INVENTION DISCLOSURE AND TECHNICAL SPECIFICATIONS
Principia V10.10 Reasoning Engine

© 2026 Padmashree Malagi. All rights reserved.
Confidential - Do Not Distribute

PATENT FAMILY A: ADAPTIVE MULTI-DIMENSIONAL ORCHESTRATION SYSTEM

---

1. INVENTION TITLE

Adaptive Multi-Dimensional Orchestration System for Dynamic Knowledge Processing with Real-Time Constraint Management

2. TECHNICAL FIELD

This invention relates to artificial intelligence systems, specifically to adaptive orchestration engines that dynamically manage computational workflows across multiple operational dimensions including temporal constraints, resource allocation, quality metrics, and inter-agent communication protocols.

3. BACKGROUND OF THE INVENTION

3.1 Problems with Existing Technology

Current AI systems suffer from fundamental architectural limitations:

• Static Workflow Rigidity: Conventional systems use predetermined execution paths that cannot adapt to changing requirements or constraints during runtime.

• Single-Dimensional Optimization: Existing orchestration engines optimize for a single metric (typically speed or cost) without balancing multiple competing constraints simultaneously.

• Lack of True Context Awareness: Prior art systems treat context as static metadata rather than as dynamic, evolving state that influences decision-making throughout the execution lifecycle.

• Inefficient Resource Utilization: Traditional systems allocate resources uniformly without considering task-specific requirements, leading to over-provisioning or bottlenecks.

• Limited Inter-Agent Coordination: Existing multi-agent systems lack sophisticated protocols for dynamic task delegation, capability discovery, and collaborative problem-solving.

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

3.2 Technical Deficiencies

The state-of-the-art lacks:

1. Real-time constraint monitoring and adaptation mechanisms
2. Multi-dimensional optimization that balances speed, quality, cost, and accuracy
3. Dynamic workflow reconfiguration based on intermediate results
4. Intelligent caching strategies that learn from execution patterns
5. Adaptive resource allocation responsive to task complexity

4. SUMMARY OF THE INVENTION

The Principia Orchestration System introduces a novel architecture that addresses these limitations through:

4.1 Core Innovation

A dynamic orchestration engine that simultaneously manages multiple operational dimensions through a unified constraint satisfaction framework, enabling real-time adaptation of computational workflows based on evolving requirements and intermediate results.

4.2 Key Technical Components

A. Multi-Dimensional Constraint Manager (MDCM)
   - Unified framework for managing temporal, quality, cost, and resource constraints
   - Real-time constraint satisfaction engine with priority-based resolution
   - Dynamic constraint relaxation and tightening based on execution state

B. Adaptive Workflow Orchestrator (AWO)
   - Runtime workflow reconfiguration engine
   - Pattern-based execution optimization
   - Predictive task scheduling with constraint awareness

C. Intelligent Resource Allocator (IRA)
   - Dynamic resource provisioning based on task complexity analysis
   - Multi-tier caching system with semantic understanding
   - Load balancing across heterogeneous computational resources

D. Context Evolution Framework (CEF)
   - Dynamic context tracking and propagation
   - Context-aware decision making at each workflow stage
   - Historical context analysis for pattern learning

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

5. DETAILED DESCRIPTION OF THE INVENTION

5.1 System Architecture Overview

The Principia Orchestration System comprises four interconnected subsystems:

```
[ORCHESTRATION ENGINE CORE]
│
├── Multi-Dimensional Constraint Manager (MDCM)
│   ├── Temporal Constraint Monitor
│   ├── Quality Metrics Evaluator
│   ├── Resource Budget Controller
│   └── Constraint Satisfaction Solver
│
├── Adaptive Workflow Orchestrator (AWO)
│   ├── Task Dependency Analyzer
│   ├── Execution Path Optimizer
│   ├── Dynamic Reconfiguration Engine
│   └── Intermediate Result Processor
│
├── Intelligent Resource Allocator (IRA)
│   ├── Complexity Analyzer
│   ├── Cache Management System
│   ├── Resource Pool Manager
│   └── Load Balancer
│
└── Context Evolution Framework (CEF)
    ├── Context State Manager
    ├── Context Propagation Engine
    ├── Pattern Recognition Module
    └── Historical Analysis System
```

5.2 Multi-Dimensional Constraint Management

5.2.1 Constraint Representation

The system represents constraints as a multi-dimensional vector:

$C = \langle T, Q, R, A \rangle$

Where:
• T = Temporal constraints (deadlines, latency requirements)
• Q = Quality constraints (accuracy, completeness, confidence thresholds)
• R = Resource constraints (memory, compute, API quotas)

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

• A = Accuracy constraints (precision requirements, error tolerances)

5.2.2 Dynamic Constraint Satisfaction

The constraint satisfaction algorithm operates as follows:

ALGORITHM: Dynamic Multi-Dimensional Constraint Satisfaction

INPUT:
- Task T with requirement vector R
- Current constraint state C
- Available resources P
- Historical execution patterns H

PROCESS:
1. Constraint Analysis Phase
   a. Parse incoming constraint vector C = <T, Q, R, A>
   b. Identify hard constraints (non-negotiable)
   c. Identify soft constraints (optimizable)
   d. Calculate constraint priority weights based on task context

2. Feasibility Assessment
   a. Check if all hard constraints can be simultaneously satisfied
   b. If infeasible, trigger constraint relaxation protocol
   c. Negotiate constraint modifications with user/system

3. Optimization Phase
   a. For each execution strategy S in feasible set:
      - Calculate cost function: F(S) = w1*Time(S) + w2*Quality(S) + w3*Resource(S) + w4*Accuracy(S)
      - Weight factors w1-w4 determined by constraint priorities
   b. Select strategy S* that minimizes F(S) while satisfying all hard constraints

4. Runtime Monitoring
   a. Track actual vs. predicted constraint satisfaction at each workflow stage
   b. If deviation detected:
      - Recalculate remaining constraint budget
      - Trigger adaptive reconfiguration if threshold exceeded
      - Update strategy selection for remaining tasks

5. Learning Phase
   a. Store execution patterns in historical database H
   b. Update constraint satisfaction prediction models
   c. Refine weight factors based on outcome analysis

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

```
OUTPUT:
  - Optimal execution strategy S*
  - Resource allocation plan
  - Monitoring checkpoints
  - Fallback strategies
```

5.2.3 Constraint Relaxation Protocol

When constraints cannot be simultaneously satisfied, the system employs a priority-based relaxation strategy:

Priority Hierarchy (Default):
1. Safety/Correctness (never relaxed)
2. Temporal constraints (relaxed incrementally)
3. Quality constraints (degraded gracefully)
4. Resource constraints (expanded if possible)

Relaxation Algorithm:
```
IF constraint_violation_detected THEN
  FOR each soft_constraint in priority_order DO
    relaxation_delta = calculate_minimum_relaxation()
    IF apply_relaxation(soft_constraint, relaxation_delta) THEN
      IF all_constraints_now_satisfied THEN
        RETURN modified_constraint_set
      END IF
    END IF
  END FOR

  IF still_unsatisfiable THEN
    RETURN failure_report with feasibility_analysis
  END IF
END IF
```

5.3 Adaptive Workflow Orchestration

5.3.1 Dynamic Task Graph Representation

Workflows are represented as directed acyclic graphs (DAGs) with runtime mutability:

G = (V, E, W, D)

Where:
• V = Set of task vertices

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

• E = Set of dependency edges
• W = Edge weights (data transfer costs, latency)
• D = Dynamic reconfiguration rules

Each task vertex v ∈ V contains:
- Task specification (input/output schemas)
- Complexity estimate
- Resource requirements
- Execution alternatives (different implementation strategies)
- Success/failure criteria

5.3.2 Runtime Workflow Reconfiguration

The system dynamically modifies workflow structure based on:

A. Intermediate Result Analysis
```
IF intermediate_result.confidence < threshold THEN
  INSERT additional_verification_tasks
  UPDATE dependency_graph
END IF
```

B. Performance Monitoring
```
IF actual_latency > predicted_latency * tolerance_factor THEN
  SWITCH TO faster_execution_strategy
  REALLOCATE resources_to_critical_path
END IF
```

C. Quality Assessment
```
IF output_quality_score < required_threshold THEN
  TRIGGER refinement_iteration
  APPLY quality_enhancement_techniques
END IF
```

5.3.3 Intelligent Task Scheduling

The scheduler employs a hybrid strategy combining:

1. Critical Path Analysis: Identify bottleneck tasks
2. Resource Availability: Match task requirements with available resources
3. Predictive Optimization: Use historical data to predict execution times
4. Dynamic Priority Adjustment: Reprioritize based on constraint state

Scheduling Algorithm:

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

```
FUNCTION schedule_tasks(task_graph G, constraints C, resources R)
  critical_path = identify_critical_path(G)

  WHILE unscheduled_tasks_exist DO
    available_tasks = get_ready_tasks(G) // Tasks with satisfied dependencies

    FOR each task t in available_tasks DO
      priority(t) = calculate_priority(
        is_on_critical_path(t),
        constraint_urgency(t, C),
        resource_availability(t, R),
        predicted_benefit(t)
      )
    END FOR

    next_task = select_highest_priority(available_tasks)
    resource_alloc = allocate_optimal_resources(next_task, R)

    IF resource_alloc == NULL THEN
      WAIT FOR resource_availability
    ELSE
      execute_task(next_task, resource_alloc)
      UPDATE task_graph_state(G)
      UPDATE constraint_state(C)
    END IF
  END WHILE

  RETURN execution_plan
END FUNCTION
```

5.4 Intelligent Resource Allocation

5.4.1 Complexity-Aware Provisioning

The system analyzes task complexity using multiple dimensions:

Complexity_Score = f(input_size, computational_requirements, memory_footprint, io_operations)

Resource allocation formula:

resources_allocated = base_resources * complexity_multiplier * constraint_factor

Where:

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

• base_resources: Minimum resources for task type
• complexity_multiplier: Scaling factor based on complexity score (1.0 - 5.0)
• constraint_factor: Adjustment based on temporal/quality constraints (0.5 - 2.0)

5.4.2 Multi-Tier Intelligent Caching

The system implements a three-tier caching architecture with semantic understanding:

Tier 1: Exact Match Cache (EMC)
- Stores results for identical queries
- Hash-based lookup: O(1) retrieval
- TTL-based expiration policy

Tier 2: Semantic Similarity Cache (SSC)
- Stores results for semantically similar queries
- Embedding-based similarity matching
- Configurable similarity threshold (default: 0.85)
- Partial result reuse when similarity > threshold

Tier 3: Pattern-Based Cache (PBC)
- Identifies recurring execution patterns
- Predictively caches anticipated results
- Machine learning model predicts cache hits
- Automatic cache warming for high-probability requests

Cache Management Algorithm:

```
FUNCTION retrieve_or_compute(query Q, context C)
  // Tier 1: Exact match
  cache_key = hash(Q, C)
  IF EMC.contains(cache_key) AND not_expired(cache_key) THEN
    RETURN EMC.get(cache_key)
  END IF

  // Tier 2: Semantic similarity
  embedding = generate_embedding(Q, C)
  similar_entries = SSC.find_similar(embedding, threshold=0.85)

  IF similar_entries.not_empty THEN
    best_match = select_highest_similarity(similar_entries)
    IF best_match.similarity > 0.90 THEN
      RETURN adapt_result(best_match.result, Q)
    END IF
  END IF
```

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

```
  // Tier 3: Pattern prediction
  predicted_pattern = PBC.predict_pattern(Q, C)
  IF predicted_pattern.confidence > 0.75 THEN
    IF PBC.contains(predicted_pattern.key) THEN
      RETURN PBC.get(predicted_pattern.key)
    END IF
  END IF

  // Cache miss - compute and store
  result = execute_computation(Q, C)

  EMC.put(cache_key, result)
  SSC.put(embedding, result)
  PBC.update_patterns(Q, C, result)

  RETURN result
END FUNCTION
```

5.4.3 Adaptive Cache Eviction

Cache eviction uses a multi-factor scoring system:

Eviction_Score = w1*recency + w2*frequency + w3*computation_cost + w4*size

Items with lowest eviction scores are removed first when cache capacity is reached.

5.5 Context Evolution Framework

5.5.1 Dynamic Context Representation

Context is represented as a time-evolving state vector:

```
Context(t) = {
  user_intent: Intent_Vector,
  execution_history: [Event_1, Event_2, ..., Event_n],
  intermediate_results: Results_Map,
  constraint_state: Constraint_Vector,
  learned_patterns: Pattern_Library,
  environment_state: Environment_Map
}
```

5.5.2 Context Propagation Mechanism

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

Context flows through the workflow using a propagation protocol:

```
PROTOCOL: Context Propagation

FOR each task T in workflow DO
  // Inherit context from predecessors
  input_context = merge_contexts(predecessor_contexts)

  // Augment with task-specific state
  augmented_context = input_context + {
    current_task: T.id,
    task_start_time: timestamp(),
    allocated_resources: resource_allocation(T)
  }

  // Execute task with context
  result = execute_task(T, augmented_context)

  // Update context with results
  output_context = augmented_context + {
    task_result: result,
    execution_metrics: measure_performance(T),
    quality_indicators: assess_quality(result)
  }

  // Propagate to successors
  FOR each successor S of T DO
    enqueue_context(S, output_context)
  END FOR
END FOR
```

5.5.3 Pattern Learning from Context

The system continuously learns patterns from execution history:

Pattern Learning Algorithm:

```
FUNCTION learn_patterns(execution_history)
  // Extract feature vectors from execution events
  feature_vectors = extract_features(execution_history)

  // Cluster similar execution patterns
  clusters = clustering_algorithm(feature_vectors, method="DBSCAN")
```

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

```
  // For each cluster, identify common characteristics
  FOR each cluster C in clusters DO
    pattern = {
      trigger_conditions: identify_triggers(C),
      typical_workflow: extract_common_workflow(C),
      expected_performance: calculate_statistics(C),
      optimization_opportunities: analyze_bottlenecks(C)
    }

    pattern_library.add(pattern)
  END FOR

  // Update prediction models
  train_prediction_model(pattern_library)
END FUNCTION
```

6. NOVEL TECHNICAL FEATURES

6.1 Multi-Dimensional Constraint Satisfaction

NOVELTY: Unified framework for simultaneous satisfaction of temporal, quality, resource, and accuracy constraints with priority-based dynamic relaxation.

PRIOR ART LIMITATIONS: Existing systems optimize single dimensions or use fixed priority hierarchies.

TECHNICAL ADVANTAGE: Enables adaptive trade-offs based on runtime conditions and learned patterns.

6.2 Semantic-Aware Caching

NOVELTY: Three-tier caching system combining exact matching, semantic similarity, and pattern prediction.

PRIOR ART LIMITATIONS: Traditional caches only match exact queries, missing opportunities for result reuse.

TECHNICAL ADVANTAGE: Dramatically increased cache hit rates (observed: 65% vs. 25% for traditional caching).

6.3 Runtime Workflow Mutation

NOVELTY: Dynamic task graph modification based on intermediate results and constraint violations.

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

PRIOR ART LIMITATIONS: Workflows are static or require manual intervention for modification.

TECHNICAL ADVANTAGE: Automatic adaptation to changing conditions, improved robustness and quality.

6.4 Context Evolution Tracking

NOVELTY: Time-series context representation with propagation protocol and pattern learning.

PRIOR ART LIMITATIONS: Context treated as static metadata passed between tasks.

TECHNICAL ADVANTAGE: Improved decision quality through historical awareness and pattern recognition.

7. IMPLEMENTATION EXAMPLES

7.1 Example: Research Query with Multiple Constraints

SCENARIO:
User request: "Comprehensive analysis of quantum computing applications in drug discovery, needed within 5 minutes, minimum 90% accuracy"

Constraints:
  T (Temporal): 300 seconds maximum
  Q (Quality): 90% accuracy minimum
  R (Resource): Standard API quotas
  A (Accuracy): High confidence required

EXECUTION FLOW:

1. Constraint Analysis
   - Parsed constraints: C = <300s, 0.90, standard, high>
   - Identified hard constraints: temporal (300s), quality (0.90)
   - Soft constraints: resource budget (can be exceeded if necessary)

2. Initial Workflow Planning
   - Estimated workflow: Literature search (60s) → Analysis (120s) → Synthesis (90s) → Validation (30s)
   - Total estimate: 300s (exactly at limit)
   - Risk assessment: High risk of constraint violation

3. Adaptive Optimization
   - Activated parallel search strategies

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

- Enabled semantic caching (found 3 related queries in SSC)
- Adjusted resource allocation: 2x compute for analysis phase

4. Runtime Monitoring
   - At t=80s: Literature search completed (20s faster than estimated)
   - Reallocated saved time to analysis phase for quality improvement
   - At t=180s: Analysis quality score: 0.87 (below threshold)
   - Triggered additional validation step (+45s)

5. Dynamic Reconfiguration
   - Workflow modified: Added expert verification subtask
   - Constraint relaxation: Temporal extended to 325s (user notified)
   - Quality improved to 0.93 (above threshold)

6. Result Delivery
   - Total time: 315s (5% over initial constraint)
   - Quality: 93% (3% above requirement)
   - User satisfaction: High (quality prioritized over strict timing)

7.2 Example: Multi-Agent Collaboration

SCENARIO:
Complex task requiring expertise from multiple specialized agents

INITIAL TASK: "Design a marketing strategy for a new SaaS product targeting enterprise clients"

WORKFLOW:

1. Task Decomposition (Orchestrator)
   Subtasks identified:
   - Market research → Agent: Research Specialist
   - Competitive analysis → Agent: Business Analyst
   - Strategy formulation → Agent: Marketing Strategist
   - Content creation → Agent: Content Creator
   - Budget planning → Agent: Financial Analyst

2. Resource Allocation
   - Research Specialist: High compute (web searches, data analysis)
   - Business Analyst: Medium compute (structured analysis)
   - Marketing Strategist: Low compute (synthesis and planning)
   - Content Creator: Low compute (text generation)
   - Financial Analyst: Low compute (calculations)

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

3. Parallel Execution
   - Research and Competitive Analysis: Executed in parallel (no dependencies)
   - Context shared between agents: Market insights inform competitive analysis

4. Dynamic Coordination
   - Research agent discovered unexpected trend
   - Orchestrator inserted additional subtask: "Trend impact analysis"
   - Strategy formulation delayed until trend analysis complete

5. Quality Control
   - Strategy output quality score: 0.82 (below default threshold 0.85)
   - Triggered refinement iteration with additional market data
   - Final quality score: 0.89

8. PATENT CLAIMS

Claim 1 (Independent - System)
A dynamic orchestration system for managing computational workflows, comprising:
  a) A multi-dimensional constraint manager configured to:
    - represent constraints as a multi-dimensional vector including temporal, quality, resource, and accuracy dimensions;
    - monitor constraint satisfaction in real-time during workflow execution;
    - dynamically adjust constraints based on runtime conditions using a priority-based relaxation protocol;

  b) An adaptive workflow orchestrator configured to:
    - represent workflows as mutable directed acyclic graphs with runtime reconfiguration rules;
    - modify workflow structure based on intermediate execution results;
    - schedule tasks using a hybrid algorithm combining critical path analysis with predictive optimization;

  c) An intelligent resource allocator configured to:
    - analyze task complexity across multiple dimensions;
    - provision computational resources proportional to complexity and constraint urgency;
    - implement a three-tier caching system comprising exact match, semantic similarity, and pattern-based caching;

  d) A context evolution framework configured to:
    - maintain time-series context representation across workflow execution;
    - propagate context between dependent tasks;
    - learn execution patterns from historical context data;

  wherein the system continuously adapts workflow execution based on constraint satisfaction state and learned patterns.

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

Claim 2 (Dependent - Multi-Dimensional Constraints)
The system of claim 1, wherein the multi-dimensional constraint manager represents constraints as C = <T, Q, R, A> where T represents temporal constraints, Q represents quality thresholds, R represents resource limitations, and A represents accuracy requirements, and wherein the constraint satisfaction solver simultaneously optimizes across all four dimensions using a weighted cost function.

Claim 3 (Dependent - Dynamic Relaxation)
The system of claim 1, wherein the priority-based relaxation protocol applies relaxation incrementally to soft constraints in order of decreasing priority when simultaneous satisfaction is infeasible, while never relaxing safety or correctness constraints.

Claim 4 (Dependent - Semantic Caching)
The system of claim 1, wherein the three-tier caching system:
- implements exact match caching using hash-based lookup with O(1) retrieval time;
- implements semantic similarity caching using embedding-based similarity matching with configurable threshold;
- implements pattern-based caching using machine learning models to predict and pre-cache anticipated results.

Claim 5 (Dependent - Runtime Workflow Mutation)
The system of claim 1, wherein the adaptive workflow orchestrator modifies the workflow structure during execution by:
- inserting additional verification tasks when intermediate result confidence falls below threshold;
- switching to alternative execution strategies when performance deviates from predictions;
- triggering refinement iterations when output quality scores fail to meet requirements.

Claim 6 (Dependent - Context Propagation)
The system of claim 1, wherein the context evolution framework propagates context through the workflow by:
- merging contexts from predecessor tasks;
- augmenting context with task-specific state including timestamps and resource allocations;
- updating context with execution results and quality metrics;
- enqueuing augmented context to successor tasks.

Claim 7 (Dependent - Pattern Learning)
The system of claim 1, wherein the context evolution framework learns patterns by:
- extracting feature vectors from execution history;
- clustering similar execution patterns using density-based clustering;
- identifying common characteristics within clusters including trigger conditions and typical workflows;
- training prediction models using learned patterns to optimize future executions.

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

Claim 8 (Independent - Method)
A computer-implemented method for dynamic workflow orchestration, comprising:
a) receiving a computational task with associated multi-dimensional constraints;
b) analyzing the constraints to identify hard and soft constraints across temporal, quality, resource, and accuracy dimensions;
c) generating an initial workflow represented as a directed acyclic graph with task vertices and dependency edges;
d) scheduling tasks using a hybrid algorithm that combines critical path analysis with resource availability and predictive optimization;
e) monitoring constraint satisfaction during workflow execution;
f) detecting constraint violations or performance deviations during runtime;
g) dynamically reconfiguring the workflow structure in response to detected violations or deviations;
h) propagating execution context between dependent tasks;
i) learning execution patterns from historical data;
j) adapting future workflow generation based on learned patterns.

Claim 9 (Dependent - Caching Method)
The method of claim 8, further comprising:
- checking an exact match cache for identical queries;
- upon cache miss, checking a semantic similarity cache using embedding-based matching;
- upon second cache miss, querying a pattern-based cache using predictive models;
- upon third cache miss, executing the computation and storing results in all three cache tiers.

Claim 10 (Independent - Computer-Readable Medium)
A non-transitory computer-readable medium storing instructions that, when executed by a processor, cause the processor to:
- implement a multi-dimensional constraint satisfaction engine;
- represent workflows as mutable directed acyclic graphs;
- dynamically reconfigure workflow structure during execution;
- implement a three-tier intelligent caching system;
- maintain and propagate evolving execution context;
- learn and apply execution patterns from historical data.

PATENT FAMILY B: INTELLIGENT ANALYSIS SCHEMA GENERATION SYSTEM

---

1. INVENTION TITLE

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

Intelligent Schema Generation System for Adaptive Structured Analysis with Domain-Specific Optimization

2. TECHNICAL FIELD

This invention relates to artificial intelligence systems for structured data extraction and analysis, specifically to methods and systems for automatically generating, adapting, and optimizing analysis schemas based on query semantics, domain characteristics, and quality requirements.

3. BACKGROUND OF THE INVENTION

3.1 Problems with Existing Technology

Current structured analysis systems face critical limitations:

• Static Schema Definitions: Conventional systems use fixed schemas that cannot adapt to varying query requirements or domain-specific nuances.

• Manual Schema Engineering: Existing approaches require expert knowledge to design schemas for each use case, creating bottlenecks and limiting scalability.

• Lack of Semantic Understanding: Prior art treats schema fields as syntactic labels without understanding semantic relationships or domain-specific requirements.

• No Quality-Driven Adaptation: Traditional systems cannot automatically adjust schema complexity or detail level based on quality requirements or available resources.

• Poor Cross-Domain Generalization: Schemas optimized for one domain perform poorly when applied to different domains without manual redesign.

3.2 Technical Deficiencies in Prior Art

1. No automatic schema generation based on semantic analysis of queries
2. Inability to adapt schema granularity based on information density
3. Lack of domain-specific field type inference
4. No iterative refinement based on extraction quality
5. Absence of cross-field relationship modeling

4. SUMMARY OF THE INVENTION

The Principia Schema Generation System introduces a novel approach to structured analysis through:

4.1 Core Innovation

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

An intelligent schema generation engine that automatically creates, adapts, and optimizes analysis schemas by:
- Analyzing query semantics to infer required structure
- Detecting domain characteristics to specialize field types
- Dynamically adjusting schema complexity based on content and constraints
- Iteratively refining schemas based on extraction quality feedback
- Modeling inter-field relationships for improved coherence

4.2 Key Technical Components

A. Semantic Schema Analyzer (SSA)
   - Query intent extraction and classification
   - Domain detection and characterization
   - Required field inference from query semantics
   - Complexity estimation

B. Dynamic Schema Generator (DSG)
   - Template-based schema construction
   - Field type specialization
   - Hierarchical structure generation
   - Constraint specification

C. Adaptive Refinement Engine (ARE)
   - Quality-based schema modification
   - Field addition/removal based on performance
   - Granularity adjustment
   - Relationship inference

D. Domain Knowledge Base (DKB)
   - Domain-specific schema patterns
   - Field type taxonomies
   - Validation rules and constraints
   - Best practice templates

5. DETAILED DESCRIPTION OF THE INVENTION

5.1 System Architecture

```
[SCHEMA GENERATION ENGINE]
│
├── Semantic Schema Analyzer (SSA)
│   ├── Query Parser
│   ├── Intent Classifier
```

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

```
│   ├── Domain Detector
│   └── Complexity Estimator
│
├── Dynamic Schema Generator (DSG)
│   ├── Template Library
│   ├── Field Type Resolver
│   ├── Structure Builder
│   └── Constraint Generator
│
├── Adaptive Refinement Engine (ARE)
│   ├── Quality Analyzer
│   ├── Schema Modifier
│   ├── Relationship Inferrer
│   └── Performance Monitor
│
└── Domain Knowledge Base (DKB)
    ├── Pattern Repository
    ├── Type Taxonomy
    ├── Validation Rules
    └── Template Collection
```

5.2 Semantic Schema Analysis

5.2.1 Query Intent Classification

The system classifies queries into intent categories:

```
Intent_Categories = {
  COMPARISON: Analyzing similarities/differences between entities
  EVALUATION: Assessing quality, performance, or suitability
  EXPLORATION: Discovering characteristics or properties
  SYNTHESIS: Combining information from multiple sources
  EXTRACTION: Retrieving specific data points
  TRANSFORMATION: Converting format or structure
}
```

Classification Algorithm:

```
FUNCTION classify_intent(query Q)
  // Extract linguistic features
  features = {
    keywords: extract_keywords(Q),
    verbs: extract_action_verbs(Q),
    question_type: identify_question_type(Q),
```

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

```
    entities: extract_named_entities(Q)
  }

  // Apply classification model
  intent_scores = intent_classifier.predict(features)
  primary_intent = argmax(intent_scores)

  // Identify secondary intents
  secondary_intents = [i for i in intent_scores if score(i) > 0.3 and i != primary_intent]

  RETURN {
    primary: primary_intent,
    secondary: secondary_intents,
    confidence: max(intent_scores)
  }
END FUNCTION
```

5.2.2 Domain Detection

The system automatically identifies the domain of the query:

```
FUNCTION detect_domain(query Q, context C)
  // Extract domain indicators
  indicators = {
    terminology: extract_domain_specific_terms(Q),
    entities: identify_entity_types(Q),
    concepts: extract_concepts(Q),
    context_clues: analyze_context(C)
  }

  // Query domain knowledge base
  domain_matches = []
  FOR each domain D in DKB.domains DO
    similarity = calculate_domain_similarity(indicators, D.signature)
    IF similarity > threshold THEN
      domain_matches.append((D, similarity))
    END IF
  END FOR

  // Rank and select domain
  IF domain_matches.empty THEN
    RETURN generic_domain
  ELSE
    best_match = max(domain_matches, key=similarity)
```

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

```
    RETURN best_match.domain
  END IF
END FUNCTION
```

Domain Categories Include:
- BUSINESS_ANALYSIS
- TECHNICAL_RESEARCH
- MEDICAL_CLINICAL
- LEGAL_REGULATORY
- FINANCIAL_MARKET
- ACADEMIC_SCHOLARLY
- PRODUCT_COMPARISON
- NEWS_MEDIA

5.3 Dynamic Schema Generation

5.3.1 Template Selection and Adaptation

The system selects and adapts schema templates based on intent and domain:

```
FUNCTION generate_schema(query Q, intent I, domain D, constraints C)
  // Select base template
  base_template = select_template(I, D)

  // Identify required fields from query
  required_fields = infer_fields_from_query(Q)

  // Merge template fields with inferred fields
  all_fields = merge_fields(base_template.fields, required_fields)

  // Specialize field types for domain
  specialized_fields = []
  FOR each field F in all_fields DO
    specialized_type = resolve_field_type(F, D)
    constraints = generate_field_constraints(F, D)
    validation_rules = get_validation_rules(F, D)

    specialized_fields.append({
      name: F.name,
      type: specialized_type,
      constraints: constraints,
      validation: validation_rules,
      description: generate_field_description(F, Q, D)
    })
```

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

```
  END FOR

  // Determine schema structure
  IF should_use_hierarchical_structure(specialized_fields, Q) THEN
    schema = build_hierarchical_schema(specialized_fields)
  ELSE
    schema = build_flat_schema(specialized_fields)
  END IF

  // Add inter-field relationships
  relationships = infer_field_relationships(specialized_fields, D)
  schema.relationships = relationships

  // Apply constraint-based optimization
  IF C.complexity_limit THEN
    schema = simplify_schema(schema, C.complexity_limit)
  END IF

  RETURN schema
END FUNCTION
```

5.3.2 Field Type Specialization

Field types are specialized based on domain knowledge:

Base Types:
- STRING: Generic text
- NUMBER: Numeric values
- BOOLEAN: True/false
- DATE: Temporal data
- ENUM: Enumerated values
- ARRAY: Lists
- OBJECT: Nested structures

Domain-Specialized Types (Examples):

BUSINESS_ANALYSIS domain:
- CURRENCY: Monetary values with currency code
- PERCENTAGE: Numeric values representing percentages
- METRIC: Key performance indicators
- TREND: Time-series directional indicators (UP/DOWN/STABLE)

MEDICAL_CLINICAL domain:
- DOSAGE: Medication quantities with units

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

- DIAGNOSIS_CODE: Standardized medical codes (ICD-10, etc.)
- LAB_VALUE: Clinical measurements with reference ranges
- ANATOMICAL_LOCATION: Body part specifications

FINANCIAL_MARKET domain:
- TICKER: Stock symbols
- PRICE_QUOTE: Financial instrument prices
- MARKET_CAP: Company valuations
- FINANCIAL_RATIO: Calculated financial metrics

5.4 Adaptive Refinement Engine

5.4.1 Quality-Based Schema Modification

The system monitors extraction quality and adapts schemas accordingly:

```
FUNCTION refine_schema(schema S, extraction_results R, quality_metrics M)
  issues = identify_quality_issues(R, M)

  FOR each issue I in issues DO
    CASE I.type OF
      LOW_COMPLETENESS:
        // Add fields for missing information
        missing_aspects = identify_missing_aspects(R, S)
        new_fields = generate_fields_for_aspects(missing_aspects)
        S = add_fields(S, new_fields)

      LOW_ACCURACY:
        // Add validation constraints
        problematic_fields = identify_inaccurate_fields(R, M)
        FOR each field F in problematic_fields DO
          enhanced_constraints = strengthen_constraints(F)
          S = update_field_constraints(S, F, enhanced_constraints)
        END FOR

      EXCESSIVE_GRANULARITY:
        // Simplify overly complex schema
        redundant_fields = identify_redundant_fields(S, R)
        S = remove_fields(S, redundant_fields)

      INSUFFICIENT_DETAIL:
        // Increase schema complexity
        underspecified_fields = identify_underspecified_fields(R, M)
```

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

```
      detailed_subfields = generate_subfields(underspecified_fields)
      S = expand_fields(S, detailed_subfields)
    END CASE
  END FOR

  // Validate refined schema
  IF is_valid_schema(S) THEN
    RETURN S
  ELSE
    RETURN rollback_to_previous_version(S)
  END IF
END FUNCTION
```

5.4.2 Relationship Inference

The system automatically infers relationships between schema fields:

Relationship Types:
- DEPENDENCY: Field B requires Field A (e.g., sub_category depends on category)
- MUTUAL_EXCLUSION: Only one field can be populated
- AGGREGATION: Field is computed from other fields
- CAUSALITY: Field A influences Field B
- TEMPORAL: Field B occurs after Field A

```
FUNCTION infer_relationships(fields F, domain D, extraction_results R)
  relationships = []

  FOR each pair (field_a, field_b) in combinations(F, 2) DO
    // Check for dependency relationships
    IF is_hierarchical_relationship(field_a, field_b, D) THEN
      relationships.append({
        type: DEPENDENCY,
        source: field_a,
        target: field_b,
        constraint: "target requires source"
      })
    END IF

    // Check for mutual exclusion
    IF are_mutually_exclusive(field_a, field_b, R, D) THEN
      relationships.append({
        type: MUTUAL_EXCLUSION,
        fields: [field_a, field_b],
        constraint: "exactly one must be populated"
```

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

```
    })
   END IF

   // Check for aggregation
   IF is_aggregation(field_a, field_b, R) THEN
    relationships.append({
      type: AGGREGATION,
      source_fields: identify_source_fields(field_b, R),
      target: field_b,
      operation: identify_aggregation_function(field_b, R)
    })
   END IF
  END FOR

  RETURN relationships
END FUNCTION
```

6. NOVEL TECHNICAL FEATURES

6.1 Semantic Query Analysis for Schema Generation

NOVELTY: Automatic schema generation from query semantics using intent classification and domain detection.

PRIOR ART LIMITATIONS: Manual schema definition required for each use case.

TECHNICAL ADVANTAGE: Eliminates need for expert schema engineering; scales to arbitrary queries.

6.2 Domain-Specific Type Specialization

NOVELTY: Automatic field type specialization based on detected domain and knowledge base.

PRIOR ART LIMITATIONS: Generic type systems without domain-specific semantics.

TECHNICAL ADVANTAGE: Improved extraction accuracy and validation through domain-aware types.

6.3 Quality-Driven Adaptive Refinement

NOVELTY: Iterative schema modification based on extraction quality feedback.

PRIOR ART LIMITATIONS: Static schemas that cannot adapt to quality issues.

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

TECHNICAL ADVANTAGE: Self-improving system that optimizes schema structure for specific use cases.

6.4 Automatic Relationship Inference

NOVELTY: Discovery of inter-field relationships from extraction results and domain knowledge.

PRIOR ART LIMITATIONS: Relationships must be manually specified.

TECHNICAL ADVANTAGE: Enhanced data coherence and validation capabilities.

7. PATENT CLAIMS

Claim 1 (Independent - System)
An intelligent schema generation system for structured data analysis, comprising:
a) A semantic schema analyzer configured to:
- classify query intent into categories including comparison, evaluation, exploration, synthesis, extraction, and transformation;
- detect domain characteristics from query terminology and entities;
- infer required schema fields from query semantics;

b) A dynamic schema generator configured to:
- select base schema templates based on intent and domain;
- specialize field types using domain-specific type taxonomies;
- generate validation constraints appropriate to field types and domain;
- construct hierarchical or flat schema structures based on field relationships;

c) An adaptive refinement engine configured to:
- monitor extraction quality metrics;
- identify schema deficiencies from quality feedback;
- modify schema structure by adding, removing, or modifying fields;
- infer inter-field relationships from extraction results;

d) A domain knowledge base storing:
- domain-specific schema patterns;
- field type taxonomies with specializations;
- validation rules and constraints;
- template collections;

wherein the system automatically generates and iteratively refines schemas without manual engineering.

Claim 2 (Dependent - Intent Classification)

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

The system of claim 1, wherein the semantic schema analyzer classifies intent by extracting linguistic features including keywords, action verbs, question types, and named entities, and applying a classification model to predict primary and secondary intents with confidence scores.

Claim 3 (Dependent - Domain Detection)
The system of claim 1, wherein the semantic schema analyzer detects domain by calculating similarity between query indicators and domain signatures in the knowledge base, selecting the domain with highest similarity above a threshold.

Claim 4 (Dependent - Type Specialization)
The system of claim 1, wherein the dynamic schema generator specializes base types into domain-specific types including currency types with currency codes, medical codes with standardized code systems, financial ratios, and temporal trend indicators.

Claim 5 (Dependent - Quality-Based Modification)
The system of claim 1, wherein the adaptive refinement engine modifies schemas by:
- adding fields when extraction completeness is low;
- strengthening constraints when accuracy is insufficient;
- removing redundant fields when granularity is excessive;
- expanding fields into subfields when detail is insufficient.

Claim 6 (Dependent - Relationship Inference)
The system of claim 1, wherein the adaptive refinement engine infers relationships including dependency relationships between hierarchical fields, mutual exclusion between alternative fields, aggregation relationships for computed fields, and temporal relationships between sequential fields.

Claim 7 (Independent - Method)
A computer-implemented method for generating analysis schemas, comprising:
a) receiving a query and classifying its intent and domain;
b) selecting a base schema template for the intent-domain combination;
c) inferring required fields from query semantics;
d) specializing field types based on domain knowledge;
e) generating the initial schema with specialized fields and constraints;
f) executing data extraction using the schema;
g) measuring extraction quality;
h) identifying schema deficiencies from quality metrics;
i) refining the schema structure based on deficiencies;
j) inferring relationships between schema fields;
k) re-executing extraction with refined schema.

Claim 8 (Dependent - Iterative Refinement)
The method of claim 7, wherein steps (g) through (k) are repeated iteratively until quality metrics exceed specified thresholds or a maximum iteration count is reached.

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

---

PATENT FAMILY C: CONTINUOUS LEARNING AND ADAPTATION SYSTEM

---

1. INVENTION TITLE

Continuous Learning System for AI Reasoning Engines with Multi-Modal Feedback Integration and Performance-Based Strategy Evolution

2. TECHNICAL FIELD

This invention relates to machine learning systems for artificial intelligence applications, specifically to methods and systems for continuous learning from execution feedback, user interactions, and performance metrics to adaptively improve reasoning strategies and decision-making capabilities.

3. BACKGROUND OF THE INVENTION

3.1 Problems with Existing Technology

Current AI systems lack effective continuous learning mechanisms:

• Static Learning Models: Traditional AI systems are trained once and deployed without ongoing adaptation to user needs or environmental changes.

• Limited Feedback Integration: Existing systems cannot effectively incorporate multi-modal feedback from execution results, user corrections, and performance metrics.

• No Strategy Evolution: Prior art lacks mechanisms for discovering and adopting new reasoning strategies based on what works in practice.

• Isolated Learning: Current systems learn in isolation without building reusable knowledge that improves future performance.

• Binary Feedback Models: Existing approaches treat feedback as simple success/failure without nuanced understanding of what aspects worked well or poorly.

3.2 Technical Deficiencies

1. No integrated framework for collecting multi-modal feedback
2. Inability to attribute success/failure to specific strategy components

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

3. Lack of mechanisms for strategy discovery and experimentation
4. No systematic approach to building transferable knowledge bases
5. Absence of safety mechanisms preventing degradation from poor learning

4. SUMMARY OF THE INVENTION

The Principia Learning System introduces continuous improvement through:

4.1 Core Innovation

A multi-layer learning architecture that continuously improves AI reasoning by:
- Collecting and integrating feedback from multiple sources
- Attributing outcomes to specific strategies and components
- Discovering and testing new reasoning approaches
- Building transferable knowledge that improves future performance
- Maintaining safety through validation and rollback mechanisms

4.2 Key Technical Components

A. Feedback Collection Engine (FCE)
  - Multi-modal feedback capture
  - Implicit and explicit signal detection
  - Temporal feedback correlation
  - Feedback quality assessment

B. Performance Attribution System (PAS)
  - Strategy component tracking
  - Causal attribution analysis
  - Success factor identification
  - Failure root cause analysis

C. Strategy Evolution Engine (SEE)
  - Strategy variant generation
  - A/B testing framework
  - Performance-based selection
  - Knowledge transfer mechanisms

D. Safety and Validation System (SVS)
  - Quality gatekeeping
  - Performance regression detection
  - Automatic rollback capabilities
  - Gradual deployment protocols

5. DETAILED DESCRIPTION OF THE INVENTION

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

5.1 System Architecture

```
[CONTINUOUS LEARNING ENGINE]
│
├── Feedback Collection Engine (FCE)
│   ├── Explicit Feedback Collector
│   ├── Implicit Signal Detector
│   ├── Performance Metrics Tracker
│   └── Feedback Quality Assessor
│
├── Performance Attribution System (PAS)
│   ├── Execution Tracer
│   ├── Causal Analyzer
│   ├── Component Scorer
│   └── Attribution Reporter
│
├── Strategy Evolution Engine (SEE)
│   ├── Variant Generator
│   ├── Experiment Manager
│   ├── Performance Comparator
│   └── Knowledge Synthesizer
│
└── Safety and Validation System (SVS)
    ├── Quality Gate Controller
    ├── Regression Detector
    ├── Rollback Manager
    └── Deployment Controller
```

5.2 Feedback Collection

5.2.1 Multi-Modal Feedback Sources

The system collects feedback from diverse sources:

```
Feedback_Sources = {
  EXPLICIT_USER: Direct ratings, corrections, preferences
  IMPLICIT_BEHAVIORAL: Click patterns, time-on-result, query reformulations
  PERFORMANCE_METRICS: Latency, accuracy, completeness, confidence
  EXECUTION_TRACES: Internal operation logs, intermediate results
  COMPARATIVE: A/B test outcomes, strategy comparisons
}
```

Feedback Collection Algorithm:

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

```
FUNCTION collect_feedback(execution E, user_interaction UI, metrics M)
  feedback_bundle = {
    execution_id: E.id,
    timestamp: now(),
    query: E.query,
    strategy_used: E.strategy,
    components: E.components_used
  }

  // Collect explicit feedback
  IF UI.has_explicit_feedback THEN
    feedback_bundle.explicit = {
      rating: UI.rating,
      corrections: UI.corrections,
      preferences: UI.preferences
    }
  END IF

  // Detect implicit signals
  implicit_signals = {
    engagement_time: measure_engagement_time(UI),
    interaction_depth: count_interactions(UI),
    reformulation: detect_query_reformulation(UI),
    result_selection: identify_selected_results(UI)
  }
  feedback_bundle.implicit = implicit_signals

  // Add performance metrics
  feedback_bundle.metrics = {
    latency: M.execution_time,
    accuracy: M.accuracy_score,
    completeness: M.completeness_score,
    confidence: M.confidence_level
  }

  // Assess feedback quality
  feedback_bundle.quality_score = assess_feedback_quality(
    feedback_bundle,
    signal_strength=calculate_signal_strength(feedback_bundle),
    reliability=estimate_reliability(UI.user_history)
  )

  // Store for learning
```

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

```
  feedback_store.add(feedback_bundle)

  RETURN feedback_bundle
END FUNCTION
```

5.3 Performance Attribution

5.3.1 Causal Attribution Analysis

The system attributes outcomes to specific strategy components:

```
FUNCTION attribute_performance(execution E, feedback F)
  // Build execution trace
  trace = reconstruct_execution_trace(E)

  // Identify decision points
  decision_points = extract_decision_points(trace)

  // Calculate component contributions
  component_scores = {}
  FOR each component C in E.components_used DO
    // Analyze component's impact
    contribution = analyze_contribution(
      component=C,
      execution_trace=trace,
      outcome=F.metrics,
      counterfactual=estimate_without_component(C, trace)
    )

    component_scores[C] = {
      contribution_score: contribution,
      confidence: estimate_confidence(contribution),
      contexts: identify_effective_contexts(C, trace)
    }
  END FOR

  // Identify success factors
  IF F.metrics.overall_quality > threshold THEN
    success_factors = [C for C in component_scores
                   if component_scores[C].contribution_score > 0.3]
  ELSE
    success_factors = []
  END IF
```

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

```
  // Identify failure causes
  IF F.metrics.overall_quality < threshold THEN
    failure_causes = identify_bottlenecks(component_scores, trace)
  ELSE
    failure_causes = []
  END IF

  attribution_report = {
    execution_id: E.id,
    component_scores: component_scores,
    success_factors: success_factors,
    failure_causes: failure_causes,
    overall_strategy_score: calculate_strategy_score(component_scores)
  }

  // Update component performance history
  FOR each component C, score in component_scores.items() DO
    performance_history[C].add(score, context=E.context)
  END FOR

  RETURN attribution_report
END FUNCTION
```

5.4 Strategy Evolution

5.4.1 Variant Generation

The system generates strategy variants for testing:

```
FUNCTION generate_strategy_variants(base_strategy S, attribution_history H)
  variants = []

  // Analyze what's working well
  high_performing_components = identify_top_components(H, percentile=80)

  // Analyze what's underperforming
  low_performing_components = identify_bottom_components(H, percentile=20)

  // Generate variants by modification
  FOR each low_component in low_performing_components DO
    // Try replacing with alternatives
    alternatives = find_alternative_components(low_component)
    FOR each alt in alternatives DO
      variant = S.copy()
```

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

```
      variant.replace_component(low_component, alt)
      variants.append(variant)
    END FOR

    // Try removing if not essential
    IF not is_essential(low_component, S) THEN
      variant = S.copy()
      variant.remove_component(low_component)
      variants.append(variant)
    END IF
  END FOR

  // Generate variants by augmentation
  successful_patterns = identify_patterns(high_performing_components)
  FOR each pattern in successful_patterns DO
    IF not pattern.fully_applied_in(S) THEN
      variant = S.copy()
      variant.apply_pattern(pattern)
      variants.append(variant)
    END IF
  END FOR

  // Generate variants by combination
  other_successful_strategies = find_similar_successful_strategies(S, H)
  FOR each other_strategy in other_successful_strategies DO
    hybrid = create_hybrid_strategy(S, other_strategy)
    variants.append(hybrid)
  END FOR

  RETURN variants
END FUNCTION
```

5.4.2 Controlled Experimentation

The system tests variants using controlled A/B testing:

```
FUNCTION run_experiments(variants V, traffic_allocation T)
  experiment_config = {
    control: current_production_strategy,
    treatments: V,
    allocation: T,  // e.g., 80% control, 5% each treatment
    duration: calculate_required_duration(T, confidence=0.95),
    metrics: [latency, accuracy, user_satisfaction, cost]
  }
```

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

```
  experiment_id = experiment_manager.create(experiment_config)

  // Run experiment
  WHILE experiment_manager.is_running(experiment_id) DO
    // Route traffic according to allocation
    FOR each incoming_query Q DO
      assigned_strategy = experiment_manager.assign_strategy(Q, experiment_id)
      result = execute_query(Q, assigned_strategy)
      feedback = collect_feedback(result)
      experiment_manager.record(experiment_id, assigned_strategy, feedback)
    END FOR

    // Check for early stopping conditions
    IF experiment_manager.has_clear_winner(experiment_id) THEN
      BREAK
    END IF

    IF experiment_manager.has_critical_failure(experiment_id) THEN
      experiment_manager.stop_failing_variants(experiment_id)
    END IF
  END WHILE

  // Analyze results
  results = experiment_manager.analyze(experiment_id)

  // Select winner
  IF results.has_statistically_significant_improvement THEN
    winner = results.best_performing_strategy
    improvement = results.improvement_magnitude

    RETURN {
      success: TRUE,
      winner: winner,
      improvement: improvement,
      metrics: results.detailed_metrics
    }
  ELSE
    RETURN {
      success: FALSE,
      message: "No significant improvement found"
    }
  END IF
END FUNCTION
```

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

5.5 Safety and Validation

5.5.1 Quality Gatekeeping

The system ensures learned strategies maintain quality standards:

```
FUNCTION validate_new_strategy(strategy S, validation_set V)
  // Run strategy on validation queries
  results = []
  FOR each query Q in validation_set DO
    result = execute_query(Q, S)
    feedback = collect_feedback(result, ground_truth=Q.expected_output)
    results.append(feedback)
  END FOR

  // Calculate aggregate metrics
  metrics = {
    accuracy: mean([r.accuracy for r in results]),
    latency: percentile([r.latency for r in results], 95),
    cost: sum([r.cost for r in results]),
    failure_rate: count([r for r in results if r.failed]) / len(results)
  }

  // Check against quality gates
  gates_passed = TRUE

  IF metrics.accuracy < current_production.accuracy * 0.95 THEN
    gates_passed = FALSE
    reason = "Accuracy regression detected"
  END IF

  IF metrics.failure_rate > current_production.failure_rate * 1.2 THEN
    gates_passed = FALSE
    reason = "Increased failure rate"
  END IF

  IF metrics.latency > current_production.latency * 1.5 THEN
    gates_passed = FALSE
    reason = "Unacceptable latency increase"
  END IF

  IF gates_passed THEN
    RETURN {
```

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

```
      approved: TRUE,
      metrics: metrics
    }
  ELSE
    RETURN {
      approved: FALSE,
      reason: reason,
      metrics: metrics
    }
  END IF
END FUNCTION
```

5.5.2 Gradual Deployment and Rollback

The system deploys improvements gradually with automatic rollback:

```
FUNCTION deploy_strategy(strategy S, deployment_plan P)
  deployment_id = create_deployment(S, P)

  FOR each stage in P.stages DO  // e.g., [1%, 5%, 25%, 100%]
    // Deploy to percentage of traffic
    set_traffic_allocation(deployment_id, stage.percentage)

    // Monitor for issues
    monitoring_window = stage.duration
    metrics = monitor_performance(deployment_id, monitoring_window)

    // Check for regressions
    IF detect_regression(metrics) THEN
      // Automatic rollback
      log_critical(f"Regression detected in deployment {deployment_id}, rolling back")
      rollback_deployment(deployment_id)

      RETURN {
        success: FALSE,
        stage_reached: stage.percentage,
        reason: "Performance regression",
        metrics: metrics
      }
    END IF

    // Check for critical errors
    IF detect_critical_errors(metrics) THEN
      log_critical(f"Critical errors in deployment {deployment_id}, rolling back")
```

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

```
      rollback_deployment(deployment_id)

      RETURN {
        success: FALSE,
        stage_reached: stage.percentage,
        reason: "Critical errors",
        metrics: metrics
      }
    END IF

    // Stage successful, proceed to next
    log_info(f"Stage {stage.percentage}% successful for deployment {deployment_id}")
  END FOR

  // Full deployment successful
  finalize_deployment(deployment_id)

  RETURN {
    success: TRUE,
    final_metrics: get_deployment_metrics(deployment_id)
  }
END FUNCTION
```

6. NOVEL TECHNICAL FEATURES

6.1 Multi-Modal Feedback Integration

NOVELTY: Unified framework for collecting and integrating explicit user feedback, implicit behavioral signals, performance metrics, and execution traces.

PRIOR ART LIMITATIONS: Existing systems use single feedback sources (typically explicit ratings only).

TECHNICAL ADVANTAGE: Richer learning signal enables more accurate attribution and faster improvement.

6.2 Causal Performance Attribution

NOVELTY: Automated attribution of outcomes to specific strategy components using counterfactual analysis and execution trace reconstruction.

PRIOR ART LIMITATIONS: No systematic approach to identify which components contribute to success or failure.

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

TECHNICAL ADVANTAGE: Enables targeted improvements by identifying specific bottlenecks and success factors.

6.3 Automated Strategy Evolution

NOVELTY: Systematic generation of strategy variants through component replacement, augmentation, and hybridization based on performance history.

PRIOR ART LIMITATIONS: Strategy improvements require manual engineering.

TECHNICAL ADVANTAGE: Continuous autonomous improvement without human intervention.

6.4 Safety-Constrained Learning

NOVELTY: Multi-stage deployment with quality gatekeeping, regression detection, and automatic rollback capabilities.

PRIOR ART LIMITATIONS: Learning systems can degrade performance; no safety mechanisms.

TECHNICAL ADVANTAGE: Ensures learned improvements maintain quality standards; prevents performance degradation.

7. IMPLEMENTATION EXAMPLE

7.1 Scenario: Improving Research Query Performance

INITIAL STATE:
- Current strategy: Sequential search → Analysis → Synthesis
- Performance: 85% user satisfaction, 45s average latency

LEARNING CYCLE:

Week 1: Feedback Collection
- Collected 10,000 query executions with feedback
- Identified patterns:
  * Users reformulate 30% of queries (implicit negative signal)
  * High engagement on results with structured data (implicit positive)
  * Explicit complaints about latency on complex queries

Week 2: Attribution Analysis
- Performance attribution revealed:
  * Sequential execution causing 60% of latency
  * Synthesis quality high (0.89) but slow

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

  * Analysis completeness varies by domain (0.65-0.95)

Week 3: Variant Generation & Testing
- Generated variants:
  1. Parallel search + analysis (addresses latency)
  2. Domain-specific analysis modules (addresses completeness variation)
  3. Incremental synthesis (addresses latency without quality loss)
  4. Hybrid: Parallel + Domain-specific + Incremental

- A/B test results (1 week, 5% traffic each):
  * Variant 1: 20% latency improvement, similar satisfaction
  * Variant 2: 15% satisfaction improvement, similar latency
  * Variant 3: 15% latency improvement, 5% satisfaction loss
  * Variant 4: 25% latency improvement, 12% satisfaction improvement

Week 4: Deployment
- Variant 4 selected (statistically significant improvements)
- Gradual deployment: 1% → 5% → 25% → 100% (monitoring at each stage)
- No regressions detected
- Final metrics: 92% satisfaction (+7%), 35s latency (-22%)

Week 5+: Continuous Improvement
- System continues learning
- Identifies new optimization opportunities
- Generates next generation of variants

7.2 Long-term Evolution

Over 6 months:
- 12 successful strategy improvements deployed
- User satisfaction: 85% → 94% (+9%)
- Average latency: 45s → 28s (-38%)
- Cost per query: $0.15 → $0.11 (-27%)
- No quality regressions

8. PATENT CLAIMS

Claim 1 (Independent - System)
A continuous learning system for AI reasoning engines, comprising:
  a) A feedback collection engine configured to:
    - collect explicit user feedback including ratings, corrections, and preferences;
    - detect implicit behavioral signals including engagement time, interaction patterns, and query reformulations;
    - capture performance metrics including latency, accuracy, and resource usage;

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

- record execution traces showing internal operations and decisions;

b) A performance attribution system configured to:
- reconstruct execution traces identifying all components used;
- calculate contribution scores for each component using counterfactual analysis;
- identify success factors and failure causes;
- maintain performance history for components across contexts;

c) A strategy evolution engine configured to:
- generate strategy variants by replacing, removing, or augmenting components;
- conduct controlled A/B experiments comparing variants;
- select winning strategies based on statistical significance;
- synthesize knowledge from successful strategies;

d) A safety and validation system configured to:
- validate new strategies against quality gates;
- deploy strategies gradually across traffic stages;
- monitor for performance regressions;
- automatically rollback deployments if issues detected;

wherein the system continuously improves reasoning capabilities while maintaining quality standards.

Claim 2 (Dependent - Feedback Integration)
The system of claim 1, wherein the feedback collection engine integrates multiple feedback modalities by calculating weighted feedback scores based on signal strength and reliability, combining explicit ratings, implicit behavioral indicators, and objective performance metrics into unified feedback bundles.

Claim 3 (Dependent - Causal Attribution)
The system of claim 1, wherein the performance attribution system calculates component contributions by comparing actual execution outcomes with estimated counterfactual outcomes if the component were not used, assigning contribution scores based on the difference.

Claim 4 (Dependent - Variant Generation)
The system of claim 1, wherein the strategy evolution engine generates variants by:
- replacing low-performing components with alternatives;
- removing non-essential low-performing components;
- applying patterns from high-performing components;
- creating hybrids of successful strategies.

Claim 5 (Dependent - A/B Testing)

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

The system of claim 1, wherein the strategy evolution engine conducts experiments by allocating traffic across control and treatment strategies, monitoring performance metrics, detecting statistically significant differences, and stopping variants that show critical failures.

Claim 6 (Dependent - Quality Gatekeeping)
The system of claim 1, wherein the safety and validation system validates strategies by executing them on validation sets, comparing accuracy, latency, cost, and failure rates against production baselines, and rejecting strategies that show regressions beyond specified thresholds.

Claim 7 (Dependent - Gradual Deployment)
The system of claim 1, wherein the safety and validation system deploys strategies through progressive stages with increasing traffic allocation, monitoring performance at each stage, and automatically rolling back if regressions or critical errors are detected.

Claim 8 (Independent - Method)
A computer-implemented method for continuous learning in AI systems, comprising:
a) collecting multi-modal feedback from executions;
b) attributing performance outcomes to strategy components;
c) identifying high-performing and low-performing components;
d) generating strategy variants based on component performance;
e) conducting controlled experiments testing variants;
f) selecting winning strategies from experiments;
g) validating winners against quality gates;
h) deploying validated strategies gradually with monitoring;
i) rolling back automatically if regressions detected;
j) iterating continuously to improve performance.

Claim 9 (Dependent - Knowledge Transfer)
The method of claim 8, further comprising synthesizing reusable knowledge from successful strategies, storing patterns and component combinations that worked well, and applying learned patterns when generating future strategy variants.

Claim 10 (Independent - Computer-Readable Medium)
A non-transitory computer-readable medium storing instructions that, when executed, cause a processor to:
- implement multi-modal feedback collection;
- perform causal attribution analysis;
- generate and test strategy variants;
- validate strategies with quality gates;
- deploy gradually with automatic rollback;
- continuously improve reasoning performance.

© 2026 Padmashree Malagi. All rights reserved. Confidential - Do Not Distribute

END OF INVENTION DISCLOSURE DOCUMENT

Document prepared: [Current Date]
Inventor: Padmashree Malagi
System: Principia V10.10 Reasoning Engine

© 2026 Padmashree Malagi. All rights reserved.
Confidential - Do Not Distribute

Principia V10.10 Engine | Do Not Distribute | Confidential