2 Model Selection Rules

```
select_model(Domain, Stakes, Model)
```

Primary Rule: Select the model with the highest proficiency score for the determined domain.

Fallback Rule: For unknown domains, default to qwen_coder_32b (most versatile).

```
% Model Selection Predicates
% File: router.pl

% Model proficiency scores
model_proficiency(qwen_coder_32b, coding_architecture, 0.95).
model_proficiency(qwen_coder_32b, coding_implementation, 0.85).
model_proficiency(qwen_coder_32b, reasoning, 0.70).
model_proficiency(qwen_coder_32b, creative_writing, 0.40).
model_proficiency(qwen_coder_32b, documentation, 0.90).

model_proficiency(nemotron_30b, coding_architecture, 0.75).
model_proficiency(nemotron_30b, coding_implementation, 0.95).
model_proficiency(nemotron_30b, reasoning, 0.65).
model_proficiency(nemotron_30b, creative_writing, 0.35).
model_proficiency(nemotron_30b, documentation, 0.70).

model_proficiency(gpt_oss_20b, coding_architecture, 0.70).
model_proficiency(gpt_oss_20b, coding_implementation, 0.60).
model_proficiency(gpt_oss_20b, reasoning, 0.90).
model_proficiency(gpt_oss_20b, creative_writing, 0.75).
model_proficiency(gpt_oss_20b, documentation, 0.80).

model_proficiency(mythomax_13b, coding_architecture, 0.30).
model_proficiency(mythomax_13b, coding_implementation, 0.40).
model_proficiency(mythomax_13b, reasoning, 0.60).
model_proficiency(mythomax_13b, creative_writing, 0.95).
```

```
model_proficiency(mythomax_13b, documentation, 0.65).

% Available models
available_model(qwen_coder_32b).
available_model(nemotron_30b).
available_model(gpt_oss_20b).
available_model(mythomax_13b).

% Model selection with proficiency ranking
select_model(Domain, _Stakes, Model) :-
    Domain \= unknown,
    findall(Score-Mod,
            (available_model(Mod), model_proficiency(Mod, Domain, Score)),
            ScoredModels),
    sort(ScoredModels, Sorted),
    reverse(Sorted, [_BestScore-Model | _]), !.

% Fallback for unknown domain
select_model(unknown, _Stakes, qwen_coder_32b) :- !.

% Alternative: Stakes-aware selection (conservative for high stakes)
select_model_conservative(Domain, high, Model) :-
    % For high stakes, prefer larger models even if slightly less proficient
    Domain = coding_implementation,
    Model = qwen_coder_32b, !.  % Prefer 32B over Nemotron for safety
select_model_conservative(Domain, _Stakes, Model) :-
    select_model(Domain, _Stakes, Model).
```

5. Validation Policy

5.1 Validation Levels

Level	Granularity	Applied To	Description
block_by_block	5-10 lines / 1 function	High Stakes	Validate every logical block before proceeding
end_stage	Complete output	Medium Stakes	Validate entire output once complete
none	N/A	Low Stakes	No validation required

5.2 Validation Depth by Domain

Domain	High Stakes	Medium Stakes	Low Stakes
coding_architecture	Per-line	Per-function	None
coding_implementation	Per-function	Per-file	None
reasoning	Per-paragraph	Per-output	None

creative_writing	Per-section	Per-output	None
documentation	Per-section	Per-output	None

```
% Validation Policy Predicates
% File: router.pl

% Validation policy by stakes
determine_validation_policy(high, block_by_block).
determine_validation_policy(medium, end_stage).
determine_validation_policy(low, none).

% Validation depth by domain and stakes
validation_depth(coding_architecture, high, per_line).
validation_depth(coding_architecture, medium, per_function).
validation_depth(coding_architecture, low, none).

validation_depth(coding_implementation, high, per_function).
validation_depth(coding_implementation, medium, per_file).
validation_depth(coding_implementation, low, none).

validation_depth(reasoning, high, per_paragraph).
validation_depth(reasoning, medium, per_output).
validation_depth(reasoning, low, none).

validation_depth(creative_writing, high, per_section).
validation_depth(creative_writing, medium, per_output).
validation_depth(creative_writing, low, none).

validation_depth(documentation, high, per_section).
validation_depth(documentation, medium, per_output).
validation_depth(documentation, low, none).

validation_depth(unknown, _, per_output).  % Conservative default

% Validator requirement check
validator_required(high, true).
validator_required(medium, true).
validator_required(low, false).

% Get complete validation configuration
get_validation_config(Domain, Stakes, Config) :-
    determine_validation_policy(Stakes, Policy),
    validation_depth(Domain, Stakes, Depth),
    validator_required(Stakes, Required),
    Config = validation_config{
        policy: Policy,
        depth: Depth,
        validator_required: Required
    }.
```

6. Tool Detection

6.1 Available Tools

Tool	Trigger Conditions	Input	Output
ocr	PDF/Image attachment + keywords (scan, document, extract)	Image/ PDF file	Extracted text with bounding boxes
vision	Image attachment + keywords (image, visual, describe)	Image file	Caption, objects, features
embeddings	Retrieval keywords (find similar, context, related) OR unknown domain	Text query	Similar documents from memory

6.2 Tool Detection Rules

```
detect_tools(Request, Attachments, Tools)
```

```
% Tool Detection Predicates
% File: router.pl

% OCR detection rules
tool_required(ocr, Request, Attachments) :-
    member(File, Attachments),
    (file_type(File, pdf) ; file_type(File, image)),
    ocr_keywords(Request).

ocr_keywords(Request) :-
    string_lower(Request, Lower),
    (sub_string(Lower, _, _, _, 'scan') ;
     sub_string(Lower, _, _, _, 'document') ;
     sub_string(Lower, _, _, _, 'extract text') ;
     sub_string(Lower, _, _, _, 'ocr')).

% Vision detection rules
tool_required(vision, Request, Attachments) :-
    member(File, Attachments),
    file_type(File, image),
    vision_keywords(Request).

vision_keywords(Request) :-
    string_lower(Request, Lower),
    (sub_string(Lower, _, _, _, 'image') ;
     sub_string(Lower, _, _, _, 'visual') ;
     sub_string(Lower, _, _, _, 'describe') ;
     sub_string(Lower, _, _, _, 'what is in') ;
     sub_string(Lower, _, _, _, 'picture')).

% Embeddings detection rules
```

```
tool_required(embeddings, Request, _Attachments) :-
    embedding_keywords(Request).

tool_required(embeddings, _Request, _Attachments, unknown) :-
    % Always use embeddings for unknown domain
    true.

embedding_keywords(Request) :-
    string_lower(Request, Lower),
    (sub_string(Lower, _, _, _, 'similar') ;
     sub_string(Lower, _, _, _, 'related') ;
     sub_string(Lower, _, _, _, 'find') ;
     sub_string(Lower, _, _, _, 'context') ;
     sub_string(Lower, _, _, _, 'retrieve')).

% File type detection
file_type(File, pdf) :-
    sub_string(File, _, _, _, '.pdf').
file_type(File, image) :-
    (sub_string(File, _, _, _, '.jpg') ;
     sub_string(File, _, _, _, '.jpeg') ;
     sub_string(File, _, _, _, '.png') ;
     sub_string(File, _, _, _, '.gif')).

% Main tool detection
detect_tools(Request, Attachments, Tools) :-
    findall(Tool, tool_required(Tool, Request, Attachments), Tools).

detect_tools(Request, Attachments, Domain, Tools) :-
    findall(Tool,
            (tool_required(Tool, Request, Attachments) ;
             (Domain = unknown, Tool = embeddings)),
            Tools).
```

7. Confidence Scoring

7.1 Confidence Levels

Level	Score Range	Action
Acceptable	≥ 0.75	Proceed with routing decision
Uncertain	0.60 - 0.74	Trigger embedding fallback for similar past routes
Critical	< 0.60	Halt and request user clarification

7.2 Confidence Calculation

```
calculate_confidence(Domain, Stakes, Model, Confidence)
```

Factors Affecting Confidence:

- **Domain Clarity** (0.0-1.0): How clearly the request matches domain keywords
- **Keyword Match Ratio** (0.0-1.0): Matched keywords / total domain keywords
- **Model Proficiency** (0.0-1.0): Selected model's proficiency for domain
- **Historical Success** (0.0-1.0): Past success rate for similar routes

```
% Confidence Scoring Predicates
% File: router.pl

% Calculate confidence for routing decision
calculate_confidence(Request, Domain, Stakes, Model, Confidence) :-
    % Domain clarity (based on keyword match strength)
    domain_clarity(Request, Domain, DomainClarity),

    % Model proficiency for domain
    model_proficiency(Model, Domain, ModelProficiency),

    % Stakes clarity (higher stakes = more conservative confidence)
    stakes_clarity(Stakes, StakesClarity),

    % Historical success (from memory, default 0.8)
    historical_success(Domain, Stakes, Model, HistoricalSuccess),

    % Weighted combination
    Confidence is (DomainClarity * 0.3) +
                  (ModelProficiency * 0.3) +
                  (StakesClarity * 0.2) +
                  (HistoricalSuccess * 0.2).

% Domain clarity calculation
domain_clarity(Request, Domain, Clarity) :-
    domain_keywords(Domain, Keywords),
    count_matching_keywords(Request, Keywords, MatchCount),
    length(Keywords, TotalKeywords),
    (TotalKeywords > 0 ->
        RawClarity is MatchCount / TotalKeywords,
        min(RawClarity, 1.0, Clarity)
    ;
        Clarity = 0.5  % Default for unknown domains
    ).

% Stakes clarity (higher stakes require more certainty)
stakes_clarity(high, 0.7).    % Conservative for high stakes
stakes_clarity(medium, 0.85).
stakes_clarity(low, 0.95).    % Confident for low stakes

% Historical success (placeholder - would query memory)
historical_success(_Domain, _Stakes, _Model, 0.8).  % Default assumption

% Confidence level determination
confidence_level(Confidence, acceptable) :- Confidence >= 0.75, !.
confidence_level(Confidence, uncertain) :- Confidence >= 0.60, !.
confidence_level(_, critical).
```

```
% Action based on confidence
confidence_action(acceptable, proceed).
confidence_action(uncertain, embedding_fallback).
confidence_action(critical, request_clarification).
```

8. Main Routing Predicate

8.1 Route/2 - The Constitutional Entry Point

```
route(Request, Decision)
```

This is the primary entry point for all routing decisions. It encapsulates all domain logic and returns a complete routing decision.

```
% Main Routing Predicate
% File: router.pl
% This is the CONSTITUTION - no Python code may override these decisions

% Route/2: Main entry point
% Request: string or request{} structure
% Decision: decision{} structure with all routing parameters

route(Request, Decision) :-
    % Step 1: Extract request components
    extract_request_components(Request, Text, Attachments, Context),

    % Step 2: Classify domain
    classify_domain(Text, Domain),

    % Step 3: Assess stakes
    assess_stakes(Text, RawStakes),
    stakes_with_override(Domain, RawStakes, Stakes),

    % Step 4: Select model
    select_model(Domain, Stakes, Model),

    % Step 5: Determine validation policy
    get_validation_config(Domain, Stakes, ValidationConfig),

    % Step 6: Detect required tools
    detect_tools(Text, Attachments, Domain, Tools),

    % Step 7: Calculate confidence
    calculate_confidence(Text, Domain, Stakes, Model, Confidence),
    confidence_level(Confidence, ConfidenceLevel),
    confidence_action(ConfidenceLevel, Action),

    % Step 8: Assemble decision
    Decision = decision{
        domain: Domain,
        stakes: Stakes,
        recommended_model: Model,
        validation_policy: ValidationConfig.policy,
        validation_depth: ValidationConfig.depth,
        validator_required: ValidationConfig.validator_required,
        tools_required: Tools,
```

```
        confidence: Confidence,
        confidence_level: ConfidenceLevel,
        action: Action,
        timestamp: _CurrentTime
    }.

% Extract components from request (handles both string and dict formats)
extract_request_components(Request, Text, Attachments, Context) :-
    is_dict(Request), !,
    get_dict(text, Request, Text),
    get_dict(attachments, Request, Attachments),
    get_dict(context, Request, Context).

extract_request_components(Text, Text, [], []) :-
    string(Text), !.

extract_request_components(Atom, Text, [], []) :-
    atom(Atom),
    atom_string(Atom, Text).
```

8.2 Decision Structure Schema

Field	Type	Description	Example
domain	atom	Classified domain	coding_architecture
stakes	atom	Risk level	high
recommended_model	atom	Selected worker model	qwen_coder_32b
validation_policy	atom	Validation approach	block_by_block
validation_depth	atom	Granularity of validation	per_function
validator_required	boolean	Whether validator must run	true
tools_required	list	Required multimodal tools	[ocr, embeddings]
confidence	float	Decision confidence score	0.87
confidence_level	atom	Confidence category	acceptable
action	atom	Recommended action	proceed
timestamp	datetime	Decision time	2026-02-11T10:30:00Z

9. Python-Prolog Integration

9.1 Interface Contract

The Python orchestrator queries Prolog through the pyswip library. The interface is strictly one-way: Python queries, Prolog decides.

```
# Python-Prolog Interface
# File: src/router/prolog_interface.py

from pyswip import Prolog
import json

class PrologRouter:
    """
    Python interface to the Prolog routing constitution.

    RULE: This class only QUERIES Prolog. It never overrides decisions.
    """

    def __init__(self, prolog_file='router.pl'):
        self.prolog = Prolog()
        self.prolog.consult(prolog_file)

    def route(self, request_text: str, attachments: list = None) -> dict:
        """
        Query Prolog for routing decision.

        Args:
            request_text: The user's request
            attachments: List of attached file paths

        Returns:
            Decision dict with all routing parameters
        """
        # Build Prolog query
        request_term = f"route({self._to_prolog_term(request_text)}, Decision)"

        # Execute query
        results = list(self.prolog.query(request_term))

        if not results:
            raise RoutingError("Prolog routing failed")

        # Extract decision from Prolog result
        decision_term = results[0]['Decision']
        return self._decision_to_dict(decision_term)

    def _to_prolog_term(self, text: str) -> str:
        """Escape string for Prolog."""
        escaped = text.replace("'", "\\'")
        return f"'{escaped}'"

    def _decision_to_dict(self, term) -> dict:
```

```
        """Convert Prolog decision term to Python dict."""
        # Extract fields from Prolog dict term
        return {
            'domain': str(term['domain']),
            'stakes': str(term['stakes']),
            'recommended_model': str(term['recommended_model']),
            'validation_policy': str(term['validation_policy']),
            'validation_depth': str(term['validation_depth']),
            'validator_required': bool(term['validator_required']),
            'tools_required': list(term['tools_required']),
            'confidence': float(term['confidence']),
            'confidence_level': str(term['confidence_level']),
            'action': str(term['action'])
        }
```

9.2 Enforcement Rules

> **CRITICAL ENFORCEMENT RULES:**
>
> 1. Python code CANNOT override a Prolog routing decision
> 2. If Prolog says validator_required=true, validation MUST occur
> 3. If Prolog says action=request_clarification, pipeline MUST halt
> 4. Python may only add operational context (e.g., current system load)
> 5. All overrides must be logged as "Constitutional Violations"

10. Test-Driven Development (TDD) Integration

10.1 Prolog Test Structure

Every domain logic predicate must have corresponding unit tests in Prolog using the plunit framework.

```
% Test Suite for Domain Logic
% File: tests/test_router.pl

:- begin_tests(router).

% Test domain classification
test(classify_architecture) :-
    Request = "Design a microservice architecture",
    classify_domain(Request, Domain),
    assertion(Domain == coding_architecture).

test(classify_implementation) :-
    Request = "Implement a Python function",
    classify_domain(Request, Domain),
    assertion(Domain == coding_implementation).

% Test stakes assessment
```

```
test(high_stakes_medical) :-
    Request = "Patient diagnosis system",
    assess_stakes(Request, Stakes),
    assertion(Stakes == high).

test(low_stakes_experiment) :-
    Request = "Experiment with colors",
    assess_stakes(Request, Stakes),
    assertion(Stakes == low).

% Test model selection
test(select_architecture_model) :-
    select_model(coding_architecture, high, Model),
    assertion(Model == qwen_coder_32b).

test(select_creative_model) :-
    select_model(creative_writing, medium, Model),
    assertion(Model == mythomax_13b).

% Test validation policy
test(high_stakes_requires_validator) :-
    validator_required(high, Required),
    assertion(Required == true).

test(low_stakes_no_validator) :-
    validator_required(low, Required),
    assertion(Required == false).

% Test confidence calculation
test(confidence_acceptable) :-
    calculate_confidence("Design a system", coding_architecture,
                         high, qwen_coder_32b, Confidence),
    Confidence >= 0.75.

% Test full routing
test(full_route) :-
    Request = "Design a Python API",
    route(Request, Decision),
    get_dict(domain, Decision, coding_architecture),
    get_dict(recommended_model, Decision, qwen_coder_32b).

:- end_tests(router).
```

10.2 Running Prolog Tests

```
# Run Prolog unit tests
$ swipl -s tests/test_router.pl -g run_tests -t halt

% Expected output:
% ?- run_tests.
% PL-Unit: router ........ done
% All 10 tests passed
% true.
```

11. Appendices

Appendix A: Complete router.pl

```
% ============================================================
% router.pl - The Sovereign AI Infrastructure Constitution
% ============================================================
% This file contains the domain logic for routing decisions.
% NO PYTHON CODE MAY OVERRIDE THESE DECISIONS.
% ============================================================

:- module(router, [
    route/2,
    classify_domain/2,
    assess_stakes/2,
    select_model/3,
    determine_validation_policy/2,
    validation_depth/3,
    validator_required/2,
    detect_tools/3,
    calculate_confidence/5,
    confidence_level/2
]).

:- use_module(library(dicts)).

% ============================================================
% Domain Taxonomy
% ============================================================

domain_keywords(coding_architecture, [
    'design', 'architecture', 'refactor', 'structure', 'pattern',
    'microservice', 'system design', 'class diagram', 'module'
]).

domain_keywords(coding_implementation, [
    'implement', 'write code', 'function', 'class', 'debug',
    'fix', 'method', 'algorithm', 'optimize code'
]).

domain_keywords(reasoning, [
    'explain', 'compare', 'analyze', 'why', 'how', 'trade-off',
    'difference between', 'advantages', 'disadvantages'
]).

domain_keywords(creative_writing, [
    'story', 'write', 'creative', 'narrative', 'poem',
    'tone', 'style', 'imagine', 'fiction'
]).

domain_keywords(documentation, [
    'document', 'readme', 'comment', 'explain code',
    'api doc', 'tutorial', 'guide'
```

```
]).

precedence_order([
    coding_architecture,
    coding_implementation,
    reasoning,
    creative_writing,
    documentation
]).

% ============================================================
% Model Proficiency Matrix
% ============================================================

model_proficiency(qwen_coder_32b, coding_architecture, 0.95).
model_proficiency(qwen_coder_32b, coding_implementation, 0.85).
model_proficiency(qwen_coder_32b, reasoning, 0.70).
model_proficiency(qwen_coder_32b, creative_writing, 0.40).
model_proficiency(qwen_coder_32b, documentation, 0.90).

model_proficiency(nemotron_30b, coding_architecture, 0.75).
model_proficiency(nemotron_30b, coding_implementation, 0.95).
model_proficiency(nemotron_30b, reasoning, 0.65).
model_proficiency(nemotron_30b, creative_writing, 0.35).
model_proficiency(nemotron_30b, documentation, 0.70).

model_proficiency(gpt_oss_20b, coding_architecture, 0.70).
model_proficiency(gpt_oss_20b, coding_implementation, 0.60).
model_proficiency(gpt_oss_20b, reasoning, 0.90).
model_proficiency(gpt_oss_20b, creative_writing, 0.75).
model_proficiency(gpt_oss_20b, documentation, 0.80).

model_proficiency(mythomax_13b, coding_architecture, 0.30).
model_proficiency(mythomax_13b, coding_implementation, 0.40).
model_proficiency(mythomax_13b, reasoning, 0.60).
model_proficiency(mythomax_13b, creative_writing, 0.95).
model_proficiency(mythomax_13b, documentation, 0.65).

available_model(qwen_coder_32b).
available_model(nemotron_30b).
available_model(gpt_oss_20b).
available_model(mythomax_13b).

% ============================================================
% Main Routing Predicate
% ============================================================

route(Request, Decision) :-
    extract_request_components(Request, Text, Attachments, _Context),
    classify_domain(Text, Domain),
    assess_stakes(Text, RawStakes),
    stakes_with_override(Domain, RawStakes, Stakes),
    select_model(Domain, Stakes, Model),
    get_validation_config(Domain, Stakes, ValidationConfig),
    detect_tools(Text, Attachments, Domain, Tools),
    calculate_confidence(Text, Domain, Stakes, Model, Confidence),
    confidence_level(Confidence, ConfidenceLevel),
```

```
    confidence_action(ConfidenceLevel, Action),
    get_time(Timestamp),
    Decision = decision{
        domain: Domain,
        stakes: Stakes,
        recommended_model: Model,
        validation_policy: ValidationConfig.policy,
        validation_depth: ValidationConfig.depth,
        validator_required: ValidationConfig.validator_required,
        tools_required: Tools,
        confidence: Confidence,
        confidence_level: ConfidenceLevel,
        action: Action,
        timestamp: Timestamp
    }.

% ============================================================
% Helper Predicates (see sections above for full implementations)
% ============================================================

% classify_domain/2, assess_stakes/2, select_model/3, etc.
% [Full implementations from previous sections]
```

Appendix B: Glossary

Term	Definition
Constitution	The Prolog domain logic that governs all routing decisions
Domain	Semantic category of a request (coding, reasoning, creative, etc.)
Stakes	Risk level of a request (high, medium, low)
Validation Policy	Rules determining when and how validation occurs
Confidence Score	Numeric measure (0.0-1.0) of routing decision certainty
Wiggum Loop	Naive persistence loop for TDD-based code generation
Bicameral Architecture	GPU for generation, CPU for validation split

Appendix C: Version History

Version	Date	Changes
1.0	2026-02-11	Initial specification

> **Document Control:** This specification is part of the Phase One documentation set. Changes require review by the Domain Logic Architect and approval by the Project Lead.

End of Document

Sovereign AI Infrastructure - Domain Logic Specification v1.0